

Methods and Applications of Deep Neural Networks

Yao Lu

A thesis submitted for the degree of
Doctor of Philosophy in
Computer Science
The Australian National University

December 2021

© Yao Lu 2021

Except where otherwise indicated, this thesis is my own original work.

Yao Lu
26 December 2021

To my parents

Acknowledgments

Research is tough. It cannot be done alone. Despite the solitude and the hermit lifestyle of some scientists, scientific research is a largely social activity. While science can be learned from books, the craft of scientific research is often taught hand-in-hand from the mentors to the apprentices. As luck would have it, I was supervised by wonderful mentors: Richard Hartley, Mehrtash Harandi and Hongdong Li, who gave me complete freedom and guided me patiently in my adventure. I thank my collaborators Stephen Gould and Thalaiyasingam Ajanthan, with whom I had great discussions in developing the ideas in this thesis. I also thank Hanna Suominen for her support and Thomas Brox for hosting my research visit in his lab at University of Freiburg. Finally, I thank my parents Sen Jia and Hongda Lu for their unconditional support and love.

Abstract

Neural networks are universal function approximators and have been widely used in performing tasks for artificial intelligence. Despite their generality, neural networks are also known to be hard to harness due to their complicated mathematical nature and the sophistication of an application domain. In this thesis, we first address neural network training. Classical optimization literature often fails to provide effective algorithms in practice. This is because the optimization problems associated to neural networks are difficult for their non-linearity and non-convexity. We propose to solve two problems in neural network training: vanishing/exploding gradients and scalability of second-order methods. For each of the problem, we provide a principled approach and provable results. Then, we look at an application of neural networks in computer vision, optical flow estimation. This application was often address with classical optimization techniques such as Markov random fields. However, neural networks when fueled with sufficient training data often outperform the classical techniques. We propose a novel neural network model for optical flow estimation with the principle to solve the “small things moving fast” problem. Experiments on both synthetic and real-world datasets are performed to demonstrate the above methods.

Contents

Acknowledgments	vii
Abstract	ix
1 Introduction	1
1.1 Neural Networks	1
1.2 Problem Statement	2
1.2.1 Vanishing/Exploding Gradients	2
1.2.2 Scalability of Second-Order Optimization Methods	4
1.2.3 Optical Flow Estimation	7
1.3 Contributions	7
2 Bidirectionally Self-Normalizing Neural Networks	9
2.1 Method	9
2.1.1 Constraints on Weight Matrices	10
2.1.2 Constraints on Activation Functions	10
2.1.3 Norm-Preservation Theorems	12
2.2 Experiments	14
2.2.1 Synthetic Data	14
2.2.2 Real-World Data	15
2.3 Discussion	15
2.4 Appendix	23
3 Block-Mean Approximation for Efficient Second-Order Optimization	29
3.1 Method	29
3.1.1 Fast Inversion Theorems	32
3.1.2 BMA for Neural Networks	32
3.1.3 BMA for AdaGrad	33
3.2 Experiments	35
3.3 Discussion	38
3.4 Appendix	38
4 Deformable Volume Network for Learning Optical Flow	43
4.1 Method	44
4.1.1 Deformable Cost Volume	44
4.1.2 Deformable Volume Network	47
4.1.2.1 Encoding Module	49

4.1.2.2	Relation Module	49
4.1.2.3	Decoding Module	51
4.2	Experiments	51
4.2.1	Training	52
4.2.2	Main Results	53
4.2.3	Ablation Analysis	54
4.2.4	Runtime	54
4.3	Discussion	55
5	Conclusion	63

List of Figures

1.1	Gradient norm for each layer.	3
1.2	The effect of parametrization in convergence behavior of gradient descent (with line search).	4
2.1	Results on synthetic data with different activation functions.	17
2.2	Results on synthetic data with different activation functions.	18
2.3	Results on synthetic data with different activation functions.	19
2.4	Gradient norm ratio for different layer width on synthetic data.	20
2.5	Histogram of $\phi'(h_i^{(l)})$	21
2.6	Test accuracy (percentage) during training on MNIST.	22
2.7	Test accuracy (percentage) during training on CIFAR-10.	22
3.1	Expansion matrices.	30
3.2	Block mean approximation of a matrix.	31
3.3	Block mean approximations of the empirical Fisher information matrix of a neural network model.	33
3.4	Performance of AdaGrad and its approximations on two standard datasets.	37
4.1	Artifacts of using image warping.	45
4.2	Cost Volumes	47
4.3	Deformable Volume Network (Devon) with three stages.	48
4.4	Encoding module f	49
4.5	Relation module R	50
4.6	Concatenation of deformable cost volumes creates a retinal structure of correspondences.	50
4.7	Decoding module g	51
4.8	FlyingChairs (validation set).	56
4.9	FlyingChairs (validation set).	56
4.10	FlyingChairs (validation set).	57
4.11	FlyingChairs (validation set).	57
4.12	FlyingChairs (validation set).	58
4.13	FlyingChairs (validation set).	58
4.14	Sintel (training set).	59
4.15	Sintel (training set).	59
4.16	Sintel (training set).	59
4.17	Sintel (training set).	60

4.18 KITTI 2015 (training set).	60
4.19 KITTI 2015 (training set).	60
4.20 KITTI 2015 (training set).	61
4.21 Results on Middlebury.	61
4.22 Results on Middlebury.	62

List of Tables

2.1	Constants for Gaussian-Poincaré normalization of activation functions.	12
2.2	Accuracy (percentage) of neural networks of depth 200 with different activation functions on real-world data.	20
3.1	Small model	36
3.2	Large model	36
3.3	Model parameters	36
3.4	Runtime comparison (ms/iteration)	36
4.1	Hyperparameters of deformable cost volumes in Devon.	50
4.2	Results on Sintel (end-point error).	53
4.3	Detailed results on Sintel (end-point error) for different distances from motion boundaries (d) and velocities (s).	53
4.4	Results on KITTI.	54
4.5	Results of ablation experiments after training on FlyingChairs (end-point error).	55
4.6	Runtime (ms).	55

Introduction

Functions are mathematical objects to represent input-output relationships. Elementary functions such as power functions and trigonometric functions are well studied such that their properties are well understood. On the other hand, some functions are rather complicated. They are hard to grasp analytically and to compute numerically. We therefore hope to approximate these complicated functions by the composition of simpler functions.

The superposition of power functions is known as polynomials. The Weierstrass approximation theorem and its generalizations state that polynomials can approximate any continuous function defined on a closed hypercube with arbitrary accuracy. Despite the celebrated results, approximating multivariate functions by polynomials can be rather cumbersome for high-dimensional inputs in practice. One reason is that the number of terms of multivariate polynomials grows exponentially with the dimensionality of input. It is computationally prohibitive to compute all the terms or to verify whether some terms can be discarded without sacrificing much accuracy. To handle dimensionality gracefully, let us consider another function class with strong approximation power, neural networks.

1.1 Neural Networks

A neural network is a hierarchical parametric function. This is no universally accepted definition of neural networks. But we give the following example.

$$f(\mathbf{x}) = \mathbf{W}^{(3)}\phi(\mathbf{W}^{(2)}\phi(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) + \mathbf{b}^{(2)}) \quad (1.1)$$

where $\mathbf{x}$ is the input vector, $\mathbf{W}^{(l)}$ is a matrix called weight matrix, $\mathbf{b}^{(l)}$ is a vector called bias vector and $\phi : \mathbb{R} \rightarrow \mathbb{R}$ is a differentiable function applied element-wisely to a vector. $\mathbf{W}^{(l)}$ and $\mathbf{b}^{(l)}$ are parameters to be tuned to approximate a target function. The superscript (l) refers to the l -th layer. We do not specify the size of the vectors and the matrices as they can be arbitrary. $f(\mathbf{x})$ is a neural network. More specifically, it is a multi-layer perceptron with two hidden layers.

The universal approximation theorems state that neural networks of the above kind can approximate any continuous functions defined on a compact subset of $\mathbb{R}^d$ [Cybenko, 1989; Hornik et al., 1989]. Different from polynomials, neural networks

work surprisingly well for approximating multivariate functions. One possible explanation is that the hierarchical structure of neural networks enables them to represent certain classes of functions efficiently [Rolnick and Tegmark, 2018].

We aim at approximating a target function $g(\mathbf{x})$ by tuning the parameters of $f(\mathbf{x})$. The target function is often unknown and we can collect samples of input-output $\{\mathbf{x}_i, \mathbf{t}_i\}$. Once an objective function $E(\cdot, \cdot)$ is defined, we tune the neural network parameters $\theta = \{\mathbf{W}^{(l)}, \mathbf{b}^{(l)}\}$ by solving the following minimization problem.

$$\min_{\theta} \sum_i E(f(\mathbf{x}_i; \theta), \mathbf{t}_i) \quad (1.2)$$

The process is called training.

The most popular method for training neural networks is gradient-based. Suppose E is a differentiable function. Then once we compute the gradient, we can update the parameters with a gradient descent algorithm such as

$$\theta \leftarrow \theta - \eta \sum_i \frac{\partial E(f(\mathbf{x}_i), \mathbf{t}_i)}{\partial \theta} \quad (1.3)$$

where η is the step size. Stochastic gradient descent algorithms allow one to update the parameters with the gradient of a single instance of the training set, for which we have the following update rule,

$$\theta \leftarrow \theta - \eta \frac{\partial E(f(\mathbf{x}_i), \mathbf{t}_i)}{\partial \theta}. \quad (1.4)$$

On the application side, neural networks have brought unprecedented performance in various artificial intelligence tasks such as image classification [Ciregan et al., 2012; Krizhevsky et al., 2012], speech recognition [Graves et al., 2013] and game playing [Silver et al., 2017].

1.2 Problem Statement

In this thesis, we address three problems in the optimization methods and applications of neural networks.

1.2.1 Vanishing/Exploding Gradients

Training neural networks is difficult [LeCun et al., 2012]. A major difficulty is the vanishing/exploding gradients problem [Hochreiter, 1991; Bengio et al., 1994; Glorot and Bengio, 2010; Pascanu et al., 2013; Philipp et al., 2018]. That is, the norm of the gradient in each layer is either growing or shrinking from the top layer to bottom layer. For deep neural networks, this problem might cause numerical overflow and make the optimization problem intrinsically difficult, as the gradient in each layer has vastly different magnitude and therefore the optimization landscape becomes pathological.

Consider a simple neural networks of 20 layers. Each layers has width 500. The activation function is $\phi(x) = \frac{1}{1+\exp(-x)}$. Let input $\mathbf{x}$ and $\mathbf{t}$ be two independent random vectors of 500 dimensions sampled from $\mathcal{N}(0, \mathbf{I})$. We use objective function $E(f(\mathbf{x}), \mathbf{t}) = \frac{1}{2} \|\mathbf{f}(\mathbf{x}) - \mathbf{t}\|_2^2$. The gradients are averaged over a mini-batch of size 500.

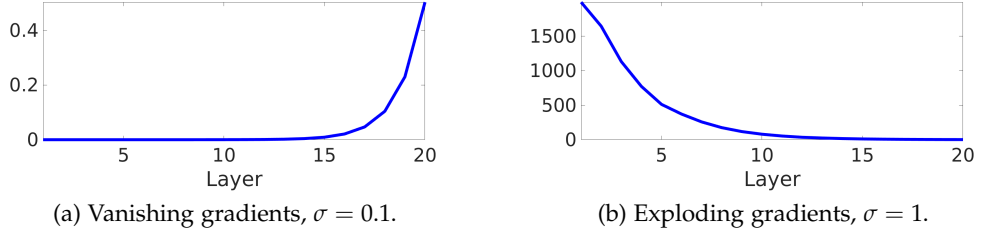


Figure 1.1: The horizontal axis denotes layer index. The vertical axis denotes $\|\frac{\partial E}{\partial \mathbf{W}_l}\|_F$. Each $\mathbf{W}_l$ is sampled i.i.d from $\mathcal{N}(0, \sigma^2 \mathbf{I})$.

To see how the vanishing/exploding gradients problem can affect neural network training, consider the following regression problem. Let $f(\mathbf{x}; \boldsymbol{\theta}) : \mathbb{R}^d \rightarrow \mathbb{R}$ be a neural network where $\boldsymbol{\theta}$ is the vectorization of its parameters. Given a training set $\{(\mathbf{x}_i, t_i)\}$ where each $\mathbf{x}_i \in \mathbb{R}^d$ and each $t_i \in \mathbb{R}$, we aim to minimize the following objective function,

$$\sum_i \frac{1}{2} (f(\mathbf{x}_i; \boldsymbol{\theta}) - t_i)^2 \quad (1.5)$$

Let $g(\mathbf{x}; \boldsymbol{\theta})$ be the first order Taylor expansion of $f(\mathbf{x}; \boldsymbol{\theta})$ at $\boldsymbol{\theta}_0$.

$$g(\mathbf{x}; \boldsymbol{\theta}) = f(\mathbf{x}, \boldsymbol{\theta}_0) + \nabla_{\boldsymbol{\theta}} f(\mathbf{x}; \boldsymbol{\theta}_0)^\top (\boldsymbol{\theta} - \boldsymbol{\theta}_0) \quad (1.6)$$

Then for $\boldsymbol{\theta} \approx \boldsymbol{\theta}_0$, $g(\mathbf{x}; \boldsymbol{\theta}) \approx f(\mathbf{x}; \boldsymbol{\theta})$. Substitute $f(\mathbf{x}; \boldsymbol{\theta})$ with $g(\mathbf{x}; \boldsymbol{\theta})$, we have the new objective function

$$\sum_i \frac{1}{2} (g(\mathbf{x}_i; \boldsymbol{\theta}) - t_i)^2 \quad (1.7)$$

which approximates the original objective function. The Hessian matrix of the new objective function is

$$\mathbf{H} = \sum_i \nabla_{\boldsymbol{\theta}} f(\mathbf{x}_i; \boldsymbol{\theta}_0) \nabla_{\boldsymbol{\theta}} f(\mathbf{x}_i; \boldsymbol{\theta}_0)^\top \quad (1.8)$$

The condition number of $\mathbf{H}$ is the ratio of the largest eigenvalue of $\mathbf{H}$ to the smallest eigenvalue, assuming both eigenvalues are positive. If the condition number is small, it implies that the steepest direction does not change significantly after a small step of update and therefore means we can make a larger stepsize.

If the gradients are balanced, then one would expect the diagonal elements of $\mathbf{H}$ to be equal, which implies smaller condition number. For any Hessian matrix $\mathbf{H}$, we

have always find a diagonal matrix $\mathbf{D}$ such that matrix $\mathbf{DHD}$ has equal diagonal elements. This transformation is called diagonal preconditioning, which often reduces the condition number of $\mathbf{H}$.

To enable the trainability of deep neural networks, batch normalization [Ioffe and Szegedy, 2015] was proposed in recent years and achieved widespread empirical success. Batch normalization is a differentiable operation which normalizes its inputs based on mini-batch statistics and is inserted between the linear and nonlinear layers. It is reported that batch normalization can accelerate neural network training significantly [Ioffe and Szegedy, 2015]. However, batch normalization does not solve the vanishing/exploding gradients problem [Philipp et al., 2018]. Indeed it is proved that batch normalization can actually worsen the problem [Yang et al., 2019]. Besides, batch normalization requires separate training and testing phases and can be ineffective when the mini-batch size is small [Ioffe, 2017].

1.2.2 Scalability of Second-Order Optimization Methods

In practice, gradient descent often converges slowly and its performance depends critically on how $f(\theta)$ is parameterized. That is, if one chooses $\theta = g(\xi)$, then the behavior of $\theta \leftarrow \theta - \eta \nabla_{\theta} f(\theta)$ can be significantly different from that of $\xi \leftarrow \xi - \eta \nabla_{\xi} f(g(\xi))$.

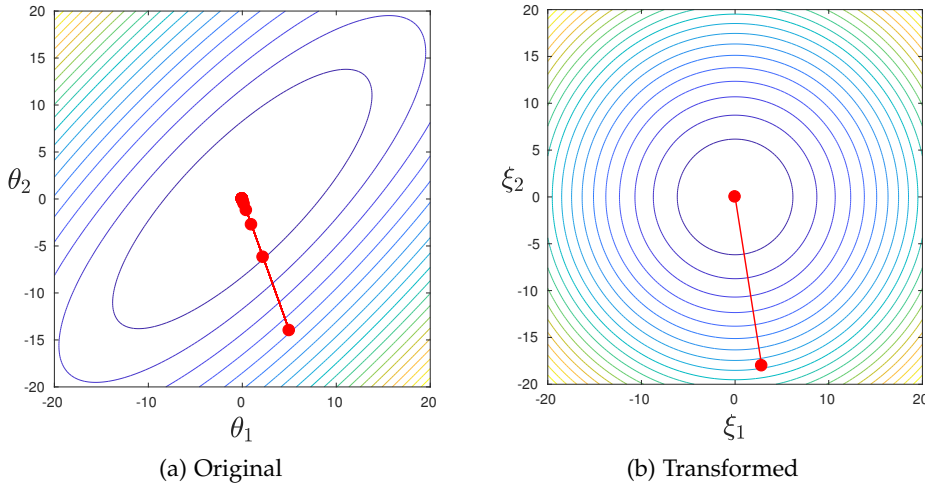


Figure 1.2: (a) Gradient descent moves in a zig-zag fashion as it approaches to the minimum. (b) By reparameterization the same problem, gradient descent finds the minimum in only one step (see the text for more details).

To show how the dependency of parameters affects the speed of gradient descent, consider a simple quadratic minimization problem

$$\min_{\theta} f(\theta) = \frac{1}{2} \theta^T \mathbf{Q} \theta \quad (1.9)$$

where

$$\boldsymbol{\theta} = [\theta_1, \theta_2]^\top, \quad \mathbf{Q} = \begin{bmatrix} 1.0 & -0.8 \\ -0.8 & 1.0 \end{bmatrix}. \quad (1.10)$$

The eigenvalues of $\mathbf{Q}$ are $\lambda_1 = 1.8$ and $\lambda_2 = 0.2$. The minimum is obtained at $\boldsymbol{\theta} = \mathbf{0}$. We note that the scale of θ_1 equals that of θ_2 . However, the gradient of θ_1 depends on the value of θ_2 and vice versa as

$$\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) = \mathbf{Q}\boldsymbol{\theta}. \quad (1.11)$$

The gradient descent method, even with the optimal step size for each iteration, converges with a rate of $(\frac{\lambda_1 - \lambda_2}{\lambda_1 + \lambda_2})^2$ with such a parameterization. However, with eigen-decomposition $\mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top = \mathbf{Q}$ and reparameterization with $\boldsymbol{\xi} = [\xi_1, \xi_2]^\top = \mathbf{\Lambda}^{\frac{1}{2}}\mathbf{U}\boldsymbol{\theta}$, then the optimization problem becomes

$$\min_{\boldsymbol{\xi}} f(\boldsymbol{\xi}) = \frac{1}{2}\boldsymbol{\xi}^\top \boldsymbol{\xi}. \quad (1.12)$$

The gradient of ξ_1 is not dependent on ξ_2 anymore and vice versa. As a result, gradient descent with the optimal step size converges to the minimum solution in only one iteration (see Figure 1.2 for an illustration). The take home message from this textbook example is that algorithms that exploit the dependency between the parameters can prevail over the ones that totally ignore such information.

To accelerate gradient descent and deal better with the parameterization issue, one can resort to second order optimization methods, in which the second order derivative (Hessian) of $f(\cdot)$ or second order statistics of gradients is incorporated in $\mathbf{G}$. In the Newton method, $\mathbf{G}$ is chosen as the Hessian matrix of $f(\cdot)$ according to

$$\mathbf{G}_{ij} = \frac{\partial^2 f(\boldsymbol{\theta})}{\partial \theta_i \partial \theta_j}. \quad (1.13)$$

The Newton method approximates the loss function locally by a quadratic function, in which the Hessian matrix measures the curvature of the loss function. This in turn yields better descent directions than the ones obtained by solely considering the gradient directions. In fact, under mild conditions, the Newton method converges to a local minimum at a quadratic rate while gradient descent has a linear convergence rate [Nocedal and Wright, 2006]. Besides, the Newton method is invariant to affine re-parameterization (see the derivation in Appendix).

The concept of natural gradient [Amari, 1998] provides another perspective by conditioning gradient step on the KL-divergence variations induced by the model output distribution. That is, any given step in the natural gradient method produces an equal amount of variation in terms of the KL-divergence. It is shown that the method of natural gradient is invariant to the parameterization of the model [Pascanu and Bengio, 2014]. Approximating the KL-divergence variations by its second order Taylor series, one arrives at a form that looks like the Newton method, albeit with $\mathbf{G}$

becoming the Fisher information matrix

$$\mathbf{G} = \mathbb{E}_{\mathbf{x} \sim p(\mathbf{x}|\boldsymbol{\theta})} \left[\frac{\partial \log p(\mathbf{x}|\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \frac{\partial \log p(\mathbf{x}|\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}^\top \right] \quad (1.14)$$

where $p(\mathbf{x}|\boldsymbol{\theta})$ is the probabilistic model we try to optimize. It has been shown that the Fisher information matrix can be viewed as an approximated Hessian matrix [Martens, 2014]. Various studies suggest that natural gradient can have a better convergence rate than that of the gradient descent method (e.g., , blind signal separation [Amari et al., 1996], reinforcement learning [Peters and Schaal, 2008] and variational inference [Honkela et al., 2010]).

For stochastic optimization, the AdaGrad algorithm [Duchi et al., 2011] incorporates the previously computed gradients to guide its current descent direction. In AdaGrad, $\mathbf{G}$ is the matrix square root of an outer product matrix of gradient vectors

$$\mathbf{G} = \left(\sum_{\tau=1}^t \mathbf{g}_\tau \mathbf{g}_\tau^\top \right)^{\frac{1}{2}} \quad (1.15)$$

where $\mathbf{g}_\tau = \nabla_{\boldsymbol{\theta}} f_\tau(\boldsymbol{\theta}_\tau)$ is the gradient estimated from a mini-batch of data at step τ and t is the current step. The relationship between the matrix $\sum_{\tau=1}^t \mathbf{g}_\tau \mathbf{g}_\tau^\top$ and the Hessian matrix is discussed in [Hazan et al., 2007].

Despite their intriguing properties and fast convergence rates, the aforementioned second order optimization methods become overwhelming for high-dimensional $\boldsymbol{\theta}$. This is because construction and inversion of $\mathbf{G}$ have a time complexity of $O(d^3)$. As such, various studies resort to approximation techniques when it comes to high-dimensional problems.

A simple, yet effective approximation is the diagonal approximation, where one only keeps the diagonal elements of $\mathbf{G}$. A classic method is the Levenberg-Marquardt algorithm [Levenberg, 1944; Marquardt, 1963] which uses the diagonal approximation of the Hessian matrix. Its stochastic version for training neural networks is proposed in [Becker and LeCun, 1988]. The diagonal approximation of AdaGrad and its variants such as AdaDelta [Zeiler, 2012], RMSprop [Tieleman and Hinton, 2012] and Adam [Kingma and Ba, 2014] have seen increasing popularity in training neural networks recently.

The diagonal approximation of $\mathbf{G}$ amounts to setting an individual learning rate for each parameter. This balances the scale of parameters and often accelerates the convergence. Nevertheless, one wonders whether disregarding the correlation between the parameters, as captured by the off-diagonal elements of $\mathbf{G}$, has negative effects on the convergence speed. In Chapter 3, we will show that the answer to this question is a firm yes. A simple example will be given. Experiments also show the benefits of capturing off-diagonal elements of $\mathbf{G}$.

1.2.3 Optical Flow Estimation

Optical flow estimation is the problem of finding pixel motions between two images. It is a classic computer vision problem and has been studied for more than 30 years. Since Horn and Schunck’s variational method [Horn and Schunck, 1981], a large number of optical flow algorithms [Beauchemin and Barron, 1995; Fleet and Weiss, 2006; Fortun et al., 2015] has been developed. Nevertheless, the problem of estimating optical flow is not yet solved. One can even claim that we still do not have a practical algorithm which is fast, robust and accurate for real-world images.

Recently, supervised learning of optical flow with neural networks has shown great promises [Dosovitskiy et al., 2015; Ilg et al., 2017; Ranjan and Black, 2017; Teney and Hebert, 2016; Thewlis et al., 2016; Sun et al., 2018b; Hui et al., 2018]. With graphics engines, large scale datasets with synthetic images and ground-truth optical flow are generated [Butler et al., 2012; Dosovitskiy et al., 2015; Mayer et al., 2016], which enables end-to-end learning optical flow. We note that while training images are synthetic, neural networks seem to generalize reasonably well on real-world images [Mayer et al., 2018]. Compared to classic methods, the neural network approaches have the advantage of offering robust features and fast inference. For example, models in [Ilg et al., 2017] have a running speed from 8-140 frames-per-second and accuracy competitive with the state-of-the-art methods.

1.3 Contributions

For the vanishing/exploding gradients problem, we have introduced bidirectionally self-normalizing neural networks (BSNNs) which constrain both forward and backward signal propagation using Gaussian-Poincaré normalized activation functions and orthogonal weight matrices. The work is described in the following paper.

- Bidirectionally Self-Normalizing Neural Networks
Yao Lu, Stephen Gould, Thalaiyasingam Ajanthan
arXiv preprint arXiv:2006.12169, 2020.

For the scalability of second-order optimization methods, in this work, we propose a new matrix approximation method which allows efficient storage and computation of matrix inverse and inverse square root that are required by second-order optimization methods. The work is described in the following paper.

- Block Mean Approximation for Efficient Second Order Optimization
Yao Lu, Mehrtash Harandi, Richard Hartley, Razvan Pascanu
International Conference on Machine Learning Workshop on Modern Trends in Nonconvex Optimization for Machine Learning, 2018.

For optical flow estimation, we proposed the Deformable Volume Network, which is demonstrated to be effective in estimating optical flow. The work is described in the following paper.

- Devon: Deformable Volume Network for Learning Optical Flow
Yao Lu, Jack Valmadre, Heng Wang, Juho Kannala, Mehrtaash Harandi, Philip Torr
IEEE Winter Conference on Applications of Computer Vision, 2020.

Bidirectionally Self-Normalizing Neural Networks

In this chapter, we propose a new type of neural networks that consist of orthogonal weight matrices and a new class of activation functions which we call Gaussian-Poincaré normalized (GPN) functions. We show many common activation functions can be easily transformed into their respective GPN versions. Next, we rigorously prove that the vanishing/exploding gradients problem disappears with high probability in the neural networks if the width of each layer is sufficiently large. Finally, with experiments on synthetic and real-world data, we confirm that the vanishing/-exploding gradients problem is solved to large extent in the neural networks while nonlinear functionality is maintained. All the proofs are left in the appendix.

2.1 Method

To simplify the analysis, we define neural network in a restricted sense as the following.

Definition 1 (Neural Network). *A neural network is a function $f : \mathbb{R}^d \rightarrow \mathbb{R}^d$ that for $l = 1, \dots, L$*

$$\begin{aligned} \mathbf{h}^{(l)} &= \mathbf{W}^{(l)} \mathbf{x}^{(l)}, \\ \mathbf{x}^{(l+1)} &= \phi(\mathbf{h}^{(l)}), \end{aligned} \tag{2.1}$$

where $\mathbf{W}^{(l)} \in \mathbb{R}^{d \times d}$, $\phi : \mathbb{R} \rightarrow \mathbb{R}$ is a differentiable function applied element-wisely to a vector, $\mathbf{x}^{(1)}$ is the input and $\mathbf{x}^{(L+1)}$ is the output.

Under this definition, ϕ is called the activation function, $\{\mathbf{W}^{(l)}\}_{l=1}^L$ are called the parameters, d is called the width and L is called the depth and superscript (l) denotes the l -th layer of a neural network. The above formulation is similar to [Pennington et al., 2017] but we omit the bias term in (2.1) for simplicity as it plays no role in our analysis.

Let E be the objective function of $\{\mathbf{W}^{(l)}\}_{l=1}^L$, $h_i^{(l)}$ be the i -th element of $\mathbf{h}^{(l)}$ and

$$\mathbf{D}^{(l)} = \text{diag}(\phi'(h_1^{(l)}), \dots, \phi'(h_d^{(l)})), \quad (2.2)$$

where ϕ' denotes the derivative of ϕ . Now, the backward signal $\mathbf{y}^{(l)}$ is propagated via

$$\begin{aligned} \mathbf{y}^{(L)} &= \mathbf{D}^{(L)} \frac{\partial E}{\partial \mathbf{x}^{(L+1)}}, \\ \mathbf{y}^{(l)} &= \mathbf{D}^{(l)} (\mathbf{W}^{(l+1)})^T \mathbf{y}^{(l+1)}, \end{aligned} \quad (2.3)$$

and the gradient of the weight matrix for layer l can be computed as

$$\frac{\partial E}{\partial \mathbf{W}^{(l)}} = \mathbf{y}^{(l)} (\mathbf{x}^{(l)})^T. \quad (2.4)$$

To solve the vanishing/exploding gradients problem, we constrain the forward signal $\mathbf{x}^{(l)}$ and the backward signal $\mathbf{y}^{(l)}$ in order to constrain the norm of the gradient. This leads to the following.

Definition 2 (Bidirectional Self-Normalization). *A neural network is bidirectionally self-normalizing if*

$$\|\mathbf{x}^{(1)}\|_2 = \dots = \|\mathbf{x}^{(L)}\|_2 = \sqrt{d}, \quad (2.5)$$

$$\|\mathbf{y}^{(1)}\|_2 = \dots = \|\mathbf{y}^{(L)}\|_2 = \left\| \frac{\partial E}{\partial \mathbf{x}^{(L+1)}} \right\|_2. \quad (2.6)$$

Proposition 1. *If a neural network is bidirectionally self-normalizing, then*

$$\left\| \frac{\partial E}{\partial \mathbf{W}^{(1)}} \right\|_F = \dots = \left\| \frac{\partial E}{\partial \mathbf{W}^{(L)}} \right\|_F. \quad (2.7)$$

In the rest of this section, we derive the conditions under which bidirectional self-normalization is achievable for a neural network.

2.1.1 Constraints on Weight Matrices

We constrain the weight matrices to be orthogonal since multiplication by an orthogonal matrix preserves the norm of a vector. For linear neural networks, this guarantees bidirectional self-normalization and its further benefits are discussed in [Saxe et al., 2014]. Even for nonlinear neural networks, orthogonal constraints are shown to improve the trainability with proper scaling [Mishkin and Matas, 2016; Pennington et al., 2017].

2.1.2 Constraints on Activation Functions

To achieve bidirectional self-normalization for a nonlinear network, it is not enough only to constrain the weight matrices. We also need to constrain the activation func-

tion in such a way that both forward and backward signals are normalized. To this end, we propose the following constraint.

Definition 3 (Gaussian-Poincaré Normalization). *A function $\phi : \mathbb{R} \rightarrow \mathbb{R}$ is Gaussian-Poincaré normalized if it is differentiable and*

$$\mathbb{E}_{x \sim \mathcal{N}(0,1)}[\phi(x)^2] = \mathbb{E}_{x \sim \mathcal{N}(0,1)}[\phi'(x)^2] = 1. \quad (2.8)$$

The definition is motivated by the following theorem which shows the fundamental relationship between a function and its derivative under Gaussian measure.

Theorem 1 (Gaussian-Poincaré Inequality [Bogachev, 1998]). *If function $\phi : \mathbb{R} \rightarrow \mathbb{R}$ is differentiable with bounded $\mathbb{E}_{x \sim \mathcal{N}(0,1)}[\phi(x)^2]$ and $\mathbb{E}_{x \sim \mathcal{N}(0,1)}[\phi'(x)^2]$, then*

$$\text{Var}_{x \sim \mathcal{N}(0,1)}[\phi(x)] \leq \mathbb{E}_{x \sim \mathcal{N}(0,1)}[\phi'(x)^2]. \quad (2.9)$$

Note that there is an implicit assumption that the input is approximately Gaussian for a Gaussian-Poincaré normalized (GPN) function. Even though this is standard in the literature [Klambauer et al., 2017; Pennington et al., 2017; Schoenholz et al., 2017], we will rigorously prove that this assumption is valid when orthogonal weight matrices are used in (2.1). Next, we state a property of GPN functions.

Proposition 2. *Function $\phi : \mathbb{R} \rightarrow \mathbb{R}$ is Gaussian-Poincaré normalized and $\mathbb{E}_{x \sim \mathcal{N}(0,1)}[\phi(x)] = 0$ if and only if $\phi(x) = x$ or $\phi(x) = -x$.*

This result indicates that any nonlinear function with zero mean under Gaussian distribution (e.g., Tanh and SELU) is not GPN. Now we show that a large class of activation functions can be converted into their respective GPN versions using an affine transformation.

Proposition 3. *For any differentiable function $\phi : \mathbb{R} \rightarrow \mathbb{R}$ with non-zero and bounded $\mathbb{E}_{x \sim \mathcal{N}(0,1)}[\phi(x)^2]$ and $\mathbb{E}_{x \sim \mathcal{N}(0,1)}[\phi'(x)^2]$, there exist two constants a and b such that $a\phi(x) + b$ is Gaussian-Poincaré normalized.*

To obtain a and b , one can use numerical procedure to compute the values of $\mathbb{E}_{x \sim \mathcal{N}(0,1)}[\phi'(x)^2]$, $\mathbb{E}_{x \sim \mathcal{N}(0,1)}[\phi(x)^2]$ and $\mathbb{E}_{x \sim \mathcal{N}(0,1)}[\phi(x)]$ and then solve the quadratic equations

$$\mathbb{E}_{x \sim \mathcal{N}(0,1)}[a^2 \phi'(x)^2] = 1, \quad (2.10)$$

$$\mathbb{E}_{x \sim \mathcal{N}(0,1)}[(a\phi(x) + b)^2] = 1. \quad (2.11)$$

We computed a and b (not unique) for several common activation functions [Nair and Hinton, 2010; Maas et al., 2013; Clevert et al., 2016; Klambauer et al., 2017; Hendrycks and Gimpel, 2016] with their default hyperparameters¹ and the results are listed in Table 1. Note that ReLU, LeakyReLU and SELU are not differentiable at $x = 0$ but

	Tanh	ReLU	LeakyReLU	ELU	SELU	GELU
a	1.4674	1.4142	1.4141	1.2234	0.9660	1.4915
b	0.3885	0.0000	0.0000	0.0742	0.2585	-0.9097

Table 2.1: Constants for Gaussian-Poincaré normalization of activation functions.

they can be regarded as approximations of their smooth counterparts. We ignore such point and evaluate the integrals for $x \in (-\infty, 0) \cup (0, \infty)$.

With the orthogonal constraint on the weight matrices and the Gaussian-Poincaré normalization on the activation function, we prove that bidirectional self-normalization is achievable with high probability under mild conditions in the next subsection.

2.1.3 Norm-Preservation Theorems

Bidirectional self-normalization may not be achievable precisely in general unless the neural network is a linear one. Therefore, we investigate the properties of neural networks in a probabilistic framework. Random matrix theory and high-dimensional probability theory allow us to characterize the behavior of a large class of neural networks by their mean behavior, which is significantly simpler to analyze. Therefore, we study neural networks of random weights whose properties may shed light on the trainability of neural networks in practice.

First, we need a probabilistic version of the vector norm constraint.

Definition 4 (Thin-Shell Concentration). *Random vector $\mathbf{x} \in \mathbb{R}^d$ is thin-shell concentrated if for any $\epsilon > 0$*

$$\mathbb{P}\left\{\left|\frac{1}{d}\|\mathbf{x}\|_2^2 - 1\right| \geq \epsilon\right\} \rightarrow 0 \quad (2.12)$$

as $d \rightarrow \infty$.

The definition is modified from the one in [Bobkov, 2003]. Examples of thin-shell concentrated distributions include standard multivariate Gaussian and any distribution on the d -dimensional sphere of radius $\sqrt{d}$.

To prove the main results, i.e., the norm-preservation theorems, we require the following assumptions.

Assumptions

1. Random vector $\mathbf{x} \in \mathbb{R}^d$ is thin-shell concentrated.
2. Random orthogonal matrix $\mathbf{W} = (\mathbf{w}_1, \dots, \mathbf{w}_d)^T$ is uniformly distributed.
3. Function $\phi : \mathbb{R} \rightarrow \mathbb{R}$ is Gaussian-Poincaré normalized.
4. Function $\phi : \mathbb{R} \rightarrow \mathbb{R}$ and its derivative are Lipschitz continuous.

¹We use $\phi(x) = \max(0, x) + 0.01 \min(0, x)$ for LeakyReLU, $\phi(x) = \max(0, x) + \min(0, \exp(x) - 1)$ for ELU and $\phi(x) = x / (1 + \exp(-1.702x))$ for GELU.

The above assumptions are not restrictive. For Assumption 1, one can always normalize the input vectors of a neural network. For Assumption 2, orthogonal constraint or its relaxation has already been employed in neural network training [Brock et al., 2017]. Note, in Assumption 2, uniformly distributed means that $\mathbf{W}$ is distributed under Haar measure, which is the unique rotation invariant probability measure on orthogonal matrix group. We refer the reader to [Meckes, 2019] for details. Furthermore, all the activation functions or their smooth counterparts listed in Table 2.1 satisfy Assumptions 3 and 4.

With the above assumptions, we can prove the following norm-preservation theorems.

Theorem 2 (Forward Norm-Preservation). *Under Assumptions 1-4, random vector*

$$(\phi(\mathbf{w}_1^T \mathbf{x}), \dots, \phi(\mathbf{w}_d^T \mathbf{x})) \quad (2.13)$$

is thin-shell concentrated.

This result shows the transformation (orthogonal matrix followed by the GPN activation function) can preserve the norm of its input with high probability. Since the output is thin-shell concentrated, it serves as the input for the next layer and so on. Hence, the forward pass can preserve the norm of its input in each layer along the forward path when d is sufficiently large.

Theorem 3 (Backward Norm-Preservation). *Under Assumptions 1-4, let*

$$\mathbf{D} = \text{diag}(\phi'(\mathbf{w}_1^T \mathbf{x}), \dots, \phi'(\mathbf{w}_d^T \mathbf{x})) \quad (2.14)$$

and $\mathbf{y} \in \mathbb{R}^d$ be a fixed vector with bounded $\|\mathbf{y}\|_\infty$. Then for any $\epsilon > 0$

$$\mathbb{P}\left\{\frac{1}{d}\left|\|\mathbf{D}\mathbf{y}\|_2^2 - \|\mathbf{y}\|_2^2\right| \geq \epsilon\right\} \rightarrow 0 \quad (2.15)$$

as $d \rightarrow \infty$.

This result shows that the multiplication by the diagonal matrix $\mathbf{D}$ preserves the norm of its input with high probability. Since orthogonal matrix $\mathbf{W}$ also preserves the norm of its input, when the gradient error signal is propagated backwards as in (2.3), the norm is preserved in each layer along the backward path when d is sufficiently large.

Hence, combining Theorems 2 and 3, we proved that bidirectional self-normalization is achievable with high probability if the neural network is wide enough and the conditions in the Assumptions are satisfied. Then by Proposition 1, the vanishing/-exploding gradients problem disappears with high probability.

The proofs of Theorems 2 and 3 are mainly based on a phenomenon in high-dimensional probability spaces, concentration of measure. We refer the reader to [Vershynin, 2018] for an introduction to the subject. Briefly, it can be shown that for some high-dimensional probability distributions, most mass is concentrated around

certain range. For example, while most mass of a low-dimensional standard multivariate Gaussian distribution is concentrated around the center, most mass of a high-dimensional standard multivariate Gaussian distribution is concentrated around a thin-shell. Based on this phenomenon, it can be shown that random vector $\mathbf{W}\mathbf{x}$ in high dimensions is approaching a random vector uniformly distributed on a sphere. Then the random vector uniformly distributed on a high-dimensional sphere is approximately Gaussian. Then the Gaussian random variables transformed by Lipschitz and GPN functions are subgaussian with unit variance. And the random vector of the subgaussian random variables has the concentration of norm property in high dimensions. Each of these steps is rigorously proved in Appendix.

2.2 Experiments

We verify our theory on both synthetic and real-world data. More experimental results can be found in Appendix. In short, while very deep neural networks with non-GPN activations show vanishing/exploding gradients, GPN versions show stable gradients and improved trainability in both synthetic and real data. Furthermore, compared to dynamical isometry theory, BSNNs do not exhibit pseudo-linearity and maintain nonlinear functionality.

2.2.1 Synthetic Data

We create synthetic data to test the norm-preservation properties of the neural networks. The input $\mathbf{x}^{(1)}$ is 500 data points of random standard Gaussian vectors of 500 dimensions. The gradient error $\partial E / \partial \mathbf{x}^{(L+1)}$ is also random standard Gaussian vector of 500 dimensions. All the neural networks have depth 200. All the weight matrices are random orthogonal matrices uniformly generated. No training is performed.

In Figure 2.1, we show the norm of inputs and gradients of the neural networks of width 500. From the results, we can see that with GPN, the vanishing/exploding gradients problem is eliminated to large extent. The neural network with Tanh activation function does not show the vanishing/exploding gradients problem either. However, $\|\mathbf{x}^{(l)}\|$ is close to zero for large l and each layer is close to a linear one since $\text{Tanh}(x) \approx x$ when $x \approx 0$ (pseudo-linearity), for which dynamical isometry is achieved.

One might wonder if bidirectional self-normalization has the same effect as dynamical isometry in solving the vanishing/exploding gradients problem, that is, to make the neural network close to an orthogonal matrix. To answer this question, we show the histogram of $\phi'(h_i^{(l)})$ in Figure 2.5. If the functionality of a neural network is close to an orthogonal matrix, since the weight matrices are orthogonal, then the values of $\phi'(h_i^{(l)})$ would concentrate around one (Figure 2.5 (a)), which is not the case for BSNNs (Figure 2.5 (b)). This shows that BSNNs do not suffer from the vanishing/exploding gradients problem while exhibiting nonlinear functionality.

In Figure 2.4, we show the gradient norm of BSNNs with varying width. Notice,

as the width increases, the norm of gradient in each layer of the neural network becomes more equalized, as predicted by our theory.

2.2.2 Real-World Data

We run experiments on real-world image datasets MNIST and CIFAR-10. The neural networks have width 500 and depth 200 (plus one unconstrained layer at bottom and one at top to fit the dimensionality of the input and output). We use stochastic gradient descent of momentum 0.5 with mini-batch size 64 and learning rate 0.0001. The training is run for 50 epochs for MNIST and 100 epochs for CIFAR-10. We do not use data augmentation. Since it is computationally expensive to enforce the orthogonality constraint, we simply constrain each row of the weight matrix to have l_2 norm one as a relaxation of orthogonality by the following parametrization $\mathbf{W} = (\mathbf{v}_1/\|\mathbf{v}_1\|_2, \dots, \mathbf{v}_d/\|\mathbf{v}_d\|_2)^T$ and optimize $\mathbf{V} = (\mathbf{v}_1, \dots, \mathbf{v}_d)^T$ as an unconstrained problem.

We summarize the results in Table 2.2. We can see that, for activation functions ReLU, LeakyReLU and GELU, the neural networks are not trainable. But once these functions are GPN, the neural network can be trained. GPN activation functions consistently outperform their unnormalized counterparts in terms of the trainability, as the training accuracy is increased, but not necessarily generalization ability.

We show the test accuracy during training in Figure 2.7, from which we can see the training is accelerated when SELU is GPN. ReLU, LeakyReLU and GELU, if not GPN, are completely untrainable due to the vanished gradients (see Appendix).

We observe that batch normalization leads to gradient explosion when combining with any of the activation functions. This confirms the claim of [Philipp et al., 2018; Yang et al., 2019] that batch normalization does not solve the vanishing/exploding gradients problem. On the other hand, without batch normalization the neural network with any GPN activation function has stable gradient magnitude throughout training (see Appendix). This indicates that BSNs can dispense batch normalization and therefore avoid its shortcomings.

2.3 Discussion

We compare our theory to several most relevant theories in literature. A key distinguishing feature of our theory is that we provide rigorous proofs of the conditions under which the vanishing/exploding gradients problem disappears. To the best of our knowledge, this is the first time that the problem is provably solved for nonlinear neural networks.

Self-normalizing neural networks enforce zero mean and unit variance for the output of each unit with the SELU activation function [Klambauer et al., 2017]. However, as pointed out in [Philipp et al., 2018], only constraining forward signal propagation does not solve the vanishing/exploding gradients problem since the norm of the backward signal can grow or shrink. In [Philipp et al., 2018] and our experiments,

SELU is indeed shown to cause gradient exploding. To solve the problem, the signal propagation in both directions need to be constrained, as in our theory.

Our theory is largely developed from the deep signal propagation theory [Poole et al., 2016; Schoenholz et al., 2017]. Both theories require $\mathbb{E}_{x \sim \mathcal{N}(0,1)}[\phi'(x)^2] = 1$. However, ours also requires the quantity $\mathbb{E}_{x \sim \mathcal{N}(0,1)}[\phi(x)^2]$ to be one while in [Poole et al., 2016; Schoenholz et al., 2017] it can be an arbitrary positive number. We emphasize that it is desirable to enforce $\mathbb{E}_{x \sim \mathcal{N}(0,1)}[\phi(x)^2] = 1$ to avoid trivial solutions. For example, if $\phi(x) = \text{Tanh}(\epsilon x)$ with $\epsilon \approx 0$, then $\phi(\epsilon x) \approx \epsilon x$ and the neural network becomes essentially a linear one for which depth is unnecessary (pseudo-linearity [Philipp et al., 2018]). This is observed in Figure 2.1 (a). Moreover, in [Poole et al., 2016; Schoenholz et al., 2017] the signal propagation analysis is done based on random weights under i.i.d. Gaussian distribution whereas we proved how one can solve the vanishing/exploding gradients problem assuming the weight matrices are orthogonal and uniformly distributed under Haar measure.

Dynamical isometry theory [Pennington et al., 2017] enforces the Jacobian matrix of the input-output function of a neural network to have all singular values close to one. Since the weight matrices are constrained to be orthogonal, it is equivalent to enforce each $\mathbf{D}^{(l)}$ in (2.3) to be close to the identity matrix, which implies the functionality of neural network at initialization is close to an orthogonal matrix (pseudo-linearity). This indeed enables trainability since linear neural networks with orthogonal weight matrices do not suffer from the vanishing/exploding gradients problem. As neural networks need to learn a nonlinear input-output functionality to solve certain tasks, during training the weights of a neural network are unconstrained so that the neural network would move to a nonlinear region where the vanishing/exploding gradients problem might return. In our theory, although the orthogonality of weight matrices is also required, we approach the problem from a different perspective. We do not encourage the linearity at initialization. The neural network can be initialized to be nonlinear and stay nonlinear during the training even when the weights are constrained. This is shown in §2.2.1.

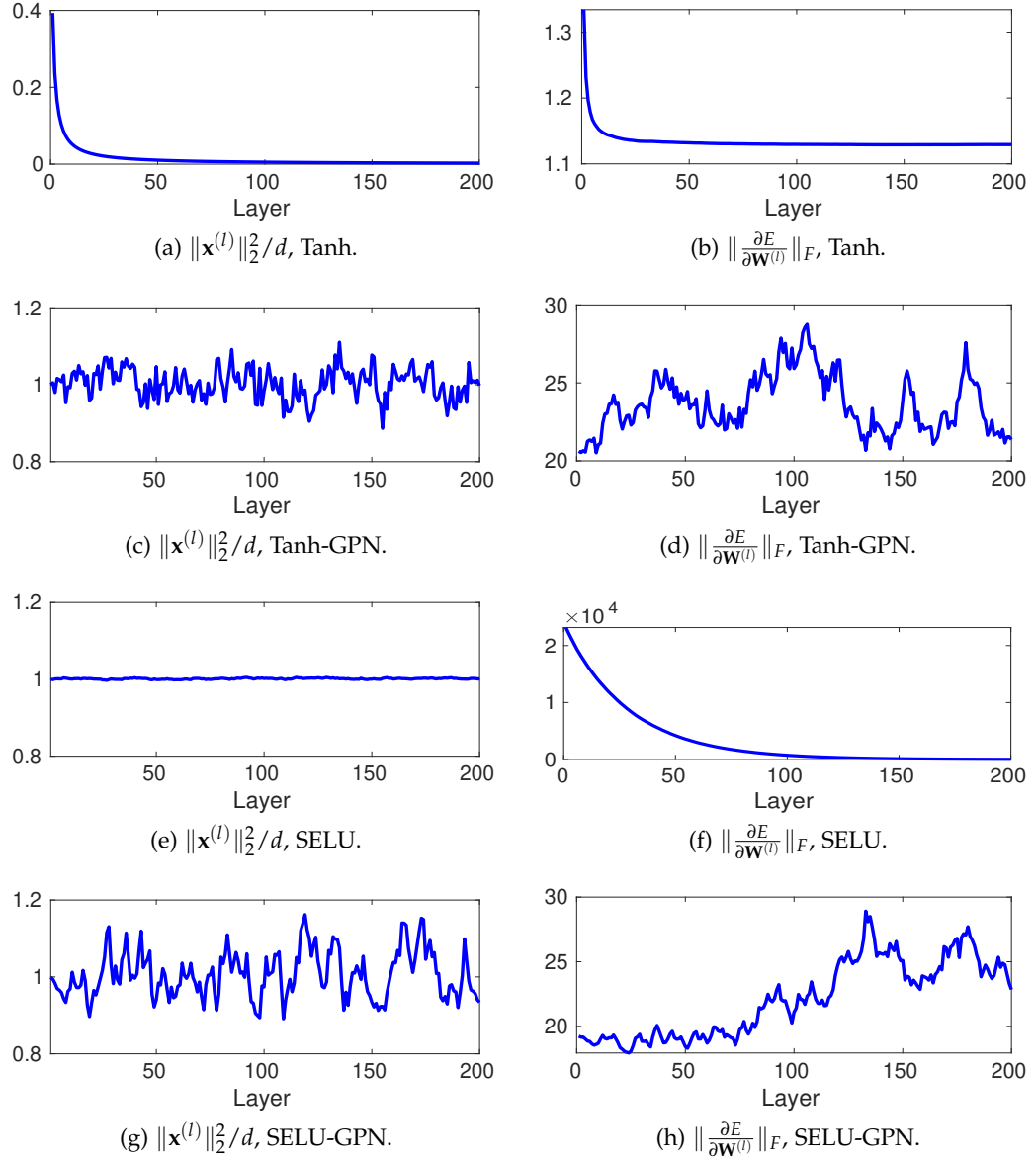


Figure 2.1: “-GPN” denotes the function is Gaussian-Poincaré normalized. $\|\mathbf{x}^{(l)}\|_2$ denotes the l_2 norm of the outputs of the l -th layer. d denotes the width. $\|\frac{\partial E}{\partial \mathbf{W}^{(l)}}\|_F$ is the Frobenius norm of the gradient of the weight matrix in the l -th layer.

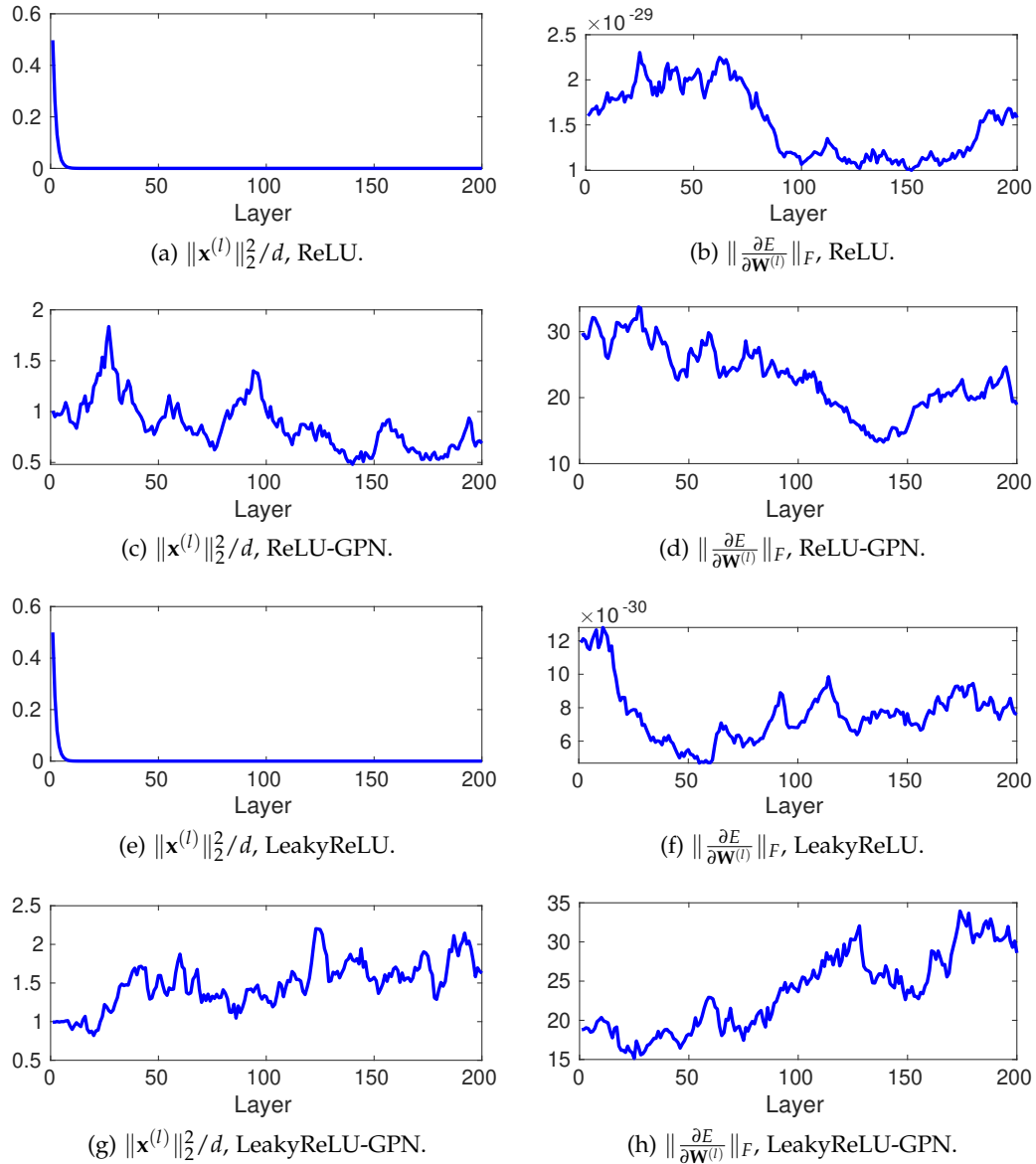


Figure 2.2: “-GPN” denotes the function is Gaussian-Poincaré normalized. $\|\mathbf{x}^{(l)}\|_2$ denotes the l_2 norm of the outputs of the l -th layer. d denotes the width. $\|\frac{\partial E}{\partial \mathbf{W}^{(l)}}\|_F$ is the Frobenius norm of the gradient of the weight matrix in the l -th layer.

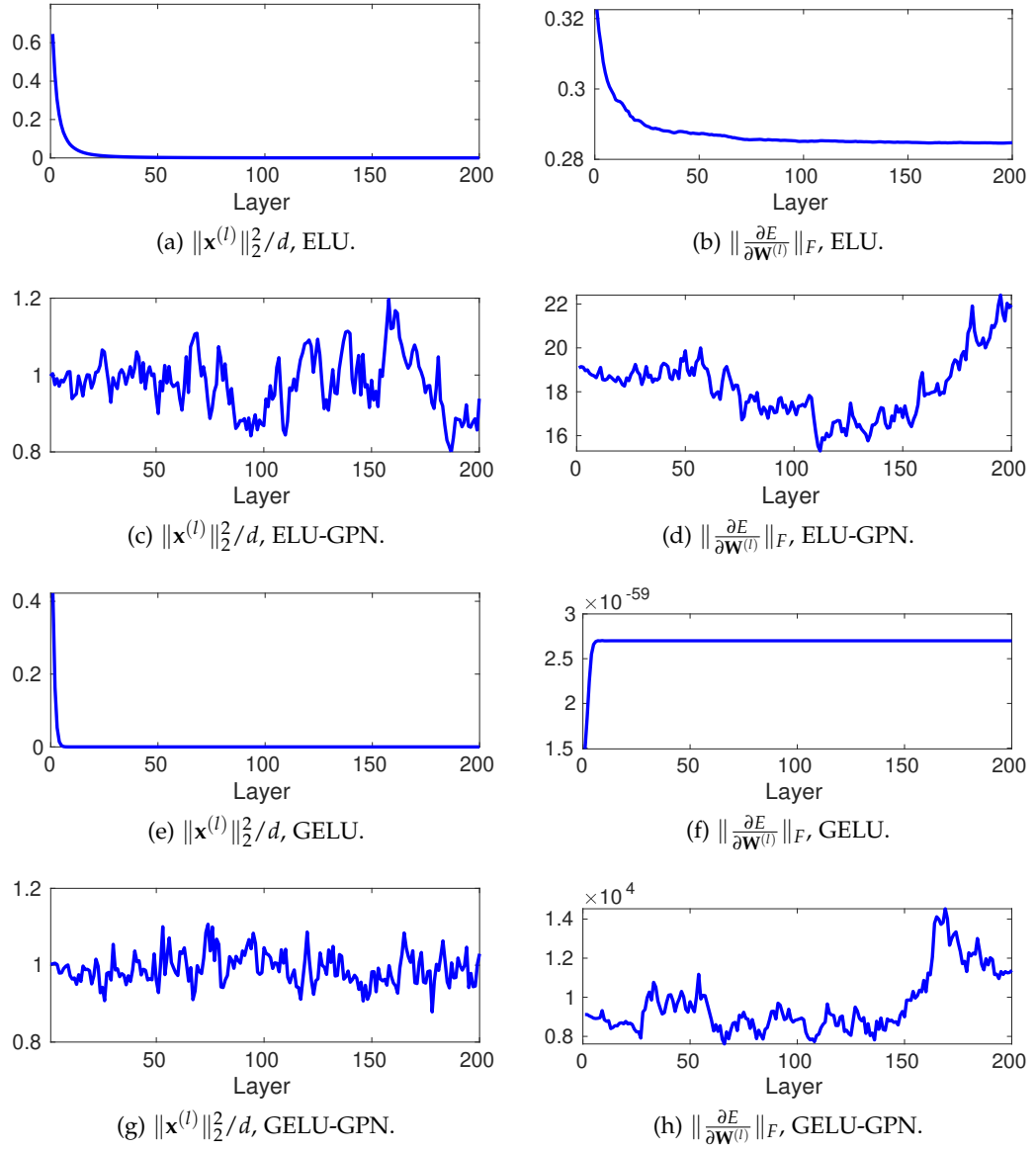


Figure 2.3: “-GPN” denotes the function is Gaussian-Poincaré normalized. $\|\mathbf{x}^{(l)}\|_2$ denotes the l_2 norm of the outputs of the l -th layer. d denotes the width. $\|\frac{\partial E}{\partial \mathbf{W}^{(l)}}\|_F$ is the Frobenius norm of the gradient of the weight matrix in the l -th layer.

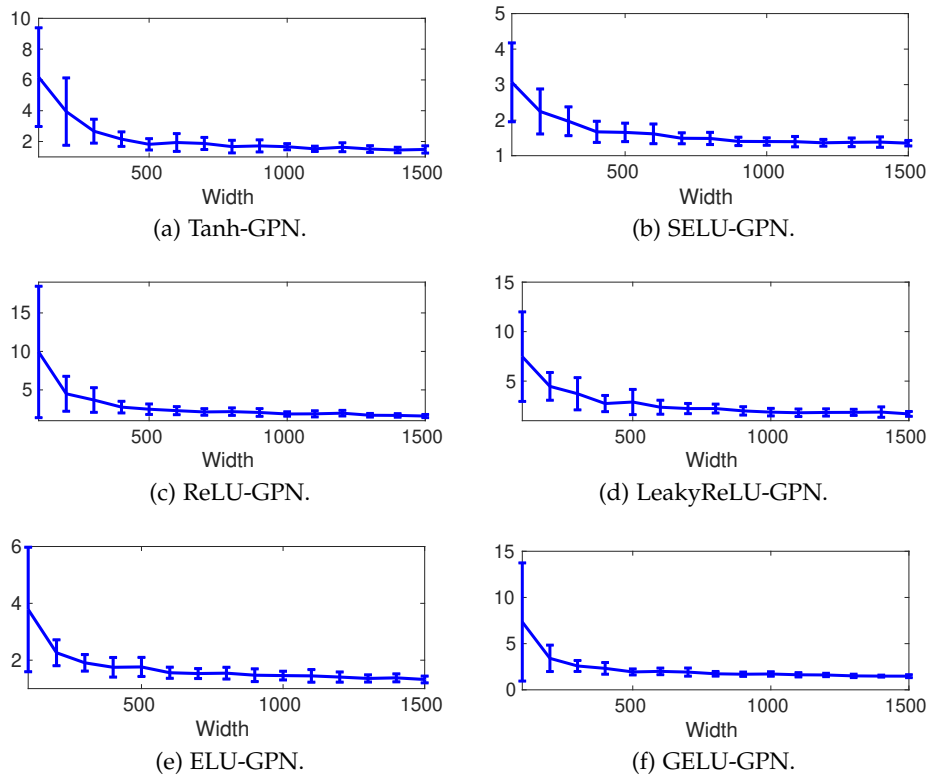


Figure 2.4: The ratio is defined as $\max_l \|\frac{\partial E}{\partial \mathbf{w}^{(l)}}\|_F / \min_l \|\frac{\partial E}{\partial \mathbf{w}^{(l)}}\|_F$. The width ranges from 100 to 1500. The error bars show standard deviation.

	MNIST		CIFAR-10	
	Train	Test	Train	Test
Tanh	99.05 (87.39)	96.57 (89.32)	80.84 (27.90)	42.71 (29.32)
Tanh-GPN	99.81 (84.93)	95.54 (87.11)	96.39 (25.13)	40.95 (26.58)
ReLU	11.24 (11.24)	11.35 (11.42)	10.00 (10.00)	10.00 (10.00)
ReLU-GPN	33.28 (11.42)	28.13 (11.34)	46.60 (10.09)	34.96 (9.96)
LeakyReLU	11.24 (11.24)	11.35 (11.63)	10.00 (10.21)	10.00 (10.06)
LeakyReLU-GPN	43.17 (11.19)	49.28 (11.66)	51.85 (9.89)	39.38 (10.00)
ELU	99.06 (98.24)	95.41 (97.48)	80.73 (42.39)	45.76 (44.16)
ELU-GPN	100.00 (97.86)	96.56 (96.69)	99.37 (43.35)	43.12 (44.36)
SELU	99.86 (97.82)	97.33 (97.38)	29.23 (46.47)	29.55 (45.88)
SELU-GPN	99.92 (97.91)	96.97 (97.39)	98.24 (47.74)	45.90 (45.52)
GELU	11.24 (12.70)	11.35 (10.28)	10.00 (10.43)	10.00 (10.00)
GELU-GPN	97.67 (11.22)	95.82 (9.74)	90.51 (10.00)	36.94 (10.00)

Table 2.2: The numbers in parenthesis denote the results when batch normalization is applied before the activation function.

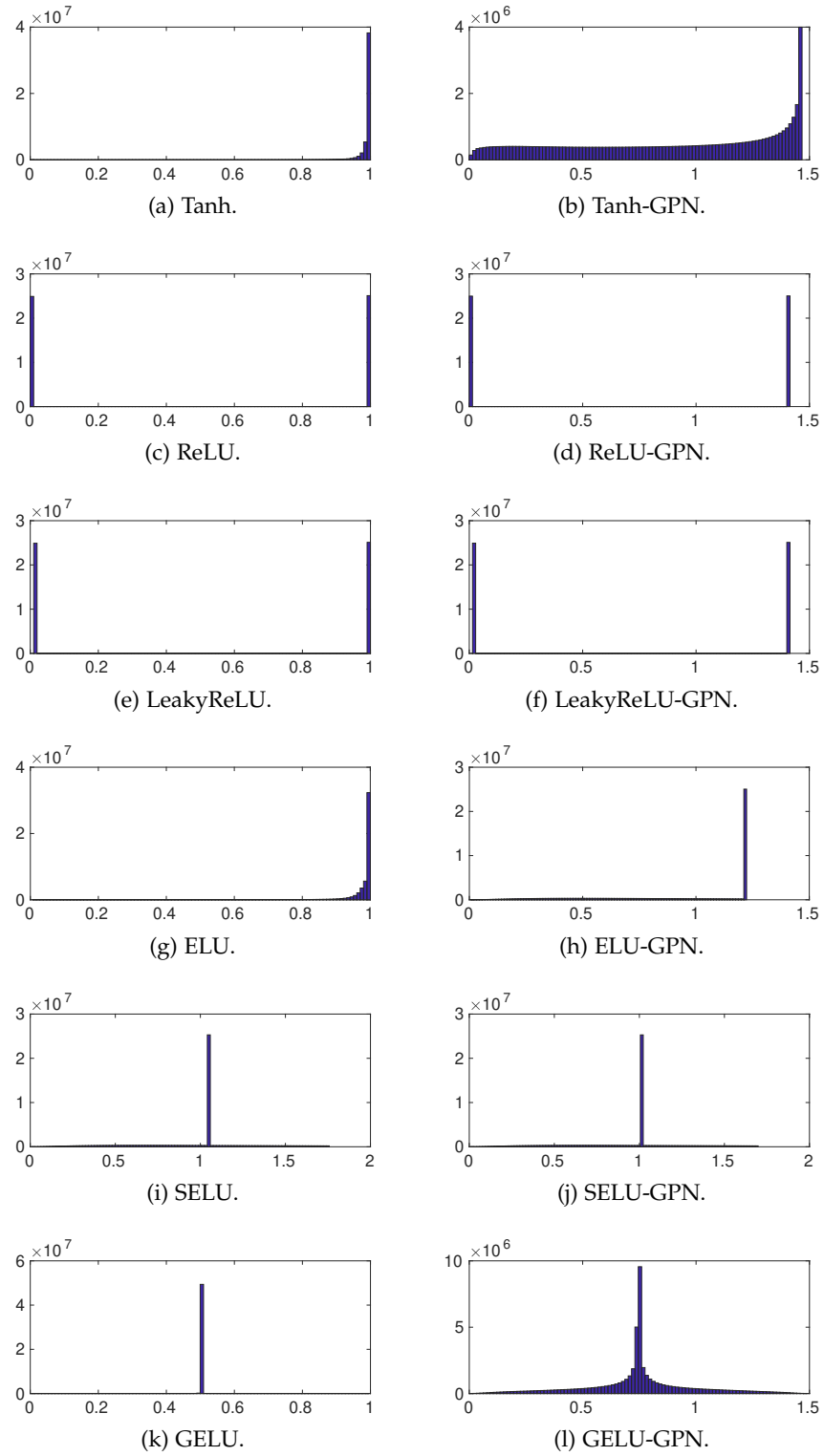


Figure 2.5: The values of $\phi'(h_i^{(l)})$ are accumulated for all units, all layers and all samples in the histogram.

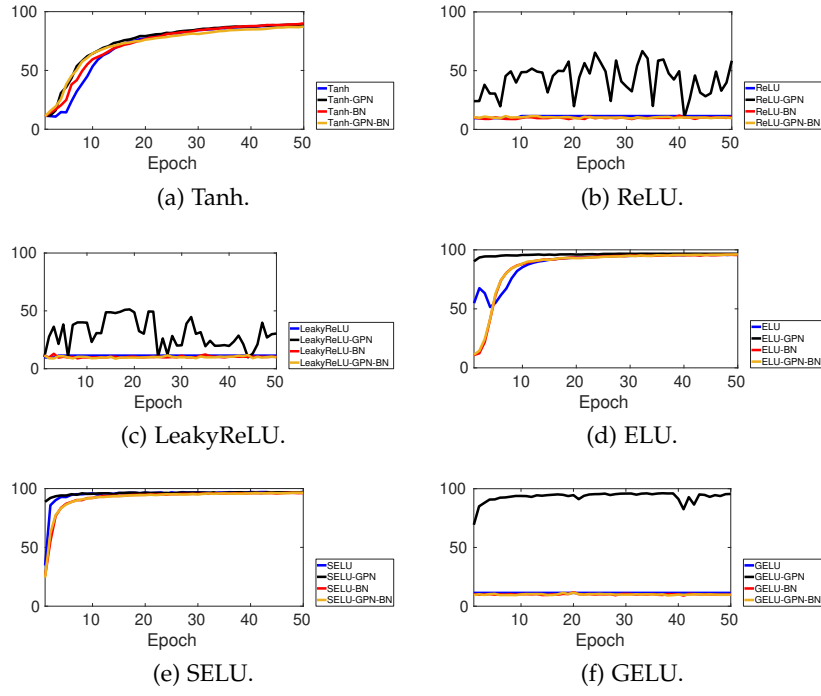


Figure 2.6: Test accuracy (percentage) during training on MNIST.

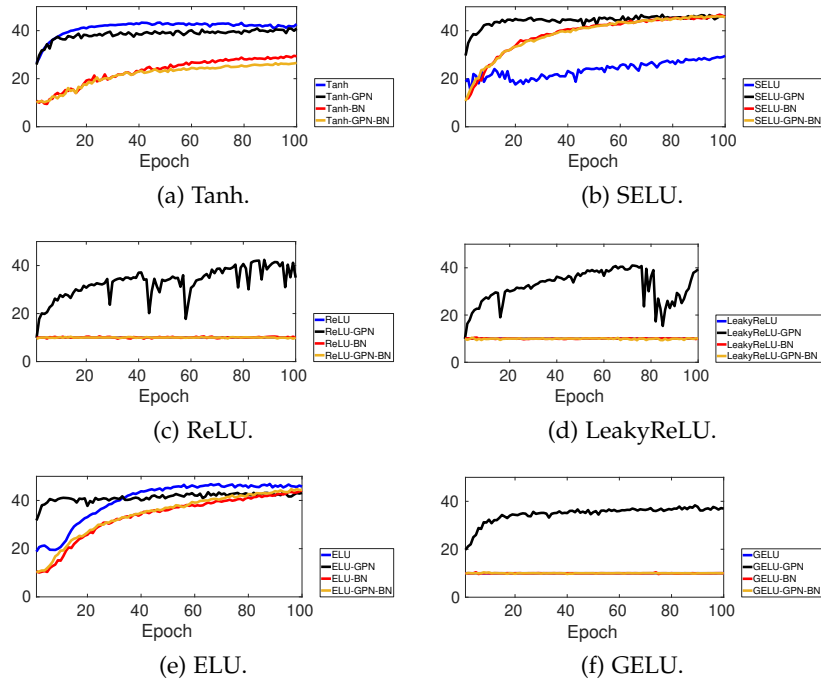


Figure 2.7: “-BN” denotes that batch normalization is applied before the activation function.

2.4 Appendix

Proposition 1. *If a neural network is bidirectionally self-normalizing, then*

$$\left\| \frac{\partial E}{\partial \mathbf{W}^{(1)}} \right\|_F = \dots = \left\| \frac{\partial E}{\partial \mathbf{W}^{(L)}} \right\|_F. \quad (2.14)$$

Proof. For each l , we have

$$\left\| \frac{\partial E}{\partial \mathbf{W}^{(l)}} \right\|_F = \sqrt{\text{trace} \left(\frac{\partial E}{\partial \mathbf{W}^{(l)}} \left(\frac{\partial E}{\partial \mathbf{W}^{(l)}} \right)^T \right)} \quad (2.15)$$

$$= \sqrt{\text{trace}(\mathbf{y}^{(l)} (\mathbf{x}^{(l)})^T \mathbf{x}^{(l)} (\mathbf{y}^{(l)})^T)} \quad (2.16)$$

$$= \sqrt{\text{trace}((\mathbf{x}^{(l)})^T \mathbf{x}^{(l)} (\mathbf{y}^{(l)})^T \mathbf{y}^{(l)})} \quad (2.17)$$

$$= \sqrt{(\mathbf{x}^{(l)})^T \mathbf{x}^{(l)}} \sqrt{(\mathbf{y}^{(l)})^T \mathbf{y}^{(l)}} \quad (2.18)$$

$$= \|\mathbf{x}^{(l)}\|_2 \|\mathbf{y}^{(l)}\|_2. \quad (2.19)$$

By the definition of bidirectional self-normalization, we have $\left\| \frac{\partial E}{\partial \mathbf{W}^{(1)}} \right\|_F = \dots = \left\| \frac{\partial E}{\partial \mathbf{W}^{(L)}} \right\|_F$. $\square$

Proposition 2. *Function $\phi : \mathbb{R} \rightarrow \mathbb{R}$ is Gaussian-Poincaré normalized and $\mathbb{E}_{x \sim \mathcal{N}(0,1)}[\phi(x)] = 0$ if and only if $\phi(x) = x$ or $\phi(x) = -x$.*

Proof. Since $\mathbb{E}_{x \sim \mathcal{N}(0,1)}[\phi(x)^2] < \infty$ and $\mathbb{E}_{x \sim \mathcal{N}(0,1)}[\phi'(x)^2] < \infty$, $\phi(x)$ and $\phi'(x)$ can be expanded in terms of Hermite polynomials. Let the Hermite polynomial of degree k be

$$H_k(x) = \frac{(-1)^k}{\sqrt{k!}} \exp\left(\frac{x^2}{2}\right) \frac{d^k}{dx^k} \exp\left(-\frac{x^2}{2}\right) \quad (2.20)$$

and due to $H'_k(x) = \sqrt{k} H_{k-1}(x)$, we have

$$\phi(x) = \sum_{k=0}^{\infty} a_k H_k(x), \quad (2.21)$$

$$\phi'(x) = \sum_{k=1}^{\infty} \sqrt{k} a_k H_{k-1}(x). \quad (2.22)$$

Since $\mathbb{E}_{x \sim \mathcal{N}(0,1)}[\phi(x)] = 0$, we have

$$a_0 = \mathbb{E}_{x \sim \mathcal{N}(0,1)}[H_0(x) \phi(x)] \quad (2.23)$$

$$= \mathbb{E}_{x \sim \mathcal{N}(0,1)}[\phi(x)] \quad (2.24)$$

$$= 0. \quad (2.25)$$

Since

$$\mathbb{E}_{x \sim \mathcal{N}(0,1)}[\phi(x)^2] = \mathbb{E}_{x \sim \mathcal{N}(0,1)}[\phi'(x)^2] = 1 \quad (2.26)$$

and Hermite polynomials are orthonormal, we have

$$\mathbb{E}_{x \sim \mathcal{N}(0,1)}[\phi(x)^2] = \sum_{k=1}^{\infty} a_k^2 = \mathbb{E}_{x \sim \mathcal{N}(0,1)}[\phi'(x)^2] = \sum_{k=1}^{\infty} k a_k^2 = 1. \quad (2.27)$$

Therefore, we have

$$\sum_{k=1}^{\infty} k a_k^2 - \sum_{k=1}^{\infty} a_k^2 = 0 \quad (2.28)$$

that is

$$\sum_{k=2}^{\infty} (k-1) a_k^2 = 0. \quad (2.29)$$

Since each term in $\sum_{k=2}^{\infty} (k-1) a_k^2$ is nonnegative, the only solution is $a_k = 0$ for $k \geq 2$. And since $\mathbb{E}_{x \sim \mathcal{N}(0,1)}[\phi(x)^2] = a_1^2 = 1$, we have $a_1 = \pm 1$. Hence, $\phi(x) = \pm H_1(x) = \pm x$. $\square$

Proposition 3. *For any differentiable function $\phi : \mathbb{R} \rightarrow \mathbb{R}$ with non-zero and bounded $\mathbb{E}_{x \sim \mathcal{N}(0,1)}[\phi(x)^2]$ and $\mathbb{E}_{x \sim \mathcal{N}(0,1)}[\phi'(x)^2]$, there exist two constants a and b such that $a\phi(x) + b$ is Gaussian-Poincaré normalized.*

Proof. Let $\varphi(x) = \phi(x) + c$. Then let

$$\psi(c) = \mathbb{E}_{x \sim \mathcal{N}(0,1)}[\varphi(x)^2] - \mathbb{E}_{x \sim \mathcal{N}(0,1)}[(\phi'(x))^2] \quad (2.30)$$

$$= \text{Var}_{x \sim \mathcal{N}(0,1)}[\phi(x)] + (\mathbb{E}_{x \sim \mathcal{N}(0,1)}[\phi(x)])^2 - \mathbb{E}_{x \sim \mathcal{N}(0,1)}[(\phi'(x))^2] \quad (2.31)$$

$$= \text{Var}_{x \sim \mathcal{N}(0,1)}[\phi(x)] + (\mathbb{E}_{x \sim \mathcal{N}(0,1)}[\phi(x)] + c)^2 - \mathbb{E}_{x \sim \mathcal{N}(0,1)}[(\phi'(x))^2]. \quad (2.32)$$

Therefore, $\psi(c)$ is a quadratic function of c . We also have $\psi(c) > 0$ as $c \rightarrow \infty$ and $\psi(-\mathbb{E}_{x \sim \mathcal{N}(0,1)}[\phi(x)]) \leq 0$ due to Gaussian-Poincaré inequality. Hence, there exists c for which $\psi(c) = 0$ such that $\mathbb{E}_{x \sim \mathcal{N}(0,1)}[(\phi(x) + c)^2] = \mathbb{E}_{x \sim \mathcal{N}(0,1)}[(\phi'(x))^2]$. Let $a = (\mathbb{E}_{x \sim \mathcal{N}(0,1)}[(\phi'(x))^2])^{-1/2}$ and $b = ac$, we have $\mathbb{E}_{x \sim \mathcal{N}(0,1)}[(a\phi(x) + b)^2] = \mathbb{E}_{x \sim \mathcal{N}(0,1)}[(a\phi'(x))^2] = 1$. $\square$

The proof is largely due to [Eldredge, 2020] with minor modification in here.

Assumptions.

1. Random vector $\mathbf{x} \in \mathbb{R}^d$ is thin-shell concentrated.
2. Random orthogonal matrix $\mathbf{W} = (\mathbf{w}_1, \dots, \mathbf{w}_d)^T$ is uniformly distributed.
3. Function $\phi : \mathbb{R} \rightarrow \mathbb{R}$ is Gaussian-Poincaré normalized.
4. Function ϕ and its derivative are Lipschitz continuous.

Theorem 2 (Forward Norm-Preservation). *Under Assumptions 1-4, random vector*

$$(\phi(\mathbf{w}_1^T \mathbf{x}), \dots, \phi(\mathbf{w}_d^T \mathbf{x})) \quad (2.33)$$

is thin-shell concentrated.

Theorem 3 (Backward Norm-Preservation). *Under Assumptions 1-4, let*

$$\mathbf{D} = \text{diag}(\phi'(\mathbf{w}_1^T \mathbf{x}), \dots, \phi'(\mathbf{w}_d^T \mathbf{x})) \quad (2.34)$$

and $\mathbf{y} \in \mathbb{R}^d$ be a fixed vector with bounded $\|\mathbf{y}\|_\infty$. Then for any $\epsilon > 0$

$$\mathbb{P}\left\{\frac{1}{d} \left| \|\mathbf{D}\mathbf{y}\|_2^2 - \|\mathbf{y}\|_2^2 \right| \geq \epsilon\right\} \rightarrow 0 \quad (2.35)$$

as $d \rightarrow \infty$.

Notation. $\mathbb{S}^{d-1} = \{\mathbf{x} \in \mathbb{R}^d : \|\mathbf{x}\|_2 = 1\}$. $\mathbf{O}(d)$ is the orthogonal matrix group of size d . $\mathbf{1}_{\{\cdot\}}$ denotes the indicator function. $\mathbf{0}_d$ denotes the vector of dimension d and all elements equal to zero. $\mathbf{I}_d$ denotes the identity matrix of size $d \times d$.

Lemma 1. *If random variable $x \sim \mathcal{N}(0, 1)$ and function $f : \mathbb{R} \rightarrow \mathbb{R}$ is Lipschitz continuous, then random variable $f(x)$ is sub-gaussian.*

Proof. Due to the Gaussian concentration theorem (Theorem 5.2.2 in [Vershynin, 2018]), we have

$$\|f(x) - \mathbb{E}[f(x)]\|_{\psi_2} \leq CK \quad (2.36)$$

where $\|\cdot\|_{\psi_2}$ denotes sub-gaussian norm, C is a constant and K is the Lipschitz constant of f . This implies $f(x) - \mathbb{E}[f(x)]$ is sub-gaussian (Proposition 2.5.2 in [Vershynin, 2018]). Therefore $f(x)$ is sub-gaussian (Lemma 2.6.8 in [Vershynin, 2018]). $\square$

Lemma 2. *Let $\mathbf{x} = (x_1, \dots, x_d) \in \mathbb{R}^d$ be a random vector that each coordinate x_i is independent and sub-gaussian and $\mathbb{E}[x_i^2] = 1$. Let $\mathbf{y} = (y_1, \dots, y_d) \in \mathbb{R}^d$ be a fixed vector with bounded $\|\mathbf{y}\|_\infty$. Then*

$$\mathbb{P}\left\{\frac{1}{d} \left| \sum_i x_i^2 y_i^2 - \sum_i y_i^2 \right| \geq \epsilon\right\} \rightarrow 0 \quad (2.37)$$

as $d \rightarrow \infty$.

Proof. Since $y_i x_i$ is sub-gaussian, then $y_i^2 x_i^2$ is sub-exponential (Lemma 2.7.6 in [Vershynin, 2018]). Since $\mathbb{E}[y_i^2 x_i^2] = y_i^2 \mathbb{E}[x_i^2] = y_i^2$, $y_i^2 x_i^2 - y_i^2$ is sub-exponential with zero mean (Exercise 2.7.10 in [Vershynin, 2018]). Applying Bernstein's inequality (Corollary 2.8.3 in [Vershynin, 2018]), we proved the lemma. $\square$

Lemma 3. *Let $\mathbf{z} \sim \mathcal{N}(\mathbf{0}_d, \mathbf{I}_d)$. Then for any $0 < \delta < 1$*

$$\mathbb{P}\{\mathbf{z} \in \mathbb{R}^d : (1 - \delta)\sqrt{d} \leq \|\mathbf{z}\|_2 \leq (1 + \delta)\sqrt{d}\} \geq 1 - 2\exp(-d\delta^2). \quad (2.38)$$

Proof. See [Alberts and Khoshnevisan, 2018] (Theorem 1.2). $\square$

Lemma 4. Let $\mathbf{z} \sim \mathcal{N}(\mathbf{0}_d, \mathbf{I}_d)$. Then $\mathbf{z}/\|\mathbf{z}\|_2$ is uniformly distributed on $\mathbb{S}^{d-1}$.

Proof. See [Dawkins, 2016]. $\square$

Lemma 5. Let $\mathbf{z} = (z_1, \dots, z_d) \sim \mathcal{N}(\mathbf{0}_d, \mathbf{I}_d)$, $\mathbf{a} = (a_1, \dots, a_d)$ be a fixed vector with bounded $\|\mathbf{a}\|_\infty$ and $f : \mathbb{R} \rightarrow \mathbb{R}$ be a continuous function. Then for any $\epsilon > 0$

$$\mathbb{P}\left\{\frac{1}{d}\left|\sum_i a_i f(\sqrt{d}/\|\mathbf{z}\|_2 z_i) - \sum_i a_i f(z_i)\right| \geq \epsilon\right\} \rightarrow 0 \quad (2.39)$$

as $d \rightarrow \infty$.

Proof. Since

$$\frac{1}{d}\left|\sum_i a_i f(\sqrt{d}/\|\mathbf{z}\|_2 z_i) - \sum_i a_i f(z_i)\right| \leq \frac{1}{d} \sum_i |a_i| \cdot |f(\sqrt{d}/\|\mathbf{z}\|_2 z_i) - f(z_i)|, \quad (2.40)$$

if, as $d \rightarrow \infty$,

$$\mathbb{P}\left\{\frac{1}{d} \sum_i |a_i| \cdot |f(\sqrt{d}/\|\mathbf{z}\|_2 z_i) - f(z_i)| \geq \epsilon\right\} \rightarrow 0, \quad (2.41)$$

then

$$\mathbb{P}\left\{\frac{1}{d}\left|\sum_i a_i f(\sqrt{d}/\|\mathbf{z}\|_2 z_i) - \sum_i a_i f(z_i)\right| \geq \epsilon\right\} \rightarrow 0. \quad (2.42)$$

For $0 < \delta < 1$, let

$$A = \left\{\mathbf{z} \in \mathbb{R}^d : \frac{1}{d} \sum_i |a_i| \cdot |f(\sqrt{d}/\|\mathbf{z}\|_2 z_i) - f(z_i)| \geq \epsilon\right\}, \quad (2.43)$$

$$\mathcal{U}_\delta = \left\{\mathbf{z} \in \mathbb{R}^d : (1 - \delta)\sqrt{d} \leq \|\mathbf{z}\|_2 \leq (1 + \delta)\sqrt{d}\right\}. \quad (2.44)$$

Then

$$\mathbb{P}\left\{\frac{1}{d} \sum_i |a_i| \cdot |f(\sqrt{d}/\|\mathbf{z}\|_2 z_i) - f(z_i)| \geq \epsilon\right\} = \int_{\mathbb{R}^d} \mathbf{1}_{\{\mathbf{z} \in A\}} \gamma(\mathbf{z}) d\mathbf{z} \quad (2.45)$$

$$= \int_{\mathbb{R}^d \setminus \mathcal{U}_\delta} \mathbf{1}_{\{\mathbf{z} \in A\}} \gamma(\mathbf{z}) d\mathbf{z} + \int_{\mathcal{U}_\delta} \mathbf{1}_{\{\mathbf{z} \in A\}} \gamma(\mathbf{z}) d\mathbf{z} \quad (2.46)$$

where $\gamma(\mathbf{z})$ denotes the density function of $\mathbf{z}$.

Let $\delta = d^{-1/4}$. From Lemma 3, we have, as $d \rightarrow \infty$,

$$\int_{\mathbb{R}^d \setminus \mathcal{U}_\delta} \mathbf{1}_{\{\mathbf{z} \in A\}} \gamma(\mathbf{z}) d\mathbf{z} \leq \int_{\mathbb{R}^d \setminus \mathcal{U}_\delta} \gamma(\mathbf{z}) d\mathbf{z} = 1 - \mathbb{P}\{\mathbf{z} \in \mathcal{U}_\delta\} \leq 2 \exp(-d\delta^2) \rightarrow 0. \quad (2.47)$$

For $\mathbf{z} \in \mathcal{U}_\delta$ and $\delta = d^{-1/4}$, we have $\|\mathbf{z}\|_2 \rightarrow \sqrt{d}$, $\sqrt{d}/\|\mathbf{z}\|_2 z_i \rightarrow z_i$ and therefore $f(\sqrt{d}/\|\mathbf{z}\|_2 z_i) \rightarrow f(z_i)$ as $d \rightarrow \infty$. Since $|a_i|$ is bounded, we have $\frac{1}{d} \sum_i |a_i| \cdot |f(\sqrt{d}/\|\mathbf{z}\|_2 z_i) - f(z_i)| \rightarrow 0$ and therefore $\int_{\mathcal{U}_\delta} \mathbf{1}_{\{\mathbf{z} \in A\}} d\mathbf{z} \rightarrow 0$, as $d \rightarrow \infty$. $\square$

Lemma 6. Let random matrix $\mathbf{W}$ be uniformly distributed on $\mathbf{O}(d)$ random vector $\boldsymbol{\theta}$ be uniformly distributed on $\mathbb{S}^{d-1}$ and random vector $\mathbf{x} \in \mathbb{R}^d$ be thin-shell concentrated. Then $\mathbf{W}\mathbf{x} \rightarrow \sqrt{d}\boldsymbol{\theta}$ as $d \rightarrow \infty$.

Proof. Let $\mathbf{y} \in \mathbb{R}^d$ be any vector with $\|\mathbf{y}\|_2 = \sqrt{d}$ and $\mathbf{a} = (\sqrt{d}, 0, \dots, 0) \in \mathbb{R}^d$. Since $\mathbf{W}$ is uniformly distributed, $\mathbf{W}\mathbf{y}$ has the same distribution as $\mathbf{W}\mathbf{a}$. $\mathbf{W}\mathbf{a}$ is the first column of $\sqrt{d}\mathbf{W}$, which is equivalent to random vector $\sqrt{d}\boldsymbol{\theta}$ [Meckes, 2019]. Since $\mathbf{x}$ is thin-shell concentrated, $\mathbf{x} \rightarrow \sqrt{d}/\|\mathbf{x}\|_2 \mathbf{x} = \mathbf{y}$ and therefore $\mathbf{W}\mathbf{x} \rightarrow \sqrt{d}\boldsymbol{\theta}$ as $d \rightarrow \infty$. $\square$

Proof of Theorem 2. Let $\mathbf{z} = (z_1, \dots, z_d) \sim \mathcal{N}(\mathbf{0}_d, \mathbf{I}_d)$. Due to Lemma 1, random variable $\phi(z_i)$ is sub-gaussian. Since ϕ is Gaussian-Poincaré normalized, $\mathbb{E}_{z_i \sim \mathcal{N}(0,1)}[\phi(z_i)^2] = 1$. Applying Lemma 2 with each $y_i = 1$, we have for $\epsilon > 0$

$$\mathbb{P}\left\{\left|\frac{1}{d} \sum_i \phi(z_i)^2 - 1\right| \geq \epsilon\right\} \rightarrow 0 \quad (2.48)$$

as $d \rightarrow \infty$.

Due to Lemma 4 and 5 (with each $a_i = 1$), for random vector $\boldsymbol{\theta} = (\theta_1, \dots, \theta_d)$ uniformly distributed on $\mathbb{S}^{d-1}$, we have

$$\mathbb{P}\left\{\left|\frac{1}{d} \sum_i \phi(\sqrt{d}\theta_i)^2 - \frac{1}{d} \sum_i \phi(z_i)^2\right| \geq \epsilon\right\} \rightarrow 0 \quad (2.49)$$

and therefore

$$\mathbb{P}\left\{\left|\frac{1}{d} \sum_i \phi(\sqrt{d}\theta_i)^2 - 1\right| \geq \epsilon\right\} \rightarrow 0 \quad (2.50)$$

as $d \rightarrow \infty$.

Then from Lemma 6, we have $\mathbf{W}\mathbf{x} \rightarrow \sqrt{d}\boldsymbol{\theta}$ and therefore

$$\mathbb{P}\left\{\left|\frac{1}{d} \sum_i \phi(\mathbf{w}_i^T \mathbf{x})^2 - 1\right| \geq \epsilon\right\} \rightarrow 0 \quad (2.51)$$

as $d \rightarrow \infty$. $\square$

Proof of Theorem 3. Let $\mathbf{z} = (z_1, \dots, z_d) \sim \mathcal{N}(\mathbf{0}_d, \mathbf{I}_d)$. Due to Lemma 1, random variable $\phi'(z_i)$ is sub-gaussian. Since ϕ is Gaussian-Poincaré normalized, $\mathbb{E}_{z_i \sim \mathcal{N}(0,1)}[\phi'(z_i)^2] = 1$. Applying Lemma 2, we have for $\epsilon > 0$

$$\mathbb{P}\left\{\left|\frac{1}{d} \sum_i y_i^2 \phi'(z_i)^2 - y_i^2\right| \geq \epsilon\right\} \rightarrow 0 \quad (2.52)$$

as $d \rightarrow \infty$.

Due to Lemma 4 and 5 (with each $a_i = y_i^2$), for random vector $\theta = (\theta_1, \dots, \theta_d)$ uniformly distributed on $\mathbb{S}^{d-1}$, we have

$$\mathbb{P}\left\{\left|\frac{1}{d}\sum_i y_i^2 \phi'(\sqrt{d}\theta_i)^2 - \frac{1}{d}\sum_i y_i^2 \phi'(z_i)^2\right| \geq \epsilon\right\} \rightarrow 0 \quad (2.53)$$

and therefore

$$\mathbb{P}\left\{\left|\frac{1}{d}\sum_i y_i^2 \phi'(\sqrt{d}\theta_i)^2 - y_i^2\right| \geq \epsilon\right\} \rightarrow 0 \quad (2.54)$$

as $n \rightarrow \infty$.

Then from Lemma 6, we have $\mathbf{W}\mathbf{x} \rightarrow \sqrt{d}\theta$ and therefore

$$\mathbb{P}\left\{\left|\frac{1}{d}\sum_i y_i^2 \phi'(\mathbf{w}_i^T \mathbf{x})^2 - y_i^2\right| \geq \epsilon\right\} \rightarrow 0 \quad (2.55)$$

as $d \rightarrow \infty$.

□

Block-Mean Approximation for Efficient Second-Order Optimization

A simple, yet effective approximation is the diagonal approximation, where one only keeps the diagonal elements of $\mathbf{G}$. A classic method is the Levenberg-Marquardt algorithm Levenberg [1944]; Marquardt [1963] which uses the diagonal approximation of the Hessian matrix. Its stochastic version for training neural networks is proposed in Becker and LeCun [1988]. The diagonal approximation of AdaGrad and its variants such as AdaDelta Zeiler [2012], RMSprop Tieleman and Hinton [2012] and Adam Kingma and Ba [2014] have seen increasing popularity in training neural networks recently.

The diagonal approximation of $\mathbf{G}$ amounts to setting an individual learning rate for each parameter. This balances the scale of parameters and often accelerates the convergence. Nevertheless, one wonders whether disregarding the correlation between the parameters, as captured by the off-diagonal elements of $\mathbf{G}$, has negative effects on the convergence speed. We will show that the answer to this question is a firm yes. A simple example will be given. Experiments in the chapter also show the benefits of capturing off-diagonal elements of $\mathbf{G}$.

As such, in this chapter we propose a new matrix approximation technique. The main idea is to split $\mathbf{G}$ into smaller blocks and approximate each block by one or two numbers. This, in return, will enable us to approximate the inverse of $\mathbf{G}$ by inverting small matrices, drastically reducing the time complexity while benefiting from the off-diagonal elements of $\mathbf{G}$. We incorporate our method into AdaGrad for training deep neural networks and empirically observe that the resulting algorithm outperforms AdaGrad with diagonal approximation in terms of the convergence speed.

3.1 Method

We need the following definitions before introducing our proposed block mean approximation (BMA).

Notation 1. $\mathbf{M}^{ij}$ denotes the (i, j) -th block of the matrix $\mathbf{M}$. $\mathbf{M}_{mn}^{ij}$ denotes the (m, n) -th element of the matrix $\mathbf{M}^{ij}$.

Definition 1. For a diagonal matrix

$$\mathbf{\Lambda} = \begin{bmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & \lambda_L \end{bmatrix}, \quad (3.1)$$

its diagonal expansion matrix with partition vector $\mathbf{s} = (s_1, \dots, s_L)$ is

$$\bar{\mathbf{\Lambda}} = \begin{bmatrix} \bar{\mathbf{\Lambda}}^{11} & 0 & \dots & 0 \\ 0 & \bar{\mathbf{\Lambda}}^{22} & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & \bar{\mathbf{\Lambda}}^{LL} \end{bmatrix} \quad (3.2)$$

where each $\bar{\mathbf{\Lambda}}^{ii}$ is a diagonal matrix of size $s_i \times s_i$ with fixed diagonal elements of λ_i .

Definition 2. For a matrix

$$\mathbf{B} = \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1L} \\ b_{21} & b_{22} & \dots & b_{2L} \\ \dots & \dots & \dots & \dots \\ b_{L1} & b_{L2} & \dots & b_{LL} \end{bmatrix}, \quad (3.3)$$

its full expansion matrix with partition vector $\mathbf{s} = (s_1, \dots, s_L)$ is

$$\bar{\mathbf{B}} = \begin{bmatrix} \bar{\mathbf{B}}^{11} & \bar{\mathbf{B}}^{12} & \dots & \bar{\mathbf{B}}^{1L} \\ \bar{\mathbf{B}}^{21} & \bar{\mathbf{B}}^{22} & \dots & \bar{\mathbf{B}}^{2L} \\ \dots & \dots & \dots & \dots \\ \bar{\mathbf{B}}^{L1} & \bar{\mathbf{B}}^{L2} & \dots & \bar{\mathbf{B}}^{LL} \end{bmatrix} \quad (3.4)$$

where each block $\bar{\mathbf{B}}^{ij}$ is a matrix of constant value b_{ij} and has s_i number of rows.

The above definitions are illustrated in Figure 3.1.

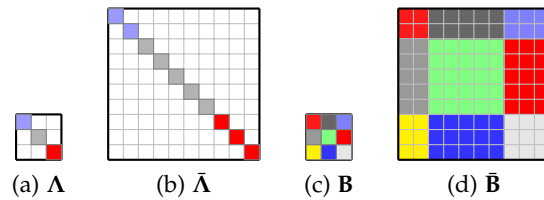


Figure 3.1: (a) Diagonal matrix $\mathbf{\Lambda}$. (b) Diagonal expansion of $\mathbf{\Lambda}$. (c) Full matrix $\mathbf{B}$. (d) Full expansion of $\mathbf{B}$. The partition vector in both cases is $\mathbf{s} = (2, 5, 3)$.

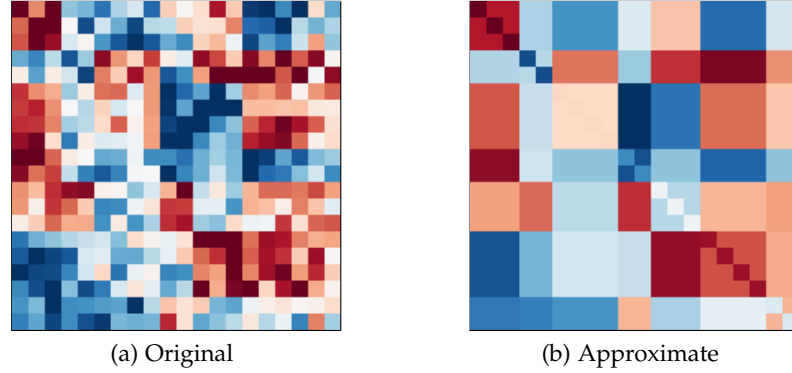


Figure 3.2: Block mean approximation of a matrix.

Definition 3. The block mean approximation of a matrix $\mathbf{M}$ with the partition vector $\mathbf{s}$ is

$$\hat{\mathbf{M}} = \bar{\mathbf{A}} + \bar{\mathbf{B}} \approx \mathbf{M} \quad (3.5)$$

where $\bar{\mathbf{A}}$ and $\bar{\mathbf{B}}$ are the diagonal and full expansion matrices with partition vector $\mathbf{s}$, respectively.

The block mean approximation, as illustrated in Figure 3.2, allows one to efficiently store a big matrix as only $\mathbf{A}$, $\mathbf{B}$ and a partition vector $\mathbf{s}$ need to be kept. Given a square matrix, its optimal block mean approximation under Frobenius norm is given by the following.

Proposition 1. The optimal block mean approximation of $\mathbf{M}$ with the partition vector $\mathbf{s}$ according to the Frobenius norm

$$\min_{\bar{\mathbf{A}}, \bar{\mathbf{B}}} \|\bar{\mathbf{A}} + \bar{\mathbf{B}} - \mathbf{M}\|_F^2 \quad (3.6)$$

is given by

$$b_{ij} = \begin{cases} 0, & i = j, s_i = 1, \\ \frac{\sum_{mn} \mathbf{M}_{mn}^{ii} - \sum_m \mathbf{M}_{mm}^{ii}}{s_i(s_i - 1)}, & i = j, s_i \neq 1, \\ \frac{\sum_{mn} \mathbf{M}_{mn}^{ij}}{s_i s_j}, & i \neq j, \end{cases} \quad (3.7)$$

$$\lambda_i = \frac{1}{s_i} \sum_m \mathbf{M}_{mm}^{ii} - b_{ii}. \quad (3.8)$$

Proposition 1 can be understood as follows. According to (3.7) and (3.8), the non-diagonal block $\mathbf{M}^{ij}$ is approximated by $\hat{\mathbf{M}}^{ij} = \mathbf{B}^{ij}$, whose value is the mean value of $\mathbf{M}^{ij}$, which minimizes the Frobenius form. The diagonal block $\mathbf{M}^{ii}$ is approximated by $\hat{\mathbf{M}}^{ii} = \mathbf{A}^{ii} + \mathbf{B}^{ii}$, whose diagonal values are equal to the mean diagonal values of $\mathbf{M}^{ii}$ and its off-diagonal values are equal to the mean off-diagonal values of $\mathbf{M}^{ii}$, which again minimizes the Frobenius form.

3.1.1 Fast Inversion Theorems

The power of block mean approximation lies in the ease of computing its inverse and inverse square root matrices, as shown by the following theorems. All the proofs are relegated to the Appendix.

Theorem 1. For invertible matrix $\bar{\mathbf{A}} + \bar{\mathbf{B}}$, where $\bar{\mathbf{A}}$ and $\bar{\mathbf{B}}$ are the diagonal and full expansion of $\mathbf{A}$ and $\mathbf{B}$ with respect to the partition vector $\mathbf{s}$,

$$(\bar{\mathbf{A}} + \bar{\mathbf{B}})^{-1} = \bar{\mathbf{A}}^{-1} + \bar{\mathbf{D}} \quad (3.9)$$

where $\bar{\mathbf{D}}$ is the full expansion matrix with partition vector $\mathbf{s}$ of

$$\mathbf{D} = (\mathbf{A}\mathbf{S} + \mathbf{S}\mathbf{B}\mathbf{S})^{-1} - (\mathbf{A}\mathbf{S})^{-1} \quad (3.10)$$

where $\mathbf{S} = \text{diag}(\mathbf{s})$.

Theorem 2. For invertible matrix $\bar{\mathbf{A}} + \bar{\mathbf{B}}$, where $\bar{\mathbf{A}}$ and $\bar{\mathbf{B}}$ are the diagonal and full expansion of $\mathbf{A}$ and $\mathbf{B}$ with respect to the partition vector $\mathbf{s}$,

$$(\bar{\mathbf{A}} + \bar{\mathbf{B}})^{-\frac{1}{2}} = \bar{\mathbf{A}}^{-\frac{1}{2}} + \bar{\mathbf{D}} \quad (3.11)$$

where $\bar{\mathbf{D}}$ is the full expansion matrix with partition vector $\mathbf{s}$ of

$$\mathbf{D} = \mathbf{S}^{-\frac{1}{2}} \left[(\mathbf{A} + \mathbf{S}^{\frac{1}{2}}\mathbf{B}\mathbf{S}^{\frac{1}{2}})^{-\frac{1}{2}} - \mathbf{A}^{-\frac{1}{2}} \right] \mathbf{S}^{-\frac{1}{2}} \quad (3.12)$$

where $\mathbf{S} = \text{diag}(\mathbf{s})$.

The importance of the above theorems can be understood by noting that inverting $\mathbf{G} \in \mathbb{R}^{d \times d}$ by splitting it into $L \times L$ blocks has a complexity of $O(L^3)$, which can be significantly faster than $O(d^3)$ flops required to obtain $\mathbf{G}^{-1}$ or $\mathbf{G}^{-\frac{1}{2}}$.

3.1.2 BMA for Neural Networks

As block mean approximation divides a matrix into $L \times L$ blocks, a natural question to ask is *how much prior knowledge is needed to determine such a block structure?* In training neural networks, we recommend to group the parameters in each layer (or even each unit) into a block such that $\mathbf{G}$ with block mean approximation represents the scale and dependency between layers (or units). This comes naturally as the output and gradient of each layer often depends on one another.

There are several work that use heuristics to set an individual learning rate for layers of a deep network Singh et al. [2015]; You et al. [2017]; Abu-El-Haija [2017]. Nevertheless, even with such heuristics, the dependency between parameters is ignored. In contrast, the BMA gives a principled way to set learning rates per layers in a deep network while capturing the dependency between layers.

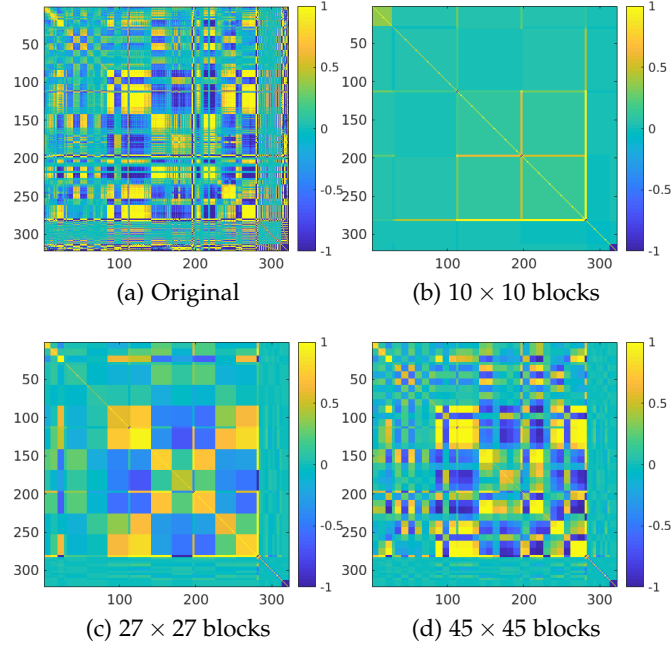


Figure 3.3: The values are normalized for better visualization. (a) The original matrix has a size of 322×322 . (b) Each block represents weights or bias in a layer. MSE (mean square error) = 0.1036. (c) Each block represents weights or bias in a unit. MSE=0.1024. (d) Each block represents a group of weights or bias in a unit. MSE=0.0998.

In Figure 3.3, we compute the empirical Fisher information matrix

$$\mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} \left[\frac{\partial \log p(\mathbf{x}|\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \frac{\partial \log p(\mathbf{x}|\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}^\top \right] \quad (3.13)$$

for a convolutional neural network (described in Table 1) and compare BMA with different block partitions. Even for the finest approximation in Figure 3.3, only 45×45 matrices need to be constructed and inverted.

3.1.3 BMA for AdaGrad

For unconstrained stochastic optimization problems, the full version of AdaGrad Duchi et al. [2011] has the following update rule,

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \mathbf{G}_t^{-1} \mathbf{g}_t \quad (3.14)$$

where $\mathbf{G}_t = \mathbf{H}_t^{\frac{1}{2}}$ and $\mathbf{H}_t = \sum_{\tau=1}^t \mathbf{g}_\tau \mathbf{g}_\tau^\top$.

We approximate the gradient outer product matrix in the following form

$$\hat{\mathbf{H}}_t = \mathbf{Z}_t \mathbf{F}_t \mathbf{Z}_t \approx \mathbf{H}_t \quad (3.15)$$

where $\mathbf{Z}_t = \text{diag}(\mathbf{z}_t)$ is a diagonal matrix and $\mathbf{F}_t$ is a positive definite matrix. As

AdaGrad requires the computation of $\mathbf{H}_t^{-\frac{1}{2}}$, one needs to choose $\mathbf{F}_t$ such that $\mathbf{F}_t^{-\frac{1}{2}}$ is easy to obtain.

To derive the algorithm, we first define the following notations.

Notation 2. We divide vector $\mathbf{g}$ into L blocks such that $\mathbf{g} = (\mathbf{g}^1, \dots, \mathbf{g}^L)$. $\mathbf{g}^i$ denotes the i -th block of vector $\mathbf{g}$. g_m^i denotes the m -th element of vector $\mathbf{g}^i$. $(g_m^i)^2$ denotes the square value of g_m^i . $\mathbf{g}_t$ denote a vector at step t . $\mathbf{g}_t^i$ denote the i -th block of $\mathbf{g}_t$. $g_{t,m}^i$ denote the m -th element of $\mathbf{g}_t^i$. $\mathbf{g}^2$ denote elementwise square of $\mathbf{g}$. $\sqrt{\mathbf{g}}$ denotes elementwise square root of $\mathbf{g}$. $\mathbf{g} + a$ denotes each element of $\mathbf{g}$ is added by scalar a .

If we set $\mathbf{z}_t$ with elements

$$z_{t,i} = \sqrt{\sum_{\tau=1}^t (g_{\tau,i})^2} \quad (3.16)$$

and $\mathbf{F}_t = \mathbf{I}$, $\hat{\mathbf{H}}_t$ is reduced to diagonal approximation.

In order to capture the off-diagonal elements of $\mathbf{H}$, we approximate $\mathbf{F}_t$ with the block mean approximation proposed above. We set $\mathbf{z}_t$ as in (3.16) and seek

$$\mathbf{F}_t = \bar{\mathbf{A}} + \bar{\mathbf{B}} \approx \mathbf{Z}_t^{-1} \left(\sum_{\tau=1}^t \mathbf{g}_\tau \mathbf{g}_\tau^\top \right) \mathbf{Z}_t^{-1}. \quad (3.17)$$

With the block mean approximation, (3.15) can be interpreted as follows: $\mathbf{Z}_t$ gives an individual learning rate of each parameter and $\mathbf{F}_t$ captures the dependency between the groups of parameters.

To realize block mean approximation, we partition the parameters into L groups.

Let $\mathbf{g}_t = (\mathbf{g}_t^1, \mathbf{g}_t^2, \dots, \mathbf{g}_t^L)$. Define $\mathbf{u}_t$ and $\mathbf{v}_t$ with

$$u_{t,i} = \sum_m g_{t,m}^i, \quad v_{t,i} = \sum_m z_{t,m}^i \quad (3.18)$$

respectively. Let $\mathbf{U}_t = \sum_{\tau=1}^t \mathbf{u}_\tau \mathbf{u}_\tau^\top$. To approximate $\mathbf{Z}_t^{-1} (\sum_{\tau=1}^t \mathbf{g}_\tau \mathbf{g}_\tau^\top) \mathbf{Z}_t^{-1}$, we choose $\bar{\mathbf{A}}$ and $\bar{\mathbf{B}}$ to be the expansion matrices of $\mathbf{A}$ and $\mathbf{B}$

$$\mathbf{A} = \mathbf{I}, \quad \mathbf{B} = \mathbf{S}^{-\frac{1}{2}} \frac{\mathbf{U}_t - \text{diag}(\mathbf{U}_t)}{\mathbf{v}_t \mathbf{v}_t^\top} \mathbf{S}^{-\frac{1}{2}} \quad (3.19)$$

where the division is elementwise. Based on Theorem 2, the inverse square root is $\mathbf{I} + \bar{\mathbf{D}}$ where $\bar{\mathbf{D}}$ is the expansion matrix of

$$\mathbf{D} = \mathbf{S}^{-\frac{1}{2}} \left[\left(\mathbf{I} + \frac{\mathbf{U}_t - \text{diag}(\mathbf{U}_t)}{\mathbf{v}_t \mathbf{v}_t^\top} \right)^{-\frac{1}{2}} - \mathbf{I} \right] \mathbf{S}^{-\frac{1}{2}}. \quad (3.20)$$

The inverse square root in (3.20) can be computed as follows. Let $\mathbf{R}\mathbf{V}\mathbf{R}^\top$ be the eigendecomposition of a matrix, then its inverse square root is $\mathbf{R}\mathbf{V}^{-\frac{1}{2}}\mathbf{R}^\top$. In case

Algorithm 1 AdaGrad-BMA

```

1: Input: Objective function  $f(\theta)$  with parameters  $\theta$ 
2: Input: A partition of parameters  $\theta = \{\theta^1, \dots, \theta^L\}$ 
3: Input: Partition vector  $\mathbf{s}$  that  $s_i = \text{size}(\theta^i)$ 
4: Input: Hyperparameters  $\eta$  and  $\epsilon$ 
5: Initialize  $\mathbf{U} = \mathbf{0}$ ,  $\mathbf{v} = \mathbf{0}$ ,  $\mathbf{r} = \mathbf{0}$ ,  $\mathbf{S} = \text{diag}(\mathbf{s})$ 
6: for  $t = 1$  to  $T$  do
7:    $\mathbf{g} \leftarrow \nabla_{\theta} f_t(\theta_t)$ 
8:    $\mathbf{r} \leftarrow \mathbf{r} + \mathbf{g}^2$ 
9:    $\mathbf{z} \leftarrow \sqrt{\mathbf{r} + \epsilon}$ 
10:  for  $i = 1$  to  $L$  do
11:     $u_i \leftarrow \sum_m g_m^i$ 
12:     $v_i \leftarrow \sum_m z_m^i$ 
13:  end for
14:   $\mathbf{U} \leftarrow \mathbf{U} + \mathbf{u}\mathbf{u}^\top$ 
15:  Compute  $\mathbf{D}$  according to (3.20)
16:   $\mathbf{g} \leftarrow \mathbf{Z}^{-\frac{1}{2}} \mathbf{g}$ 
17:  for  $i = 1$  to  $L$  do
18:     $\mathbf{g}^i \leftarrow \mathbf{g}^i + \sum_j \mathbf{D}_{ij} u_j$ 
19:  end for
20:   $\mathbf{g} \leftarrow \mathbf{Z}^{-\frac{1}{2}} \mathbf{g}$ 
21:   $\theta \leftarrow \theta - \eta \mathbf{g}$ 
22: end for

```

where the eigenvalues are zeros or too small, we clamp the eigenvalues to have a minimal value before computing $\mathbf{V}^{-\frac{1}{2}}$.

We call the above algorithm AdaGrad-BMA, which is summarized in Algorithm 1. The eigendecomposition has time complexity $O(L^3)$. Therefore, for parameters of dimension d and partitioned into L blocks, AdaGrad-BMA has time complexity $O(L^3 + d)$ per iteration.

3.2 Experiments

We evaluate AdaGrad-BMA in training convolutional neural networks, against the full version of AdaGrad (AdaGrad-full) and AdaGrad with diagonal approximation (AdaGrad-diag). For AdaGrad-BMA, we group the weights and the bias parameters separately for each layer. For a model of l convolution or fully connected layers, we partition $\mathbf{G}$ into $L \times L$ blocks with BMA where $L = 2l$.

The experiments are done on two standard datasets MNIST and CIFAR-10. We use two simple models: small and large, as described in Table 1 and 2. We choose the architecture of the small model to ensure that AdaGrad-full is applicable. For the large model, AdaGrad-full is too computationally expensive to use. Each convolution layer has kernel size 3×3 , stride 1 and zero-padding 1. Each convolution layer is

followed by a hyperbolic tangent activation function. The number of parameters of each model is listed in Table 3. As MNIST and CIFAR-10 have different input image size, the fully connected layer in each model has different size of inputs, therefore resulting in different number of parameters.

For each algorithm on each dataset, we tried learning rates $\eta \in \{1, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}\}$ and report the best performance achieved by the algorithm on the dataset.

The results are shown in Figure 3.4, from which we can see AdaGrad-BMA outperforms AdaGrad-diag and achieves similar speed of convergence to AdaGrad-full. The comparison of runtime of each iteration on MNIST is shown in Table 3.4. Although AdaGrad-BMA has longer runtime than AdaGrad-diag for each iteration, in practice one can update $\mathbf{G}^{-1}$ for every several steps to amortize the cost.

The code for the experiments is included in the supplementary materials.

Table 3.1: Small model

Conv 3x3, 3
Max Pooling 2x2
Conv 3x3, 3
Max Pooling 2x2
Conv 3x3, 3
Max Pooling 2x2
Conv 3x3, 3
Max Pooling 2x2
Fully Connected, 10
Softmax, 10

Table 3.2: Large model

Conv 3x3, 32
Conv 3x3, 32
Conv 3x3, 32
Conv 3x3, 32
Max Pooling 2x2
Conv 3x3, 32
Conv 3x3, 32
Conv 3x3, 32
Conv 3x3, 32
Max Pooling 2x2
Conv 3x3, 32
Conv 3x3, 32
Conv 3x3, 32
Conv 3x3, 32
Conv 3x3, 32
Max Pooling 2x2
Conv 3x3, 32
Conv 3x3, 32
Conv 3x3, 32
Conv 3x3, 32
Max Pooling 2x2
Fully Connected, 10
Softmax, 10

Table 3.3: Model parameters

	MNIST	CIFAR-10
Small model	322	466
Large model	139370	140906

Table 3.4: Runtime comparison (ms/iteration)

	Small model	Large model
AdaGrad-full	16.85	-
AdaGrad-diag	5.70	7.55
AdaGrad-BMA	10.07	19.12

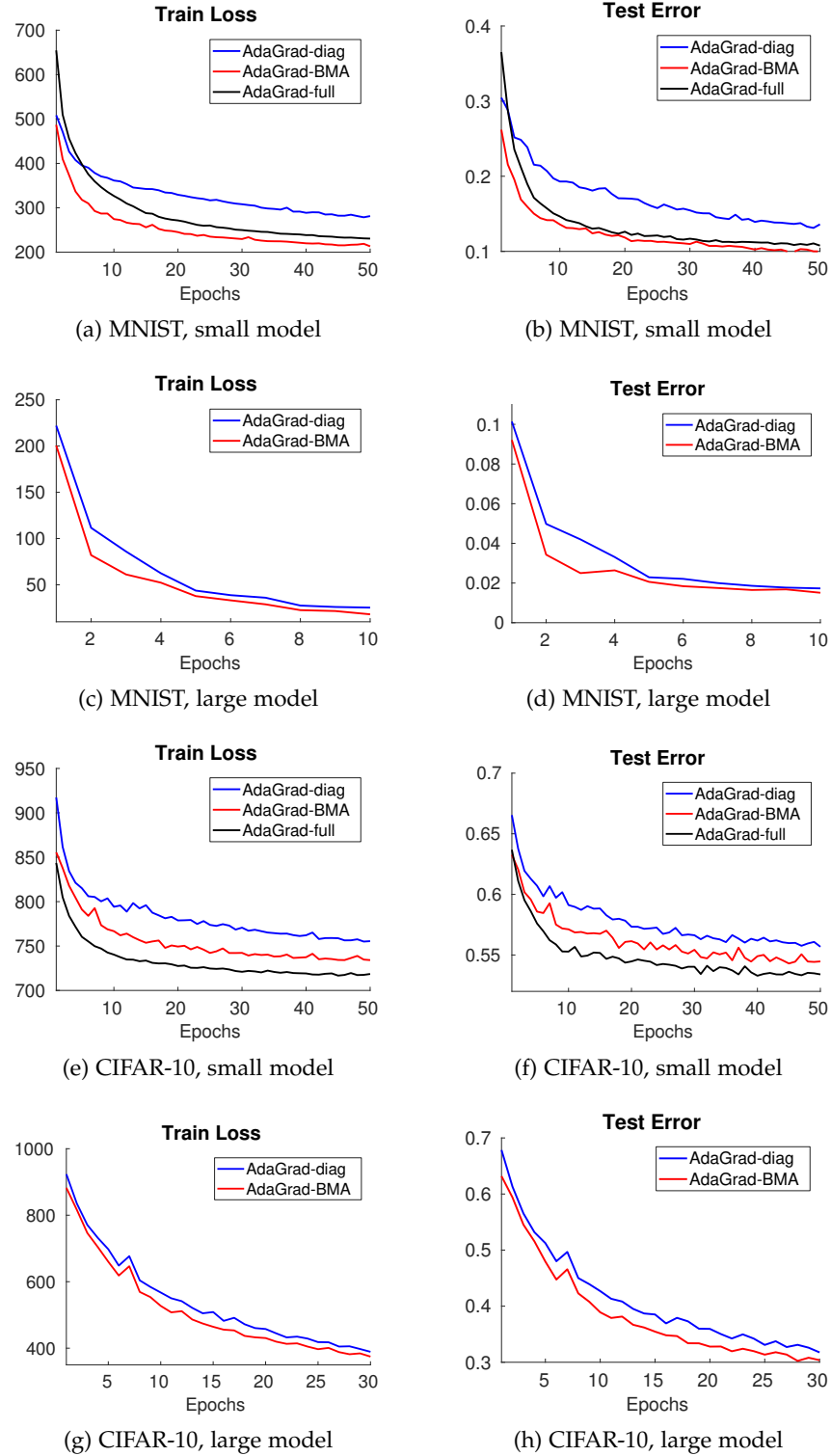


Figure 3.4: Performance of AdaGrad and its approximations on two standard datasets.

3.3 Discussion

In this work, we propose a new matrix approximation method which allows efficient storage and computation of matrix inverse and inverse square root. The method is applied to AdaGrad and achieves promising results.

In the numerical linear algebra literature, there are two work relevant but different from ours. Chow and Saad [1997] proposed an approximate inverse technique which generates sparse solutions for block partitioned matrices. However, in our method the exact inverse of the block mean approximation matrix can be computed, as proved in Theorem 1 and 2. Our method does not assume sparse solutions either. Guillaume et al. [2003] proposed an approximate inverse technique which incorporates block constant structure in the preconditioning matrix. This is different from ours as in our method the block constant structure is incorporated the approximated matrix and its inverse and the analytic solution of the inverse is explicitly given.

Second order optimization methods for training neural networks have many theoretical appeals, as discussed in this work and reviewed in Shepherd [2012]; Martens [2016]. We hope our method makes one more step towards their practical implementation.

3.4 Appendix

To show how Newton method is invariant of affine re-parameterization, let ∇^2 be the Hessian operator, $\xi = \mathbf{A}\theta$ and $g(\theta) = f(\mathbf{A}\theta)$ where $\mathbf{A}$ is an invertible square matrix.

$$\theta_{t+1} = \theta_t - \eta(\nabla_{\theta}^2 g(\theta))^{-1} \nabla_{\theta} g(\theta_t) \quad (3.21)$$

$$= \theta_t - \eta(\mathbf{A}^T \nabla_{\xi}^2 f(\mathbf{A}\theta_t) \mathbf{A})^{-1} \mathbf{A}^T \nabla_{\xi} f(\mathbf{A}\theta_t) \quad (3.22)$$

$$= \theta_t - \eta \mathbf{A}^{-1} (\nabla_{\xi}^2 f(\mathbf{A}\theta_t))^{-1} \nabla_{\xi} f(\mathbf{A}\theta_t) \quad (3.23)$$

we obtain

$$\mathbf{A}\theta_{t+1} = \mathbf{A}\theta_t - \eta(\nabla_{\xi}^2 f(\mathbf{A}\theta_t))^{-1} \nabla_{\xi} f(\mathbf{A}\theta_t) \quad (3.24)$$

which is equivalent to

$$\xi_{t+1} = \xi_t - \eta(\nabla_{\xi}^2 f(\xi_t))^{-1} \nabla_{\xi} f(\xi_t). \quad (3.25)$$

Proposition 1. *The optimal block mean approximation of $\mathbf{M}$ with the partition vector $\mathbf{s}$ according to the Frobenius norm*

$$\min_{\bar{\mathbf{A}}, \bar{\mathbf{B}}} \|\bar{\mathbf{A}} + \bar{\mathbf{B}} - \mathbf{M}\|_F^2 \quad (3.26)$$

is given by

$$b_{ij} = \begin{cases} 0, & i = j, s_i = 1, \\ \frac{\sum_{mn} \mathbf{M}_{mn}^{ii} - \sum_m \mathbf{M}_{mm}^{ii}}{s_i(s_i-1)}, & i = j, s_i \neq 1, \\ \frac{\sum_{mn} \mathbf{M}_{mn}^{ij}}{s_i s_j}, & i \neq j, \end{cases} \quad (3.27)$$

$$\lambda_i = \frac{1}{s_i} \sum_m \mathbf{M}_{mm}^{ii} - b_{ii}. \quad (3.28)$$

Proof. For $i \neq j$, $\bar{\Lambda}^{ij} = 0$ by construction. Hence, $b_{ij} = \frac{\sum_{mn} \mathbf{M}_{mn}^{ij}}{s_i s_j}$ is the minimum solution for $\|\bar{\mathbf{B}}^{ij} - \mathbf{M}^{ij}\|_F$. For $i = j$, since $b_{ii} = \frac{\sum_{mn} \mathbf{M}_{mn}^{ii} - \sum_m \mathbf{M}_{mm}^{ii}}{s_i(s_i-1)}$, the off-diagonal elements of $\mathbf{M}^{ii}$ are minimized under the Frobenius norm. And since $\lambda_i + b_{ii} = \frac{1}{s_i} \sum_m \mathbf{M}_{mm}^{ii}$, the diagonal elements of $\mathbf{M}^{ii}$ are also minimized. $\square$

Theorem 1. For invertible matrix $\bar{\Lambda} + \bar{\mathbf{B}}$, where $\bar{\Lambda}$ is the diagonal expansion matrix of Λ and $\bar{\mathbf{B}}$ is the full expansion matrix of $\mathbf{B}$, both of which have the same partition vector $\mathbf{s}$,

$$(\bar{\Lambda} + \bar{\mathbf{B}})^{-1} = \bar{\Lambda}^{-1} + \bar{\mathbf{D}} \quad (3.29)$$

where $\bar{\mathbf{D}}$ is the full expansion matrix with partition vector $\mathbf{s}$ of

$$\mathbf{D} = (\Lambda \mathbf{S} + \mathbf{S} \mathbf{B} \mathbf{S})^{-1} - (\Lambda \mathbf{S})^{-1} \quad (3.30)$$

where $\mathbf{S} = \text{diag}(\mathbf{s})$.

Proof. First we prove $(\bar{\Lambda} + \bar{\mathbf{B}})(\bar{\Lambda}^{-1} + \bar{\mathbf{D}}) = \mathbf{I}$.

$$\mathbf{D} = (\Lambda \mathbf{S} + \mathbf{S} \mathbf{B} \mathbf{S})^{-1} - (\Lambda \mathbf{S})^{-1} \quad (3.31)$$

$$\begin{aligned} &= (\Lambda \mathbf{S})^{-1} \\ &\quad - (\Lambda \mathbf{S})^{-1} \mathbf{S} (\mathbf{I} + \mathbf{B} \mathbf{S} (\Lambda \mathbf{S})^{-1} \mathbf{S})^{-1} \mathbf{B} \mathbf{S} (\Lambda \mathbf{S})^{-1} \\ &\quad - (\Lambda \mathbf{S})^{-1} \end{aligned} \quad (3.32)$$

$$= -\Lambda^{-1} (\mathbf{I} + \mathbf{B} \Lambda^{-1} \mathbf{S})^{-1} \mathbf{B} \Lambda^{-1} \quad (3.33)$$

$$= -(\Lambda + \mathbf{B} \mathbf{S})^{-1} \mathbf{B} \Lambda^{-1} \quad (3.34)$$

(3.32) follows from the Kailath variant of Woodbury identity. (3.33) follows from $\Lambda \mathbf{S} = \mathbf{S} \Lambda$ since $\mathbf{S}$ and Λ are both diagonal. Multiply both sides of (3.34) by $\Lambda + \mathbf{B} \mathbf{S}$, after some manipulation, we have

$$\mathbf{B} \Lambda^{-1} + \Lambda \mathbf{D} + \mathbf{B} \mathbf{S} \mathbf{D} = 0 \quad (3.35)$$

Since $\bar{\Lambda}$, $\bar{\mathbf{B}}$ and $\bar{\mathbf{D}}$ are the expansion matrices with partition vector $\mathbf{s}$ of Λ , $\mathbf{B}$ and $\mathbf{D}$,

respectively, we have equivalently

$$\bar{\mathbf{B}}\bar{\mathbf{\Lambda}}^{-1} + \bar{\mathbf{\Lambda}}\bar{\mathbf{D}} + \bar{\mathbf{B}}\bar{\mathbf{D}} = 0 \quad (3.36)$$

$$\mathbf{I} + \bar{\mathbf{B}}\bar{\mathbf{\Lambda}}^{-1} + \bar{\mathbf{\Lambda}}\bar{\mathbf{D}} + \bar{\mathbf{B}}\bar{\mathbf{D}} = \mathbf{I} \quad (3.37)$$

$$(\bar{\mathbf{\Lambda}} + \bar{\mathbf{B}})(\bar{\mathbf{\Lambda}}^{-1} + \bar{\mathbf{D}}) = \mathbf{I} \quad (3.38)$$

Second, we prove $(\bar{\mathbf{\Lambda}}^{-1} + \bar{\mathbf{D}})(\bar{\mathbf{\Lambda}} + \bar{\mathbf{B}}) = \mathbf{I}$.

$$\mathbf{D} = (\mathbf{\Lambda}\mathbf{S} + \mathbf{S}\mathbf{B}\mathbf{S})^{-1} - (\mathbf{\Lambda}\mathbf{S})^{-1} \quad (3.39)$$

$$= (\mathbf{\Lambda}\mathbf{S})^{-1} \quad (3.40)$$

$$- (\mathbf{\Lambda}\mathbf{S})^{-1}\mathbf{S}\mathbf{B}(\mathbf{I} + \mathbf{S}(\mathbf{\Lambda}\mathbf{S})^{-1}\mathbf{S}\mathbf{B})^{-1}\mathbf{S}(\mathbf{\Lambda}\mathbf{S})^{-1} \quad (3.41)$$

$$- (\mathbf{\Lambda}\mathbf{S})^{-1} \quad (3.42)$$

$$= -\mathbf{\Lambda}^{-1}\mathbf{B}(\mathbf{I} + \mathbf{\Lambda}^{-1}\mathbf{S}\mathbf{B})^{-1}\mathbf{\Lambda}^{-1} \quad (3.43)$$

$$= -\mathbf{\Lambda}^{-1}\mathbf{B}(\mathbf{\Lambda} + \mathbf{S}\mathbf{B})^{-1} \quad (3.44)$$

Therefore

$$\mathbf{\Lambda}^{-1}\mathbf{B} + \mathbf{D}\mathbf{\Lambda} + \mathbf{D}\mathbf{S}\mathbf{B} = 0 \quad (3.45)$$

$$\bar{\mathbf{\Lambda}}^{-1}\bar{\mathbf{B}} + \bar{\mathbf{D}}\bar{\mathbf{\Lambda}} + \bar{\mathbf{D}}\bar{\mathbf{B}} = 0 \quad (3.46)$$

$$\mathbf{I} + \bar{\mathbf{\Lambda}}^{-1}\bar{\mathbf{B}} + \bar{\mathbf{D}}\bar{\mathbf{\Lambda}} + \bar{\mathbf{D}}\bar{\mathbf{B}} = \mathbf{I} \quad (3.47)$$

$$(\bar{\mathbf{\Lambda}}^{-1} + \bar{\mathbf{D}})(\bar{\mathbf{\Lambda}} + \bar{\mathbf{B}}) = \mathbf{I} \quad (3.48)$$

□

Lemma 1. For invertible matrix $\bar{\mathbf{\Lambda}} + \bar{\mathbf{B}}$, where $\bar{\mathbf{\Lambda}}$ is the diagonal expansion matrix of $\mathbf{\Lambda}$ and $\bar{\mathbf{B}}$ is the full expansion matrix of $\mathbf{B}$, both of which have the same partition vector $\mathbf{s}$,

$$(\bar{\mathbf{\Lambda}} + \bar{\mathbf{B}})^{\frac{1}{2}} = \bar{\mathbf{\Lambda}}^{\frac{1}{2}} + \bar{\mathbf{D}} \quad (3.49)$$

where $\bar{\mathbf{D}}$ is the full expansion matrix with partition vector $\mathbf{s}$ of

$$\mathbf{D} = \mathbf{S}^{-\frac{1}{2}} \left[(\mathbf{\Lambda} + \mathbf{S}^{\frac{1}{2}}\mathbf{B}\mathbf{S}^{\frac{1}{2}})^{\frac{1}{2}} - \mathbf{\Lambda}^{\frac{1}{2}} \right] \mathbf{S}^{-\frac{1}{2}} \quad (3.50)$$

where $\mathbf{S} = \text{diag}(\mathbf{s})$.

Proof.

$$\mathbf{D} = \mathbf{S}^{-\frac{1}{2}} \left[(\mathbf{\Lambda} + \mathbf{S}^{\frac{1}{2}}\mathbf{B}\mathbf{S}^{\frac{1}{2}})^{\frac{1}{2}} - \mathbf{\Lambda}^{\frac{1}{2}} \right] \mathbf{S}^{-\frac{1}{2}} \quad (3.51)$$

Left and right multiply both side by $\mathbf{S}^{\frac{1}{2}}$,

$$\mathbf{S}^{\frac{1}{2}}\mathbf{D}\mathbf{S}^{\frac{1}{2}} = (\mathbf{\Lambda} + \mathbf{S}^{\frac{1}{2}}\mathbf{B}\mathbf{S}^{\frac{1}{2}})^{\frac{1}{2}} - \mathbf{\Lambda}^{\frac{1}{2}} \quad (3.52)$$

$$\mathbf{S}^{\frac{1}{2}}\mathbf{D}\mathbf{S}^{\frac{1}{2}} + \mathbf{\Lambda}^{\frac{1}{2}} = (\mathbf{\Lambda} + \mathbf{S}^{\frac{1}{2}}\mathbf{B}\mathbf{S}^{\frac{1}{2}})^{\frac{1}{2}} \quad (3.53)$$

$$(\mathbf{S}^{\frac{1}{2}}\mathbf{D}\mathbf{S}^{\frac{1}{2}} + \mathbf{\Lambda}^{\frac{1}{2}})^2 = (\mathbf{\Lambda} + \mathbf{S}^{\frac{1}{2}}\mathbf{B}\mathbf{S}^{\frac{1}{2}}) \quad (3.54)$$

Expanding the square,

$$\mathbf{S}^{\frac{1}{2}}\mathbf{D}\mathbf{S}\mathbf{D}\mathbf{S}^{\frac{1}{2}} + \mathbf{S}^{\frac{1}{2}}\mathbf{D}\mathbf{S}^{\frac{1}{2}}\mathbf{\Lambda}^{\frac{1}{2}} + \mathbf{\Lambda}^{\frac{1}{2}}\mathbf{S}^{\frac{1}{2}}\mathbf{D}\mathbf{S}^{\frac{1}{2}} + \mathbf{\Lambda} \quad (3.55)$$

$$= \mathbf{\Lambda} + \mathbf{S}^{\frac{1}{2}}\mathbf{B}\mathbf{S}^{\frac{1}{2}} \quad (3.56)$$

Left and right multiply both side by $\mathbf{S}^{-\frac{1}{2}}$,

$$\mathbf{D}\mathbf{S}\mathbf{D} + \mathbf{D}\mathbf{\Lambda}^{\frac{1}{2}} + \mathbf{\Lambda}^{\frac{1}{2}}\mathbf{D} = \mathbf{B} \quad (3.57)$$

Since $\bar{\mathbf{\Lambda}}$, $\bar{\mathbf{B}}$ and $\bar{\mathbf{D}}$ are the expansion matrices with partition vector $\mathbf{s}$ of $\mathbf{\Lambda}$, $\mathbf{B}$ and $\mathbf{D}$, respectively, we have equivalently

$$\bar{\mathbf{D}}\bar{\mathbf{D}} + \bar{\mathbf{D}}\bar{\mathbf{\Lambda}}^{\frac{1}{2}} + \bar{\mathbf{\Lambda}}^{\frac{1}{2}}\bar{\mathbf{D}} = \bar{\mathbf{B}} \quad (3.58)$$

$$\bar{\mathbf{\Lambda}} + \bar{\mathbf{D}}\bar{\mathbf{D}} + \bar{\mathbf{D}}\bar{\mathbf{\Lambda}}^{\frac{1}{2}} + \bar{\mathbf{\Lambda}}^{\frac{1}{2}}\bar{\mathbf{D}} = \bar{\mathbf{\Lambda}} + \bar{\mathbf{B}} \quad (3.59)$$

$$(\bar{\mathbf{\Lambda}}^{\frac{1}{2}} + \bar{\mathbf{D}})^2 = \bar{\mathbf{\Lambda}} + \bar{\mathbf{B}} \quad (3.60)$$

$$\bar{\mathbf{\Lambda}}^{\frac{1}{2}} + \bar{\mathbf{D}} = (\bar{\mathbf{\Lambda}} + \bar{\mathbf{B}})^{\frac{1}{2}} \quad (3.61)$$

□

Theorem 2. For invertible matrix $\bar{\mathbf{\Lambda}} + \bar{\mathbf{B}}$, where $\bar{\mathbf{\Lambda}}$ is the diagonal expansion matrix of $\mathbf{\Lambda}$ and $\bar{\mathbf{B}}$ is the full expansion matrix of $\mathbf{B}$, both of which have the same partition vector $\mathbf{s}$,

$$(\bar{\mathbf{\Lambda}} + \bar{\mathbf{B}})^{-\frac{1}{2}} = \bar{\mathbf{\Lambda}}^{-\frac{1}{2}} + \bar{\mathbf{D}} \quad (3.62)$$

where $\bar{\mathbf{D}}$ is the full expansion matrix with partition vector $\mathbf{s}$ of

$$\mathbf{D} = \mathbf{S}^{-\frac{1}{2}} \left[(\mathbf{\Lambda} + \mathbf{S}^{\frac{1}{2}}\mathbf{B}\mathbf{S}^{\frac{1}{2}})^{-\frac{1}{2}} - \mathbf{\Lambda}^{-\frac{1}{2}} \right] \mathbf{S}^{-\frac{1}{2}} \quad (3.63)$$

where $\mathbf{S} = \text{diag}(\mathbf{s})$.

Proof. The theorem can be proved by combining results of Theorem 1 and Lemma 1.

Substituting $\mathbf{B}$ in (3.50) with $\mathbf{D}$ in (3.30) and substituting $\mathbf{\Lambda}$ in (3.50) with $\mathbf{\Lambda}^{-1}$, we get

$$\mathbf{D} = \mathbf{S}^{-\frac{1}{2}} [(\mathbf{\Lambda} + \mathbf{S}^{\frac{1}{2}} ((\mathbf{\Lambda} \mathbf{S} + \mathbf{S} \mathbf{B} \mathbf{S})^{-1} \quad (3.64)$$

$$- (\mathbf{\Lambda} \mathbf{S})^{-1}) \mathbf{S}^{\frac{1}{2}})^{\frac{1}{2}} - \mathbf{\Lambda}^{-\frac{1}{2}}] \mathbf{S}^{-\frac{1}{2}} \quad (3.65)$$

$$= \mathbf{S}^{-\frac{1}{2}} [\mathbf{S}^{\frac{1}{2}} (\mathbf{\Lambda} \mathbf{S} + \mathbf{S} \mathbf{B} \mathbf{S})^{-1} \mathbf{S}^{\frac{1}{2}})^{\frac{1}{2}} - \mathbf{\Lambda}^{-\frac{1}{2}}] \mathbf{S}^{-\frac{1}{2}} \quad (3.66)$$

$$= \mathbf{S}^{-\frac{1}{2}} [\mathbf{S}^{\frac{1}{2}} \mathbf{S}^{-\frac{1}{2}} (\mathbf{\Lambda} + \mathbf{S}^{\frac{1}{2}} \mathbf{B} \mathbf{S}^{\frac{1}{2}})^{-1} \mathbf{S}^{-\frac{1}{2}} \mathbf{S}^{\frac{1}{2}})^{\frac{1}{2}} - \mathbf{\Lambda}^{-\frac{1}{2}}] \mathbf{S}^{-\frac{1}{2}} \quad (3.67)$$

$$= \mathbf{S}^{-\frac{1}{2}} \left[(\mathbf{\Lambda} + \mathbf{S}^{\frac{1}{2}} \mathbf{B} \mathbf{S}^{\frac{1}{2}})^{-\frac{1}{2}} - \mathbf{\Lambda}^{-\frac{1}{2}} \right] \mathbf{S}^{-\frac{1}{2}} \quad (3.68)$$

□

Deformable Volume Network for Learning Optical Flow

One of the major difficulties in estimating optical flow is large displacements. Learning optical flow with large displacements cannot be achieved by traditional Convolutional Neural Networks (CNNs). This is due to the fact that CNNs make use of small filters (sizes are rarely larger than 11×11). One cannot afford to increase filter sizes naïvely to cover large displacements, as the number of parameters and the computational cost both increase drastically.

To handle large displacements, multi-resolution representations are employed in several neural network models. SpyNet [Ranjan and Black, 2017] downsamples the original images into multiple scale levels and each level is handled by a CNN. FlowNet [Dosovitskiy et al., 2015], FlowNet2 [Ilg et al., 2017], PWC-Net [Sun et al., 2018b] and LiteFlowNet [Hui et al., 2018] create multi-resolution representation of images by striding during convolution. As a result, a convolution layer in CNNs that receives lower resolution feature maps has effectively larger receptive size. However, downsampling may lead to loss of valuable information such as small objects and detailed textures. As pointed out in [Yang and Soatto, 2017], the multi-resolution representation faces the “small things move fast” problem as small objects disappear in lower resolution and cannot be recovered in higher resolution due to their large motion. Although the problem is not severe in terms of performance on the current public benchmarks, it might limit the use of optical flow for high precision applications.

Another technique to handle large displacements in neural network models such as [Ilg et al., 2017; Ranjan and Black, 2017; Sun et al., 2018b] is warping, which propagates optical flow between two stages in a model. However, warping often creates distortions and artifacts.

Warping has been used in variational methods Brox et al. [2004]; Lucas and Kanade [1981] and neural network models Ilg et al. [2017]; Ranjan and Black [2017]; Sun et al. [2018b] for iteratively refining optical flow estimations in a multi-stage framework. The first stage covers large displacements and outputs a rough estimation. Then the second image (or its feature maps) is warped by the roughly estimated optical flow such that pixels of large displacements in the second image are moved

closer to their correspondences in the first image. As a result, the next stage, which receives the original first image and the warped second image as inputs, only needs to handle smaller displacements to refine the estimation.

Let $I : \mathbb{R}^2 \rightarrow \mathbb{R}^3$ denote the first image, $J : \mathbb{R}^2 \rightarrow \mathbb{R}^3$ denote the second image and $F : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ denote the optical flow field of the first image. The warped second image is defined as

$$\tilde{J}(\mathbf{p}) = J(\mathbf{p} + F(\mathbf{p})) \quad (4.1)$$

for image location $\mathbf{p} \in \mathbb{R}^2$ Ilg et al. [2017].

The warping operation creates a transformed image reasonably well if the new pixel locations $\mathbf{p} + F(\mathbf{p})$ do not occlude or collide with each other. For example, this is the case with the affine transform $F(\mathbf{p}) = \mathbf{A}\mathbf{p} + \mathbf{t}$ where $\mathbf{A}$ and $\mathbf{t}$ are the transformation parameters. However, for real-world images, occlusions are common (e.g., when an object moves and the background is still). If an image is warped with the optical flow which induces occlusions, duplicates will be created. The effect is demonstrated in Figure 4.1. The artifacts cannot be cleaned simply by subtracting the first or the second image from the warped image, as shown in Figure 4.1 (e) and (f). The artifacts induced by warping have been previously observed in Brox et al. [2009]; Wang et al. [2018]; Janai et al. [2018]. Intuitively, if a pixel which is moved by warping to a new location and no other pixel are moved to fill in its old location, the pixel will appear twice in the warped image.

Mathematically, consider the following example. Assume the value of $J(\mathbf{p}_1)$ is unique in J , that is, $J(\mathbf{p}) \neq J(\mathbf{p}_1)$ for all $\mathbf{p} \neq \mathbf{p}_1$. Then for an optical flow field in which

$$F(\mathbf{p}_1) = 0, \quad F(\mathbf{p}_2) = \mathbf{p}_1 - \mathbf{p}_2, \quad (4.2)$$

we have

$$\tilde{J}(\mathbf{p}_1) = J(\mathbf{p}_1 + F(\mathbf{p}_1)) \quad (4.3)$$

$$= J(\mathbf{p}_1 + 0) = J(\mathbf{p}_1), \quad (4.4)$$

$$\tilde{J}(\mathbf{p}_2) = J(\mathbf{p}_2 + F(\mathbf{p}_2)) \quad (4.5)$$

$$= J(\mathbf{p}_2 + \mathbf{p}_1 - \mathbf{p}_2) = J(\mathbf{p}_1). \quad (4.6)$$

Therefore $\tilde{J}(\mathbf{p}_1) = \tilde{J}(\mathbf{p}_2) = J(\mathbf{p}_1)$. Since the value of $J(\mathbf{p}_1)$ is unique in image J but not unique in $\tilde{J}$, a duplicate is created on the warped second image $\tilde{J}$.

4.1 Method

4.1.1 Deformable Cost Volume

Let I denote the first image, J denote the second image and $f_I : \mathbb{R}^2 \rightarrow \mathbb{R}^d$ and $f_J : \mathbb{R}^2 \rightarrow \mathbb{R}^d$ denote their feature maps of dimensionality d , respectively. The standard

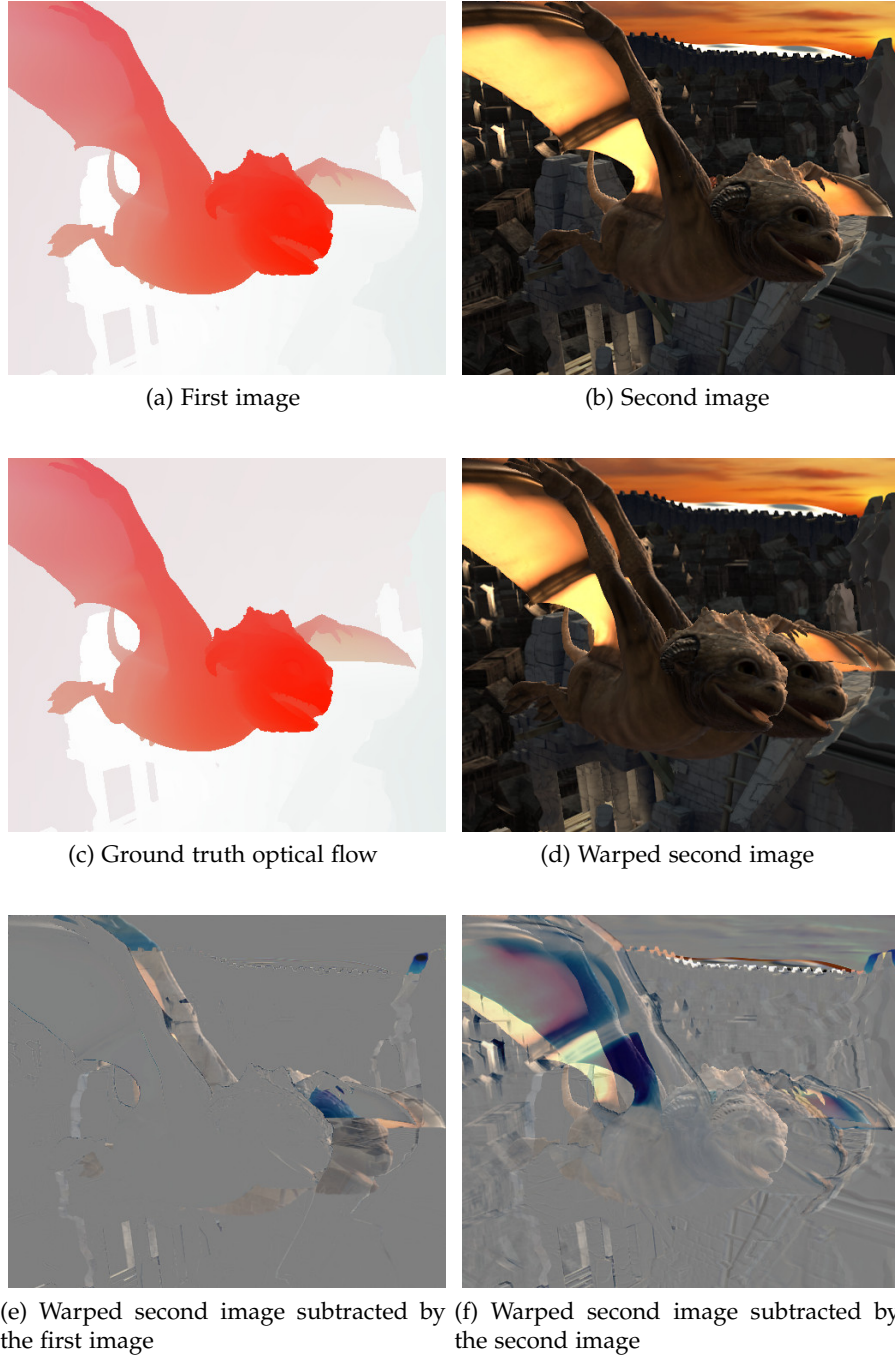


Figure 4.1: From (d), we can see the duplicates of the dragon head and wings. The images and the ground truth optical flow are from the Sintel dataset Butler et al. [2012]. Warping is done with function `image.warp()` in the Torch-image toolbox.

cost volume is defined as

$$C(\mathbf{p}, \mathbf{v}) = \|f_I(\mathbf{p}) - f_J(\mathbf{p} + \mathbf{v})\|, \quad (4.7)$$

for image location $\mathbf{p} \in \mathbb{R}^2$, neighbor $\mathbf{v} \in [-\frac{k-1}{2}, \frac{k-1}{2}]^2$ of neighborhood size k and a given vector norm $\|\cdot\|$.

The cost volume gives an explicit representation of displacements. The idea of using cost volume goes back to stereo matching Scharstein and Szeliski [2002]. When using the feature maps learned by neural networks, construction and processing of a fully connected cost volume, in which the neighborhood is large enough to cover the maximum displacement, leads to high performance in stereo matching Zbontar and LeCun [2016] and optical flow Xu et al. [2017]. However, the displacements in stereo matching are one-dimensional while in optical flow they are two-dimensional. For two images (and their feature maps) of resolution $m \times n$, the construction of the cost volume in equation (4.7) has time and space complexity $O(mndk^2)$. Naïvely increasing neighborhood size k to cover large displacements increases the computation quadratically. As a result, DCFlow requires several seconds to compute optical flow for a pair of images on a GPU and large memory usage Xu et al. [2017].

To reduce the computational burden, in Sun et al. [2018b]; Hui et al. [2018], multi-resolution feature maps of two images are created and a cost volume of a small neighborhood is constructed at each resolution. Warping is used to propagate the optical flow between two resolutions. However, as discussed before, the multi-resolution estimation of optical flow leads to the “small objects move fast” problem and warping induces artifacts and distortion. To avoid the drawbacks, we propose a new neural network module, the deformable cost volume. The key idea is: instead of deforming images or their feature maps, we deform the cost volume and leave the images and the feature maps unchanged.

The proposed deformable cost volume is defined as

$$C(\mathbf{p}, \mathbf{v}, r, F) = \|f_I(\mathbf{p}) - f_J(\mathbf{p} + r \cdot \mathbf{v} + F(\mathbf{p}))\| \quad (4.8)$$

where r is the dilation rate and $F(\cdot)$ is an external flow field. The dilation rate r is introduced to enlarge the size of the neighborhood to handle large displacements without reducing the resolution. This is inspired by the dilated convolution Chen et al. [2016]; Yu and Koltun [2015] which enlarges its receptive field in a similar way. $F(\cdot)$ can be obtained from the optical flow estimated from a previous stage or an external algorithm. If $F(\mathbf{p}) = 0$ for all $\mathbf{p}$ and $r = 1$, then the deformable cost volume is reduced to the standard cost volume. For non-integer $F(\mathbf{p})$, bilinear interpolation is used. The deformable cost volume is illustrated in Figure 4.2.

Since the deformable cost volume does not distort f_I or f_J , the artifacts associated with warping will not be created. Optical flow can be inferred from the deformable cost volume solely without resorting to the feature maps of the first image to counter the duplicates.

The deformable cost volume is differentiable with respect to $f_I(\mathbf{p})$ and $f_J(\mathbf{p} + r \cdot \mathbf{v} + F(\mathbf{p}))$ for each image location $\mathbf{p}$. Due to bilinear interpolation, the deformable cost volume is also differentiable with respect to $F(\mathbf{p})$, using the same technique as in Ilg et al. [2017]; Jaderberg et al. [2015]. Therefore, the deformable cost volume can be inserted in a neural network for end-to-end learning optical flow.

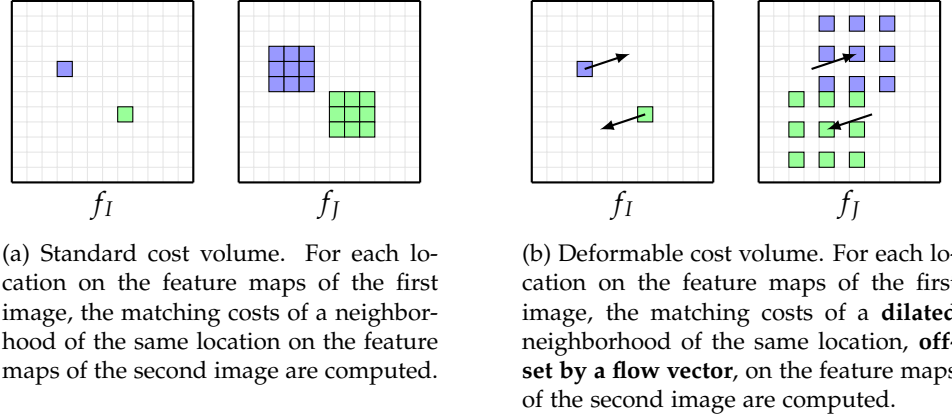


Figure 4.2: Cost Volumes

To see how the deformable cost volume avoids the artifacts of warping, consider the following. Assume $F(\mathbf{p}_1) = \mathbf{0}$ and $F(\mathbf{p}_2) = \mathbf{p}_1 - \mathbf{p}_2$. The standard cost volume (4.7) with warping (4.1) gives

$$C(\mathbf{p}_1, \mathbf{0}) = \|f_I(\mathbf{p}_1) - \tilde{f}_J(\mathbf{p}_1 + \mathbf{0})\| \quad (4.9)$$

$$= \|f_I(\mathbf{p}_1) - f_J(\mathbf{p}_1 + F(\mathbf{p}_1))\| \quad (4.10)$$

$$= \|f_I(\mathbf{p}_1) - f_J(\mathbf{p}_1)\| \quad (4.11)$$

$$C(\mathbf{p}_1, \mathbf{p}_2 - \mathbf{p}_1) = \|f_I(\mathbf{p}_1) - \tilde{f}_J(\mathbf{p}_1 + \mathbf{p}_2 - \mathbf{p}_1)\| \quad (4.12)$$

$$= \|f_I(\mathbf{p}_1) - \tilde{f}_J(\mathbf{p}_2)\| \quad (4.13)$$

$$= \|f_I(\mathbf{p}_1) - f_J(\mathbf{p}_2 + F(\mathbf{p}_2))\| \quad (4.14)$$

$$= \|f_I(\mathbf{p}_1) - f_J(\mathbf{p}_1)\| \quad (4.15)$$

That $C(\mathbf{p}_1, \mathbf{0}) = C(\mathbf{p}_1, \mathbf{p}_2 - \mathbf{p}_1)$ implies $f_I(\mathbf{p}_1)$ has the same matching cost for $f_J(\mathbf{p}_1)$ and $f_J(\mathbf{p}_2)$, which does not hold in general and makes the matching ambiguous. On the other hand, with deformable cost volume (4.8) of dilation rate one, we have

$$C(\mathbf{p}_1, \mathbf{0}) = \|f_I(\mathbf{p}_1) - f_J(\mathbf{p}_1)\| \quad (4.16)$$

$$C(\mathbf{p}_1, \mathbf{p}_2 - \mathbf{p}_1) = \|f_I(\mathbf{p}_1) - f_J(\mathbf{p}_2 + F(\mathbf{p}_1))\| \quad (4.17)$$

$$= \|f_I(\mathbf{p}_1) - f_J(\mathbf{p}_2)\| \quad (4.18)$$

As $C(\mathbf{p}_1, \mathbf{0}) \neq C(\mathbf{p}_1, \mathbf{p}_2 - \mathbf{p}_1)$ in general, the artifact is avoided.

4.1.2 Deformable Volume Network

Our proposed model is the Deformable Volume Network (Devon), as illustrated in Figure 4.3. Devon has multiple stages. Each stage is a neural network with an identical Siamese architecture, which consists of an encoding module, a relation module and a decoding module. Each stage outputs the optical flow field of 1/4 resolution

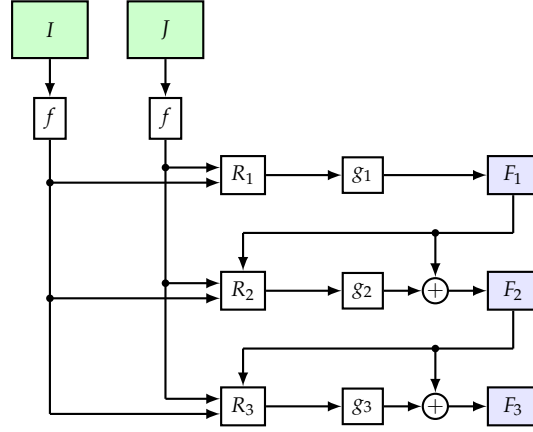


Figure 4.3: I denotes the first image, J denotes the second image, f denotes the encoding module (§4.1.2.1), R_t denotes the relation module (§4.1.2.2), g_t denotes the decoding module (§4.1.2.3) and F_t denotes the estimated optical flow for stage t .

and then the flow from last stage is bilinearly upsampled to obtain the final prediction. The optical flow estimated from a previous stage is propagated to the current one through the deformable cost volume and residual connections.

Compared to previous neural network models Dosovitskiy et al. [2015]; Ilg et al. [2017]; Ranjan and Black [2017]; Sun et al. [2018b]; Hui et al. [2018], Devon is different in the following major ways: (1) All stages in Devon output optical flow of the same resolution. While Devon handles multi-scale motion by the deformable cost volume, it does not use the multi-resolution representation of motion. Extensive downsampling leads to the loss of information and deficiency in handling the “small objects move fast” problem. The advantage of estimating motion direct on a single high resolution is also shown in Yang and Soatto [2017]. (2) Each stage acts on the undistorted images. No warping is used. Therefore, the artifacts discussed in §?? can be avoided. (3) The decoding module only receives inputs from the relation module. Therefore, neural networks infer the optical flow solely from the relations between two images, rather than memorize the optical flow pattern of a single image as a short-cut. The short-cut issue has appeared when applying neural networks to learn monocular stereo Ummenhofer et al. [2017]. On the contrary, in FlowNetC, PWC-Net and LiteFlowNet, the decoding module also receives inputs from the encoding module of the first image. (4) The encoding module is shared in all stages.

We describe the details of each module structure below. All convolution layers have zero-padding size one. Besides, all convolution layers, except the last one in the encoding module and the last one in the decoding module, are followed by a leaky ReLU function He et al. [2015] with leakiness 0.1.

4.1.2.1 Encoding Module

The encoding module has a U-Net structure Ronneberger et al. [2015] with residual connections He et al. [2016], as shown in Figure 4.4. Since the model output optical flow of 1/4 resolution, the encoding module has 6 convolution layers of stride 2 but 4 upsampling layers. We also experimented with a simpler module and result is shown in the ablation analysis in §4.2.3.

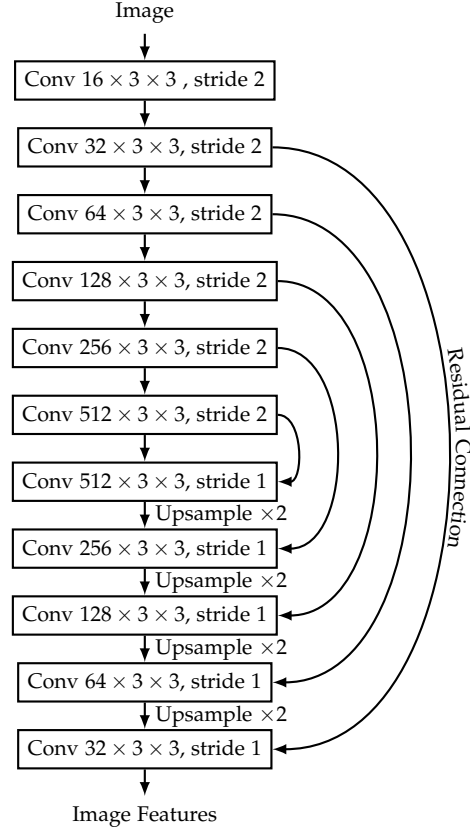


Figure 4.4: The residual connection denotes the output of a layer is added to the output of another layer.

4.1.2.2 Relation Module

The relation module is illustrated in Figure 4.5. It concatenates the outputs of five deformable cost volumes, which have different neighborhood size k or dilation rate r , as shown in Table 4.1. Such combination enables dense correspondences nearby the center of an image location and sparse correspondences in peripheral to capture multi-scale motion. This is consistent with the fact that small displacements are more frequent in natural videos Roth and Black [2007] and resembles the structure of retina, as illustrated in Figure 4.6.

Since Devon is a multi-stage model which performs coarse-to-fine estimation of

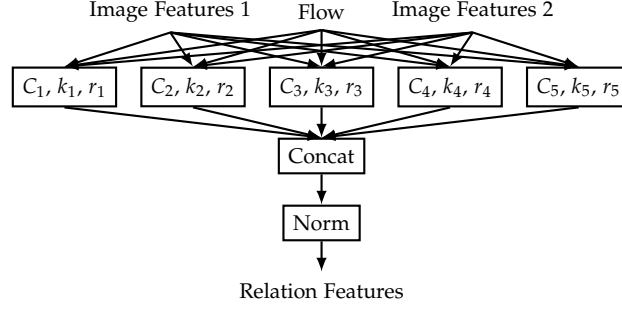


Figure 4.5: $C_1 \sim C_5$ denote the deformable cost volumes. $k_1 \sim k_5$ denote the neighborhood sizes. $r_1 \sim r_5$ denote the dilation rates. Concat denotes concatenation. Norm denotes normalization.

	$(k_1, k_2, k_3, k_4, k_5)$	$(r_1, r_2, r_3, r_4, r_5)$
R_1	$(5, 5, 5, 5, 9)$	$(1, 3, 8, 12, 20)$
R_2	$(5, 5, 5, 5, 9)$	$(1, 3, 8, 10, 12)$
R_3	$(5, 5, 5, 5, 9)$	$(1, 3, 4, 5, 7)$

Table 4.1: Hyperparameters of deformable cost volumes in Devon.

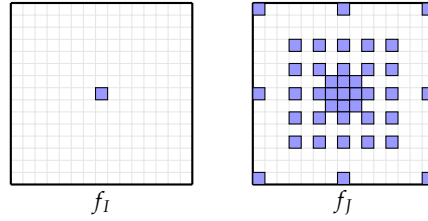


Figure 4.6: In this example, three cost volumes of neighborhood sizes $(k_1, k_2, k_3) = (3, 5, 3)$ and dilation rates $(r_1, r_2, r_3) = (1, 2, 7)$ respectively are concatenated.

optical flow, we gradually decrease the dilation rates as the optical flow is expected to get finer in later stages. We also experimented with using the cost volumes of the same neighbor sizes and dilation rates for all three stages, the result of which is reported in §4.2.3.

Next, for each location in the concatenated feature maps, a normalization method is applied across the channels. We apply $f(C) = \exp(-C)$ elements-wise, which encourages the elements representing the minimum cost to pop-up. Such normalization improves the estimation accuracy as shown in §4.2.3.

The output of this module has size $(k_1^2 + k_2^2 + k_3^2 + k_4^2 + k_5^2) \times m \times n$, where m is the height and n is the width of the feature maps. When the module does not receive an optical flow as one of the inputs (in the first stage), it is set to receive a zero-valued optical flow field.

Since the relation module captures multi-scale motion without reducing the resolution, the presence and precise location of small objects which move fast are re-

tained. This allows Devon to have better chance in solving the “small objects move fast” problem.

4.1.2.3 Decoding Module

In the decoding module, we again use the U-Net structure with residual connections. The whole module structure is illustrated in Figure 4.7. Each stage has its own decoder. We also experimented with sharing decoder in all three stages. The result is reported in §4.2.3.

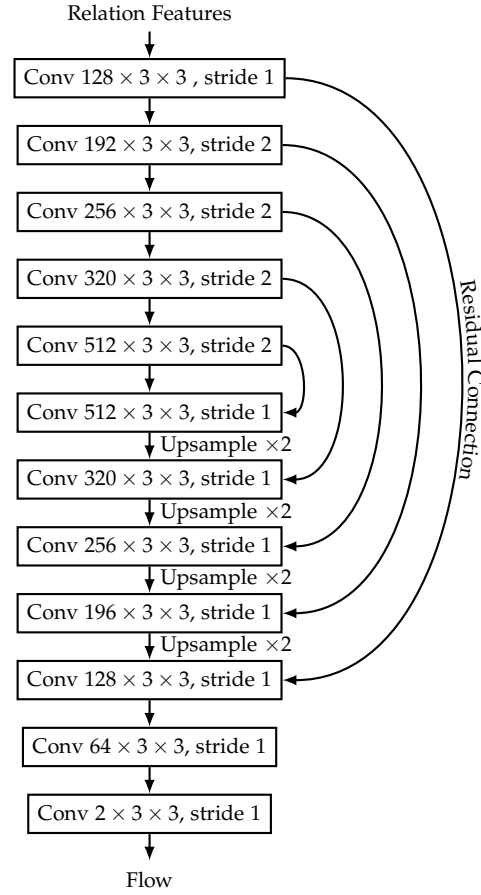


Figure 4.7: The residual connection denotes the output of a layer is added to the output of another layer.

4.2 Experiments

We evaluate Devon on two challenging benchmarks which contain large motions: Sintel Butler et al. [2012] and KITTI Geiger et al. [2012], as in Dosovitskiy et al. [2015]; Ranjan and Black [2017]; Ilg et al. [2017]; Sun et al. [2018b]. We compare Devon with

the previous neural network models: FlowNetS Dosovitskiy et al. [2015], FlowNetC Dosovitskiy et al. [2015], FlowNet2 Ilg et al. [2017], SpyNet Ranjan and Black [2017], PWC-Net Sun et al. [2018b] and LiteFlowNet Hui et al. [2018]. We use Devon with three stages. We use l_1 norm for the deformable cost volumes.

4.2.1 Training

Our training procedure largely follows from Ilg et al. [2017]; Sun et al. [2018b]. We first train our network on FlyingChairs. We use the l_2 loss function

$$L = \sum_t \gamma_t |F_{\text{GT}} - \hat{F}_t|_2 \quad (4.19)$$

where F_{GT} denotes the ground-truth optical flow and $\hat{F}_t$ denotes the bilinear upsampled network output at stage t . For Devon with three stages, we choose $\gamma_1 = 0.2$, $\gamma_2 = 0.3$ and $\gamma_3 = 0.5$. All pixel values of the images are multiplied by $1/255$. Empirically, such normalization is found to accelerate the training. For optimization algorithm, we use Adam Kingma and Ba [2014] with $\beta_1 = 0.9$, $\beta_2 = 0.999$ and weight decay factor 0.0004. We use the weight initialization method in He et al. [2015]. We use data augmentation which is consisted of random cropping of size 448×384 , translation, rotation, color jittering and Gaussian noise. There are totally 500000 mini-batch updates with mini-batch size 8. The learning rate starts from $1e - 4$ and halves at 200000, 300000 and 400000 mini-batch updates.

After training on FlyingChairs, we fine-tune our model on FlyingThings3D (final pass) with the robust loss function

$$L = \sum_t \gamma_t (|F_{\text{GT}} - \hat{F}_t|_1 + \epsilon)^q \quad (4.20)$$

where $q = 0.4$ and $\epsilon = 0.01$. There are totally 500000 mini-batch updates with mini-batch size 4. We use data augmentation which is consisted of random cropping of size 786×384 , translation and color jittering. We do not use Gaussian noise during data augmentation since the images have motion blur already. The learning rate starts from $1e - 5$ and halves at 200000, 300000 and 400000 mini-batch updates.

For the evaluation on Sintel, we fine-tune the model, which is trained on FlyingChairs and fine-tuned on FlyingThings3D, on the Sintel training set. We use the robust loss function described above and learning rate schedule in Sun et al. [2018a]. The data augmentation consists of random cropping of size 786×384 , rotation, horizontal flipping and color jittering. The mini-batch size is 4.

For the evaluation on KITTI, we fine-tune the network initially fine-tuned on FlyingThings3D, on the KITTI training set. We use the robust loss function described above and learning rate schedule in Sun et al. [2018a]. The data augmentation consists of random cropping of size 896×320 , horizontal flipping and color jittering. The mini-batch size is 4. We use the same learning schedule as the one for Sintel. The training set is mixed with KITTI 2012 and KITTI 2015. Invalid pixels are excluded in computing the loss.

	Train		Test	
	Clean	Final	Clean	Final
EpicFlow Revaud et al. [2015]	-	-	4.12	6.29
MRFlow Wulff et al. [2017]	1.83	3.59	2.53	5.38
FlowFields Bailer et al. [2015]	-	-	3.78	5.36
DCFlow Xu et al. [2017]	-	-	3.54	5.12
FlowNetS	4.35	5.46	-	-
FlowNetS (ft)	(3.66)	(4.44)	6.96	7.52
FlowNetC	3.52	5.00	-	-
FlowNetC (ft)	(3.50)	(3.89)	6.85	8.51
FlowNet2	2.02	3.14	3.96	6.02
FlowNet2 (ft)	(1.45)	(2.01)	4.16	5.74
SpyNet	4.12	5.57	6.69	8.43
SpyNet (ft)	(3.17)	(4.32)	6.64	8.36
PWC-Net	2.55	3.93	-	-
PWC-Net (ft)	(2.02)	(2.08)	4.39	5.04
LiteFlowNet	2.48	4.04	-	-
LiteFlowNet (ft)	(1.35)	(1.78)	4.54	5.38
Devon	2.45	3.72	-	-
Devon (ft)	(1.97)	(2.67)	4.34	6.35

Table 4.2: (ft) denotes the fine-tuning.

All the experiments are conducted with PyTorch. The deformable cost volume is implemented in CUDA with PyTorch interface.

Clean	d0-10	d10-60	d60-140	s0-10	s10-40	s40+
SpyNet	6.69	4.37	3.29	1.40	5.53	49.71
FlowNet2	4.82	2.56	1.74	0.96	3.23	35.54
PWC-Net	4.68	2.08	1.52	0.90	2.99	31.28
LiteFlowNet	3.27	1.44	0.93	0.50	1.73	31.41
Devon	4.12	1.53	0.82	0.76	2.45	26.72
Final	d0-10	d10-60	d60-140	s0-10	s10-40	s40+
SpyNet	5.50	3.12	1.72	0.83	3.34	43.44
FlowNet2	3.27	1.46	0.86	0.60	1.89	27.35
PWC-Net	3.83	1.31	0.56	0.70	2.19	23.56
LiteFlowNet	4.09	2.10	1.73	0.75	2.75	34.72
Devon	5.34	2.88	2.30	1.12	3.83	38.38

Table 4.3: Detailed results on Sintel (end-point error) for different distances from motion boundaries (d) and velocities (s).

4.2.2 Main Results

In Figure 4.8, 4.14 and 4.20, we show visualization results of situations where small objects move fast. All models were trained on FlyingChairs and then fine-tuned on FlyingThings3D. No additional fine-tuning is applied.

From Figure 4.8, 4.14 and 4.20, we can see Devon gives much more accurate estimation of the small objects. The results on Sintel and KITTI are listed in Table 4.2, 4.3 and 4.4, from which we can see Devon outperforms PWC-Net and LiteFlowNet on Sintel clean pass, though not on Sintel final pass and KITTI.

	KITTI 2012		KITTI 2015	
	Train EPE	Test EPE	Train EPE	Test F1-all
EpicFlow	3.09	3.8	27.18%	27.10%
MRFlow	-	-	14.09%	12.19%
FlowFields	-	3.0	-	19.80%
DCFlow	-	-	15.09%	14.83%
FlowNetS	8.26	-	-	-
FlowNetS (ft)	-	9.1	-	-
FlowNetC	9.35	-	-	-
FlowNetC (ft)	-	-	-	-
FlowNet2	4.09	-	10.08	-
FlowNet2 (ft)	(1.28)	1.8	(2.30)	10.41%
SpyNet	9.12	-	-	-
SpyNet (ft)	(4.13)	4.7	-	35.07%
PWC-Net	4.14	-	10.35	-
PWC-Net (ft)	(1.08)	1.7	(2.16)	9.16%
LiteFlowNet	4.00	-	10.39	-
LiteFlowNet (ft)	(1.05)	1.6	(1.62)	9.38%
Devon	4.73	-	10.65	-
Devon (ft)	(1.29)	2.6	(2.00)	14.31 %

Table 4.4: (ft) denotes the fine-tuning. EPE denotes end-point error. F1-all denotes the ratio of pixels where the flow estimate is incorrect by both ≥ 3 pixels and $\geq 5\%$.

4.2.3 Ablation Analysis

We perform an ablation analysis of Devon trained on FlyingChairs. There are six ablation cases: (1) With warping. We replace the deformable cost volumes with warping and standard cost volumes (with dilation). (2) With shortcut. We additionally concatenate the relation module with the feature maps of the first image and feed them into the decoder. (3) Without dilation. We replace the concatenated deformable cost volumes with one deformable cost volume of neighborhood size 13×13 and dilation rate one. (4) Without norm. We remove the normalization in the relation modules. (5) Shared decoder. We let all stages share one decoder and set the hyperparameters of the relation modules in second and third stage the same as the first stage. (6) Simple encoder. We replace the U-Net structure encoding module with a simpler one: 4 convolutional layers of 32 units receptive field size 3×3 . The first two layers have stride 2 and the last two have stride 1. For (1)~(5), the changes are applied to all stages in Devon. The results are listed in Table 4.5, from which we can see the architecture of Devon is robust to various changes.

4.2.4 Runtime

We report the runtime of Devon and its variants in Table 4.6. The timing was recorded on a NVIDIA TITAN Xp graphics card for processing a pair of RGB images of size 1024×448 .

	FlyingChairs Valid	Sintel clean Train	KITTI 2015 Train
Full model	1.87	2.99	13.25
With warping	1.88	2.98	13.73
With shortcut	1.79	2.97	15.31
Without dilation	1.95	3.12	13.84
Without norm	1.99	3.30	15.64
Shared decoder	1.96	2.90	13.89
Simple encoder	1.88	3.00	14.04

Table 4.5: Results of ablation experiments after training on FlyingChairs (end-point error).

	Forward	Backward
Full model	50.51	177.17
With warping	57.75	182.75
With shortcut	51.62	181.47
Without dilation	49.07	147.74
Without norm	52.33	177.25
Shared decoder	51.15	178.54
Simple encoder	49.98	180.78

Table 4.6: Runtime (ms).

4.3 Discussion

While Devon achieves better results in handling small objects moving fast in the visualization results, it does not outperform multi-resolution based methods such as PWC-Net on Sintel final pass and KITTI. We conjecture that this is due to the fact that Sintel and KITTI mostly contain large objects (e.g. human bodies, cars and buildings), for which the multi-resolution approach might be more suitable. An interesting extension of our work is to combine multi-resolution approach and Devon to handle objects of diverse sizes and speed.

From Table 4.6, we can see using deformable cost volume achieves shorter runtime than using warping and standard cost volumes. This is because the deformation and the cost volume construction are merged into one process in deformable cost volume and therefore reduces the runtime. Along with Table 4.6, the result suggests that the deformable cost volume is a suitable replacement of warping and standard cost volume in Devon and potentially other models.

The deformation in deformable cost volume is different from the one in deformable convolutional networks Dai et al. [2017]. In deformable cost volume, the cost volume is offset by an external optical flow and dilation. There is no learnable parameter while in deformable convolutional networks, the deformation is element-wise and the offset parameters are learned during training. Another related work is the deformable spatial pyramid matching Kim et al. [2013]; Hur et al. [2015] which uses deformation in the classic energy minimization framework for hierarchical dense matching. Applying normalization on hidden unit outputs is found

advantageous in modeling general image relations Lu et al. [2017].

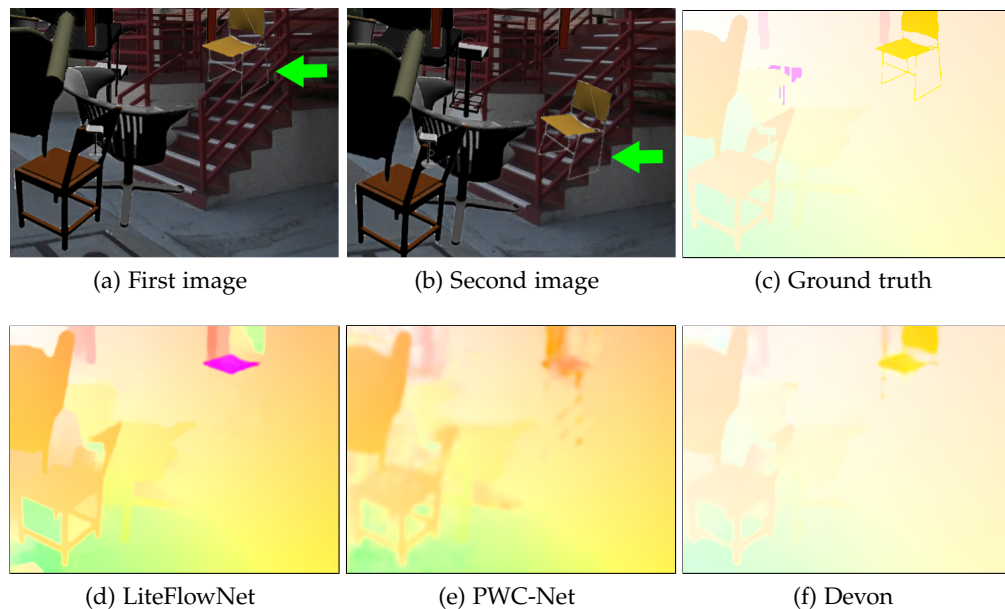


Figure 4.8: Green arrows indicate the small object that moves fast.

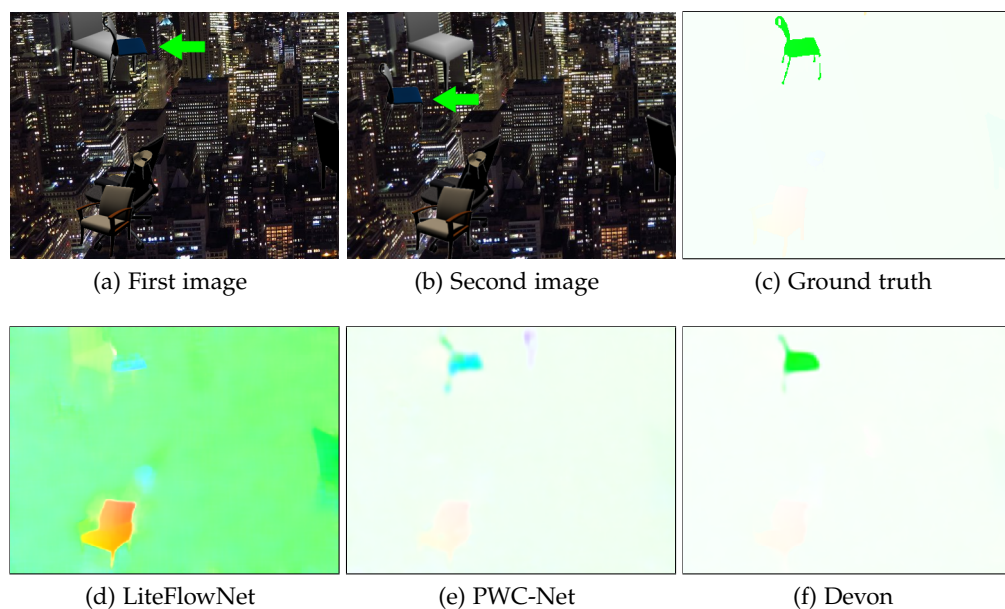


Figure 4.9: FlyingChairs (validation set).

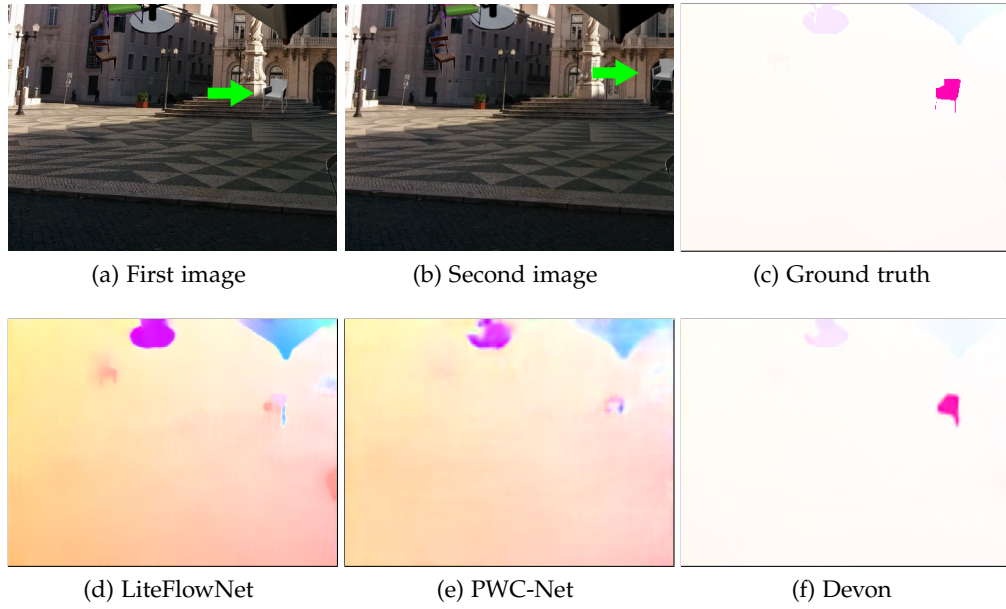


Figure 4.10: FlyingChairs (validation set).

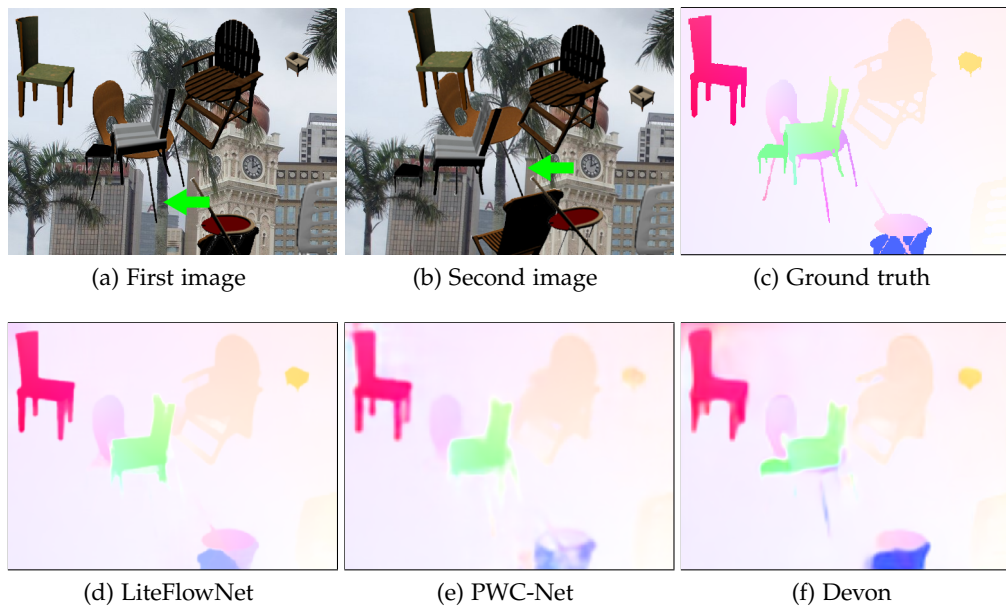


Figure 4.11: FlyingChairs (validation set).

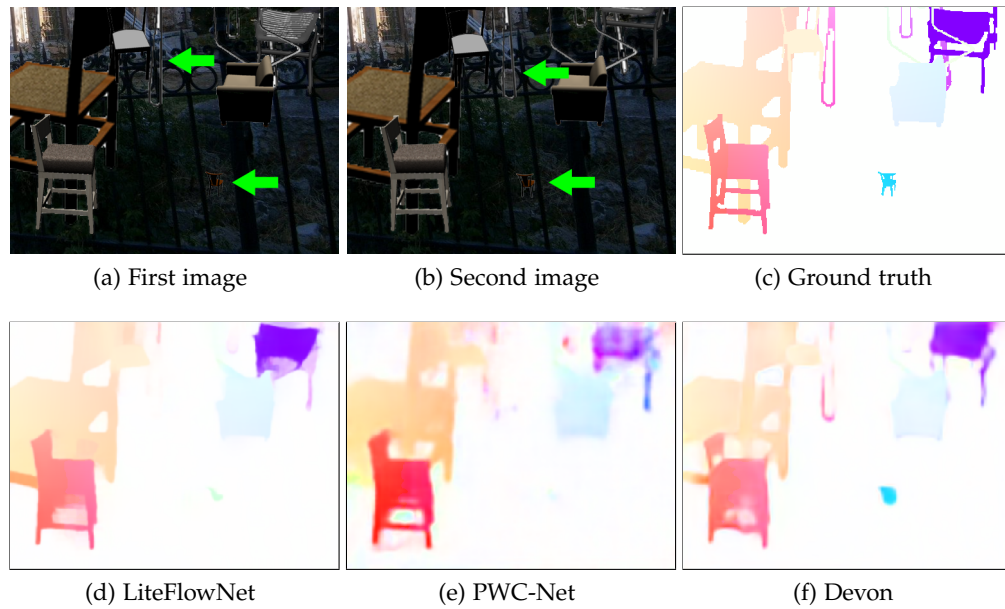


Figure 4.12: FlyingChairs (validation set).

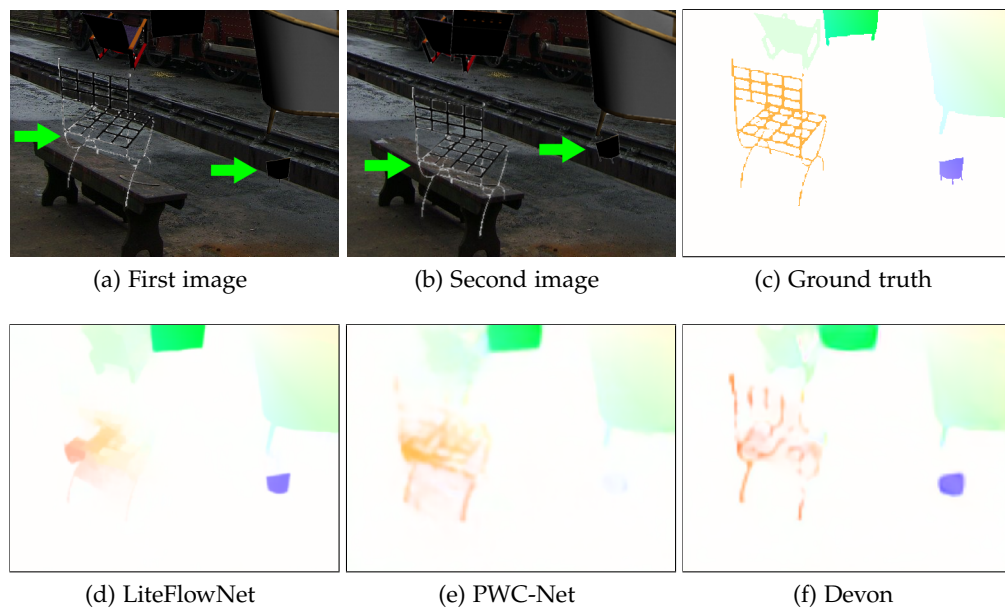


Figure 4.13: FlyingChairs (validation set).

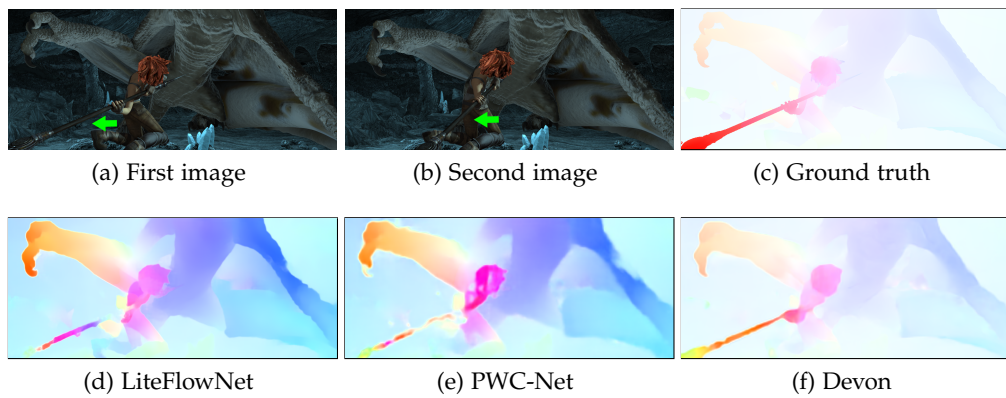


Figure 4.14: Green arrows indicate the small object that moves fast.

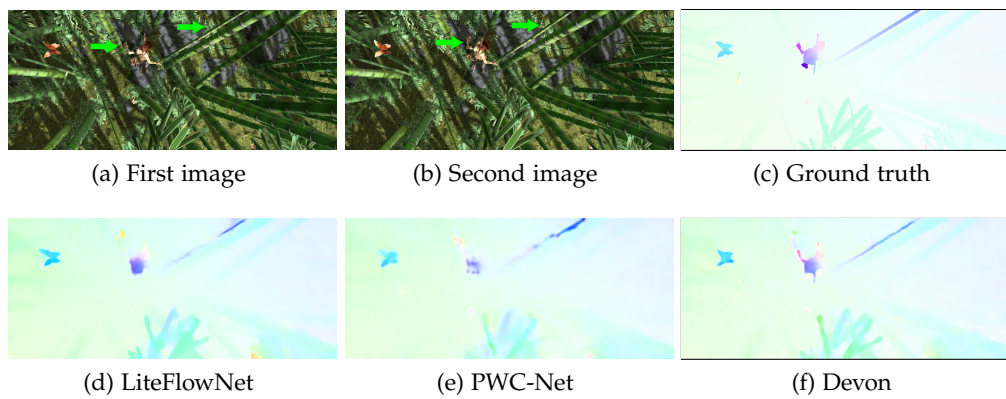


Figure 4.15: Sintel (training set).

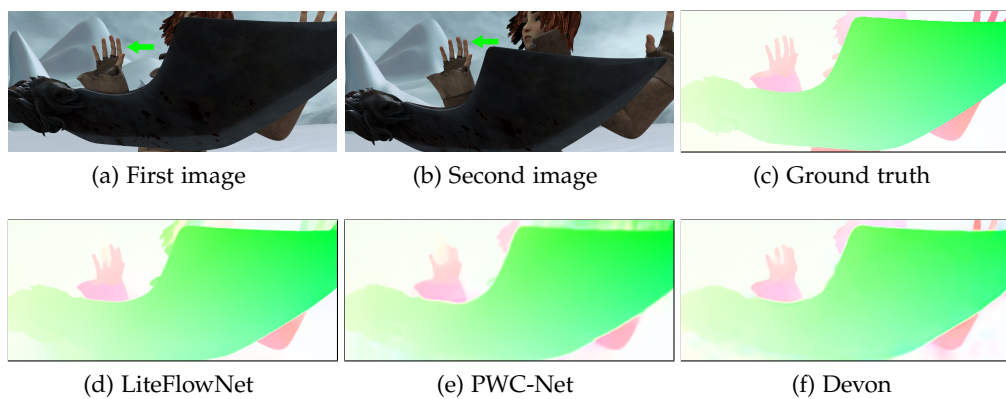


Figure 4.16: Sintel (training set).

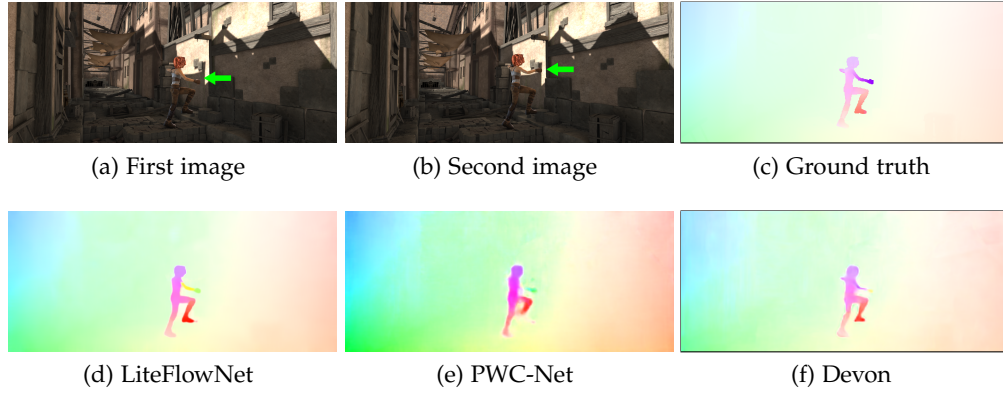


Figure 4.17: Sintel (training set).

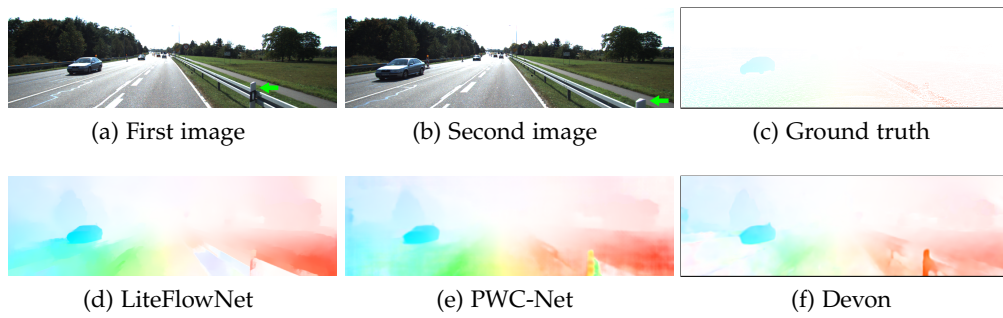


Figure 4.18: KITTI 2015 (training set).

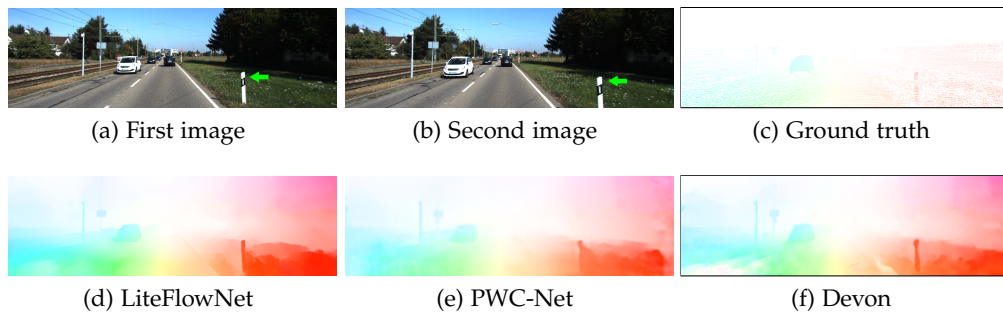


Figure 4.19: KITTI 2015 (training set).

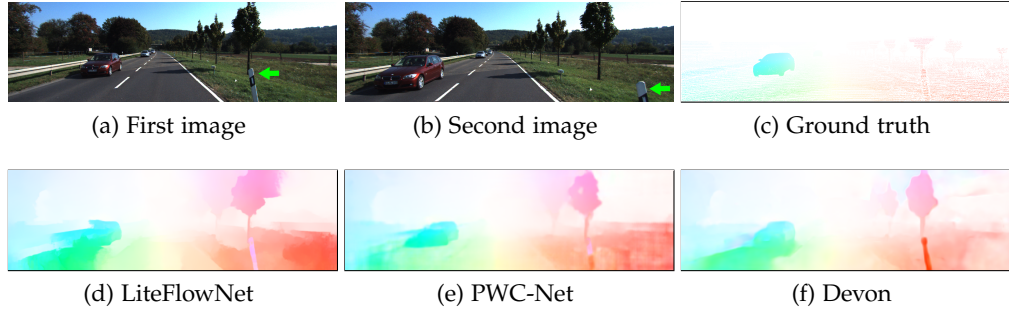


Figure 4.20: Green arrows indicate the small object that moves fast.

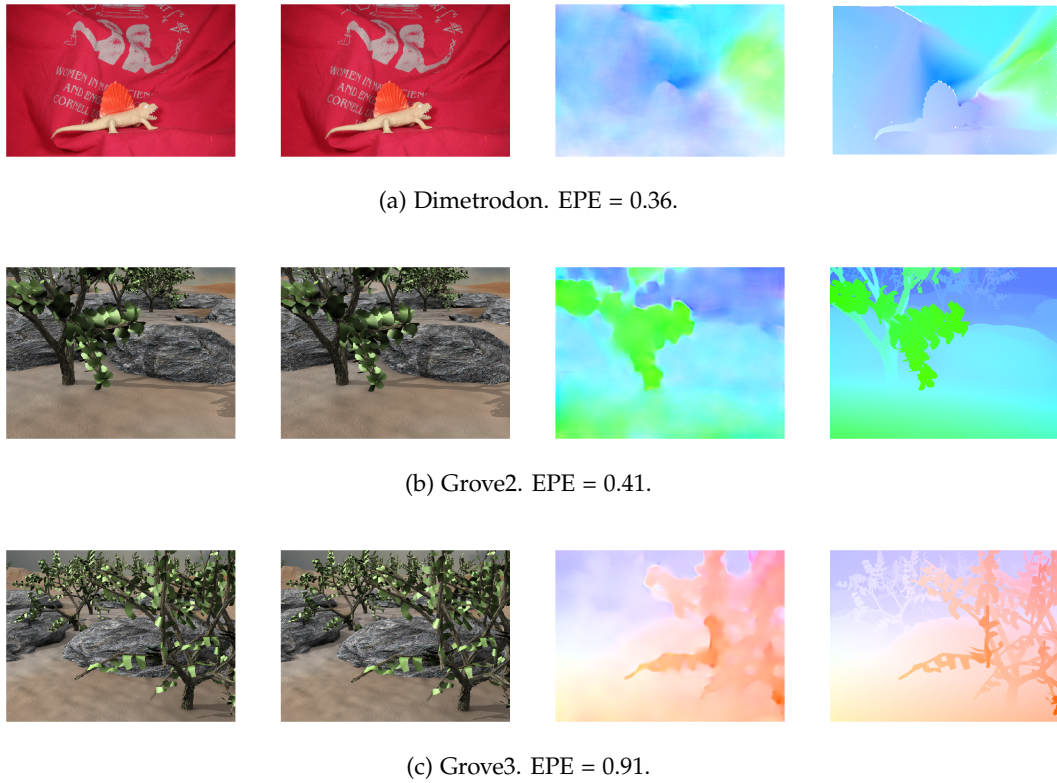
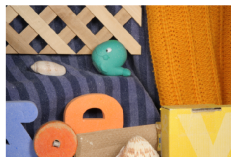
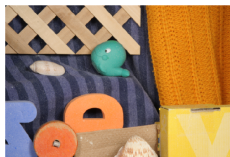


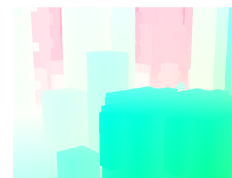
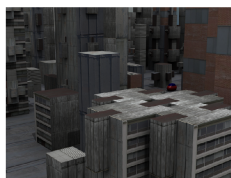
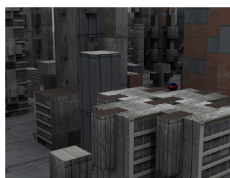
Figure 4.21: From left to right columns: first image, second image, estimation by Devon, ground-truth.



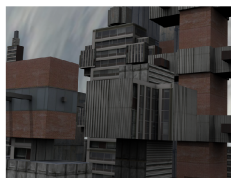
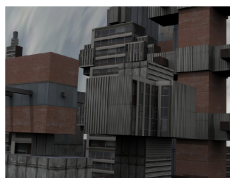
(a) Hydrangea. EPE = 0.34.



(b) RubberWhale. EPE = 0.39.



(c) Urban2. EPE = 0.59.



(d) Urban3. EPE = 1.56.



(e) Venus. EPE = 0.42.

Figure 4.22: From left to right columns: first image, second image, estimation by Devon, ground-truth.

Conclusion

In this thesis, we addressed three problems in neural networks: vanishing/exploding gradients, scalability of second-order optimization methods and optical flow estimation.

For the vanishing/exploding gradients problem, we have introduced bidirectionally self-normalizing neural networks (BSNNs) which constrain both forward and backward signal propagation using Gaussian-Poincaré normalized activation functions and orthogonal weight matrices. BSNNs are not restrictive since many commonly used activation functions can be Gaussian-Poincaré normalized. We have rigorously proved that the vanishing/exploding gradients problem disappears in BSNNs with high probability under mild conditions. Experiments on synthetic and real-world data confirm the validity of our theory and demonstrate that BSNNs have excellent trainability without batch normalization. Currently, the theoretical analysis is limited to same width, fully-connected neural networks. Future work includes extending our theory to more sophisticated networks such as convolutional architectures as well as investigating the approximation and generalization capabilities of BSNNs.

For the scalability of second-order optimization methods, in our work, we propose a new matrix approximation method which allows efficient storage and computation of matrix inverse and inverse square root that are required by second-order optimization methods.

For optical flow estimation, we proposed a new neural network module, Deformable Cost Volume, which allows the representation of multi-scale motion in a single high resolution and avoids the drawback of warping. Based on it, we designed the Deformable Volume Network, which is demonstrated to be effective in estimating optical flow, especially in situations where small objects move fast.

Deep neural networks is still an active ongoing research area. While we have made several advances in the optimization methods, training deep neural networks is far from satisfactory in methodology or being understood in theory. Moreover, applications of deep neural networks for wide range of problems are yet to be discovered. We look forward to seeing fruitful progresses in both directions.

Bibliography

- ABU-EL-HAJJA, S., 2017. Proportionate gradient updates with percentdelta. *arXiv*, (2017). (cited on page 32)
- ALBERTS, T. AND KHOSHNEVISAN, D., 2018. *Calculus on Gauss Space: An Introduction to Gaussian Analysis*. (cited on page 26)
- AMARI, S.-I., 1998. Natural gradient works efficiently in learning. *Neural Computation*, (1998). (cited on page 5)
- AMARI, S.-I.; CICHOCKI, A.; AND YANG, H. H., 1996. A new learning algorithm for blind signal separation. *NIPS*, (1996). (cited on page 6)
- BAILER, C.; TAETZ, B.; AND STRICKER, D., 2015. Flow fields: Dense correspondence fields for highly accurate large displacement optical flow estimation. *ICCV*, (2015). (cited on page 53)
- BEAUCHEMIN, S. S. AND BARRON, J. L., 1995. The computation of optical flow. *ACM computing surveys*, (1995). (cited on page 7)
- BECKER, S. AND LECUN, Y., 1988. Improving the convergence of back-propagation learning with second order methods. *Technical Report*, (1988). (cited on pages 6 and 29)
- BENGIO, Y.; SIMARD, P.; AND FRASCONI, P., 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, (1994). (cited on page 2)
- BOBKOV, S. G., 2003. On concentration of distributions of random weighted sums. *Annals of Probability*, (2003). (cited on page 12)
- BOGACHEV, V. I., 1998. *Gaussian Measures*. American Mathematical Society. (cited on page 11)
- BROCK, A.; LIM, T.; RITCHIE, J. M.; AND WESTON, N., 2017. Neural photo editing with introspective adversarial networks. *International Conference on Learning Representations*, (2017). (cited on page 13)
- BROX, T.; BREGLER, C.; AND MALIK, J., 2009. Large displacement optical flow. *CVPR*, (2009). (cited on page 44)
- BROX, T.; BRUHN, A.; PAPENBERG, N.; AND WEICKERT, J., 2004. High accuracy optical flow estimation based on a theory for warping. *ECCV*, (2004). (cited on page 43)

-
- BUTLER, D. J.; WULFF, J.; STANLEY, G. B.; AND BLACK, M. J., 2012. A naturalistic open source movie for optical flow evaluation. *ECCV*, (2012). (cited on pages 7, 45, and 51)
- CHEN, L.-C.; PAPANDREOU, G.; KOKKINOS, I.; MURPHY, K.; AND YUILLE, A. L., 2016. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *arXiv*, (2016). (cited on page 46)
- CHOW, E. AND SAAD, Y., 1997. Approximate inverse techniques for block-partitioned matrices. *SIAM Journal on Scientific Computing*, (1997). (cited on page 38)
- CIREGAN, D.; MEIER, U.; AND SCHMIDHUBER, J., 2012. Multi-column deep neural networks for image classification. *IEEE Conference on Computer Vision and Pattern Recognition*, (2012). (cited on page 2)
- CLEVERT, D.-A.; UNTERTHINER, T.; AND HOCHREITER, S., 2016. Fast and accurate deep network learning by exponential linear units (elus). *International Conference on Learning Representations*, (2016). (cited on page 11)
- CYBENKO, G., 1989. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, (1989). (cited on page 1)
- DAI, J.; QI, H.; XIONG, Y.; LI, Y.; ZHANG, G.; HU, H.; AND WEI, Y., 2017. Deformable convolutional networks. *ICCV*, (2017). (cited on page 55)
- DAWKINS, J., 2016. Normalized vector of gaussian variables is uniformly distributed on the sphere. *Math Stackexchange*, (2016). <https://math.stackexchange.com/q/1864563>. (cited on page 26)
- DOSOVITSKIY, A.; FISCHER, P.; ILG, E.; HÄUSSER, P.; HAZIRBAŞ, C.; GOLKOV, V.; VAN DER SMAGT, P.; CREMERS, D.; AND BROX, T., 2015. Flownet: Learning optical flow with convolutional networks. *ICCV*, (2015). (cited on pages 7, 43, 48, 51, and 52)
- DUCHI, J.; HAZAN, E.; AND SINGER, Y., 2011. Adaptive subgradient methods for online learning and stochastic optimization. *JMLR*, (2011). (cited on pages 6 and 33)
- ELDREDGE, N., 2020. When do ϕ^2 and ϕ'^2 have the same expectation under a gaussian random variable? *MathOverflow*, (2020). <https://mathoverflow.net/q/351553>. (cited on page 24)
- FLEET, D. AND WEISS, Y., 2006. Optical flow estimation. In *Handbook of mathematical models in computer vision*. (cited on page 7)
- FORTUN, D.; BOUTHEMY, P.; AND KERVANN, C., 2015. Optical flow modeling and computation: a survey. *CVIU*, (2015). (cited on page 7)
- GEIGER, A.; LENZ, P.; AND URTASUN, R., 2012. Are we ready for autonomous driving? the kitti vision benchmark suite. *CVPR*, (2012). (cited on page 51)

-
- GLOROT, X. AND BENGIO, Y., 2010. Understanding the difficulty of training deep feedforward neural networks. *International Conference on Artificial Intelligence and Statistics*, (2010). (cited on page 2)
- GRAVES, A.; MOHAMED, A.-R.; AND HINTON, G., 2013. Speech recognition with deep recurrent neural networks. *IEEE International Conference on Acoustics, Speech and Signal Processing*, (2013). (cited on page 2)
- GUILLAUME, P.; HUARD, A.; AND LE CALVEZ, C., 2003. A block constant approximate inverse for preconditioning large linear systems. *SIAM Journal on Matrix Analysis and Applications*, (2003). (cited on page 38)
- HAZAN, E.; AGARWAL, A.; AND KALE, S., 2007. Logarithmic regret algorithms for online convex optimization. *Machine Learning*, (2007). (cited on page 6)
- HE, K.; ZHANG, X.; REN, S.; AND SUN, J., 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CVPR*, (2015). (cited on pages 48 and 52)
- HE, K.; ZHANG, X.; REN, S.; AND SUN, J., 2016. Deep residual learning for image recognition. *CVPR*, (2016). (cited on page 49)
- HENDRYCKS, D. AND GIMPEL, K., 2016. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, (2016). (cited on page 11)
- HOCHREITER, S., 1991. Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, (1991). (cited on page 2)
- HONKELA, A.; RAIKO, T.; KUUSELA, M.; TORNIO, M.; AND KARHUNEN, J., 2010. Approximate riemannian conjugate gradient learning for fixed-form variational bayes. *JMLR*, (2010). (cited on page 6)
- HORN, B. K. AND SCHUNCK, B. G., 1981. Determining optical flow. *Artificial Intelligence*, (1981). (cited on page 7)
- HORNIK, K.; STINCHCOMBE, M.; AND WHITE, H., 1989. Multilayer feedforward networks are universal approximators. *Neural Networks*, (1989). (cited on page 1)
- HUI, T.-W.; TANG, X.; AND LOY, C. C., 2018. Liteflownet: A lightweight convolutional neural network for optical flow estimation. *CVPR*, (2018). (cited on pages 7, 43, 46, 48, and 52)
- HUR, J.; LIM, H.; PARK, C.; AND CHUL AHN, S., 2015. Generalized deformable spatial pyramid: Geometry-preserving dense correspondence estimation. *CVPR*, (2015). (cited on page 55)
- ILG, E.; MAYER, N.; SAIKIA, T.; KEUPER, M.; DOSOVITSKIY, A.; AND BROX, T., 2017. Flownet 2.0: Evolution of optical flow estimation with deep networks. *CVPR*, (2017). (cited on pages 7, 43, 44, 46, 48, 51, and 52)

- IOFFE, S., 2017. Batch renormalization: Towards reducing minibatch dependence in batch-normalized models. *Advances in Neural Information Processing Systems*, (2017). (cited on page 4)
- IOFFE, S. AND SZEGEDY, C., 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *ICML*, (2015). (cited on page 4)
- JADERBERG, M.; SIMONYAN, K.; ZISSERMAN, A.; ET AL., 2015. Spatial transformer networks. *NIPS*, (2015). (cited on page 46)
- JANAI, J.; GÜNEY, F.; RANJAN, A.; BLACK, M.; AND GEIGER, A., 2018. Unsupervised learning of multi-frame optical flow with occlusions. *ECCV*, (2018). (cited on page 44)
- KIM, J.; LIU, C.; SHA, F.; AND GRAUMAN, K., 2013. Deformable spatial pyramid matching for fast dense correspondences. *CVPR*, (2013). (cited on page 55)
- KINGMA, D. AND BA, J., 2014. Adam: A method for stochastic optimization. *ICLR*, (2014). (cited on pages 6, 29, and 52)
- KLAMBAUER, G.; UNTERTHINER, T.; MAYR, A.; AND HOCHREITER, S., 2017. Self-normalizing neural networks. *Advances in Neural Information Processing Systems*, (2017). (cited on pages 11 and 15)
- KRIZHEVSKY, A.; SUTSKEVER, I.; AND HINTON, G. E., 2012. Imagenet classification with deep convolutional neural networks. *NIPS*, (2012). (cited on page 2)
- LECUN, Y. A.; BOTTOU, L.; ORR, G. B.; AND MÜLLER, K.-R., 2012. Efficient backprop. *Neural networks: Tricks of the trade*, (2012). (cited on page 2)
- LEVENBERG, K., 1944. A method for the solution of certain non-linear problems in least squares. *Quarterly of Applied Mathematics*, (1944). (cited on pages 6 and 29)
- LU, Y.; YANG, Z.; KANNALA, J.; AND KASKI, S., 2017. Learning image relations with contrast association networks. *arXiv*, (2017). (cited on page 56)
- LUCAS, B. D. AND KANADE, T., 1981. An iterative image registration technique with an application to stereo vision. *IJCAI*, (1981). (cited on page 43)
- MAAS, A. L.; HANNUN, A. Y.; AND NG, A. Y., 2013. Rectifier nonlinearities improve neural network acoustic models. *International Conference on Machine Learning*, (2013). (cited on page 11)
- MARQUARDT, D. W., 1963. An algorithm for least-squares estimation of nonlinear parameters. *SIAM Journal on Applied Mathematics*, (1963). (cited on pages 6 and 29)
- MARTENS, J., 2014. New insights and perspectives on the natural gradient method. *arXiv*, (2014). (cited on page 6)

-
- MARTENS, J., 2016. *Second-order optimization for neural networks*. Ph.D. thesis, University of Toronto. (cited on page 38)
- MAYER, N.; ILG, E.; FISCHER, P.; HAZIRBAS, C.; CREMERS, D.; DOSOVITSKIY, A.; AND BROX, T., 2018. What makes good synthetic training data for learning disparity and optical flow estimation? *IJCV*, (2018). (cited on page 7)
- MAYER, N.; ILG, E.; HAUSSE, P.; FISCHER, P.; CREMERS, D.; DOSOVITSKIY, A.; AND BROX, T., 2016. A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. *CVPR*, (2016). (cited on page 7)
- MECKES, E. S., 2019. *The Random Matrix Theory of the Classical Compact Groups*. Cambridge University Press. (cited on pages 13 and 27)
- MISHKIN, D. AND MATAS, J., 2016. All you need is a good init. *International Conference on Learning Representations*, (2016). (cited on page 10)
- NAIR, V. AND HINTON, G. E., 2010. Rectified linear units improve restricted boltzmann machines. *International Conference on Machine Learning*, (2010). (cited on page 11)
- NOCEDAL, J. AND WRIGHT, S. J., 2006. Numerical optimization 2nd. (cited on page 5)
- PASCANU, R. AND BENGIO, Y., 2014. Revisiting natural gradient for deep networks. *ICLR*, (2014). (cited on page 5)
- PASCANU, R.; MIKOLOV, T.; AND BENGIO, Y., 2013. On the difficulty of training recurrent neural networks. *International Conference on Machine Learning*, (2013). (cited on page 2)
- PENNINGTON, J.; SCHOENHOLZ, S.; AND GANGULI, S., 2017. Resurrecting the sigmoid in deep learning through dynamical isometry: theory and practice. *Advances in Neural Information Processing Systems*, (2017). (cited on pages 9, 10, 11, and 16)
- PETERS, J. AND SCHAAL, S., 2008. Natural actor-critic. *Neurocomputing*, (2008). (cited on page 6)
- PHILIPP, G.; SONG, D.; AND CARBONELL, J. G., 2018. The exploding gradient problem demystified-definition, prevalence, impact, origin, tradeoffs, and solutions. *arXiv preprint arXiv:1712.05577*, (2018). (cited on pages 2, 4, 15, and 16)
- POOLE, B.; LAHIRI, S.; RAGHU, M.; SOHL-DICKSTEIN, J.; AND GANGULI, S., 2016. Exponential expressivity in deep neural networks through transient chaos. *Advances in Neural Information Processing Systems*, (2016). (cited on page 16)
- RANJAN, A. AND BLACK, M. J., 2017. Optical flow estimation using a spatial pyramid network. *CVPR*, (2017). (cited on pages 7, 43, 48, 51, and 52)
- REVAUD, J.; WEINZAEPFEL, P.; HARCHAOU, Z.; AND SCHMID, C., 2015. EpicFlow: Edge-Preserving Interpolation of Correspondences for Optical Flow. *CVPR*, (2015). (cited on page 53)

-
- ROLNICK, D. AND TEGMARK, M., 2018. The power of deeper networks for expressing natural functions. *International Conference on Learning Representations*, (2018). (cited on page 2)
- RONNEBERGER, O.; FISCHER, P.; AND BROX, T., 2015. U-net: Convolutional networks for biomedical image segmentation. *MICCAI*, (2015). (cited on page 49)
- ROTH, S. AND BLACK, M. J., 2007. On the spatial statistics of optical flow. *IJCV*, (2007). (cited on page 49)
- SAXE, A. M.; MCCLELLAND, J. L.; AND GANGULI, S., 2014. Exact solutions to the non-linear dynamics of learning in deep linear neural networks. *International Conference on Learning Representations*, (2014). (cited on page 10)
- SCHARSTEIN, D. AND SZELISKI, R., 2002. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *IJCV*, (2002). (cited on page 46)
- SCHOENHOLZ, S. S.; GILMER, J.; GANGULI, S.; AND SOHL-DICKSTEIN, J., 2017. Deep information propagation. *International Conference on Learning Representations*, (2017). (cited on pages 11 and 16)
- SHEPHERD, A. J., 2012. *Second-order methods for neural networks: Fast and reliable training methods for multi-layer perceptrons*. (cited on page 38)
- SILVER, D.; SCHRITTWIESER, J.; SIMONYAN, K.; ANTONOGLOU, I.; HUANG, A.; GUEZ, A.; HUBERT, T.; BAKER, L.; LAI, M.; BOLTON, A.; ET AL., 2017. Mastering the game of go without human knowledge. *Nature*, (2017). (cited on page 2)
- SINGH, B.; DE, S.; ZHANG, Y.; GOLDSTEIN, T.; AND TAYLOR, G., 2015. Layer-specific adaptive learning rates for deep networks. *ICMLA*, (2015). (cited on page 32)
- SUN, D.; YANG, X.; LIU, M.-Y.; AND KAUTZ, J., 2018a. Models matter, so does training: An empirical study of cnns for optical flow estimation. *arXiv*, (2018). (cited on page 52)
- SUN, D.; YANG, X.; LIU, M.-Y.; AND KAUTZ, J., 2018b. Pwc-net: Cnns for optical flow using pyramid, warping, and cost volume. *CVPR*, (2018). (cited on pages 7, 43, 46, 48, 51, and 52)
- TENEY, D. AND HEBERT, M., 2016. Learning to extract motion from videos in convolutional neural networks. *arXiv*, (2016). (cited on page 7)
- THEWLIS, J.; ZHENG, S.; TORR, P. H.; AND VEDALDI, A., 2016. Fully-trainable deep matching. *BMVC*, (2016). (cited on page 7)
- TIELEMAN, T. AND HINTON, G., 2012. Rmsprop: Divide the gradient by a running average of its recent magnitude. *Neural networks for machine learning, Coursera*, (2012). (cited on pages 6 and 29)

-
- UMMENHOFER, B.; ZHOU, H.; UHRIG, J.; MAYER, N.; ILG, E.; DOSOVITSKIY, A.; AND BROX, T., 2017. Demon: Depth and motion network for learning monocular stereo. *CVPR*, (2017). (cited on page 48)
- VERSHYNIN, R., 2018. *High-Dimensional Probability: An Introduction with Applications in Data Science*. Cambridge University Press. (cited on pages 13 and 25)
- WANG, Y.; YANG, Y.; YANG, Z.; ZHAO, L.; AND XU, W., 2018. Occlusion aware unsupervised learning of optical flow. (2018). (cited on page 44)
- WULFE, J.; SEVILLA-LARA, L.; AND BLACK, M. J., 2017. Optical flow in mostly rigid scenes. *CVPR*, (2017). (cited on page 53)
- XU, J.; RANFTL, R.; AND KOLTUN, V., 2017. Accurate Optical Flow via Direct Cost Volume Processing. *CVPR*, (2017). (cited on pages 46 and 53)
- YANG, G.; PENNINGTON, J.; RAO, V.; SOHL-DICKSTEIN, J.; AND SCHOENHOLZ, S. S., 2019. A mean field theory of batch normalization. *International Conference on Learning Representations*, (2019). (cited on pages 4 and 15)
- YANG, Y. AND SOATTO, S., 2017. S2f: Slow-to-fast interpolator flow. *CVPR*, (2017). (cited on pages 43 and 48)
- YOU, Y.; GITMAN, I.; AND GINSBURG, B., 2017. Scaling sgd batch size to 32k for imagenet training. *arXiv*, (2017). (cited on page 32)
- YU, F. AND KOLTUN, V., 2015. Multi-scale context aggregation by dilated convolutions. *ICLR*, (2015). (cited on page 46)
- ZBONTAR, J. AND LECUN, Y., 2016. Stereo matching by training a convolutional neural network to compare image patches. *JMLR*, (2016). (cited on page 46)
- ZEILER, M. D., 2012. Adadelat: an adaptive learning rate method. *arXiv*, (2012). (cited on pages 6 and 29)