Deep Learning: Rethinking Common Practices

Elad Hoffer

Deep Learning: Rethinking Common Practices

Research Thesis

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy

Elad Hoffer

Submitted to the Senate
of the Technion — Israel Institute of Technology
Nisan 5779 Haifa April 2019

This research was carried out in the Faculty of Electrical Engineering under the supervision of Prof. Daniel Soudry, and Prof. Nir Ailon (Faculty of Computer Science)

Some results in this thesis have been published as articles by the author and research collaborators in conferences and journals during the course of the author's doctoral research period, the most up-to-date versions of which being:

Elad Hoffer and Nir Ailon. Deep metric learning using triplet network. In *International Workshop on Similarity-Based Pattern Recognition*, pages 84–92. Springer, 2015.

Elad Hoffer, Ron Banner, Itay Golan, and Daniel Soudry. Norm matters: efficient and accurate normalization schemes in deep networks. In *Advances in Neural Information Processing Systems*, 2018.

Elad Hoffer, Itay Hubara, and Daniel Soudry. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. In *Advances in Neural Information Processing Systems*, pages 1731–1741, 2017.

Elad Hoffer, Itay Hubara, and Daniel Soudry. Fix your classifier: the marginal value of training the last weight layer. *International Conference For Learning Representations*, 2018.

Acknowledgements

I would like to thank my advisors Prof. Daniel Soudry and Prof. Nir Ailon for their guidance and mentorship throughout my studies. To my coauthors: Itay Hubara, Ron Banner and Itay Golan, thank you for fruitful and pleasant collaboration.
I would also like to thank my dear family: my parents, Aviva and Uri, who bestowed me with the desire to learn and create and stood with me whenever I needed their support. You will always serve as my role models. To my brother Barak and sister Noa, who are always a source of pride and who I'm grateful to have in my life. To my grand-parents: Hanna and Harry, Mickey and Flavia, who all have a part in my achievements.
Last but not least, I would like to thank my loving wife Nirit, for her support and partnership, together with our two sons Carmel and Shaked, who joined our lives during the course of my studies.

The generous financial help of the Technion is gratefully acknowledged.

Contents

List of Figures

List of Tables

Abstract

Deep learning has revolutionized the way machine learning is used in various domains, improving previous approaches substantially in many cases such as computer vision, speech recognition, and control. Deep models aim at learning hierarchical representations of data, substituting engineered features by neural networks structures trained end-to-end.

We will examine and shed new light on several common practices and beliefs in Deep Learning:

- Learning explicit versus implicit features – we will suggest a network and objective function capable at learning features explicitly by metric embedding using deep networks. This model will also be shown useful at unsupervised and semi-supervised settings.
- The effect of batch size on generalization – showing that contrary to previous beliefs, the batch-size used for training deep models has no inherent negative impact on their generalization capabilities. This investigation will also provide us with insights on several key properties of these models and their training.
- The purpose of the classifier in the last layer – demonstrating that a fully-connected classification layer which is prevalent in deep models, has a marginal impact on its performance. Instead, using a fixed classifier can be used to improve performance and reduce parameter redundancy of the model.
- The role of batch-normalization – we will show that batch-normalization, a common building block of deep models, can be interpreted differently than originally thought, with implication to other regularization mechanisms. Our reinterpretation will also suggest new alternatives to batch-norm with several computational advantages.

Both theoretical and empirical arguments will be used to show that several commonly used practices are often misguided - and how they can be improved.

Chapter 1

Introduction

1.1 Background

Deep learning consists of multiple machine-learning algorithms that aim at learning hierarchical representations of data [LBH15]. These models are usually composed of multiple layers separated by non-linear functions and trained end-to-end on a specific task. At the roots of these models are "Artificial Neural Networks" (ANNs) – structures loosely inspired by biological neural networks, composed of layers of interconnected nodes with weighted edges between them. Neural networks layers with no direct connection to either input or output are known as "hidden-layers", where the term "Deep Learning" is usually used to describe a network with more than one hidden layer.

Neural networks were used as machine learning models as early as the 1940s [MP43], with practical use over the last three decades [LBD$^+$89], declining in usage at the end of the century in favor of competing models [CV95]. However, the recent rise of computational resources and abundance of data made these models appealing over the past few years, with many successful results on various tasks such as vision [KSH12a], speech-recognition [CJLV16], audio-generation [vdODZ$^+$, SPW$^+$18], NLP [VSP$^+$17, DCLT18] and control [MKS$^+$15, SSS$^+$17].
To fully understand the key properties leading to the success of deep models, three topics stand out:

- *Expressiveness* – their ability to learn arbitrary functions.
- *Optimization* – the ability to fit models according to an objective defined over the training data.
- *Generalization* – their performance on unseen examples.

We will use these three topics as guidelines for this introduction.

1.2 Deep neural architectures

One well-known aspect of neural networks is their expressive power, allowing them to be used as a universal function approximation tool. The representation power of neural networks is reflected in the universal approximation theorem [Hor91], which states that feed-forward network with a single hidden layer containing a finite number of neurons, can approximate continuous functions on compact subsets of $\mathbb{R}^n$, (under mild assumptions on the activation function). The expressiveness of neural network models has since been explored in various works [CSS16, RPK$^+$17]. Although highly expressive even in their most simple form, neural networks are known to benefit from domain-specific engineering of their underlying architecture vastly.

From the perspective of statistical learning theory, by specifying a neural network architecture, we obtain a hypothesis class, consisting of all prediction rules obtained by using the same network architecture while changing its weights. Learning the class involves finding a specific set of weights, based on training examples. We are additionally interested in generalizing, such that our predictor will yield good performance on unseen examples. Choosing a domain specific architecture for a network thus creates a strong inductive-bias, that may significantly impact both the ability to fit the training samples, as well as to generalize to those yet unseen.

1.2.1 Convolutional neural networks

Deep convolutional neural networks [LBD$^+$89] are the leading models of deep learning in computer vision tasks. Inspired by the structure in the vision circuits observed in biological systems [HW62], it is based on the notion of local connectivity over its spatial input, also known as the "receptive field" of the model.

The central premise is that stationary properties of images allow the use of a reduced set of parameters that needs to be learned. Convolutional layers consist of a set of trainable weight kernels, that produce their output value by cross-correlating with input data. This way, every spatial patch in the image is weighted using the same shared kernels. In each layer, multiple values are describing each spatial location. The different values for each position are known as the layer's *feature maps*.

To describe the convolution operation, we denote the learnable kernel weights as $w \in \mathcal{R}^{H_k \times W_k \times N \times M}$, input image $X \in \mathcal{R}^{H \times W \times N}$ with input pixels denoted as as $x_{i,j,k}$ and output pixels as $y_{i,j,k}$, where i, j is the spatial location and k is the corresponding feature map. Additionally, a bias term b_k is usually added to each output feature map k. The computed function for each feature map in a convolution layer ℓ (assuming padded boundaries) is:

$$y_{i,j,m} = \sum_{n=0}^{N-1} \sum_{k=0}^{H_k-1} \sum_{l=0}^{W_k-1} w_{k,l,n,m} \cdot x_{(i+k),(j+l),n} + b_m. \tag{1.1}$$

Since convolutional layers contain weights that are shared by multiple spatial regions, they naturally aggregate and average the gradients over large amounts of data. The same function can be applied as 1-dimensional convolution that is suited for temporal data such as audio and text.

Another main layer of ConvNets is the pooling layer. Pooling layers are used to reduce the dimensionality of the input while gaining scale and shift invariance to small amounts. This invariance is achieved by pooling (or down-sampling) spatial regions of the input, taking the average (Average-Pooling), max (Max-Pooling) or other variants. Most ConvNets are comprised of layers of Convolution & Pooling followed by fully-connected layers (dot products) and a classifier. Between layers with learned parameters, a non-linearity function is applied. Modern deep learning models usually employ $ReLU$ (rectified-linear-unit) of the form $f(x) = \max(0, x)$ or some variant of it [HZRS15b]. As input to a convolutional network is processed over local regions (kernel spatial sizes are usually on the scale of 3-11 pixels), global information is gathered by stacking multiple layers so that the final "spatial-region" of the model is at the scale of the whole image/sample.

Convolutional networks are used for different vision-related tasks such as classification [KSH12b], semantic segmentation [LSD15], detection [STE13] and control [MKS+15]. ConvNets were also shown to provide competitive results in language domain on tasks such as translation [GAG+17] and classification [ZZL15] by using convolutions on a character level. It is known that structure decisions and assumptions in the design of ConvNets are priors that form strong inductive bias concerning their input data [CS16, UVL18].

1.2.2 Recurrent neural networks

Another set of noteworthy models is that of recurrent neural networks (RNNs). RNNs are networks where connections between units form a directed cycle. For example, in its most simplest form (also known as an Elman Network [EBKS+96]):

$$h_t = \phi\left(W_i x_t + W_r h_{t-1} + b\right)$$

where $x_t \in \mathbb{R}^N$ and $h_t \in \mathbb{R}^M$ are the model's input and hidden-state at time t respectively and ϕ is the activation function (usually a bounded function such as tanh or sigmoid). RNNs are suited to work on time-series tasks, where their hidden states work as memory on previous inputs and states. RNNs are known to be able to address a wide range of time-dependencies and to generalize over sequences of varying length. RNNs were also proven to be Turing-complete [Sie95] in the sense that they can simulate arbitrary programs (when configured with the proper weights). Recurrent networks tend to suffer from the infamous problem of *vanishing or exploding gradients* [Hoc91], where a back-propagated error signals either shrink rapidly or grow out of bounds. This issue

brought forth a variety of models designed to circumvent it, where the most successful of them is known as the "Long-short-term-memory" network (LSTM) [HS97].

1.2.3 Other common patterns in deep modeling

Several additional patterns in model architectures appeared lately, demonstrating the effectiveness of structure in the design of deep networks. Residual connections, first introduced with the ResNet model by [HZRS16] have been widely adopted across different deep architectures [WSC+16, XGD+17]. Residual connections add activations across layers (e.g $x^{\ell+1} = F(x^{\ell}) + x^{\ell}$ for a given layer ℓ) and noted to improve convergence and final accuracy, as well as to enable the training of very deep networks. The merits of residual connections can be explained by their ability to allow gradients to "flow" throughout the network unhindered. Another similar improvement is that of "Attention" modules [BCB14, LPM15], allowing to compute a weighted average of a sequence representation dependent on a query activations. This proved highly useful for language-related tasks, with state-of-the-art results in machine translation [VSP+17] and language understanding [DCLT18].

1.3 Optimization in deep learning

The most common usage of deep networks is in a supervised learning setting, where the model is optimized for empirical-risk-minimization [Vap92] with regard to a set of ground-truth labels. Given training examples $\left\{(x^{(1)}, t^{(1)}), ..., (x^{(N)}, t^{(N)})\right\}$ such that $x^{(i)}$ are the inputs and $t^{(i)}$ are the targets, the model is optimized over an objective function between the targets and the network output $y = F(x)$. For example, we can perform regression using a network with a *Mean-square-error (MSE)* — computing for the network outputs $y^{(i)}$ vs. targets $t^{(i)}$

$$E_{MSE}(y, t) = \frac{1}{N}\sum_{i=1}^{N} \|y^{(i)} - t^{(i)}\|^2$$

1.3.1 Gradient based methods

As modern neural networks are composed from a vast number of underlying parameters, it is therefore imperative to optimize these models with methods that scale linearly with the dimensionality of the error surface. For this reason, deep models are commonly trained using techniques of first-order optimization, requiring only a gradient of each parameter with respect to the objective function. Additionally, the gradients necessary for each optimization step are usually computed only on a fraction of the training data at hand, leading to a noisy estimation of the true gradient. This is known as *Stochastic gradient descent (SGD)* [RM51]. The SGD update-rule at its simplest form

(vanilla-SGD) can be described as:

$$w_{t+1} = w_t - \varepsilon \cdot \frac{\partial E}{\partial w_t}$$

where w is the optimized parameter (weight), E is the error (loss function estimation) and ε is the *Learning-Rate*. This update rule requires $O(n)$ number of computations and memory use, where n is the number of parameters. Optimizing the objective over a batch of examples at each step also allow to parallelize the computations needed thus enabling the growing scale of models and training data.

Other variants of gradient-based optimizations were suggested to improve convergence time and quality, by estimating higher-order moments or preconditioning the update-rule to better suit the geometry of the underlying loss surface [KB14, MG15].

Backpropagation

In order to apply the SGD update rule, one must acquire the gradients of all parameters with respect to the error function $\frac{\partial E}{\partial w}$. This is achieved for any layer $f(x; w)$ with input x and parameters w by applying the chain-rule: Given an error gradient with regard to the output of the function $\frac{\partial E}{\partial f}$, we can compute the gradient with regard to input

$$\frac{\partial E}{\partial x} = \frac{\partial f}{\partial x} \cdot \frac{\partial E}{\partial f}$$

and similar treatment is applied to yield the gradient of the parameters $\frac{\partial E}{\partial w}$. Applying multiple invocations of the chain rule over a differentiable graph is also known as the "Back-propagation" procedure [RHW86] or reverse-mode auto differentiation. Starting at the output and the gradient of our error function, we can back-propagate our gradients towards previous layers, where each step is local – requiring the (saved) output of the previous layer and the gradient with-respect-to to the next layer.

1.4 Generalization in Deep Learning

Neural networks are notoriously known for being "under-specified" or "over-parameterized", marking the fact that they usually require much more parameters than the number of samples used to train them. For example, the popular Imagenet dataset [RDS+15] includes roughly 1.3 million images, while convolutional networks trained using it are usually composed of 10s to 100s million of parameters.

Over-parameterization, from a perspective of classical learning theory, would imply poor performance as these models can easily memorize the training set, without generalizing to other unseen samples. However, it was found that models with a larger number of weights are easier to train and reach a better generalization performance. In some respects, it can be viewed as "a blessing of dimensionality", where a high

dimensional weight space combats the input space's "curse of dimensionality".

As restricting the number parameters is not a successful strategy when considering deep architectures, the common approach is, instead

- Use a large model (as computational and time requirements allow).
- Control its capacity through regularization – thus reducing it's ability to overfit over training data.

1.4.1 Regularization of deep networks

Data augmentation

A common practice in training modern neural networks is to use data augmentation — applying different transformations to each input sample. For example, in image classification tasks, for any input image, a random crop of varying size and scale is applied to it, potentially together with rotation, mirroring and even color jittering [KSH12b]. Data augmentations were repeatedly found to provide efficient and useful regularization, often accounting for a significant portion of the final generalization performance [Zag16, DT17]. Several works even attempt to learn how to generate good data augmentations. For example, Bayesian approaches based on the training set distribution [TPC+17], generative approaches based on generative adversarial networks [ASE17, SWL18] and search methods aimed to find the best data augmentation policy [CZM+18].

Noise injection

Another set of common regularizers can be seen as "noise-injection" methods, in which model capacity is controlled by inserting noise to either activations or parameters of the network. The prime representative of these methods is "Dropout" [SHK+14], used in many deep learning models. Dropout consists of a noise-signal (usually a Bernoulli noise, but Gaussian noise was shown to work as well), that is injected as multiplication on intermediate hidden layers and removed at inference time.

For example, a Bernoulli Dropout with $p = 0.5$ will zero half of the activations of the corresponding layer at each iteration. This noise will be removed when evaluating with the model and hidden activation will be scaled by $1 - p$, so that its values will be preserved in expectation.
Several explanations were suggested to account for the success of Dropout:

- It encourages redundancy within a network – there cannot be a high-dependency on a specific unit or set of units.
- It can be seen as a model averaging technique - as 2^N different models are used under expectation when evaluating.

- Multiplicative noise (additive on gradients) - can allow better generalization, similar to benefits by data augmentations

Other similar regularization methods are used such as "Dropconnect" [WZZ+13] – inserting noise onto weights instead of activations, "DropBlock" [GLL18] – using a noise pattern of spatial blocks in convolutional networks and ZoneOut [KMK+16] – creating noise at state-transfer function of recurrent networks by keeping intermediate states unchanged at random. Noise injection was also found useful when applied to intermediate gradient calculations [NVL+15]. Another related regularization technique called "Mixup" was introduced by [ZCDLP17]. Mixup uses a mixed input from two separate samples with different classes and uses as target their labels mixed by the same amount.

Explicit regularization

Another form of regularization in deep networks is performed by explicitly adding a regularization term into the underlying optimized objective function. For example, in the well known L^2 regularization (also known as "ridge-regression" or "weight-decay") method, the loss function is added with a term that penalizes the L^2 norm of the weights:

$$L(y,t;w) = E(y,t) + \eta\frac{1}{2}\|w\|_2^2 \tag{1.2}$$

This is equivalent to decaying the weights in each gradient descent step so that the update learning rule $\Delta\hat{w}_t$ is:

$$\Delta\hat{w}_t = \Delta w_t - \eta w_t$$

reducing the weights' norm, thus smoothing the decision boundary. Other forms of this kind of regularization are L^1 and L^0 [LWK17], inducing sparsity of activations and/or of weights.

Neural-networks are also known to exhibit calibration issues [GPSW17, OAGR18], where their outputs are "over-confident", failing to express the true posterior and often missing the inherent underlying ambiguity of the task at hand. To address this issues, a label-smoothing criterion was suggested by [SVI+16] where our classification is now

$$t = \begin{cases} 1-\epsilon, & \text{if} \quad j = c_t \\ \frac{\epsilon}{C-1}, & \text{otherwise} \end{cases}$$

A similar approach was introduced by [PTC+17], where the entropy of the model's prediction is explicitly penalized to avoid over-confident outputs.

Normalization methods

Another method for model regularization is by normalization of either activation or weights. A very popular technique in modern NNs is *Batch Normalization* [IS15].

Batch normalization (batch-norm) transforms the input so that the sum of deviations from the mean would be standardized.

For a layer with $d-$dimensional input $x = (x^{(1)}, x^{(2)}, ..., x^{(d)})$, batch-norm normalizes each dimension such that

$$\hat{x}^{(k)} = \frac{x^{(k)} - \mathbb{E}x^{(k)}}{\sqrt{\text{Var}[x^{(k)}] + \epsilon}} \cdot \gamma + \beta$$

where γ and β are additional learned parameters and ϵ ensures numerical stability.

The use of batch-norm was found empirically to allow higher learning rates and to converge faster. It also usually results with better generalization and alleviates a lot of issues with properly initializing the weights of neural networks. Other normalization methods such as layer-norm [BKH16] and group-norm [WH18] suggest normalization over a different dimension of the activations, depending on the learned task. Additional techniques such as weight-norm [SK16] suggest a normalization procedure on weights rather than activations. All of these normalization schemes can be viewed as regularization, as they limit the expressiveness of the model in some sense (and hence its ability to fit the training data).

1.5 Layout of this work

In this work, we will explore several of the underlying properties of Deep models as described above. In chapter 3, we will suggest a new approach to learn explicit representations of data using deep networks by metric embedding. Instead of explicitly training to classify, we will show how a novel loss function based on the embedded distance between samples can be used to create useful representations. This notion of learning by distance embedding will lay the foundations for the work presented in chapters 7, 8, describing novel methods for unsupervised and semi-supervised learning using deep networks.

In chapter 4, we will examine the wide-held belief that the batch size used in training of deep models can have an adverse effect on the generalization capabilities of the final model. Through a careful investigation in the dynamics of the training process, we will show that the so-called "generalization gap" in large batch training is not an inherent problem, but rather a symptom reflected by the modification of the optimization regime.

In chapter 5 we examine the role of the final classifier that exists in every deep network used for classification. We will show that despite the vast number of parameters and computational resources that are required by these layers, they contribute only marginally to the performance of the model.

In chapter 6 we investigate the "batch-normalization" technique, that is used both to regularize as well as to accelerate the training process of deep networks. We will show that batch-norm can be interpreted as an explicit regularization, inserting a hard-constraint on the model's parameters while also affecting other regularization techniques. These insights also allow better-suited alternatives that will be evaluated.

Chapter 2

Research Methods

Research presented in this work was performed using a setup created for experimentation, that allowed training and evaluation of deep neural network models. This setup consisted, for the most part, of implementations done on contemporary GPU hardware using deep learning frameworks such as Torch, PyTorch and TensorFlow. Accompanying code developed and used in the following works is available as open-source in the following URLs:

- Vision based deep models - `https://github.com/eladhoffer/convNet.pytorch`
- Sequence-to-sequence models - `https://github.com/eladhoffer/seq2seq.pytorch`

Chapter 3

Deep metric learning using Triplet Network

Deep metric learning using Triplet network

Elad Hoffer, Nir Ailon
Technion - Israel Institute of Technology, Haifa, Israel
elad.hoffer@gmail.com

Abstract

Deep learning has proven itself as a successful set of models for learning useful semantic representations of data. These, however, are mostly implicitly learned as part of a classification task. In this paper we propose the *triplet network* model, which aims to learn useful representations by distance comparisons. A similar model was defined by Wang et al. (2014), tailor made for learning a ranking for image information retrieval. Here we demonstrate using various datasets that our model learns a better representation than that of its immediate competitor, the Siamese network. We also discuss future possible usage as a framework for unsupervised learning.

1 Introduction

For the past few years, deep learning models have been used extensively to solve various machine learning tasks. One of the underlying assumptions is that deep, hierarchical models such as convolutional networks create useful representation of data (1; 10), which can then be used to distinguish between available classes. This quality is in contrast with traditional approaches requiring engineered features extracted from data and then used in separate learning schemes. Features extracted by deep networks were also shown to provide useful representation (24; 20) which can be, in turn, successfully used for other tasks (19).

Despite their importance, these representations and their corresponding induced metrics are often treated as side effects of the classification task, rather than being explicitly sought. There are also many interesting open questions regarding the intermediate representations and their role in disentangling and explaining the data (2). Notable exceptions where explicit metric learning is preformed are the *Siamese Network* variants (3; 5; 9), in which a contrastive loss over the metric induced by the representation is used to train the network to distinguish between similar and dissimilar *pairs* of examples. A contrastive loss favours a small distance between pairs of examples labeled as similar, and large distances for pairs labeled dissimilar. However, the representations learned by these models provide sub-par results when used as features for classification, compared with other deep learning models including ours. Siamese networks are also sensitive to calibration in the sense that the notion of similarity vs dissimilarity requires context. For example, a person might be deemed similar to another person when a dataset of random objects is provided, but might be deemed dissimilar with respect to the same other person when we wish to distinguish between two individuals in a set of individuals only. In our model, such a calibration is not required. In fact, in our experiments here, we have experienced hands on the difficulty in using Siamese networks.

We follow a similar task to that of (4). For a set of samples $\mathbb{P}$ and a chosen rough similarity measure $r(x, x')$ given through a training oracle (e.g. how close are two images of objects semantically) we wish to learn a similarity function $S(x, x')$ induced by a normed metric. Unlike (4)'s work, our labels are of the form $r(x, x_1) > r(x, x_2)$ for triplets x, x_1, x_2 of objects. Accordingly, we try to fit a metric embedding and a corresponding similarity function satisfying:

$$S(x, x_1) > S(x, x_2), \ \ \forall x, x_1, x_2 \in \mathbb{P} \ \ \text{for which } r(x, x_1) > r(x, x_2).$$

In our experiment, we try to find a metric embedding of a multi-class labeled dataset - meaning that our similarity function is the same-class indicator. We will always take x_1 to be of the same

class as x and x_2 of a different class, although in general more complicated choices could be made. Accordingly, we will use the notation x^+ and x^- instead of x_1, x_2. We focus on finding an L_2 embedding, by learning a function $F(x)$ for which $S(x, x') = \|F(x) - F(x')\|_2$. Inspired from the recent success of deep learning, we will use a deep network as our embedding function $F(x)$.

We call our approach a *triplet network*. A similar approach was proposed in (23) for the purpose of learning a ranking function for image retrieval. Compared with the single application proposed in (23), we make a comprehensive study of the triplet architecture which is, as we shall argue below, interesting in and of itself. In fact, we shall demonstrate below that the triplet approach is a strong competitor to the Siamese approach, its most obvious competitor.

2 The Triplet network

A *Triplet network* (inspired by "Siamese network") is comprised of 3 instances of the same feed-forward network (with shared parameters). When fed with 3 samples, the network outputs 2 intermediate values - the L_2 distances between the embedded representation of two of its inputs from the representation of the third. If we will denote the 3 inputs as x, x^+ and x^-, and the embedded representation of the network as $Net(x)$, the one before last layer will be the vector:

$$TripletNet(x, x^-, x^+) = \begin{bmatrix} \|Net(x) - Net(x^-)\|_2 \\ \|Net(x) - Net(x^+)\|_2 \end{bmatrix} \in \mathbb{R}^2_+ \ .$$

In words, this encodes the pair of distances between each of x^+ and x^- against the *reference* x.

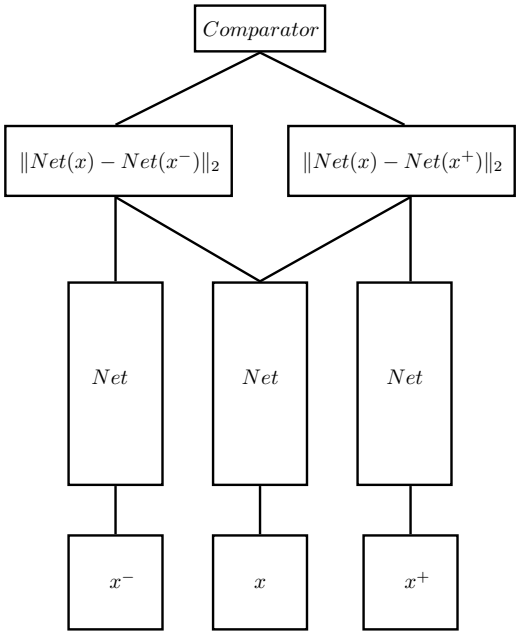


Figure 1: Triplet network structure

2.1 Training

Training is preformed by feeding the network with samples where, as explained above, x and x^+ are of the same class, and x^- is of different class. The network architecture allows the task to be expressed as a 2-class classification problem, where the objective is to correctly classify which of x^+ and x^- is of the same class as x. We stress that in a more general setting, where the objective might be to learn a metric embedding, the label determines which example is *closer* to x. Here we simply interpret "closeness" as "sharing the same label". In order to output a comparison operator from the

model, a SoftMax function is applied on both outputs - effectively creating a ratio measure. Similarly to traditional convolutional-networks, training is done by simple Stochastic Gradient Descent on a negative-log-likelihood loss with regard to the 2-class problem. We later examined that better results are achieved when the loss function is replaced by a simple Mean Squared Error on the soft-max result, compared to the $(0, 1)$ vector, so that the loss is

$$Loss(d_+, d_-) = \|(d_+, d_- - 1)\|_2^2 = const \cdot d_+^2$$

where

$$d_+ = \frac{e^{\|Net(x)-Net(x^+)\|_2}}{e^{\|Net(x)-Net(x^+)\|_2} + e^{\|Net(x)-Net(x^-)\|_2}}$$

and

$$d_- = \frac{e^{\|Net(x)-Net(x^-)\|_2}}{e^{\|Net(x)-Net(x^+)\|_2} + e^{\|Net(x)-Net(x^-)\|_2}} \ .$$

We note that $Loss(d_+, d_-) \rightarrow 0$ iff $\frac{\|Net(x)-Net(x^+)\|}{\|Net(x)-Net(x^-)\|} \rightarrow 0$, which is the required objective. By using the same shared parameters network, we allow the back-propagation algorithm to update the model with regard to all three samples simultaneously.

3 Tests and results

The Triplet network was implemented and trained using the Torch7 environment ((7)).

3.1 Datasets

We experimented with 4 datasets. The first is *Cifar10* ((11)), consisting of 60000 32x32 color images of 10 classes (of which 50000 are used for training only, and 10000 for test only). The second dataset is the original *MNIST* ((12)) consisting of 60000 28x28 gray-scale images of handwritten digits 0-9, and a corresponding set of 10000 test images. The third is the *Street-View-House-Numbers (SVHN)* of (18) consisting of 600000 32x32 color images of house-number digits 0-9. The fourth dataset is *STL10* of (6), similar to Cifar10 and consisting of 10 object classes, only with 5000 training images (instead of 50000 in Cifar) and a bigger 96x96 image size.

It is important to note that no data augmentation or whitening was applied, and the only preprocessing was a global normalization to zero mean and unit variance. Each training instance (for all four datasets) was a uniformly sampled set of 3 images, 2 of which are of the same class (x and x^+), and the third (x^-) of a different class. Each training epoch consisted of 640000 such instances (randomly chosen each epoch), and a fixed set of 64000 instances used for test. We emphasize that each test instance involves 3 images from the set of test images which was excluded from training.

3.2 The Embedding Net

For Cifar10 and SVHN we used a convolutional network, consisting of 3 convolutional and 2x2 max-pooling layers, followed by a fourth convolutional layer. A *ReLU* non-linearity is applied between two consecutive layers. Network configuration (ordered from input to output) consists of filter sizes {5,3,3,2}, and feature map dimensions {3,64,128,256,128} where a 128 vector is the final embedded representation of the network. Usually in convolutional networks, a subsequent fully-connected layer is used for classification. In our net this layer is removed, as we are interested in a feature embedding only.

The network for STL10 is identical, only with stride=3 for the first layer, to allow the bigger input size. The network used for MNIST was a smaller version consisting of smaller feature map sizes {1,32,64,128}.

3.3 Results

Training on all datasets was done by SGD, with initial learning-rate of 0.5 and a learning rate decay regime. We used a momentum value of 0.9. We also used the dropout regularization technique with $p = 0.5$ to avoid over-fitting. After training on each dataset for 10-30 epochs, the network reached a fixed error over the triplet comparisons. We then used the embedding network to extract features

Dataset	TripletNet	SiameseNet	Best known result (no data augmentation)
Mnist	99.54±0.08%	97.9±0.1%	99.61% (16),(13)
Cifar10	87.1±0.07%	-	90.22% (13)
SVHN	95.37±0.08%	-	98.18% (13)
STL10	70.67±0.1%	-	67.9% (15)

Table 1: Classification accuracy (no data augmentation)

from the full dataset, and trained a simple 1-layer network model on the full 10-class classification task (using only training set representations). The test set was then measured for accuracy. These results (Table 1) are comparable to state-of-the-art results with deep learning models, without using any artificial data augmentation ((25; 8; 14)). Noteworthy is the STL10 dataset, in which the TripletNet achieved the best known result for non-augmented data. We conjecture that data augmentation techniques (such as translations, mirroring and noising) may provide similar benefits to those described in previous works.

We also note that similar results are achieved when the embedded representations are classified using a linear SVM model or KNN classification with up to 0.5% deviance from the results in Table 1. Another side-affect noticed, is that the representation seems to be sparse - about 25% non-zero values. This is very helpful when used later as features for classification both computationally and with respect to accuracy, as each class is characterised by only a few non zero elements.

3.4 2d visualization of features

In order to examine our main premise, which is that the network embeds the images into a representation with meaningful properties, we use PCA to project the embedding into 2d euclidean space which can be easily visualized (figures 4 3 4). We can see a significant clustering by semantic meaning, confirming that the network is useful in embedding images into the euclidean space according to their content. Similarity between objects can be easily found by measuring the distance between their embedding and, as shown in the results, can reach high classification accuracy using a simple subsequent linear classifier.

3.5 Comparison with performance of the Siamese network

The Siamese network is the most obvious competitor for our approach. Our implementation of the Siamese network consisted of the same embedding network, but with the use of a contrastive loss between a pair of samples, instead of three (as explained in (5)). The generated features were then used for classification using a similar linear model as was used for the TripletNet method. We measured lower accuracy on the MNIST dataset compared to results gained using the TripletNet representations 1.

We have tried a similar comparison for the other three datasets, but unfortunately could not obtain any meaningful result using a Siamese network. We conjecture that this might be related to the problem of context described above, and leave the resolution of this conjecture to future work.

4 Future work

As the Triplet net model allows learning by comparisons of samples instead of direct data labels, usage as an unsupervised learning model is possible. Future investigations can be performed in several scenarios:

- **Using spatial information.** Objects and image patches that are spatially near are also expected to be similar from a semantic perspective. Therefore, we could use geometric distance between patches of the same image as a rough similarity oracle $r(x, x')$, in an unsupervised setting.
- **Using temporal information.** The same is applicable to time domain, where two consecutive video frames are expected to describe the same object, while a frame taken 10 minutes later is less likely to do so. Our Triplet net may provide a better embedding and improve on

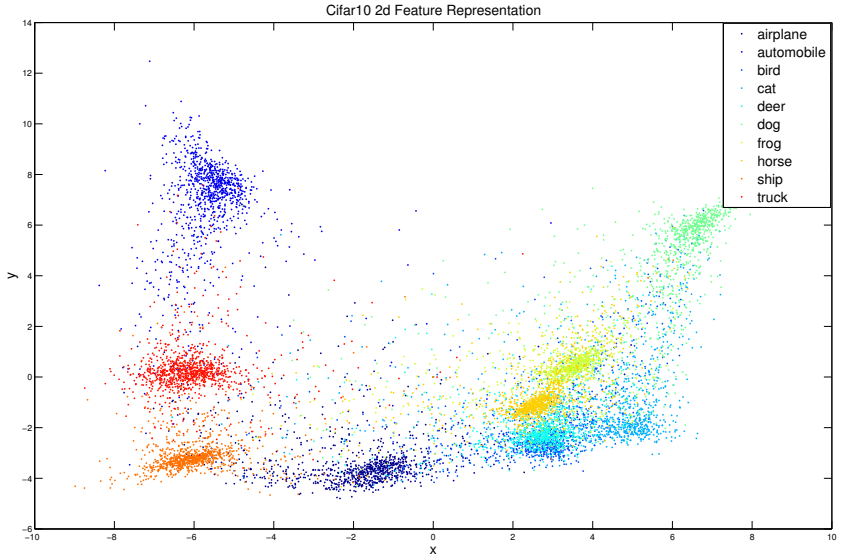


Figure 2: CIFAR10 - Euclidean representation of embedded test data, projected onto top two singular vectors

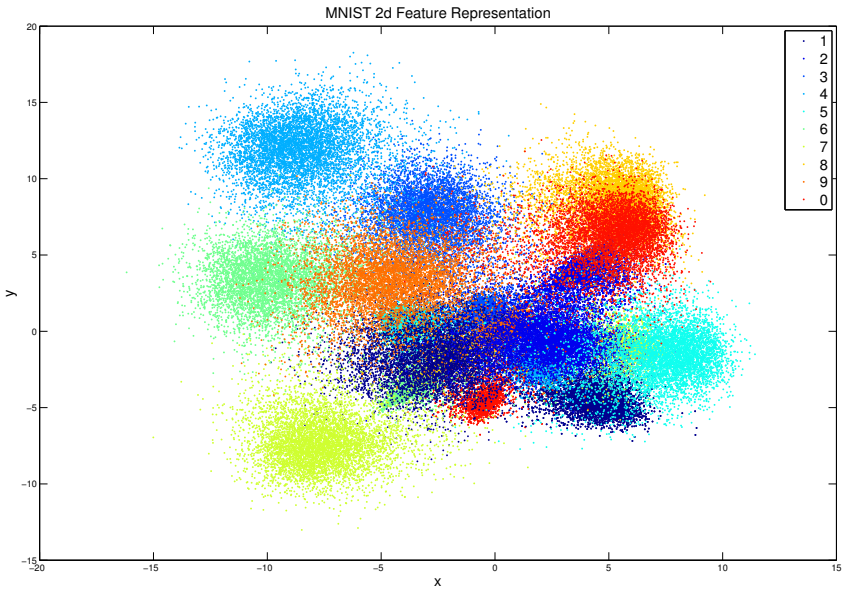


Figure 3: MNIST - Euclidean representation of embedded test data, projected onto top two singular vectors

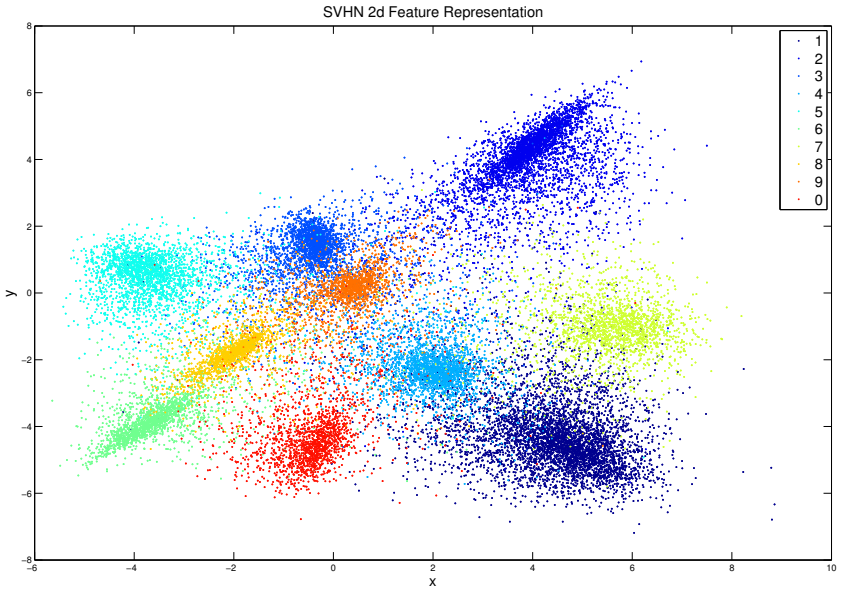


Figure 4: SVHN - Euclidean representation of embedded test data, projected onto top two singular vectors

past attempts in solving classification tasks in an unsupervised environment, such as that of ((17)).

It is also well known that humans tend to be better at accurately providing comparative labels. Our framework can be used in a crowd sourcing learning environment. This can be compared with (22), who used a different approach. Furthermore, it may be easier to collect data trainable on a Triplet network, as comparisons over similarity measures are much easier to attain (pictures taken at the same location, shared annotations, etc).

5 Conclusions

In this work we introduced the *Triplet network* model, a tool that uses a deep network to learn useful representation explicitly. The results shown on various datasets provide evidence that the representations that were learned are useful to classification in a way that is comparable with a network that was trained explicitly to classify samples. We believe that enhancement to the embedding network such as Network-in-Network model ((14)), Inception models ((21)) and others can benefit the Triplet net similarly to the way they benefited other classification tasks. Considering the fact that this method requires to know only that two out of three images are sampled from the same class, rather than knowing what that class is, we think this should be inquired further, and may provide us insights to the way deep networks learn in general. We have also shown how this model learns using only comparative measures instead of labels, which we can use in the future to leverage new data sources for which clear out labels are not known or do not make sense (e.g hierarchical labels).

Acknowledgements

We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan-Z GPU used for this research. This resesarch was additionally supported by the Israel Science Foundation (ISF) grant No. 1271/13, and by the ISF-UGC India-Israel joint research program grant No. 1932/14.

References

[1] Bengio, Yoshua. Learning Deep Architectures for AI, 2009. ISSN 1935-8237.

[2] Bengio, Yoshua. Deep learning of representations: Looking forward. In *LNCS (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 7978 LNAI, pp. 1–37, 2013. ISBN 9783642395925.

[3] Bromley, Jane, Bentz, James W, Bottou, Léon, Guyon, Isabelle, LeCun, Yann, Moore, Cliff, Säckinger, Eduard, and Shah, Roopak. Signature verification using a siamese time delay neural network. *International Journal of Pattern Recognition and Artificial Intelligence*, 7(04):669–688, 1993.

[4] Chechik, Gal, Sharma, Varun, Shalit, Uri, and Bengio, Samy. Large scale online learning of image similarity through ranking. *The Journal of Machine Learning Research*, 11:1109–1135, 2010.

[5] Chopra, Sumit, Hadsell, Raia, and LeCun, Yann. Learning a similarity metric discriminatively, with application to face verification. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, pp. 539–546, 2005. ISBN 0769523722.

[6] Coates, Adam, Ng, Andrew Y, and Lee, Honglak. An analysis of single-layer networks in unsupervised feature learning. In *International Conference on Artificial Intelligence and Statistics*, pp. 215–223, 2011.

[7] Collobert, Ronan, Kavukcuoglu, Koray, and Farabet, Clément. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, number EPFL-CONF-192376, 2011.

[8] Goodfellow, Ian J, Warde-Farley, David, Mirza, Mehdi, Courville, Aaron, and Bengio, Yoshua. Maxout networks. *arXiv preprint arXiv:1302.4389*, 2013.

[9] Hadsell, Raia, Chopra, Sumit, and LeCun, Yann. Dimensionality reduction by learning an invariant mapping. In *Computer vision and pattern recognition, 2006 IEEE computer society conference on*, volume 2, pp. 1735–1742. IEEE, 2006.

[10] Hinton, Geoffrey E. Learning multiple layers of representation, 2007. ISSN 13646613.

[11] Krizhevsky, Alex and Hinton, Geoffrey. Learning multiple layers of features from tiny images. *Computer Science Department, University of Toronto, Tech. Rep*, 2009.

[12] LeCun, Yann, Bottou, Léon, Bengio, Yoshua, and Haffner, Patrick. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[13] Lee, Chen-Yu, Xie, Saining, Gallagher, Patrick, Zhang, Zhengyou, and Tu, Zhuowen. Deeply-supervised nets. *arXiv preprint arXiv:1409.5185*, 2014.

[14] Lin, Min, Chen, Qiang, and Yan, Shuicheng. Network in network. *CoRR*, abs/1312.4400, 2013. URL `http://arxiv.org/abs/1312.4400`.

[15] Lin, Tsung-Han and Kung, HT. Stable and efficient representation learning with nonnegativity constraints. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pp. 1323–1331, 2014.

[16] Mairal, Julien, Koniusz, Piotr, Harchaoui, Zaid, and Schmid, Cordelia. Convolutional kernel networks. In *Advances in Neural Information Processing Systems*, pp. 2627–2635, 2014.

[17] Mobahi, Hossein, Collobert, Ronan, and Weston, Jason. Deep learning from temporal coherence in video. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pp. 737–744. ACM, 2009.

[18] Netzer, Yuval, Wang, Tao, Coates, Adam, Bissacco, Alessandro, Wu, Bo, and Ng, Andrew Y. Reading digits in natural images with unsupervised feature learning.

[19] Razavian, Ali Sharif, Azizpour, Hossein, Sullivan, Josephine, and Carlsson, Stefan. CNN Features off-the-shelf: an Astounding Baseline for Recognition. *Arxiv*, 2014. URL http://arxiv.org/abs/1403.6382.

[20] Sermanet, Pierre, Eigen, David, Zhang, Xiang, Mathieu, Michael, Fergus, Rob, and LeCun, Yann. OverFeat : Integrated Recognition , Localization and Detection using Convolutional Networks. *arXiv preprint arXiv:1312.6229*, pp. 1–15, 2013. URL http://arxiv.org/abs/1312.6229.

[21] Szegedy, Christian, Liu, Wei, Jia, Yangqing, Sermanet, Pierre, Reed, Scott, Anguelov, Dragomir, Erhan, Dumitru, Vanhoucke, Vincent, and Rabinovich, Andrew. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014. URL http://arxiv.org/abs/1409.4842.

[22] Tamuz, Omer, Liu, Ce, Belongie, Serge, Shamir, Ohad, and Kalai, Adam. Adaptively learning the crowd kernel. In Getoor, Lise and Scheffer, Tobias (eds.), *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, ICML '11, pp. 673–680, New York, NY, USA, June 2011. ACM. ISBN 978-1-4503-0619-5.

[23] Wang, Jiang, Song, Yang, Leung, Thomas, Rosenberg, Chuck, Wang, Jingbin, Philbin, James, Chen, Bo, and Wu, Ying. Learning fine-grained image similarity with deep ranking. In *CVPR*, 2014.

[24] Zeiler, Matthew D and Fergus, Rob. Visualizing and Understanding Convolutional Networks. *arXiv preprint arXiv:1311.2901*, 2013. URL http://arxiv.org/abs/1311.2901.

[25] Zeiler, Matthew D and Fergus, Rob. Stochastic pooling for regularization of deep convolutional neural networks. *arXiv preprint arXiv:1301.3557*, 2013.

Chapter 4

Train longer, generalize better: closing the generalization gap in large batch training of neural networks

Train longer, generalize better: closing the generalization gap in large batch training of neural networks

Elad Hoffer[*] **Itay Hubara**[*] **Daniel Soudry**
Technion - Israel Institute of Technology, Haifa, Israel
{elad.hoffer, itayhubara, daniel.soudry}@gmail.com

Abstract

Background: Deep learning models are typically trained using stochastic gradient descent or one of its variants. These methods update the weights using their gradient, estimated from a small fraction of the training data. It has been observed that when using large batch sizes there is a persistent degradation in generalization performance - known as the "generalization gap" phenomenon. Identifying the origin of this gap and closing it had remained an open problem.

Contributions: We examine the initial high learning rate training phase. We find that the weight distance from its initialization grows logarithmically with the number of weight updates. We therefore propose a "random walk on a random landscape" statistical model which is known to exhibit similar "ultra-slow" diffusion behavior. Following this hypothesis we conducted experiments to show empirically that the "generalization gap" stems from the relatively small number of updates rather than the batch size, and can be completely eliminated by adapting the training regime used. We further investigate different techniques to train models in the large-batch regime and present a novel algorithm named "Ghost Batch Normalization" which enables significant decrease in the generalization gap without increasing the number of updates. To validate our findings we conduct several additional experiments on MNIST, CIFAR-10, CIFAR-100 and ImageNet. Finally, we reassess common practices and beliefs concerning training of deep models and suggest they may not be optimal to achieve good generalization.

1 Introduction

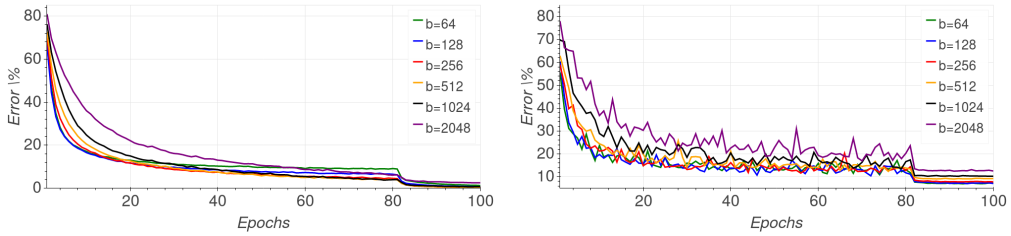
For quite a few years, deep neural networks (DNNs) have persistently enabled significant improvements in many application domains, such as object recognition from images (He et al., 2016); speech recognition (Amodei et al., 2015); natural language processing (Luong et al., 2015) and computer games control using reinforcement learning (Silver et al., 2016; Mnih et al., 2015).

The optimization method of choice for training highly complex and non-convex DNNs, is typically stochastic gradient decent (SGD) or some variant of it. Since SGD, at best, finds a local minimum of the non-convex objective function, substantial research efforts are invested to explain DNNs ground breaking results. It has been argued that saddle-points can be avoided (Ge et al., 2015) and that "bad" local minima in the training error vanish exponentially (Dauphin et al., 2014; Choromanska et al., 2015; Soudry & Hoffer, 2017). However, it is still unclear why these complex models tend to generalize well to unseen data despite being heavily over-parameterized (Zhang et al., 2017).

A specific aspect of generalization has recently attracted much interest. Keskar et al. (2017) focused on a long observed phenomenon (LeCun et al., 1998a) – that when a large batch size is used while

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

[*]Equal contribution



(a) Training error

(b) Validation error

Figure 1: Impact of batch size on classification error

training DNNs, the trained models appear to generalize less well. This remained true even when the models were trained "without any budget or limits, until the loss function ceased to improve" (Keskar et al., 2017). This decrease in performance has been named the "generalization gap".

Understanding the origin of the generalization gap, and moreover, finding ways to decrease it, may have a significant practical importance. Training with large batch size immediately increases parallelization, thus has the potential to decrease learning time. Many efforts have been made to parallelize SGD for Deep Learning (Dean et al., 2012; Das et al., 2016; Zhang et al., 2015), yet the speed-ups and scale-out are still limited by the batch size.

In this study we suggest a first attempt to tackle this issue.
First,

- We propose that the initial learning phase can be described using a high-dimensional "random walk on a random potential" process, with an "ultra-slow" logarithmic increase in the distance of the weights from their initialization, as we observe empirically.

Inspired by this hypothesis, we find that

- By simply adjusting the learning rate and batch normalization the generalization gap can be significantly decreased (for example, from 5% to $1\% - 2\%$).
- In contrast to common practices (Montavon et al., 2012) and theoretical recommendations (Hardt et al., 2016), generalization keeps improving for a long time at the initial high learning rate, even without any observable changes in training or validation errors. However, this improvement seems to be related to the distance of the weights from their initialization.
- There is no inherent "generalization gap": large-batch training can generalize as well as small batch training by adapting the number of iterations.

2 Training with a large batch

Training method. A common practice of training deep neural networks is to follow an optimization "regime" in which the objective is minimized using gradient steps with a fixed learning rate and a momentum term (Sutskever et al., 2013). The learning rate is annealed over time, usually with an exponential decrease every few epochs of training data. An alternative to this regime is to use an adaptive per-parameter learning method such as Adam (Kingma & Ba, 2014), Rmsprop (Dauphin et al.) or Adagrad (Duchi et al., 2011). These methods are known to benefit the convergence rate of SGD based optimization. Yet, many current studies still use simple variants of SGD (Ruder, 2016) for all or part of the optimization process (Wu et al., 2016), due to the tendency of these methods to converge to a lower test error and better generalization.

Thus, we focused on momentum SGD, with a fixed learning rate that decreases exponentially every few epochs, similarly to the regime employed by He et al. (2016). The convergence of SGD is also known to be affected by the batch size (Li et al., 2014), but in this work we will focus on generalization. Most of our results were conducted on the Resnet44 topology, introduced by He et al. (2016). We strengthen our findings with additional empirical results in section 6.

Empirical observations of previous work. Previous work by Keskar et al. (2017) studied the performance and properties of models which were trained with relatively large batches and reported the following observations:

- Training models with large batch size increase the generalization error (see Figure 1).
- This "generalization gap" seemed to remain even when the models were trained without limits, until the loss function ceased to improve.
- Low generalization was correlated with "sharp" minima[2] (strong positive curvature), while good generalization was correlated with "flat" minima (weak positive curvature).
- Small-batch regimes were briefly noted to produce weights that are farther away from the initial point, in comparison with the weights produced in a large-batch regime.

Their hypothesis was that a large estimation noise (originated by the use of mini-batch rather than full batch) in small mini-batches encourages the weights to exit out of the basins of attraction of sharp minima, and towards flatter minima which have better generalization.In the next section we provide an analysis that suggest a somewhat different explanation.

3 Theoretical analysis

Notation. In this paper we examine Stochastic Gradient Descent (SGD) based training of a Deep Neural Network (DNN). The DNN is trained on a finite training set of N samples. We define $\mathbf{w}$ as the vector of the neural network parameters, and $L_n(\mathbf{w})$ as loss function on sample n. We find $\mathbf{w}$ by minimizing the training loss.

$$L(\mathbf{w}) \triangleq \frac{1}{N}\sum_{n=1}^{N} L_n(\mathbf{w}) \ ,$$

using SGD. Minimizing $L(\mathbf{w})$ requires an estimate of the gradient of the negative loss.

$$\mathbf{g} \triangleq \frac{1}{N}\sum_{n=1}^{N} \mathbf{g}_n \triangleq -\frac{1}{N}\sum_{n=1}^{N} \nabla L_n(\mathbf{w})$$

where $\mathbf{g}$ is the true gradient, and $\mathbf{g}_n$ is the per-sample gradient. During training we increment the parameter vector $\mathbf{w}$ using only the mean gradient $\hat{\mathbf{g}}$ computed on some mini-batch B – a set of M randomly selected sample indices.

$$\hat{\mathbf{g}} \triangleq \frac{1}{M}\sum_{n\in B} \mathbf{g}_n \, .$$

In order to gain a better insight into the optimization process and the empirical results, we first examine simple SGD training, in which the weights at update step t are incremented according to the mini-batch gradient $\Delta\mathbf{w}_t = \eta\hat{\mathbf{g}}_t$. With respect to the randomness of SGD,

$$\mathbb{E}\hat{\mathbf{g}}_t = \mathbf{g} = -\nabla L(\mathbf{w}_t) \, ,$$

and the increments are uncorrelated between different mini-batches[3]. For physical intuition, one can think of the weight vector $\mathbf{w}_t$ as a particle performing a random walk on the loss ("potential") landscape $L(\mathbf{w}_t)$. Thus, for example, adding momentum term to the increment is similar to adding inertia to the particle.

Motivation. In complex systems (such as DNNs) where we do not know the exact shape of the loss, statistical physics models commonly assume a simpler description of the potential as a random process. For example, Dauphin et al. (2014) explained the observation that local minima tend to have

[2] It was later pointed out (Dinh et al., 2017) that certain "degenerate" directions, in which the parameters can be changed without affecting the loss, must be excluded from this explanation. For example, for any $c > 0$ and any neuron, we can multiply all input weights by c and divide the output weights by c: this does not affect the loss, but can generate arbitrarily strong positive curvature.

[3] Either exactly (with replacement) or approximately (without replacement): see appendix section A.

low error using an analogy between $L(\mathbf{w})$, the DNN loss surface, and the high-dimensional Gaussian random field analyzed in Bray & Dean (2007), which has zero mean and auto-covariance

$$\mathbb{E}\left(L\left(\mathbf{w}_1\right)L\left(\mathbf{w}_2\right)\right) = f\left(\left\|\mathbf{w}_1 - \mathbf{w}_2\right\|^2\right) \tag{1}$$

for some function f, where the expectation now is over the randomness of the loss. This analogy resulted with the hypothesis that in DNNs, local minima with high loss are indeed exponentially vanishing, as in Bray & Dean (2007). Only recently, similar results are starting to be proved for realistic neural network models (Soudry & Hoffer, 2017). Thus, a similar statistical model of the loss might also give useful insights for our empirical observations.

Model: Random walk on a random potential. Fortunately, the high dimensional case of a particle doing a "random walk on a random potential" was extensively investigated already decades ago (Bouchaud & Georges, 1990). The main result of that investigation was that the asymptotic behavior of the auto-covariance of a random potential[4],

$$\mathbb{E}\left(L\left(\mathbf{w}_1\right)L\left(\mathbf{w}_2\right)\right) \sim \left\|\mathbf{w}_1 - \mathbf{w}_2\right\|^{\alpha} \ , \ \alpha > 0 \tag{2}$$

in a certain range, determines the asymptotic behavior of the random walker in that range:

$$\mathbb{E}\left\|\mathbf{w}_t - \mathbf{w}_0\right\|^2 \sim \left(\log t\right)^{\frac{4}{\alpha}} \ . \tag{3}$$

This is called an "ultra-slow diffusion" in which, typically $\left\|\mathbf{w}_t - \mathbf{w}_0\right\| \sim \left(\log t\right)^{2/\alpha}$, in contrast to standard diffusion (on a flat potential), in which we have $\left\|\mathbf{w}_t - \mathbf{w}_0\right\| \sim \sqrt{t}$. The informal reason for this behavior (for any $\alpha > 0$), is that for a particle to move a distance d, it has to pass potential barriers of height $\sim d^{\alpha/2}$, from eq. (2). Then, to climb (or go around) each barrier takes exponentially long time in the height of the barrier: $t \sim \exp(d^{\alpha/2})$. Inverting this relation, we get eq. $d \sim (\log(t))^{2/\alpha}$. In the high-dimensional case, this type of behavior was first shown numerically and explained heuristically by Marinari et al. (1983), then rigorously proven for the case of a discrete lattice by Durrett (1986), and explained in the continuous case by Bouchaud & Comtet (1987).

3.1 Comparison with empirical results and implications

To examine this prediction of ultra slow diffusion and find the value of α, in Figure 2a, we examine $\left\|\mathbf{w}_t - \mathbf{w}_0\right\|$ during the initial training phase over the experiment shown in Figure 1. We found that the weight distance from initialization point increases logarithmically with the number of training iterations (weight updates), which matches our model with $\alpha = 2$:

$$\left\|\mathbf{w}_t - \mathbf{w}_0\right\| \sim \log t \, . \tag{4}$$

Interestingly, the value of $\alpha = 2$ matches the statistics of the loss estimated in appendix section B.

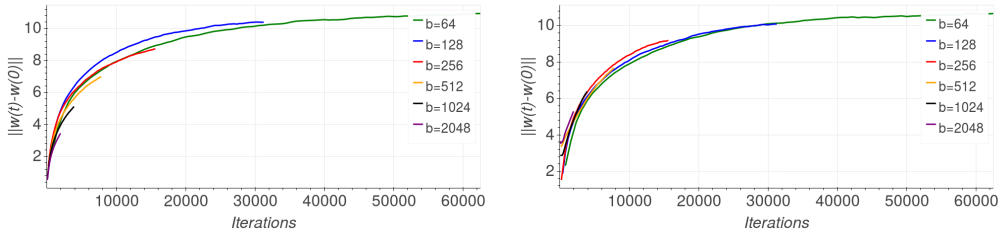
Moreover, in Figure 2a, we find that a very similar logarithmic graph is observed for all batch sizes. Yet, there are two main differences. First, each graph seems to have a somewhat different slope (i.e., it is multiplied by different positive constant), which peaks at $M = 128$ and then decreases with the mini-batch size. This indicates a somewhat different diffusion rate for different batch sizes. Second, since we trained all models for a constant number of epochs, smaller batch sizes entail more training iterations in total. Thus, there is a significant difference in the number of iterations and the corresponding weight distance reached at the end of the initial learning phase.

This leads to the following informal argument (which assumes flat minima are indeed important for generalization). During the initial training phase, to reach a minima of "width" d the weight vector $\mathbf{w}_t$ has to travel at least a distance d, and this takes a long time – about $\exp(d)$ iterations. Thus, to reach wide ("flat") minima we need to have the highest possible diffusion rates (which do not result in numerical instability) and a large number of training iterations. In the next sections we will implement these conclusions in practice.

4 Matching weight increment statistics for different mini-batch sizes

First, to correct the different diffusion rates observed for different batch sizes, we will aim to match the statistics of the weights increments to that of a small batch size.

[4]Note that this form is consistent with eq. (1), if $f(x) = x^{\alpha/2}$.



(a) Before learning rate adjustment and GBN (b) After learning rate adjustment and GBN

Figure 2: Euclidean distance of weight vector from initialization

Learning rate. Recall that in this paper we investigate SGD, possibly with momentum, where the weight updates are proportional to the estimated gradient.

$$\Delta \mathbf{w} \propto \eta \hat{\mathbf{g}} \,, \tag{5}$$

where η is the learning rate, and we ignore for now the effect of batch normalization.

In appendix section A, we show that the covariance matrix of the parameters update step $\Delta \mathbf{w}$ is,

$$\mathrm{cov}\left(\Delta \mathbf{w}, \Delta \mathbf{w}\right) \approx \frac{\eta^2}{M} \left(\frac{1}{N} \sum_{n=1}^{N} \mathbf{g}_n \mathbf{g}_n^\top \right) \tag{6}$$

in the case of uniform sampling of the mini-batch indices (with or without replacement), when $M \ll N$. Therefore, a simple way to make sure that the covariance matrix stays the same for all mini-batch sizes is to choose

$$\eta \propto \sqrt{M} \,, \tag{7}$$

i.e., we should increase the learning rate by the square root of the mini-batch size.

We note that Krizhevsky (2014) suggested a similar learning rate scaling in order to keep the variance in the gradient expectation constant, but chose to use a linear scaling heuristics as it reached better empirical result in his setting. Later on, Li (2017) suggested the same.

Naturally, such an increase in the learning rate also increases the mean steps $\mathbb{E}\left[\Delta \mathbf{w}\right]$. However, we found that this effect is negligible since $\mathbb{E}\left[\Delta \mathbf{w}\right]$ is typically orders of magnitude lower than the standard deviation.

Furthermore, we can match both the first and second order statistics by adding multiplicative noise to the gradient estimate as follows:

$$\hat{\mathbf{g}} = \frac{1}{M} \sum_{n \in B}^{N} \mathbf{g}_n z_n \,,$$

where $z_n \sim \mathcal{N}\left(1, \sigma^2\right)$ are independent random Gaussian variables for which $\sigma^2 \propto M$. This can be verified by using similar calculation as in appendix section A. This method keeps the covariance constant when we change the batch size, yet does not change the mean steps $\mathbb{E}\left[\Delta \mathbf{w}\right]$.

In both cases, for the first few iterations, we had to clip or normalize the gradients to prevent divergence. Since both methods yielded similar performance [5] (due the negligible effect of the first order statistics), we preferred to use the simpler learning rate method.

It is important to note that other types of noise (e.g., dropout (Srivastava et al., 2014), dropconnect (Wan et al., 2013), label noise (Szegedy et al., 2016)) change the structure of the covariance matrix and not just its scale, thus the second order statistics of the small batch increment cannot be accurately matched. Accordingly, we did not find that these types of noise helped to reduce the generalization gap for large batch sizes.

Lastly, note that in our discussion above (and the derivations provided in appendix section A) we assumed each per-sample gradient $\mathbf{g}_n$ does not depend on the selected mini-batch. However, this ignores the influence of batch normalization. We take this into consideration in the next subsection.

[5] a simple comparison can be seen in appendix (figure 3)

Ghost Batch Normalization. Batch Normalization (BN) (Ioffe & Szegedy, 2015), is known to accelerate the training, increase the robustness of neural network to different initialization schemes and improve generalization. Nonetheless, since BN uses the batch statistics it is bounded to depend on the choosen batch size. We study this dependency and observe that by acquiring the statistics on small virtual ("ghost") batches instead of the real large batch we can reduce the generalization error. In our experiments we found out that it is important to use the full batch statistic as suggested by (Ioffe & Szegedy, 2015) for the inference phase. Full details are given in Algorithm 1. This modification by itself reduce the generalization error substantially.

Algorithm 1: Ghost Batch Normalization (GBN), applied to activation x over a large batch B_L with virtual mini-batch B_S. Where $B_S < B_L$.

Require: Values of x over a large-batch: $B_L = \{x_{1...m}\}$ size of virtual batch $|B_S|$; Parameters to be learned: γ, β, momentum η

Training Phase:

Scatter B_L to $\{X^1, X^2, ...X^{|B_L|/|B_S|}\} = \{x_{1...|B_S|}, x_{|B_S|+1...2|B_S|}...x_{|B_L|-|B_S|...m}\}$

$\mu_B^l \leftarrow \frac{1}{|B_S|}\sum_{i=1}^{|B_S|} X_i^l$ for $l = 1, 2, 3 \ldots$ {calculate ghost mini-batches means}

$\sigma_B^l \leftarrow \sqrt{\frac{1}{|B_S|}\sum_{i=1}^{|B_S|}(X_i^l - \mu_B)^2 + \epsilon}$ for $l = 1, 2, 3 \ldots$ {calculate ghost mini-batches std}

$\mu_{run} = (1-\eta)^{|B_S|}\mu_{run} + \sum_{i=1}^{|B_L|/|B_S|}(1-\eta)^i \cdot \eta \cdot \mu_B^l$

$\sigma_{run} = (1-\eta)^{|B_S|}\sigma_{run} + \sum_{i=1}^{|B_L|/|B_S|}(1-\eta)^i \cdot \eta \cdot \sigma_B^l$

return $\gamma\frac{X^l - \mu_B^l}{\sigma_B^l} + \beta$

Test Phase:

return $\gamma\frac{X - \mu_{run}^l}{\sigma_{run}} + \beta$ {scale and shift}

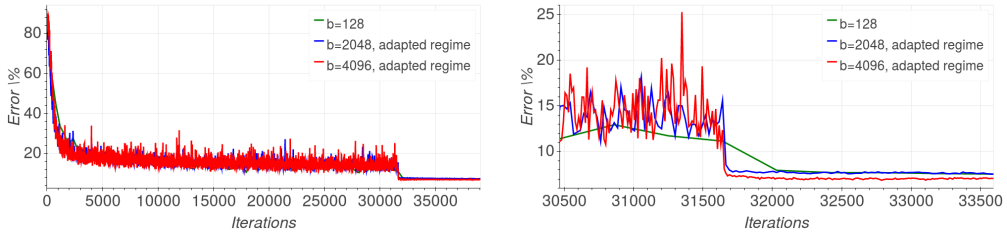
We note that in a multi-device distributed setting, some of the benefits of "Ghost BN" may already occur, since batch-normalization is often preformed on each device separately to avoid additional communication cost. Thus, each device computes the batch norm statistics using only its samples (i.e., part of the whole mini-batch). It is a known fact, yet unpublished, to the best of the authors knowledge, that this form of batch norm update helps generalization and yields better results than computing the batch-norm statistics over the entire batch. Note that GBN enables flexibility in the small (virtual) batch size which is not provided by the commercial frameworks (e.g., TensorFlow, PyTorch) in which the batch statistics is calculated on the entire, per-device, batch. Moreover, in those commercial frameworks, the running statistics are usually computed differently from "Ghost BN", by weighting each update part equally. In our experiments we found it to worsen the generalization performance.

Implementing both the learning rate and GBN adjustments seem to improve generalization performance, as we shall see in section 6. Additionally, as can be seen in Figure 6, the slopes of the logarithmic weight distance graphs seem to better matched, indicating similar diffusion rates. We also observe some constant shift, which we believe is related to the gradient clipping. Since this shift only increased the weight distances, we assume it does not harm the performance.

5 Adapting number of weight updates eliminates generalization gap

According to our conclusions in section 3, the initial high-learning rate training phase enables the model to reach farther locations in the parameter space, which may be necessary to find wider local minima and better generalization. Examining figure 2b, the next obvious step to match the graphs for different batch sizes is to increase the number of training iterations in the initial high learning rate regime. And indeed we noticed that the distance between the current weight and the initialization point can be a good measure to decide upon when to decrease the learning rate.

Note that this is different from common practices. Usually, practitioners decrease the learning rate after validation error appears to reach a plateau. This practice is due to the long-held belief that the optimization process should not be allowed to decrease the training error when validation error "flatlines", for fear of overfitting (Girosi et al., 1995). However, we observed that substantial improvement to the final accuracy can be obtained by continuing the optimization using the same



(a) Validation error

(b) Validation error - zoomed

Figure 3: Comparing generalization of large-batch regimes, adapted to match performance of small-batch training.

learning rate even if the training error decreases while the validation plateaus. Subsequent learning rate drops resulted with a sharp validation error decrease, and better generalization for the final model.

These observations led us to believe that "generalization gap" phenomenon stems from the relatively small number of updates rather than the batch size. Specifically, using the insights from Figure 2 and our model, we adapted the training regime to better suit the usage of large mini-batch. We "stretched" the time-frame of the optimization process, where each time period of e epochs in the original regime, will be transformed to $\frac{|B_L|}{|B_S|}e$ epochs according to the mini-batch size used. This modification ensures that the number of optimization steps taken is identical to those performed in the small batch regime. As can be seen in Figure 3, combining this modification with learning rate adjustment completely eliminates the generalization gap observed earlier [6].

6 Experiments

Experimental setting. We experimented with a set of popular image classification tasks:

- MNIST (LeCun et al., 1998b) - Consists of a training set of $60K$ and a test set of $10K$ 28×28 gray-scale images representing digits ranging from 0 to 9.
- CIFAR-10 and CIFAR-100 (Krizhevsky, 2009) - Each consists of a training set of size 50K and a test set of size $10K$. Instance are 32×32 color images representing 10 or 100 classes.
- ImageNet classification task Deng et al. (2009) - Consists of a training set of size 1.2M samples and test set of size 50K. Each instance is labeled with one of 1000 categories.

To validate our findings, we used a representative choice of neural network models. We used the fully-connected model, F1, as well as shallow convolutional models C1 and C3 suggested by Keskar et al. (2017). As a demonstration of more current architectures, we used the models: VGG (Simonyan, 2014) and Resnet44 (He et al., 2016) for CIFAR10 dataset, Wide-Resnet16-4 (Zagoruyko, 2016) for CIFAR100 dataset and Alexnet (Krizhevsky, 2014) for ImageNet dataset.
In each of the experiments, we used the training regime suggested by the original work, together with a momentum SGD optimizer. We use a batch of 4096 samples as "large batch" (LB) and a small batch (SB) of either 128 (F1,C1,VGG,Resnet44,C3,Alexnet) or 256 (WResnet). We compare the original training baseline for small and large batch, as well as the following methods[7]:

- Learning rate tuning (LB+LR): Using a large batch, while adapting the learning rate to be larger so that $\eta_L = \sqrt{\frac{|B_L|}{|B_S|}}\eta_S$ where η_S is the original learning rate used for small batch, η_L is the adapted learning rate and $|B_L|, |B_S|$ are the large and small batch sizes, respectively.
- Ghost batch norm (LB+LR+GBN): Additionally using the "Ghost batch normalization" method in our training procedure. The "ghost batch size" used is 128.
- Regime adaptation: Using the tuned learning rate as well as ghost batch-norm, but with an adapted training regime. The training regime is modified to have the same number of

[6] Additional graphs, including comparison to non-adapted regime, are available in appendix (figure 2).
[7] Code is available at `https://github.com/eladhoffer/bigBatch`.

iterations for each batch size used - effectively multiplying the number of epochs by the relative size of the large batch.

Results. Following our experiments, we can establish an empirical basis to our claims. Observing the final validation accuracy displayed in Table 1, we can see that in accordance with previous works the move from a small-batch (SB) to a large-batch (LB) indeed incurs a substantial generalization gap. However, modifying the learning-rate used for large-batch (+LR) causes much of this gap to diminish, following with an additional improvement by using the Ghost-BN method (+GBN). Finally, we can see that the generalization gap completely disappears when the training regime is adapted (+RA), yielding validation accuracy that is good-as or better than the one obtained using a small batch.
We additionally display results obtained on the more challenging ImageNet dataset in Table 2 which shows similar impact for our methods.

Table 1: Validation accuracy results, SB/LB represent small and large batch respectively. GBN stands for Ghost-BN, and RA stands for regime adaptation

Network	Dataset	SB	LB	+LR	+GBN	+RA
F1 (Keskar et al., 2017)	MNIST	98.27%	97.05%	97.55%	97.60%	98.53%
C1 (Keskar et al., 2017)	Cifar10	87.80%	83.95%	86.15%	86.4%	88.20%
Resnet44 (He et al., 2016)	Cifar10	92.83%	86.10%	89.30%	90.50%	93.07%
VGG (Simonyan, 2014)	Cifar10	92.30%	84.1%	88.6%	91.50%	93.03%
C3 (Keskar et al., 2017)	Cifar100	61.25%	51.50%	57.38%	57.5%	63.20%
WResnet16-4 (Zagoruyko, 2016)	Cifar100	73.70%	68.15%	69.05%	71.20%	73.57%

Table 2: ImageNet top-1 results using Alexnet topology (Krizhevsky, 2014), notation as in Table 1.

Network	LB size	Dataset	SB	LB[8]	+LR[8]	+GBN	+RA
Alexnet	4096	ImageNet	57.10%	41.23%	53.25%	54.92%	59.5%
Alexnet	8192	ImageNet	57.10%	41.23%	53.25%	53.93%	59.5%

7 Discussion

There are two important issues regarding the use of large batch sizes. First, why do we get worse generalization with a larger batch, and how do we avoid this behaviour? Second, can we decrease the training wall clock time by using a larger batch (exploiting parallelization), while retaining the same generalization performance as in small batch?

This work tackles the first issue by investigating the random walk behaviour of SGD and the relationship of its diffusion rate to the size of a batch. Based on this and empirical observations, we propose simple set of remedies to close down the generalization gap between the small and large batch training strategies: (1) Use SGD with momentum, gradient clipping, and a decreasing learning rate schedule; (2) adapt the learning rate with batch size (we used a square root scaling); (3) compute batch-norm statistics over several partitions ("ghost batch-norm"); and (4) use a sufficient number of high learning rate training iterations.

Thus, the main point arising from our results is that, in contrast to previous conception, there is no inherent generalization problem with training using large mini batches. That is, model training using large mini-batches can generalize as well as models trained using small mini-batches. Though this answers the first issues, the second issue remained open: can we speed up training by using large batch sizes?

Not long after our paper first appeared, this issue was also answered. Using a Resnet model on Imagenet Goyal et al. (2017) showed that, indeed, significant speedups in training could be achieved using a large batch size. This further highlights the ideas brought in this work and their importance to future scale-up, especially since Goyal et al. (2017) used similar training practices to those we

[8] Due to memory limitation those experiments were conducted with batch of 2048.

described above. The main difference between our works is the use of a linear scaling of the learning rate[9], similarly to Krizhevsky (2014), and as suggested by Bottou (2010). However, we found that linear scaling works less well on CIFAR10, and later work found that linear scaling rules work less well for other architectures on ImageNet (You et al., 2017).

We also note that current "rules of thumb" regarding optimization regime and explicitly learning rate annealing schedule may be misguided. We showed that good generalization can result from extensive amount of gradient updates in which there is no apparent validation error change and training error continues to drop, in contrast to common practice. After our work appeared, Soudry et al. (2017) suggested an explanation to this, and to the logarithmic increase in the weight distance observed in Figure 2. We show this behavior happens even in simple logistic regression problems with separable data. In this case, we exactly solve the asymptotic dynamics and prove that $\mathbf{w}(t) = \log(t)\hat{\mathbf{w}} + O(1)$ where $\hat{\mathbf{w}}$ is to the L_2 maximum margin separator. Therefore, the margin (affecting generalization) improves slowly (as $O(1/\log(t))$, even while the training error is very low. Future work, based on this, may be focused on finding when and how the learning rate should be decreased while training.

Conclusion. In this work we make a first attempt to tackle the "generalization gap" phenomenon. We argue that the initial learning phase can be described using a high-dimensional "random walk on a random potential" process, with a an "ultra-slow" logarithmic increase in the distance of the weights from their initialization, as we observe empirically. Following this observation we suggest several techniques which enable training with large batch without suffering from performance degradation. This implies that the problem is not related to the batch size but rather to the amount of updates. Moreover we introduce a simple yet efficient algorithm "Ghost-BN" which improves the generalization performance significantly while keeping the training time intact.

Acknowledgments

We wish to thank Nir Ailon, Dar Gilboa, Kfir Levy and Igor Berman for their feedback on the initial manuscript. The research was partially supported by the Taub Foundation, and the Intelligence Advanced Research Projects Activity (IARPA) via Department of Interior/ Interior Business Center (DoI/IBC) contract number D16PC00003. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. Disclaimer: The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of IARPA, DoI/IBC, or the U.S. Government.

References

Amodei, D., Anubhai, R., Battenberg, E., et al. Deep speech 2: End-to-end speech recognition in english and mandarin. *arXiv preprint arXiv:1512.02595*, 2015.

Bottou, L. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pp. 177–186. Springer, 2010.

Bouchaud, J. P. and Georges, A. Anomalous diffusion in disordered media: statistical mechanisms, models and physical applications. *Physics reports*, 195:127–293, 1990.

Bouchaud, J. P. and Comtet, A. Anomalous diffusion in random media of any dimensionality. *J. Physique*, 48: 1445–1450, 1987.

Bray, A. J. and Dean, D. S. Statistics of critical points of Gaussian fields on large-dimensional spaces. *Physical Review Letters*, 98(15):1–5, 2007.

Choromanska, A., Henaff, M., Mathieu, M., Arous, G. B., and LeCun, Y. The Loss Surfaces of Multilayer Networks. *AISTATS15*, 38, 2015.

Das, D., Avancha, S., Mudigere, D., et al. Distributed deep learning using synchronous stochastic gradient descent. *arXiv preprint arXiv:1602.06709*, 2016.

Dauphin, Y., de Vries, H., Chung, J., and Bengio, Y. Rmsprop and equilibrated adaptive learning rates for non-convex optimization. corr abs/1502.04390 (2015).

Dauphin, Y., Pascanu, R., and Gulcehre, C. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *NIPS*, pp. 1–9, 2014.

Dean, J., Corrado, G., Monga, R., et al. Large scale distributed deep networks. In *NIPS*, pp. 1223–1231, 2012.

Deng, J., Dong, W., Socher, R., et al. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[9] e.g., Goyal et al. (2017) also used an initial warm-phase for the learning rate, however, this has a similar effect to the gradient clipping we used, since this clipping was mostly active during the initial steps of training.

Dinh, L., Pascanu, R., Bengio, S., and Bengio, Y. Sharp minima can generalize for deep nets. *arXiv preprint arXiv:1703.04933*, 2017.

Duchi, J., Hazan, E., and Singer, Y. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

Durrett, R. Multidimensional random walks in random environments with subclassical limiting behavior. *Communications in Mathematical Physics*, 104(1):87–102, 1986.

Ge, R., Huang, F., Jin, C., and Yuan, Y. Escaping from saddle points-online stochastic gradient for tensor decomposition. In *COLT*, pp. 797–842, 2015.

Girosi, F., Jones, M., and Poggio, T. Regularization theory and neural networks architectures. *Neural computation*, 7(2):219–269, 1995.

Goyal, P., Dollár, P., Girshick, R., et al. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

Hardt, M., Recht, B., and Singer, Y. Train faster, generalize better: Stability of stochastic gradient descent. *ICML*, pp. 1–24, 2016.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., and Tang, P. T. P. On large-batch training for deep learning: Generalization gap and sharp minima. In *ICLR*, 2017.

Kingma, D. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Krizhevsky, A. Learning multiple layers of features from tiny images. 2009.

Krizhevsky, A. One weird trick for parallelizing convolutional neural networks. *arXiv preprint arXiv:1404.5997*, 2014.

LeCun, Y., Bottou, L., and Orr, G. Efficient backprop in neural networks: Tricks of the trade (orr, g. and müller, k., eds.). *Lecture Notes in Computer Science*, 1524, 1998a.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998b.

Li, M. *Scaling Distributed Machine Learning with System and Algorithm Co-design*. PhD thesis, Intel, 2017.

Li, M., Zhang, T., Chen, Y., and Smola, A. J. Efficient mini-batch training for stochastic optimization. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 661–670. ACM, 2014.

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

Marinari, E., Parisi, G., Ruelle, D., and Windey, P. Random Walk in a Random Environment and 1f Noise. *Physical Review Letters*, 50(1):1223–1225, 1983.

Mnih, V., Kavukcuoglu, K., Silver, D., et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Montavon, G., Orr, G., and Müller, K.-R. *Neural Networks: Tricks of the Trade*. 2 edition, 2012. ISBN 978-3-642-35288-1.

Ruder, S. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.

Silver, D., Huang, A., Maddison, C. J., et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

Simonyan, K. e. a. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Soudry, D., Hoffer, E., and Srebro, N. The Implicit Bias of Gradient Descent on Separable Data. *ArXiv e-prints*, October 2017.

Soudry, D. and Hoffer, E. Exponentially vanishing sub-optimal local minima in multilayer neural networks. *arXiv preprint arXiv:1702.05777*, 2017.

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Sutskever, I., Martens, J., Dahl, G., and Hinton, G. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pp. 1139–1147, 2013.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2818–2826, 2016.

Wan, L., Zeiler, M., Zhang, S., LeCun, Y., and Fergus, R. Regularization of neural networks using dropconnect. ICML'13, pp. III–1058–III–1066. JMLR.org, 2013.

Wu, Y., Schuster, M., Chen, Z., et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144, 2016.

You, Y., Gitman, I., and Ginsburg, B. Scaling sgd batch size to 32k for imagenet training. *arXiv preprint arXiv:1708.03888*, 2017.

Zagoruyko, K. Wide residual networks. In *BMVC*, 2016.

Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. Understanding deep learning requires rethinking generalization. In *ICLR*, 2017.

Zhang, S., Choromanska, A. E., and LeCun, Y. Deep learning with elastic averaging sgd. In *NIPS*, pp. 685–693, 2015.

Supplementary Material for "Train longer, generalize better: closing the generalization gap in large batch training regime of neural networks"

Appendix

A Derivation of eq. (6)

Note that we can write the mini-batch gradient as

$$\hat{\mathbf{g}} = \frac{1}{M}\sum_{n=1}^{N} \mathbf{g}_n s_n \text{ with } s_n \triangleq \begin{cases} 1 & \text{, if } n \in B \\ 0 & \text{, if } n \notin B \end{cases}$$

Clearly, $\hat{\mathbf{g}}$ is an unbiased estimator of $\mathbf{g}$, if

$$\mathbb{E}s_n = P\left(s_n = 1\right) = \frac{M}{N}\,.$$

since then

$$\mathbb{E}\hat{\mathbf{g}} = \frac{1}{M}\sum_{n=1}^{N} \mathbf{g}_n \mathbb{E}s_n = \frac{1}{N}\sum_{n=1}^{N} \mathbf{g}_n = \mathbf{g}\,.$$

First, we consider the simpler case of sampling with replacement. In this case it easy to see that different minibatches are uncorrelated, and we have

$$\begin{aligned}\mathbb{E}\left[s_n s_{n'}\right] &= P\left(s_n = 1\right)\delta_{nn'} + P\left(s_n = 1, s_{n'} = 1\right)\left(1 - \delta_{nn'}\right) \\ &= \frac{M}{N}\delta_{nn'} + \frac{M^2}{N^2}\left(1 - \delta_{nn'}\right)\,.\end{aligned}$$

and therefore

$$\begin{aligned}\mathrm{cov}\left(\hat{\mathbf{g}}, \hat{\mathbf{g}}\right) &= \mathbb{E}\left[\hat{\mathbf{g}}\hat{\mathbf{g}}^\top\right] - \mathbb{E}\hat{\mathbf{g}}\mathbb{E}\hat{\mathbf{g}}^\top \\ &= \frac{1}{M^2}\sum_{n=1}^{N}\sum_{n'=1}^{N}\mathbb{E}\left[s_n s_{n'}\right]\mathbf{g}_n\mathbf{g}_{n'}^\top - \mathbf{g}\mathbf{g}^\top \\ &= \frac{1}{M^2}\sum_{n=1}^{N}\sum_{n'=1}^{N}\left[\frac{M}{N}\delta_{nn'} + \frac{M^2}{N^2}\left(1 - \delta_{nn'}\right)\right]\mathbf{g}_n\mathbf{g}_{n'}^\top - \mathbf{g}\mathbf{g}^\top \\ &= \left(\frac{1}{M} - \frac{1}{N}\right)\frac{1}{N}\sum_{n=1}^{N}\mathbf{g}_n\mathbf{g}_n^\top,\end{aligned}$$

which confirms eq. (6).

Next, we consider the case of sampling without replacement. In this case the selector variables are now different and correlated between different mini-batches (*e.g.*, with indices t and $t + k$), since we

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

cannot select previous samples. Thus, these variables s_n^t and s_n^{t+k} have the following second-order statistics

$$\begin{aligned}\mathbb{E}\left[s_n^t s_{n'}^{t+k}\right] &= P\left(s_n^t = 1, s_{n'}^{t+k} = 1\right) \\ &= \frac{M}{N}\delta_{nn'}\delta_{k0} + \frac{M}{N}\frac{M}{N-1}\left(1 - \delta_{nn'}\delta_{k0}\right) .\end{aligned}$$

This implies

$$\begin{aligned}&\mathbb{E}\left[\hat{\mathbf{g}}_t\hat{\mathbf{g}}_{t+k}^\top\right] - \mathbb{E}\hat{\mathbf{g}}_t\mathbb{E}\hat{\mathbf{g}}_{t+k}^\top \\ &= \mathbb{E}\left[\left(\frac{1}{M}\sum_{n=1}^N s_n^t\mathbf{g}_n\right)\left(\frac{1}{M}\sum_{n'=1}^N s_{n'}^{t+k}\mathbf{g}_{n'}^\top\right)\right] - \mathbf{g}\mathbf{g}^\top \\ &= \frac{1}{M^2}\sum_{n=1}^N\sum_{n'=1}^N \mathbb{E}\left[s_n^t s_{n'}^{t+k}\right]\mathbf{g}_n\mathbf{g}_{n'}^\top - \mathbf{g}\mathbf{g}^\top \\ &= \frac{1}{M^2}\sum_{n=1}^N\sum_{n'=1}^N \left[\left(\frac{M}{N} - \frac{M^2}{N^2-N}\right)\delta_{nn'}\delta_{k0} - \frac{M^2}{N^2-N}\right]\mathbf{g}_n\mathbf{g}_{n'}^\top - \mathbf{g}\mathbf{g}^\top\end{aligned}$$

so, if $k = 0$ the covariance is

$$\begin{aligned}\mathbb{E}\left[\hat{\mathbf{g}}_t\hat{\mathbf{g}}_t^\top\right] - \mathbb{E}\hat{\mathbf{g}}_t\mathbb{E}\hat{\mathbf{g}}_t^\top &= \left(\frac{1}{M} - \frac{1}{N-1}\right)\frac{1}{N}\sum_{n=1}^N \mathbf{g}_n\mathbf{g}_n^\top + \frac{1}{N-1}\mathbf{g}\mathbf{g}^\top \\ &\overset{M\ll N}{\approx} \frac{1}{M}\left(\frac{1}{N}\sum_{n=1}^N \mathbf{g}_n\mathbf{g}_n^\top\right)\end{aligned}$$

while the covariance between different minibatches ($k \neq 0$) is much smaller for $M \ll N$

$$\mathbb{E}\left[\hat{\mathbf{g}}_t\hat{\mathbf{g}}_{t+k}^\top\right] - \mathbb{E}\hat{\mathbf{g}}_t\mathbb{E}\hat{\mathbf{g}}_{t+k}^\top = \frac{1}{N-1}\mathbf{g}\mathbf{g}^\top$$

this again confirms eq. (6).

B Estimating α from random potential

The logarithmic increase in weight distance (Figure 2 in the paper) matches a "random walk on a random potential" model with $\alpha = 2$. In such a model the loss auto-covariance asymptotically increases with the square of the weight distance, or, equivalently (Marinari et al., 1983), the standard deviation of the loss difference asymptotically increases linearly with the weight distance

$$\text{std} \triangleq \sqrt{\mathbb{E}\left(L\left(\mathbf{w}\right) - L\left(\mathbf{w}_0\right)\right)^2} \sim \|\mathbf{w} - \mathbf{w}_0\| . \tag{1}$$

In this section we examine this behavior: in Figure we indeed find such a linear behavior, confirming the prediction of our model with $\alpha = 2$.

To obtain the relevant statistics to plot eq. 1 we conducted the following experiment on Resnet44 model (He et al., 2016). We initialized the model weights, $\mathbf{w}_0$, according to Glorot & Bengio (2010), and repeated the following steps a 1000 times, given some parameter c:

- Sample a random direction $\mathbf{v}$ with norm one.
- Sample a scalar z uniformly in some range $[0, c]$.
- Choose $\mathbf{w} = \mathbf{w}_0 + z\mathbf{v}$.
- Save $||\mathbf{w} - \mathbf{w}_0||$ and $L(\mathbf{w})$.

We have set the parameter c so that the maximum weight distance from initialization $||\mathbf{w} - \mathbf{w}_0||$ is equal to the same maximal distance in Figure 2 in the paper, *i.e.,* $c \approx 10$.

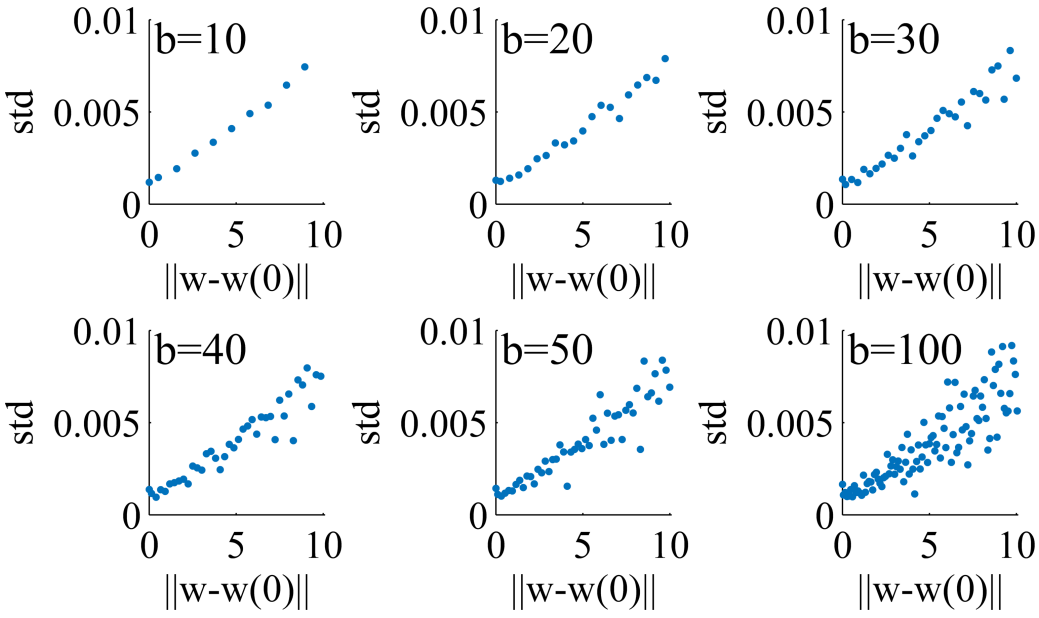


Figure 1: **The standard deviation of the loss shows linear dependence on weight distance (eq. 1) as predicted by the "random walk on a random potential" model with $\alpha = 2$ we found in the main paper.** To approximate the ensemble average in eq. 1 we divided the x-axis to b bins and calculated the empiric average in each bin. Each panel shows the resulting graph for a different value of b.

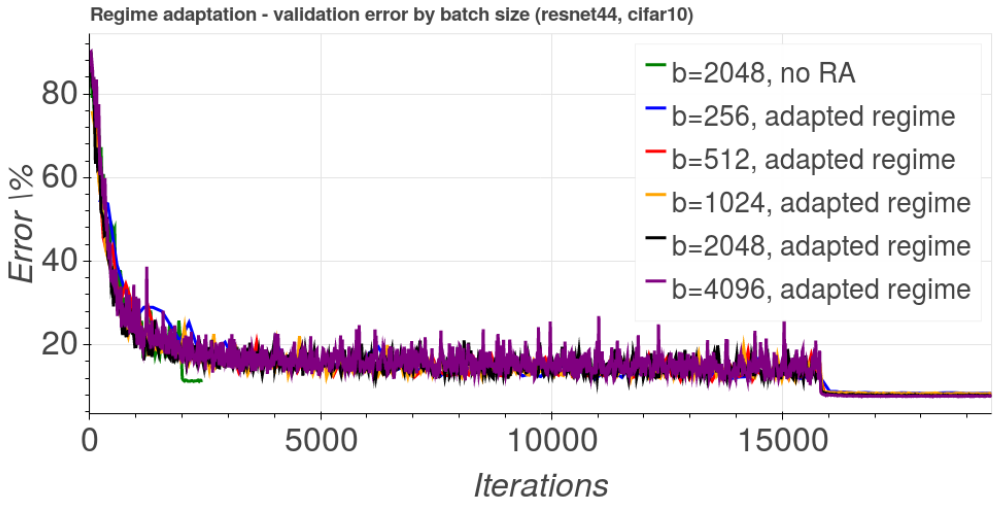


Figure 2: Comparing regime adapted large batch training vs. a 2048 batch with no adaptation.

References

Glorot, X. and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In *Aistats*, volume 9, pp. 249–256, 2010.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

Marinari, E., Parisi, G., Ruelle, D., and Windey, P. Random Walk in a Random Environment and 1f Noise. *Physical Review Letters*, 50(1):1223–1225, 1983.

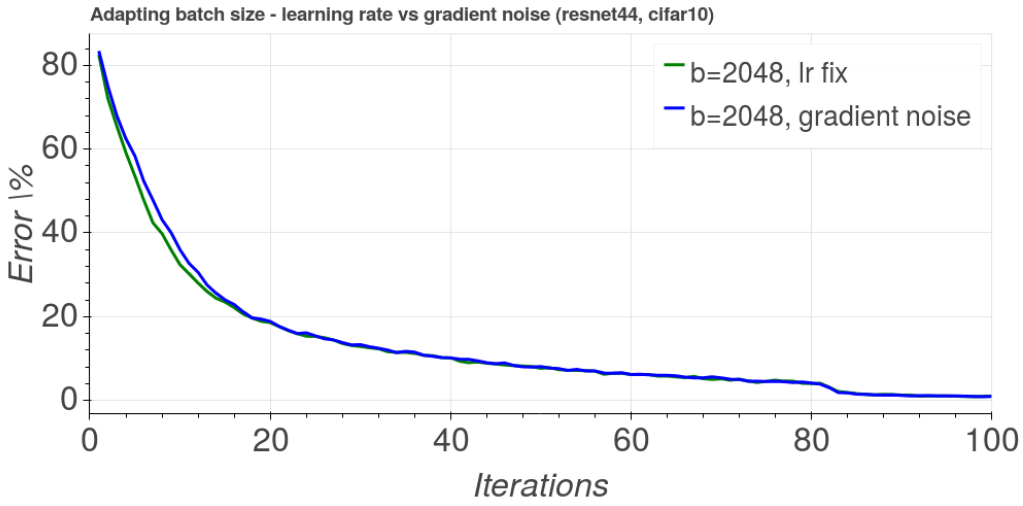


Figure 3: Comparing a learning scale fix for a 2048 batch, to a multiplicative noise to the gradient of the same scale

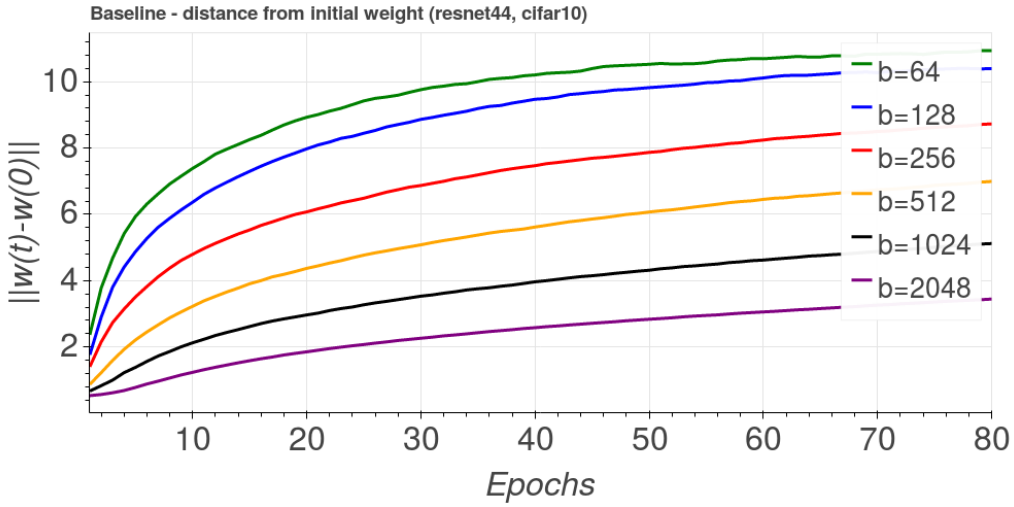


Figure 4: Comparing L_2 distance from initial weight for different batch sizes

Chapter 5

Fix your classifier: the marginal value of training the last weight layer

Fix your classifier: the marginal value of training the last weight layer

Elad Hoffer, Itay Hubara, Daniel Soudry
Department of Electrical Engineering
Technion
Haifa, 320003, Israel
elad.hoffer, itay.hubara, daniel.soudry@gmail.com

Abstract

Neural networks are commonly used as models for classification for a wide variety of tasks. Typically, a learned affine transformation is placed at the end of such models, yielding a per-class value used for classification. This classifier can have a vast number of parameters, which grows linearly with the number of possible classes, thus requiring increasingly more resources.

In this work we argue that this classifier can be fixed, up to a global scale constant, with little or no loss of accuracy for most tasks, allowing memory and computational benefits. Moreover, we show that by initializing the classifier with a Hadamard matrix we can speed up inference as well. We discuss the implications for current understanding of neural network models.

1 Introduction

Deep neural network have become a widely used model for machine learning, achieving state-of-the-art results on many tasks. The most common task these models are used for is to perform classification, as in the case of convolutional neural networks (CNNs) used to classify images to a semantic category. CNN models are currently considered the standard for visual tasks, allowing far better accuracy than preceding approaches (Krizhevsky et al., 2012; He et al., 2016; Szegedy et al., 2015).

Training NN models and using them for inference requires large amounts of memory and computational resources, thus, extensive amount of research has been done lately to reduce the size of networks. Han et al. (2015) used weight sharing and specification, Micikevicius et al. (2017) used mixed precision to reduce the size of the neural networks by half. Tai et al. (2015) and Jaderberg et al. (2014) used low rank approximations to speed up NNs.

Hubara et al. (2016b), Li et al. (2016) and Zhou et al. (2016), used a more aggressive approach, in which weights, activations and gradients were quantized to further reduce computation during training. Although aggressive quantization benefits from smaller model size, the extreme compression rate comes with a loss of accuracy.

Past work noted the fact that predefined (Park & Sandberg, 1991) and random (Huang et al., 2006) projections can be used together with a learned affine transformation to achieve competitive results on several tasks. In this study suggest the reversed proposal - that common NN models used can learn useful representation even without modifying the final output layer, which often holds a large number of parameters that grows linearly with number of classes.

1.1 Classifiers in convolutional neural networks

Convolutional neural networks (CNNs) are commonly used to solve a variety of spatial and temporal tasks. CNNs are usually composed of a stack of convolutional parameterized layers, spatial pooling

layers and fully connected layers, separated by non-linear activation functions. Earlier architectures of CNNs (LeCun et al., 1998; Krizhevsky et al., 2012) used a set of fully-connected layers at later stage of the network, presumably to allow classification based on global features of an image. The final classifier can also be replaced with a convolutional layer with output feature maps matching the number of classes, as demonstrated by Springenberg et al. (2014).

Despite the enormous number of trainable parameters these layers added to the model, they are known to have a rather marginal impact on the final performance of the network (Zeiler & Fergus, 2014) and are easily compressed and reduced after a model was trained by simple means such as matrix decomposition and sparsification (Han et al., 2015). Further more, modern architecture choices are characterized with the removal of most of the fully connected layers (Lin et al., 2013; Szegedy et al., 2015; He et al., 2016), which was found to lead to better generalization and overall accuracy, together with a huge decrease in the number of trainable parameters.

Additionally, numerous works showed that CNNs can be trained in a metric learning regime (Bromley et al., 1994; Schroff et al., 2015; Hoffer & Ailon, 2015), where no explicit classification layer was introduced and the objective regarded only distance measures between intermediate representations. Hardt & Ma (2017) suggested an all-convolutional network variant, where they kept the original initialization of the classification layer fixed with no negative impact on performance on the Cifar10 dataset. All of these properties provide evidence that fully-connected layers are in fact redundant and play a small role in learning and generalization.

Despite the apparent minor role they play, fully-connected layers are still commonly used as classification layers, transforming from the dimension of network features N to the number of required class categories C. Therefore, each classification model must hold $N \cdot C$ number of trainable parameters that grows in a linear manner with the number of classes. This property still holds when the fully-connected layer is replaced with a convolutional classifier as shown by Springenberg et al. (2014).

In this work we claim that for common use-cases of convolutional network, the parameters used for the final classification transform are completely redundant, and can be replaced with a predetermined linear transform. As we will show for the first time, this property holds even in large-scale models and classification tasks, such as recent architectures trained on the ImageNet benchmark (Deng et al., 2009).

The use of a fixed transform can, in many cases, allow a huge decrease in model parameters, and a possible computational benefit. We suggest that existing models can, with no other modification, devoid their classifier weights, which can help the deployment of those models in devices with low computation ability and smaller memory capacity. Moreover, as we keep the classifier fixed, less parameters need to be updated, reducing the communication cost for models deployed in distributed systems. The use of a fixed transform which does not depend on the number classes can allow models to scale to a large number of possible outputs, without a linear cost in the number of parameters. We also suggest that these finding might shed light on the importance of the preceding non-linear layers to learning and generalization.

2 Using a fixed classifier

2.1 Fully-connected classifiers

We focus our attention on the final representation obtained by the network (the last hidden layer), before the classifier. We denote these representation as $x = F(z; \theta)$ where F is assumed to be a deep neural network with input z and parameters θ, e.g., a convolutional network, trained by back-propagation.

In common NN models, this representation is followed by an additional affine transformation

$$y = W^T x + b$$

where W and b are also trained by back-propagation.

For input x of N length, and C different possible outputs, W is required to be a matrix of $N \times C$. Training is done using cross-entropy loss, by feeding the network outputs through a softmax

activation

$$v_i = \frac{e^{y_i}}{\sum_j^C e^{y_j}},\ i \in \{1, \ldots, C\}$$

and reducing the expected negative log likelihood with respect to ground-truth target $t \in \{1, \ldots, C\}$, by minimizing

$$\mathcal{L}(x, t) = -\log v_t = -w_t \cdot x - b_t + \log\left(\sum_j^C e^{w_j \cdot x + b_j}\right)$$

where w_i is the i-th column of W.

2.2 Choosing the projection matrix

To evaluate our conjecture regarding the importance of the final classification transformation, we replaced the trainable parameter matrix W with a fixed orthonormal projection $Q \in \mathbb{R}^{N \times C}$, such that $\forall i \neq j : q_i \cdot q_j = 0$ and $\|q_i\|_2 = 1$, where q_i is the ith column of Q. This can be ensured by a simple random sampling and singular-value decomposition

As the rows of classifier weight matrix are fixed with an equally valued L_2 norm, we find it beneficial to also restrict the representation of x by normalizing it to reside on the n-dimensional sphere

$$\hat{x} = \frac{x}{\|x\|_2} \tag{1}$$

This allows faster training and convergence, as the network does not need to account for changes in the scale of its weights.

We now face the problem that $q_i \cdot \hat{x}$ is bounded between -1 and 1. This causes convergence issues, as the softmax function is scale sensitive, and the network is affected by the inability to re-scale its input. This is similar to the phenomenon described by Vaswani et al. (2017) with respect to softmax function used for attention mechanisms. In the same spirit, we can amend this issue with a fixed scale T applied to softmax inputs $f(y) = \mathrm{softmax}(\frac{1}{T}y)$, also known as a softmax temperature. However, this introduces an additional hyper-parameter which may differ between networks and datasets. Instead, we suggest to introduce a single scalar parameter α to learn the softmax scale, effectively functioning as an inverse of the softmax temperature $\frac{1}{T}$.
Using normalized weights and an additional scale coefficient is similar in spirit to weight-normalization (Salimans & Kingma, 2016), with the difference that we use a single scale for all entries in the weight matrix, in contrast to a scale for each row that Salimans & Kingma (2016) uses.

We keep the additional vector of bias parameters $b \in \mathbb{R}^C$, and train using the same negative-log-likelihood criterion. More explicitly, our classifier output is now

$$v_i = \frac{e^{\alpha q_i \cdot \hat{x} + b_i}}{\sum_j^C e^{\alpha q_j \cdot \hat{x} + b_j}},\ i \in \{1, \ldots, C\}$$

and we minimize the loss:

$$\mathcal{L}(x, t) = -\alpha q_t \cdot \frac{x}{\|x\|_2} + b_t + \log\left(\sum_{i=1}^C \exp\left(\alpha q_i \cdot \frac{x}{\|x\|_2} + b_i\right)\right)$$

where we recall x is the final representation obtained by the network for a specific sample, and $t \in \{1, \ldots, C\}$ is the ground-truth label for that sample.

Observing the behavior of the α parameter over time revealed a logarithmic growth depicted in graph 1. Interestingly, this is the same behavior exhibited by the norm of a learned classifier, first described by Hoffer et al. (2017) and linked to the generalization of the network. This was recently explained by the under-review work of Soudry et al. (2018) as convergence to a max margin classifier. We suggest that using a single parameter will enable a simpler examination and possible further exploration of this phenomenon and its implications.

We note that as $-1 \leq q_i \cdot \hat{x} \leq 1$, we also found it possible to train the network with a simple cosine angle loss:

$$\mathcal{L}(\hat{x}, t) = \begin{cases} q_i \cdot \hat{x} - 1, & \text{if } i = t, \\ q_i \cdot \hat{x} + 1, & \text{otherwise.} \end{cases}$$

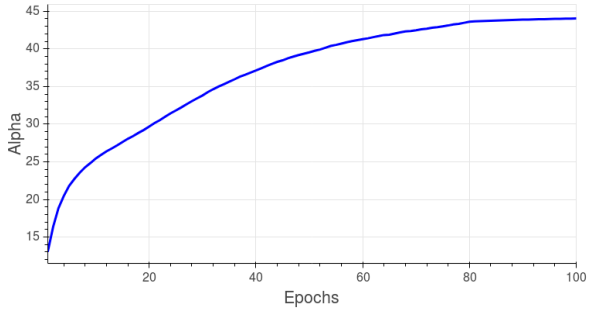


Figure 1: The softmax scale coefficient α was observed to follow a logarithmic growth over the course of training.

allowing to discard the softmax function and its scale altogether, but resulting in a slight decrease in final validation accuracy compared to original models.

2.3 Using a fixed Hadmard matrix

We further suggest the use of a Hadamard matrix (Hedayat et al., 1978) as the final classification transform. Hadamard matrix H is an $n \times n$ matrix, where all of its entries are either $+1$ or -1. Further more, H is orthogonal, such that $HH^T = nI_n$ where I_n is the identity matrix.
We can use a truncated Hadamard matrix $\hat{H} \in \{-1, 1\}^{C \times N}$ where all C rows are orthogonal as our final classification layer such that

$$y = \hat{H}\hat{x} + b$$

This usage allows two main benefits:

- A deterministic, low-memory and easily generated matrix that can be used to classify.
- Removal of the need to perform a full matrix-matrix multiplication - as multiplying by a Hadamard matrix can be done by simple sign manipulation and addition.

We note that n must be a multiple of 4, but it can be easily truncated to fit normally defined networks.

We also note the similarity of using a Hadamard matrix as a final classifier to methods of weight binarization such as the one suggested by Courbariaux et al. (2015). As the classifier weights are fixed to need only 1-bit precision, it is now possible to focus our attention on the features preceding it.

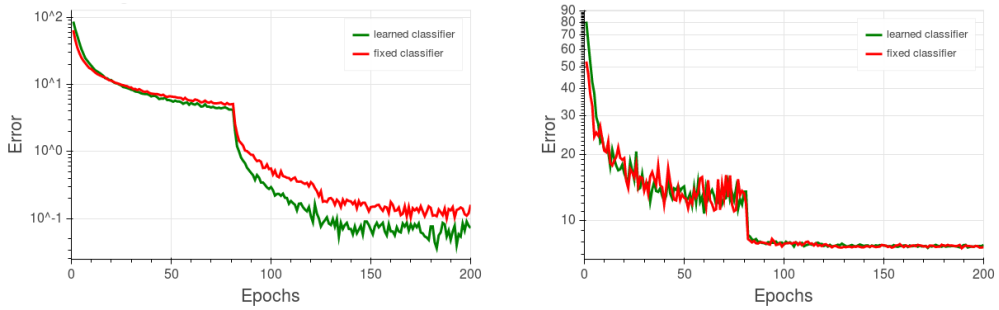
3 Experimental results

Table 1: Validation accuracy results on learned vs. fixed classifier

Network	Dataset	Learned	Fixed	# Params	% Fixed params
Resnet56 (He et al., 2016)	Cifar10	93.03%	93.14%	855,770	0.07%
DenseNet(k=12)(Huang et al., 2017)	Cifar100	77.73%	77.67%	800,032	4.2%
Resnet50 (He et al., 2016)	ImageNet	75.3%	75.3%	25,557,032	8.01%
DenseNet169(Huang et al., 2017)	ImageNet	76.2%	76%	14,149,480	11.76%
ShuffleNet(Zhang et al., 2017b)	ImageNet	65.9%	65.4%	1,826,555	52.56%

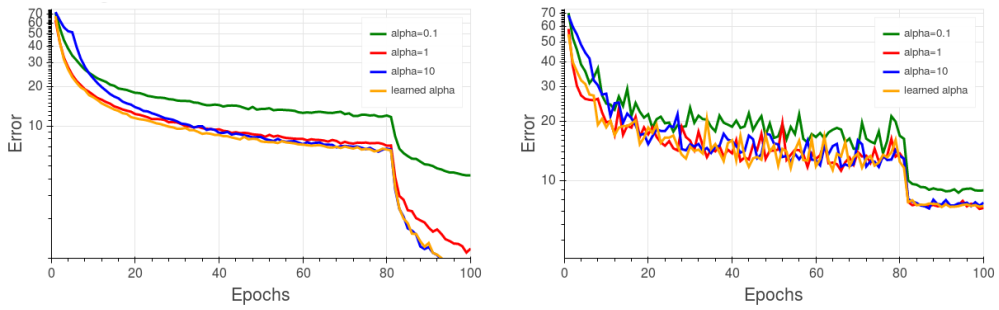
3.1 Cifar10/100

We used the well known Cifar10 and Cifar100 datasets by Krizhevsky (2009) as an initial test-bed to explore the idea of a fixed classifier. Cifar10 is an image classification benchmark dataset containing $50,000$ training images and $10,000$ test images. The images are in color and contain 32×32 pixels.



(a) Training error (b) Validation error

Figure 2: Comparing training and validation error of fixed and learned classifier (ResNet56, Cifar10)



(a) Training error (b) Validation error

Figure 3: Comparing fixed vs. trained variable scale α (ResNet56, Cifar10)

There are 10 possible classes of various animals and vehicles. Cifar100 holds the same number of images of same size, but contains 100 different classes.

We trained a residual network of He et al. (2016) on the Cifar10 dataset. We used a network of depth 56 and the same hyper-parameters used in the original work. We compared two variants: the original model with a learned classifier, and our version, where a fixed transformation is used. The results shown in figure 2 demonstrate that although the training error is considerably lower for the network with learned classifier, both models achieve the same classification accuracy on the validation set. Our conjecture is that with our new fixed parameterization, the network can no longer increase the norm of a given sample's representation - thus learning its label requires more effort. As this may happen for specific seen samples - it affects only training error.

We also compared using a fixed scale variable α at different values vs. a learned parameter. Results for $\alpha = \{0.1, 1, 10\}$ are depicted in figure 3 for both training and validation error. As can be seen, similar validation accuracy can be obtained using a fixed scale value (in this case $\alpha = 1$ or 10 will suffice) at the expense of another hyper-parameter to seek. In all our experiments we opted to train this parameter instead. In all experiments the α scale parameter was regularized with the same weight decay coefficient used on original classifier.

We then followed to train a model on the Cifar100 dataset. We used the DenseNet-BC model of Huang et al. (2017) with depth of 100 layers and $k = 12$. We continued to train according to the original regime and setting described for this network and dataset. Naturally, the higher number of classes caused the number of parameters to grow and encompass about 4% of the whole model. Validation accuracy for the fixed-classifier model remained equally good as the original model, and we continued to observe the same training curve.

3.2 Imagenet

In order to validate our results on a more challenging dataset, we used the Imagenet dataset introduced by Deng et al. (2009). The Imagenet dataset spans over 1000 visual classes, and over 1.2

million samples. CNNs used to classify Imagenet such as Krizhevsky et al. (2012), He et al. (2016), Szegedy et al. (2016) usually have a hidden representation leading to the final classifier of at least 1024 dimensions. This architectural choice, together with the large number of classes, causes the size of classifier to exceed millions of parameters and taking a sizable share from the entire model size.

We evaluated our fixed classifier method on Imagenet using Resnet50 by He et al. (2016) with the same training regime and hyper-parameters. By using a fixed classifier, approximately 2-million parameters were removed from the model, accounting for about 8% of the model parameters. Following the same procedure, we trained a Densenet169 model (Huang et al., 2017) for which a fixed classifier reduced about 12% of the parameters. Similarly to results on Cifar10 dataset, we observed the same convergence speed and approximately the same final accuracy on both the validation and training sets.

Furthermore, we were interested in evaluating more challenging models where the classifier parameters constitutes the majority amount. For this reason we chose the Shufflenet architecture (Zhang et al., 2017b), which was designed to be used in low memory and limited computing platforms. The Shufflenet network contains about 1.8 million parameters, out of which 0.96 million are part of the final classifier. Fixing the classifier resulted with a model with only 0.86 million parameters. This model was trained and found, again, to converge to similar validation accuracy as the original.

Interestingly, this method allowed Imagenet training in an under-specified regime, where there are more training samples than number of parameters. This is an unconventional regime for modern deep networks, which are usually over-specified to have many more parameters than training samples (Zhang et al., 2017a). Moreover, many recent theoretical results related to neural network training (Soudry & Hoffer, 2017; Xie et al., 2016; Safran & Shamir, 2016; Soltanolkotabi et al., 2017; Soudry & Carmon, 2016) and even generalization (Gunasekar et al., 2017; Advani & Saxe, 2017; Wilson et al., 2017) usually assume over-specification.

Table 1 summarizes our fixed-classifier results on convolutional networks, comparing to originally reported results. We offer our drop-in replacement for learned classifier that can be used to train models with fixed classifiers and replicate our results[1].

3.3 Language modeling

As language modeling requires classification of all possible tokens available in the task vocabulary, we were interested to see if a fixed classifier can be used, possible saving a very large number of trainable parameters (vocabulary size can have tens or even hundreds of thousands of different words). Recent works have already found empirically that using the same weights for both word embedding and classifier can yield equal or better results than using a separate pair of weights (Inan et al., 2016; Press & Wolf, 2017; Vaswani et al., 2017). This is compliant with our findings that the linear classifier is largely redundant. To examine further reduction in the number of parameters, we removed both classifier and embedding weights and replaced them with a fixed transform.

We trained a language model on the WikiText2 dataset described in Merity et al. (2016), using the same setting in Merity et al. (2017). We used a recurrent model with 2-layers of LSTM (Hochreiter & Schmidhuber, 1997) and embedding + hidden size of 512. As the vocabulary of WikiText2 holds about $33K$ different words, the expected number of parameters in embedding and classifier is about 34-million. This number makes for about 89% from the $38M$ parameters used for the whole model.

We found that using a random orthogonal transform yielded poor results compared to learned embedding. We suspect that, in oppose to image classification benchmarks, the embedding layer in language models holds information of the words similarities and relations, thus requiring a fine initialization. To test our intuition, we opted to use pre-trained embeddings using word2vec algorithm by Mikolov et al. (2013) or PMI factorization as suggested by Levy & Goldberg (2014). We find that using fixed word2vec embeddings, we achieve much better results. Specifically, we use 89% less parameters than the fully learned model, and obtain only somewhat worse perplexity.

We argue that this implies a required structure in word embedding that stems from semantic relatedness between words and the natural imbalance between classes. However, we suggest that with

[1] Code is available at `https://github.com/eladhoffer/fix_your_classifier`

a much more cost effective ways to train word embeddings (e.g., Mikolov et al. (2013)), we can narrow the gap and avoid their cost when training bigger models.

Table 2: Validation perplexity results

Network	Dataset	Learned	Fixed	# Params	% Fixed params
2-layer LSTM (h=512)	WikiText-2	74.1	81.2	38,312,446	88.94%

4 Discussion

4.1 Implications to future DNN models and use cases

In the last couple of years a we observe a rapid growth in the number of classes benchmark datasets contain, for example: Cifar100 (Krizhevsky, 2009), ImageNet1K, ImageNet22k (Deng et al., 2009) and language modeling (Merity et al., 2016). Therefore the computational demands of the final classifier will increase as well and should be considered no less than the architecture chosen. We use the work by Sun et al. (2017) as our use case, which introduced JFT-300M - an internal Google dataset with over 18K different classes. Using a Resnet50 (He et al., 2016), with a 2048 sized representation, this led to a model with over 36M parameters. This means that over 60% of the model parameters reside in the final classification layer.

Sun et al. (2017) further describes the difficulty in distributing this amount of parameters between the training servers, and the need to split them between 50 sub-layers. We also note the fact that the training procedure needs to account for synchronization after each parameter update - which must incur a non-trivial overhead.

Our work can help considerably in this kind of scenario - where using a fixed classifier removes the need to do any gradient synchronization for the final layer. Furthermore, using a Hadamard matrix, we can remove the need to save the transformation altogether, and make it more efficient, allowing considerable memory and computational savings.

4.2 Possible caveats

We argue that our method works due to the ability of preceding layers in the network to learn separable representations that are easily classified even when the classifier itself is fixed. This property can be affected when the ratio between learned features and number of classes is small – that is, when $C > N$. We've been experimenting with such cases, for example Imagenet classification ($C = 1000$) using mobilenet-0.5 (Howard et al., 2017) where $N = 512$, or reduced version of ResNet (He et al., 2016) where $N = 256$. In both scenarios, our method converged similarly to a fully learned classifier reaching the same final validation accuracy. This is strengthening our finding, showing that even in cases in which $C > N$, fixed classifier can provide equally good results.

Another possible issue may appear when the possible classes are highly correlated. As a fixed orthogonal classifier does not account for this kind of correlation, it may prove hard for the network to learn in this case. This may suggest another reason for the difficulties we experienced in training a language model using an orthogonal fixed classifier, as word classes tend to have highly correlated instances.

4.3 Future work

Understanding that linear classifiers used in NN models are largely redundant allows us to consider new approaches in training and understanding these models.

Recent works (Neyshabur et al., 2017; Bartlett et al., 2017) suggested a connection between generalization capabilities of models and various norm-related quantities of their weights. Such results might be potentially simplified in our model, since we have a single scalar variable (i.e., scale), which seems to be the only relevant parameter in the model (since we normalize the last hidden layer, and fix the last weight layer).

The use of fixed classifiers might be further simplified in Binarized Neural Networks (Hubara et al., 2016a), where the activations and weights are restricted to ± 1 during propagations. In this case the norm of the last hidden layer is constant for all samples (equal to the square root of the hidden layer width). This constant can be absorbed into the scale constant α, and there is no need in a per-sample normalization as in eq. 1.

We also plan to further explore more efficient ways to learn word embedding, where similar redundancy in classifier weights may suggest simpler forms of token representations - such as low-rank or sparse versions, allowing similar benefits to the fixed transformations we suggested.

5 Conclusion

In this work we suggested removing the parameters from the classification layer used in deep neural networks. We showed empirical results suggesting that keeping the classifier fixed cause little or no decline in classification performance for common balanced datasets such as Cifar and Imagenet, while allowing a noticeable reduction in trainable parameters. We argue that fixing the last layer can reduce the computational complexity for training as well as the communication cost in distributed learning. Furthermore, using a Hadamard matrix as classifier might lead to some computational benefits when properly implemented, and save memory otherwise spent on large amount of transformation coefficients. As datasets tend to become more complex by time (e.g., Cifar100, ImageNet1K, ImageNet22k, JFT-300M, and language modeling) we believe that resource hungry affine transformation should remain fixed during training, at least partially.

We also found that new efficient methods to create pre-defined word embeddings should be explored, as they require huge amount of parameters that can possibly be avoided when learning a new task. Based on these findings, we recommend future research to focus on representations learned by the non-linear part of neural networks - up to the final classifier, as it seems to be highly redundant.

Acknowledgments

The research leading to these results has received funding from the Taub Foundation, and the European Research Council under European Unions Horizon 2020 Program, ERC Grant agreement no. 682203 SpeedInfTradeoff.

References

Madhu S Advani and Andrew M Saxe. High-dimensional dynamics of generalization error in neural networks. *arXiv preprint arXiv:1710.03667*, 2017.

Peter Bartlett, Dylan J Foster, and Matus Telgarsky. Spectrally-normalized margin bounds for neural networks. *arXiv preprint arXiv:1706.08498*, 2017.

Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. Signature verification using a" siamese" time delay neural network. In *Advances in Neural Information Processing Systems*, pp. 737–744, 1994.

Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in Neural Information Processing Systems*, pp. 3123–3131, 2015.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 248–255. IEEE, 2009.

Suriya Gunasekar, Blake Woodworth, Srinadh Bhojanapalli, Behnam Neyshabur, and Nathan Srebro. Implicit regularization in matrix factorization. *arXiv preprint arXiv:1705.09280*, 2017.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

Moritz Hardt and Tengyu Ma. Identity matters in deep learning. 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

A Hedayat, WD Wallis, et al. Hadamard matrices and their applications. *The Annals of Statistics*, 6 (6):1184–1238, 1978.

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

Elad Hoffer and Nir Ailon. Deep metric learning using triplet network. In *International Workshop on Similarity-Based Pattern Recognition*, pp. 84–92. Springer, 2015.

Elad Hoffer, Itay Hubara, and Daniel Soudry. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. 2017.

Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. Extreme learning machine: theory and applications. *Neurocomputing*, 70(1):489–501, 2006.

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *Advances in Neural Information Processing Systems 29 (NIPS'16)*, 2016a.

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *arXiv preprint arXiv:1609.07061*, 2016b.

Hakan Inan, Khashayar Khosravi, and Richard Socher. Tying word vectors and word classifiers: A loss framework for language modeling. *arXiv preprint arXiv:1611.01462*, 2016.

Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014.

Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009.

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

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Omer Levy and Yoav Goldberg. Neural word embedding as implicit matrix factorization. In *Advances in neural information processing systems*, pp. 2177–2185, 2014.

Fengfu Li, Bo Zhang, and Bin Liu. Ternary weight networks. *arXiv preprint arXiv:1605.04711*, 2016.

Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

Stephen Merity, Nitish Shirish Keskar, and Richard Socher. Regularizing and Optimizing LSTM Language Models. *arXiv preprint arXiv:1708.02182*, 2017.

Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaev, Ganesh Venkatesh, et al. Mixed precision training. *arXiv preprint arXiv:1710.03740*, 2017.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed tations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pp. 3111–3119, 2013.

Behnam Neyshabur, Srinadh Bhojanapalli, David McAllester, and Nathan Srebro. Exploring generalization in deep learning. *arXiv preprint arXiv:1706.08947*, 2017.

Jooyoung Park and Irwin W Sandberg. Universal approximation using radial-basis-function networks. *Neural computation*, 3(2):246–257, 1991.

Ofir Press and Lior Wolf. Using the output embedding to improve language models. *EACL 2017*, pp. 157, 2017.

Itay Safran and Ohad Shamir. On the quality of the initial basin in overspecified neural networks. In *International Conference on Machine Learning*, pp. 774–782, 2016.

Tim Salimans and Diederik P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems*, pp. 901–909, 2016.

Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 815–823, 2015.

Mahdi Soltanolkotabi, Adel Javanmard, and Jason D Lee. Theoretical insights into the optimization landscape of over-parameterized shallow neural networks. *arXiv preprint arXiv:1707.04926*, 2017.

Daniel Soudry and Yair Carmon. No bad local minima: Data independent training error guarantees for multilayer neural networks. *arXiv preprint arXiv:1605.08361*, 2016.

Daniel Soudry and Elad Hoffer. Exponentially vanishing sub-optimal local minima in multilayer neural networks. *arXiv preprint arXiv:1702.05777*, 2017.

Daniel Soudry, Elad Hoffer, and Nathan Srebro. The implicit bias of gradient descent on separable data. 2018.

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

Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. Revisiting unreasonable effectiveness of data in deep learning era. *arXiv preprint arXiv:1707.02968*, 2017.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2818–2826, 2016.

Cheng Tai, Tong Xiao, Yi Zhang, Xiaogang Wang, et al. Convolutional neural networks with low-rank regularization. *arXiv preprint arXiv:1511.06067*, 2015.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. 2017.

Ashia C Wilson, Rebecca Roelofs, Mitchell Stern, Nathan Srebro, and Benjamin Recht. The marginal value of adaptive gradient methods in machine learning. *arXiv preprint arXiv:1705.08292*, 2017.

Bo Xie, Yingyu Liang, and Le Song. Diversity leads to generalization in neural networks. *arXiv preprint arXiv:1611.03131*, 2016.

Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pp. 818–833. Springer, 2014.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. In *ICLR*, 2017a. URL `https://arxiv.org/abs/1611.03530`.

Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *arXiv preprint arXiv:1707.01083*, 2017b.

Shuchang Zhou, Zekun Ni, Xinyu Zhou, He Wen, Yuxin Wu, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.

Chapter 6

Norm matters: efficient and accurate normalization schemes in deep networks

Norm matters: efficient and accurate normalization schemes in deep networks

Elad Hoffer[1]*, **Ron Banner**[2]*, **Itay Golan**[1]*, **Daniel Soudry**[1]
{elad.hoffer, itaygolan, daniel.soudry}@gmail.com
{ron.banner}@intel.com

(1) Technion - Israel Institute of Technology, Haifa, Israel
(2) Intel - Artificial Intelligence Products Group (AIPG)

Abstract

Over the past few years, Batch-Normalization has been commonly used in deep networks, allowing faster training and high performance for a wide variety of applications. However, the reasons behind its merits remained unanswered, with several shortcomings that hindered its use for certain tasks. In this work, we present a novel view on the purpose and function of normalization methods and weight-decay, as tools to decouple weights' norm from the underlying optimized objective. This property highlights the connection between practices such as normalization, weight decay and learning-rate adjustments. We suggest several alternatives to the widely used L^2 batch-norm, using normalization in L^1 and L^∞ spaces that can substantially improve numerical stability in low-precision implementations as well as provide computational and memory benefits. We demonstrate that such methods enable the first batch-norm alternative to work for half-precision implementations. Finally, we suggest a modification to weight-normalization, which improves its performance on large-scale tasks. [2]

1 Introduction

Deep neural networks are known to benefit from normalization between consecutive layers. This was made noticeable with the introduction of Batch-Normalization (BN) [19], which normalizes the output of each layer to have zero mean and unit variance for each channel across the training batch. This idea was later developed to act across channels instead of the batch dimension in Layer-normalization [2] and improved in certain tasks with methods such as Batch-Renormalization [18], Instance-normalization [32] and Group-Normalization [37]. In addition, normalization methods are also applied to the layer parameters instead of their outputs. Methods such as Weight-Normalization [26], and Normalization-Propagation [1] targeted the layer weights by normalizing their per-channel norm to have a fixed value.

1.1 Issues with current normalization methods

Batch-normalization, despite its merits, suffers from several issues, as pointed out by previous work [26, 18, 1]. These issues are not yet solved in current normalization methods.

Interplay with other regularization mechanisms. Batch normalization typically improves generalization performance and is therefore considered a regularization mechanism. Other regularization

*Equal contribution
[2]Source code is available at `https://github.com/eladhoffer/norm_matters`

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

mechanisms are typically used in conjunction. For example, weight decay, also known as L^2 regularization, is a common method which adds a penalty proportional to the weights' norm. Weight decay was proven to improve generalization in various problems [23, 5, 4], but, so far, not for non-linear deep neural networks. There, [39] performed an extensive set of experiments on regularization and concluded that explicit regularization, such as weight decay, may improve generalization performance, but is neither necessary nor, by itself, sufficient for reducing generalization error. Therefore, it is not clear how weight decay interacts with BN, or if weight decay is even really necessary given that batch norm already constrains the output norms [16]).

Task-specific limitations. A key assumption in BN is the independence between samples appearing in each batch. While this assumption seems to hold for most convolutional networks used to classify images in conventional datasets, it falls short when employed in domains with strong correlations between samples, such as time-series prediction, reinforcement learning, and generative modeling. For example, BN requires modifications to work in recurrent networks [6], for which alternatives such as weight-normalization [26] and layer-normalization [2] were explicitly devised, without reaching the success and wide adoption of BN. Another example is Generative adversarial networks, which are also noted to suffer from the common form of BN. GAN training with BN proved unstable in some cases, decreasing the quality of the trained model [27]. Instead, it was replaced with virtual-BN [27], weight-norm [38] and spectral normalization [31]. Also, BN may be harmful even in plain classification tasks, when using unbalanced classes, or correlated instances. In addition, while BN is defined for the training phase of the models, it requires a running estimate for the evaluation phase – causing a noticeable difference between the two [19]. This shortcoming was addressed later by batch-renormalization [18], yet still requiring the original BN at the early steps of training.

Computational costs. From the computational perspective, BN is significant in modern neural networks, as it requires several floating point operations across the activation of the entire batch for every layer in the network. Previous analysis by Gitman & Ginsburg [11] measured BN to constitute up to 24% of the computation time needed for the entire model. It is also not easily parallelized, as it is usually memory-bound on currently employed hardware. In addition, the operation requires saving the pre-normalized activations for back-propagation in the general case [25], thus using roughly twice the memory as a non-BN network in the training phase. Other methods, such as Weight-Normalization [26] have a much smaller computational cost but typically achieve significantly lower accuracy when used in large-scale tasks such as ImageNet [11].

Numerical precision. As the use of deep learning continues to evolve, the interest in low-precision training and inference increases [17, 35]. Optimized hardware was designed to leverage benefits of low-precision arithmetic and memory operations, with the promise of better, more efficient implementations [21]. Although most mathematical operations employed in neural-networks are known to be robust to low-precision and quantized values, the current normalization methods are notably not suited for these cases. As far as we know, this has remained an unanswered issue, with no suggested alternatives. Specifically, all normalization methods, including BN, use an L^2 normalization (variance computation) to control the activation scale for each layer. The operation requires a sum of power-of-two floating point variables, a square-root function, and a reciprocal operation. All of these require both high-precision to avoid zero variance, and a large range to avoid overflow when adding large numbers. This makes BN an operation that is not easily adapted to low-precision implementations. Using norm spaces other than L^2 can alleviate these problems, as we shall see later.

1.2 Contributions

In this paper we make the following contributions, to address the issues explained in the previous section:

- We find the mechanism through which weight decay before BN affects learning dynamics: we demonstrate that by adjusting the learning rate or normalization method we can exactly mimic the effect of weight decay on the learning dynamics. We suggest this happens since certain normalization methods, such as a BN, disentangle the effect of weight vector norm on the following activation layers.

- We show that we can replace the standard L^2 BN with certain L^1 and L^∞ based variations of BN, which do not harm accuracy (on CIFAR and ImageNet) and even somewhat improve training speed. Importantly, we demonstrate that such norms can work well with low precision (16bit), while L^2 does not. Notably, for these normalization schemes to work well, precise scale adjustment is required, which can be approximated analytically.
- We show that by bounding the norm in a weight-normalization scheme, we can significantly improve its performance in convnets (on ImageNet), and improve baseline performance in LSTMs (on WMT14 de-en). This method can alleviate several task-specific limitations of BN, and reduce its computational and memory costs (e.g., allowing to work with significantly larger batch sizes). Importantly, for the method to work well, we need to carefully choose the scale of the weights using the scale of the initialization.

Together, these findings emphasize that the learning dynamics in neural networks are very sensitive to the norms of the weights. Therefore, it is an important goal for future research to search for precise and theoretically justifiable methods to adjust the scale for these norms.

2 Consequences of the scale invariance of Batch-Normalization

When BN is applied after a linear layer, it is well known that the output is invariant to the channel weight vector norm. Specifically, denoting a channel weight vector with w and $\hat{w} = w/\|w\|_2$, channel input as x and BN for batch-norm, we have

$$BN(\|w\|_2 \hat{w}x) = BN(\hat{w}x). \tag{1}$$

This invariance to the weight vector norm means that a BN applied after a layer renders its norm irrelevant to the inputs of consecutive layers. The same can be easily shown for the per-channel weights of a convolutional layer. The gradient in such case is scaled by $1/\|w\|_2$:

$$\frac{\partial \text{BN}(\|w\|_2 \hat{w}x)}{\partial(\|w\|_2 \hat{w})} = \frac{1}{\|w\|_2} \frac{\partial \text{BN}(\hat{w}x)}{\partial(\hat{w})}. \tag{2}$$

When a layer is rescaling invariant, the key feature of the weight vector is its direction.

During training, the weights are typically incremented through some variant of stochastic gradient descent, according to the gradient of the loss at mini-batch t, with learning rate η

$$w_{t+1} = w_t - \eta \nabla L_t(w_t) \ . \tag{3}$$

Claim. During training, the weight direction $\hat{w}_t = w_t/\|w_t\|_2$ is updated according to

$$\hat{w}_{t+1} = \hat{w}_t - \eta \|w_t\|^{-2} \left(I - \hat{w}_t \hat{w}_t^\top\right) \nabla L(\hat{w}_t) + O\left(\eta^2\right)$$

Proof. Denote $\rho_t = \|w_t\|_2$. Note that, from eqs. 2 and 3 we have

$$\rho_{t+1}^2 = \rho_t^2 - 2\eta \hat{w}_t^\top \nabla L(\hat{w}_t) + \eta^2 \rho_t^{-2} \|\nabla L(\hat{w}_t)\|^2$$

and therefore

$$\begin{aligned}\rho_{t+1} &= \rho_t \sqrt{1 - 2\eta\rho_t^{-2} \hat{w}_t^\top \nabla L(\hat{w}_t) + \eta^2 \rho_t^{-4} \|\nabla L(\hat{w}_t)\|^2} \\ &= \rho_t - \eta \rho_t^{-1} \hat{w}_t^\top \nabla L(\hat{w}_t) + O\left(\eta^2\right) \ .\end{aligned}$$

Additionally, from eq. 3 we have

$$\rho_{t+1}\hat{w}_{t+1} = \rho_t \hat{w}_t - \eta \nabla L(\hat{w}_t \rho_t)$$

and therefore, from eq. 2,

$$\begin{aligned}\hat{w}_{t+1} &= \frac{\rho_t}{\rho_{t+1}} \hat{w}_t - \eta \frac{1}{\rho_{t+1}\rho_t} \nabla L(\hat{w}_t) \\ &= \left(1 + \eta\rho_t^{-2} \hat{w}_t^\top \nabla L(\hat{w}_t)\right) \hat{w}_t - \eta\rho_t^{-2} \nabla L(\hat{w}_t) + O\left(\eta^2\right) \\ &= \hat{w}_t - \eta\rho_t^{-2} \left(I - \hat{w}_t \hat{w}_t^\top\right) \nabla L(\hat{w}_t) + O\left(\eta^2\right) \ ,\end{aligned}$$

which proves the claim. □

Therefore, the step size of the weight direction is approximately proportional to

$$\hat{w}_{t+1} - \hat{w}_t \propto \frac{\eta}{\|w_t\|_2^2}. \tag{4}$$

in the case of linear layer followed by BN, and for small learning rate η. Note that a similar conclusion was reached by van Laarhoven [33], who implicitly assumed $||w_{t+1}|| = ||w_t||$, though this is only approximately true. Here we show this conclusion is still true without such an assumption. This analysis continues to hold for non-linear functions that do not affect scale, such as the commonly used ReLU function. In addition, although stated for the case of vanilla SGD, similar argument can be made for adaptive methods such as Adagrad [9] or Adam [20].

3 Connection between weight-decay, learning rate and normalization

We claim that when using batch-norm (BN), weight decay (WD) improves optimization only by fixing the norm to a small range of values, leading to a more stable step size for the weight direction ("effective step size"). Fixing the norm allows better control over the effective step size through the learning rate η. Without WD, the norm grows unbounded [30], resulting in a decreased effective step size, although the learning rate hyper-parameter remains unchanged.

We show empirically that the accuracy gained by using WD can be achieved without it, only by adjusting the learning rate. Given statistics on norms of each channel from a training with WD and BN, similar results can be achieved without WD by mimicking the effective step size using the following correction on the learning rate:

$$\hat{\eta}_{\text{Correction}} = \eta \frac{\|w\|_2^2}{\|w_{[\text{WD on}]}\|_2^2} \tag{5}$$

where w is the weights' vector of a single channel, and $w_{[\text{WD on}]}$ is the weights' vector of the corresponding channel in a training with WD. This correction requires access to the norms of a training with WD, hence it is not a practical method to replace WD but just a tool to demonstrate our claim on the connection between weights' norm, WD and step size.

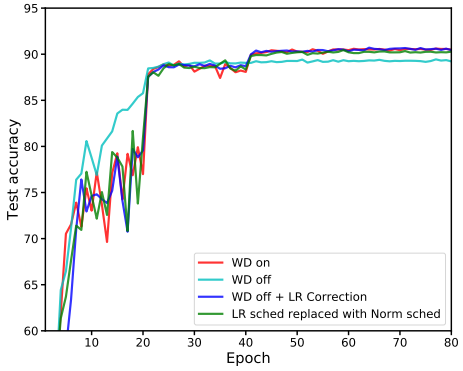


Figure 1: The connection between norm, effective step size and weight decay. *WD on*/*WD off* was trained with/without weight decay respectively. *WD off correction* was trained without weight decay but with LR correction as presented in Eq. 5. *LR sched replaced with Norm sched* is based on *WD on* norms but replacing LR scheduling with norm scheduling. (VGG11, CIFAR-10)

We conducted multiple experiments on CIFAR-10 [22] to show this connection. Figure 1 reports the test accuracy during the training of all experiments. We were able to show that WD results can be mimicked with step size adjustments using the correction formula from Eq. 5. In another experiment, we replaced the learning rate scheduling with norm scheduling. To do so, after every gradient descent step we normalized the norm of each convolution layer channel to be the same as the norm of the corresponding channel in training with WD and keep the learning rate constant. When learning rate is multiplied by 0.1 in the WD training, we instead multiply the norm by $\sqrt{10}$, leading to an effective step size of $\frac{\eta}{\|W_{\text{WD on}}\|_2^2\sqrt{10^2}} = 0.1\frac{\eta}{\|W_{\text{WD on}}\|_2^2}$. As expected, when applying the correction on step-size or replacing learning rate scheduling with norm scheduling, the accuracy is similar to the training with WD throughout the learning process, suggesting that WD affects the training process only indirectly, by modulating the learning rate. Implementation details appear in supplementary material.

4 Alternative L^p metrics for batch norm

We suggested above that the main function of BN is to neutralize the effect of the preceding layer's weights. If this hypothesis is true, then other operations might be able to replace BN, as long as they remain similarly scale invariant (as in eq. (1)) — and if we keep the same scale as BN. Following this reasoning, we next aim to replace the use of L^2 norm with scale-invariant alternatives which are more appealing computationally and for low-precision implementations.

Batch normalization aims at regularizing the input so that sum of deviations from the mean would be standardized according to the Euclidean L^2 norm metric. For a layer with $d-$dimensional input $x = (x^{(1)}, x^{(2)}, ..., x^{(d)})$, L^2 batch norm normalizes each dimension

$$\hat{x}^{(k)} = \frac{x^{(k)} - \mu^k}{\sqrt{\mathrm{Var}[x^{(k)}]}}\,, \tag{6}$$

where μ^k is the expectation over $x^{(k)}$, n is the batch size and $\mathrm{Var}[x^{(k)}] = \frac{1}{n}||x^{(k)} - \mu^k||_2^2$.

The computation toll induced by $\sqrt{\mathrm{Var}[x^{(k)}]}$ is often significant with non-negligible overheads on memory and energy consumption. In addition, as the above variance computation involves sums of squares, the quantization of the L_2 batch norm for training on optimized hardware can lead to numerical instability as well as to arithmetic overflows when dealing with large values.

In this section, we suggest alternative L^p metrics for BN. We focus on the L^1 and L^∞ due to their appealing speed and memory computations. In our simulations, we were able to train models faster and with fewer GPUs using the above normalizations. Strikingly, by proper adjustments of these normalizations, we were able to train various complicated models without hurting the classification performance. We begin with the L^1-norm metric.

4.1 L^1 batch norm.

For a layer with $d-$dimensional input $x = (x^{(1)}, x^{(2)}, ..., x^{(d)})$, L^1 batch normalization normalize each dimension

$$\hat{x}^{(k)} = \frac{x^{(k)} - \mu^k}{C_{L_1} \cdot ||x^{(k)} - \mu^k||_1 / n} \tag{7}$$

where μ^k is the expectation over $x^{(k)}$, n is the batch size and $C_{L_1} = \sqrt{\pi/2}$ is a normalization term.

Unlike traditional L^2 batch normalization that computes the *average squared deviation* from the mean (variance), L^1 batch normalization computes only the *average absolute deviation* from the mean. This has two major advantages. First, L^1 batch normalization eliminates the computational efforts required for the square and square root operations. Second, as the square of an n-bit number is generally of $2n$ bits, the absence of these square computations makes it much more suitable for low-precision training that has been recognized to drastically reduce memory size and power consumption on dedicated deep learning hardware [7].

As can be seen in equation 7, the L^1 batch normalization quantifies the variability with the normalized average absolute deviation $C_{L_1} \cdot ||x^{(k)} - \mu^k||_1 / n$. To calculate an appropriate value for the constant C_{L_1}, we assume the input $x^{(k)}$ follows Gaussian distribution $N(\mu^k, \sigma^2)$. This is a common approximation (e.g., Soudry et al. [29]), based on the fact that the neural input $x^{(k)}$ is a sum of many inputs, so we expect it to be approximately Gaussian from the central limit theorem. In this case, $\hat{x}^{(k)} = (x^{(k)} - \mu^k)$ follows the distribution $N(0, \sigma^2)$. Therefore, for each example $\hat{x}_i^{(k)} \in \hat{x}^{(k)}$ it holds that $|\hat{x}_i^{(k)}|$ follows a half-normal distribution with expectation $E(|\hat{x}_i^{(k)}|) = \sigma \cdot \sqrt{2/\pi}$. Accordingly, the expected L^1 variability measure is related to the traditional standard deviation measure σ normally used with batch normalization as follows:

$$E\left[\frac{C_{L_1}}{n} \cdot ||x^{(k)} - \mu^k||_1\right] = \frac{\sqrt{\pi/2}}{n} \cdot \sum_{i=1}^{n} E[|\hat{x}_i^{(k)}|] = \sigma.$$

Figure 2 presents the validation accuracy of ResNet-18 and ResNet-50 on ImageNet using L^1 and L^2 batch norms. While the use of L_1 batch norm is more efficient in terms of resource usage,

power, and speed, they both share the same classification accuracy. We additionally verified L^1 layer-normalization to work on Transformer architecture [34]. Using an L^1 layer-norm we achieved a final perplexity of 5.2 vs. 5.1 for original L^2 layer-norm using the base model on the WMT14 dataset.

We note the importance of C_{L_1} to the performance of L^1 normalization method. For example, using C_{L_1} helps the network to reach 20% validation error more than twice faster than an equivalent configuration without this normalization term. With C_{L_1} the network converges at the same rate and to the same accuracy as L^2 batch norm. It is somewhat surprising that this constant can have such an impact on performance, considering the fact that it is so close to one ($C_{L_1} = \sqrt{\pi/2} \approx 1.25$). A demonstration of this effect can be found in the supplementary material (Figure 1).

We also note that the use of L^1 norm improved both running time and memory consumption for models we tested. These benefits can be attributed to the fact that absolute-value operation is computationally more efficient compared to the costly square and sqrt operations. Additionally, the derivative of $|x|$ is the operation $sign(x)$. Therefore, in order to compute the gradients, we only need to cache the sign of the values (not the actual values), allowing for substantial memory savings.

4.2 L^∞ batch norm

Another alternative measure for variability that avoids the discussed limitations of the traditional L^2 batch norm is the *maximum absolute deviation*. For a layer with $d-$dimensional input $x = (x^{(1)}, x^{(2)}, ..., x^{(d)})$, L^∞ batch normalization normalize each dimension

$$\hat{x}^{(k)} = \frac{x^{(k)} - \mu^k}{C_{L_\infty}(n) \cdot ||x^{(k)} - \mu^k||_\infty}, \tag{8}$$

where μ^k is the expectation over $x^{(k)}$, n is batch size and $C_{L_\infty}(n)$ is computed similarly to $C_{L_1}(n)$ (derivation appears in appendix).

While normalizing according to the maximum absolute deviation offers a major performance advantage, we found it somewhat less robust to noise compared to L^1 and L^2 normalization.

By replacing the maximum absolute deviation with the mean of ten largest deviations, we were able to make normalization much more robust to outliers. Formally, let s_n be the n-th largest value in S, we define $\mathrm{Top}(k)$ as follows

$$\mathrm{Top}(k) = \frac{1}{k} \sum_{n=1}^{k} |s_n|$$

Given a batch of size n, the notion of $\mathrm{Top}(k)$ generalizes L^1 and L^∞ metrics. Indeed, L^∞ is precisely $\mathrm{Top}(1)$ while L^1 is by definition equivalent to $\mathrm{Top}(n)$. As we could not find a closed-form expression for the normalization term $C_{\mathrm{Top}K}(n)$, we approximated it as a linear interpolation between C_{L_1} and $C_{L_\infty}(n)$. As can be seen in figure 2, the use of $\mathrm{Top}(10)$ was sufficient to close the gap to L_2 performance. For further details on $\mathrm{Top}(10)$ implementation, see our code.

4.3 Batch norm at half precision

Due to numerical issues, prior attempts to train neural networks at low precision had to leave batch norm operations at full precision (float 32) as described by Micikevicius et al. [24], Das et al. [8], thus enabling only *mixed* precision training. This effectively means that low precision hardware still needs to support full precision data types. The sensitivity of BN to low precision operations can be attributed to both the numerical operations of square and square-root used, as well as the possible overflow of the sum of many large positive values. To overcome this overflow, we may further require a wide accumulator with full precision.

We provide evidence that by using L^1 arithmetic, batch normalization can also be quantized to half precision with no apparent effect on validation accuracy, as can be seen in figure 3. Using the standard L^2 BN in low-precision leads to overflow and significant quantization noise that quickly deteriorate the whole training process, while L^1 BN allows training with no visible loss of accuracy.

As far as we know, our work is the first to demonstrate a viable alternative to BN in half-precision accuracy. We also note that the usage of L^∞ BN or its $\mathrm{Top}(k)$ relaxation, may further help low-

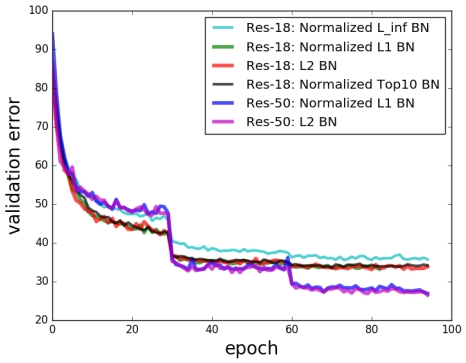


Figure 2: Classification error with L^2 batch norm (baseline) and L^1, L^∞ and $\mathrm{Top}(10)$ alternatives for ResNet-18 and ResNet-50 on ImageNet. Compared to the baselines, L^1 and $\mathrm{Top}(10)$ normalizations reached similar final accuracy (difference < 0.2%), while L^∞ had a lower accuracy, by 3%.

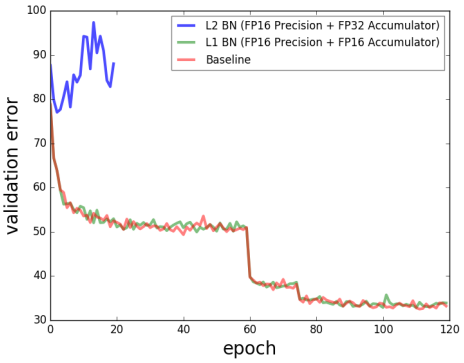


Figure 3: L^1 BN is more robust to quantization noise compared to L^2 BN as demonstrated for ResNet18 on ImageNet. The half precision run of L^2 BN was clearly diverging, even when done with a high precision accumulator, and we stopped the run before termination at epoch 20.

precision implementations by significantly lowering the extent of the reduction operation (as only k numbers need to be summed).

5 Improving weight normalization

5.1 The advantages and disadvantages of weight normalization

Trying to address several of the limitations of BN, Salimans & Kingma [26] suggested weight normalization as its replacement. As weight-norm requires an L^2 normalization over the output channels of the *weight* matrix, it alleviates both computational and task-specific shortcomings of BN, ensuring no dependency on the current batch of sample activations within a layer.

While this alternative works well for small-scale problems, as demonstrated in the original work, it was noted by Gitman & Ginsburg [11] to fall short in large-scale usage. For example, in the ImageNet classification task, weight-norm exhibited unstable convergence and significantly lower performance (67% accuracy on ResNet50 vs. 75% for original).

An additional modification of weight-norm called "normalization propagation" [1] adds additional multiplicative and additive corrections to address the change of activation distribution introduced by the ReLU non-linearity used between layers in the network. These modifications are not trivially applied to architectures with complex structure elements such as residual connections [14].

So far, we've demonstrated that the key to the performance of normalization techniques lies in their property to neutralize the effect of weight's norm. Next, we will use this reasoning to overcome the shortcoming of weight-norm.

5.2 Norm bounded weight-normalization

We return to the original parametrization suggested for weight norm, for a given initialized weight matrix V with N output channels:

$$w_i = g_i \frac{v_i}{\|v_i\|_2},$$

where w_i is a parameterized weight for the ith output channel, composed from an L^2 normalized vector v_i and scalar g_i.

Weight-norm successfully normalized each output channel's weights to reside on the L^2 sphere. However, it allowed the weights scale to change freely through the scalar g_i. Following reasoning presented earlier in this work, we wish to make the weight's norm completely disjoint from its values.

We can achieve this by keeping the norm fixed as follows:

$$w_i = \rho \frac{v_i}{\|v_i\|_2},$$

where ρ is a fixed scalar for each layer that is determined by its size (number of input and output channels). A simple choice for ρ is by the initial norm of the weights, e.g $\rho = \|V\|_F^{(t=0)}/\sqrt{N}$, thus employing the various successful heuristics used to initialize modern networks [12, 13]. We also note that when using non-linearity with no scale sensitivity (e.g ReLU), these ρ constants can be instead incorporated into only the final classifier's weights and biases throughout the network.

Previous works demonstrated that weight-normalized networks converge faster when augmented with mean only batch normalization. We follow this regime, although noting that similar final accuracy can be achieved without mean normalization but at the cost of slower convergence, or with the use of zero-mean preserving activation functions [10].

After this modification, we now find that weight-norm can be improved substantially, solving the stability issues for large-scale task observed by Gitman & Ginsburg [11] and achieving comparable accuracy (although still behind BN). Results on Imagenet using Resnet50 are described in Figure 4, using the original settings and training regime [14]. We believe the still apparent margin between the two methods can be further decreased using hyper-parameter tuning, such as a modified learning rate schedule. It is also interesting to observe BWN's effect in recurrent networks, where BN is not easily applicable [6]. We compare weight-norm vs. the common implementation (with layer-norm) of an attention-based LSTM model on the WMT14 en-de translation task [3]. The model consists of 2 LSTM cells for both encoder and decoder, with an attention mechanism. We also compared BWN on the Transformer architecture [34] to replace layer-norm, again achieving comparable final performance (26.5 vs. 27.3 BLEU score on the original base model). Both sequence-to-sequence models were tested using beam-search decoding with a beam size of 4 and length penalty of 0.6. Additional results for BWN can be found in the supplementary material (Figure 2 and Table 1).

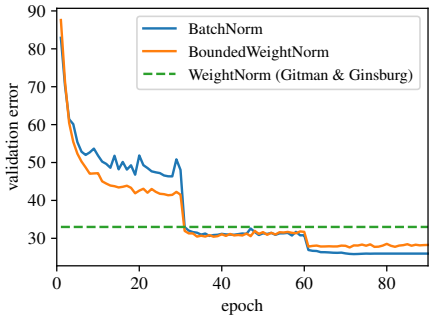


Figure 4: Comparison between batch-norm (BN), weight-norm (WN) and bounded-weight-norm (WN) on ResNet50, ImageNet. For weight-norm, we show the final results from [11]. Our implementation of WN here could not converge (similar convergence issues were reported by [11]). Final accuracy: BN - 75.3%, WN 67%, and BWN - 73.8%.

5.3 L^p weight normalization

As we did for BN, we can consider weight-normalization over norms other than L^2 such that

$$w_i = \rho \frac{v_i}{\|v_i\|_p}, \quad \rho = \|V\|_p^{(t=0)}/N^{1/p},$$

where computing the constant ρ over desired (vector) norm will ensure proper scaling that was required in the BN case. We find that similarly to BN, the L^1 norm can serve as an alternative to original L^2 weight-norm, where using L^∞ cause a noticeable degradation when using its proper form (top-1 absolute maximum).

6 Discussion

In this work, we analyzed common normalization techniques used in deep learning models, with BN as their prime representative. We considered a novel perspective on the role of these methods, as tools to decouple the weights' norm from training objective. This perspective allowed us to re-evaluate the necessity of regularization methods such as weight decay, and to suggest new methods for normalization, targeting the computational, numerical and task-specific deficiencies of current techniques.

Specifically, we showed that the use of L^1 and L^∞-based normalization schemes could provide similar results to the standard BN while allowing low-precision computation. Such methods can

be easily implemented and deployed to serve in current and future network architectures, low-precision devices. A similar L^1 normalization scheme to ours was recently introduced by Wu et al. [36], appearing in parallel to us (within a week). In contrast to Wu et al. [36], we found that the C_{L_1} normalization constant is crucial for achieving the same performance as $L2$ (see Figure 1 in supplementary). We additionally demonstrated the benefits of L^1 normalization: it allowed us to perform BN in half-precision floating-point, which was noted to fail in previous works [24, 8] and required full and mixed precision hardware.

Moreover, we suggested a bounded weight normalization method, which achieves improved results on large-scale tasks (ImageNet) and is nearly comparable with BN. Such a weight normalization scheme improves computational costs and can enable improved learning in tasks that were not suited for previous methods such as reinforcement-learning and temporal modeling.

We further suggest that insights gained from our findings can have an additional impact on the way neural networks are devised and trained. As previous works demonstrated, a strong connection appears between the batch-size used and the optimal learning rate regime [15, 28] and between the weight-decay factor and learning-rate [33]. We deepen this connection and suggest that all of these factors, including the effective norm (or temperature), are mutually affecting one another. It is plausible, given our results, that some (or all) of these hyper-parameters can be fixed given another, which can potentially ease the design and training of modern models.

Acknowledgments

This research was supported by the Israel Science Foundation (grant No. 31/1031), and by the Taub foundation. A Titan Xp used for this research was donated by the NVIDIA Corporation.

References

[1] Arpit, D., Zhou, Y., Kota, B., and Govindaraju, V. Normalization propagation: A parametric technique for removing internal covariate shift in deep networks. In *International Conference on Machine Learning*, pp. 1168–1176, 2016.

[2] Ba, J. L., Kiros, J. R., and Hinton, G. E. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

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

[4] Bös, S. Optimal weight decay in a perceptron. In *International Conference on Artificial Neural Networks*, pp. 551–556. Springer, 1996.

[5] Bos, S. and Chug, E. Using weight decay to optimize the generalization ability of a perceptron. In *Neural Networks, 1996., IEEE International Conference on*, volume 1, pp. 241–246. IEEE, 1996.

[6] Cooijmans, T., Ballas, N., Laurent, C., Gülçehre, Ç., and Courville, A. Recurrent batch normalization. *arXiv preprint arXiv:1603.09025*, 2016.

[7] Courbariaux, M., Bengio, Y., and David, J.-P. Training deep neural networks with low precision multiplications. *arXiv preprint arXiv:1412.7024*, 2014.

[8] Das, D., Mellempudi, N., Mudigere, D., et al. Mixed precision training of convolutional neural networks using integer operations. *arXiv preprint arXiv:1802.00930*, 2018.

[9] Duchi, J., Hazan, E., and Singer, Y. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[10] Eidnes, L. and Nøkland, A. Shifting mean activation towards zero with bipolar activation functions. *arXiv preprint arXiv:1709.04054*, 2017.

[11] Gitman, I. and Ginsburg, B. Comparison of batch normalization and weight normalization algorithms for the large-scale image classification. *CoRR*, abs/1709.08145, 2017.

[12] Glorot, X. and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pp. 249–256, 2010.

[13] He, K., Zhang, X., Ren, S., and Sun, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

[14] He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[15] Hoffer, E., Hubara, I., and Soudry, D. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. In *Advances in Neural Information Processing Systems*, pp. 1729–1739, 2017.

[16] Huang, L., Liu, X., Lang, B., and Li, B. Projection based weight normalization for deep neural networks. *arXiv preprint arXiv:1710.02338*, 2017.

[17] Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., and Bengio, Y. Binarized neural networks. In *Advances in neural information processing systems*, pp. 4107–4115, 2016.

[18] Ioffe, S. Batch renormalization: Towards reducing minibatch dependence in batch-normalized models. In *Advances in Neural Information Processing Systems*, pp. 1942–1950, 2017.

[19] Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pp. 448–456, 2015.

[20] Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[21] Köster, U., Webb, T., Wang, X., et al. Flexpoint: An adaptive numerical format for efficient training of deep neural networks. In *Advances in Neural Information Processing Systems*, pp. 1740–1750, 2017.

[22] Krizhevsky, A. and Hinton, G. Learning multiple layers of features from tiny images. 2009.

[23] Krogh, A. and Hertz, J. A. A simple weight decay can improve generalization. In *Advances in neural information processing systems*, pp. 950–957, 1992.

[24] Micikevicius, P., Narang, S., Alben, J., et al. Mixed precision training. In *International Conference on Learning Representations*, 2018.

[25] Rota Bulò, S., Porzi, L., and Kontschieder, P. In-place activated batchnorm for memory-optimized training of dnns. *arXiv preprint arXiv:1712.02616*, 2017.

[26] Salimans, T. and Kingma, D. P. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems*, pp. 901–909, 2016.

[27] Salimans, T., Goodfellow, I., Zaremba, W., et al. Improved techniques for training gans. In *Advances in Neural Information Processing Systems*, pp. 2234–2242, 2016.

[28] Smith, S. L., Kindermans, P.-J., and Le, Q. V. Don't decay the learning rate, increase the batch size. *arXiv preprint arXiv:1711.00489*, 2017.

[29] Soudry, D., Hubara, I., and Meir, R. Expectation backpropagation: parameter-free training of multilayer neural networks with continuous or discrete weights. In *Neural Information Processing Systems*, volume 2, pp. 963–971, dec 2014.

[30] Soudry, D., Hoffer, E., and Srebro, N. The implicit bias of gradient descent on separable data. *International Conference on Learning Representations*, 2018.

[31] Takeru Miyato, M. K. Y. Y., Toshiki Kataoka. Spectral normalization for generative adversarial networks. *International Conference on Learning Representations*, 2018.

[32] Ulyanov, D., Vedaldi, A., and Lempitsky, V. S. Instance normalization: The missing ingredient for fast stylization. *CoRR*, abs/1607.08022, 2016.

[33] van Laarhoven, T. L2 regularization versus batch and weight normalization. *arXiv preprint arXiv:1706.05350*, 2017.

[34] Vaswani, A., Shazeer, N., Parmar, N., et al. Attention is all you need. In *Advances in Neural Information Processing Systems*, pp. 6000–6010, 2017.

[35] Venkatesh, G., Nurvitadhi, E., and Marr, D. Accelerating deep convolutional networks using low-precision and sparsity. In *Acoustics, Speech and Signal Processing (ICASSP), 2017 IEEE International Conference on*, pp. 2861–2865. IEEE, 2017.

[36] Wu, S., Li, G., Deng, L., et al. L1-Norm Batch Normalization for Efficient Training of Deep Neural Networks. *ArXiv e-prints*, February 2018.

[37] Wu, Y. and He, K. Group normalization. *arXiv preprint arXiv:1803.08494*, 2018.

[38] Xiang, S. and Li, H. On the effect of batch normalization and weight normalization in generative adversarial networks. *arXiv preprint arXiv:1704.03971*, 2017.

[39] Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.

Norm matters: supplementary material

1 Implementation Details for weight-decay experiments

For all experiments, we used weight decay on the last layer with $\lambda = 0.0005$. The network architecture was VGG11 [4] with batch-norm after every convolution layer. Learning rate started from 0.1 and divided by 10 every 20 epochs (except for the norm scheduling experiment). The same random seed was used.

2 Importance of normalization constants

Figure 1 shows the scale adjustment C_{L_1} is essential even for relatively "easy" data sets with small images such as CIFAR-10, and the use of smaller/bigger adjustments degrade classification accuracy.

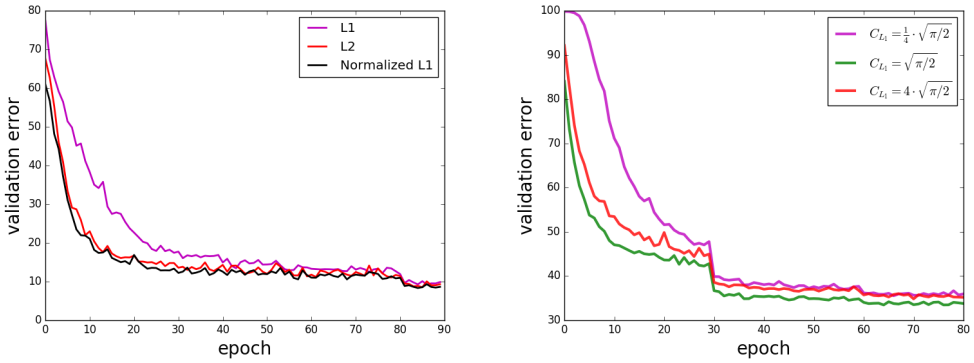


Figure 1: *Left*: The importance of normalization term C_{L_1} while training ResNet-56 on CIFAR-10. Without the use of C_{L_1} the network convergence is slower and reaches a higher final validation error. We found it somewhat surprising that a constant so close to one ($C_{L_1} = \sqrt{\pi/2} \approx 1.25$) can have such an impact on performance. *Right*: We further demonstrate, with Res18 on ImageNet, that $C_{L_1} = \sqrt{\pi/2}$ is optimal: performance is only degraded if we modify C_{L_1} to other nearby values.

2.1 Deriving C_{L_∞}

As seen in main manuescript,

$$\hat{x}^{(k)} = \frac{x^{(k)} - \mu^k}{C_{L_\infty}(n) \cdot ||x^{(k)} - \mu^k||_\infty}, \tag{1}$$

To derive $C_{L_\infty}(n)$ we assume again the input $\{x_i\}_{i=1}^n$ to the normalization layer follows a Gaussian distribution $N(\mu^k, \sigma^2)$. Then, the maximum absolute deviation is bounded on expectation as follows

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

[3]:

$$\frac{\sigma \cdot \sqrt{\ln(n)}}{\sqrt{\pi \ln(2)}} \leq ||x^{(k)} - \mu^k||_\infty \leq \sigma\sqrt{2\ln(n)}.$$

Therefore, by multiplying the three sides of inequality with the normalization term $C_{L_\infty}(n)$, the L^∞ batch norm in equation 1 approximates an expectation the original standard deviation measure σ as follows:

$$l \leq C_{L_\infty}(n) \cdot ||x^{(k)} - \mu^k||_\infty \leq u$$

where $l = \frac{1+\sqrt{\pi \ln(4)}}{\sqrt{8\pi \ln(2)}} \cdot \sigma \approx 0.793\sigma$, and $u = \frac{1+\sqrt{\pi \ln(4)}}{2} \cdot \sigma \approx 1.543\sigma$.

3 Bounded-weight-norm experiments

Figure 2 depicts the impact of bounded-weight norm for the training of recurrent network on WMT14 de-en task. Additional results are summarized in Table 1.

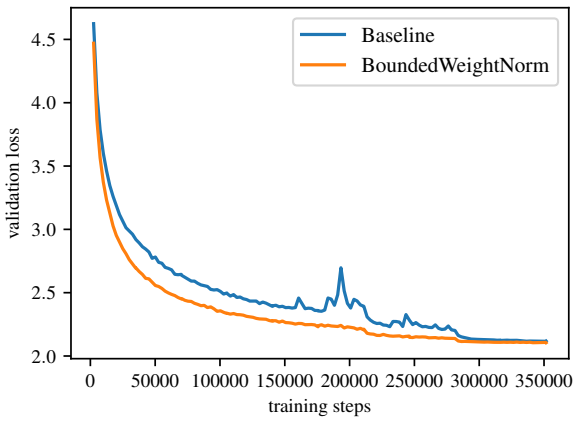


Figure 2: Comparison between bounded weight-norm and baseline with no normalization in recurrent network training (LSTM attention-seq2seq network, WMT14 de-en)

.

Table 1: Results comparing baseline, L^2 based normalization with weight-norm (WN) by Salimans & Kingma [2] and our bounded-weight-norm (BWN)

Network	Batch/Layer norm	WN	BWN
ResNet56 (Cifar10)	93.03%	92.5%	92.88%
ResNet50 (ImageNet)	75.3%	67% [1]	73.8%
Transformer (WMT14)	27.3 BLEU	-	26.5 BLEU
2-layer LSTM (WMT14)	21.5 BLEU	-	21.2 BLEU

Table 2: Results comparing baseline, and L^1 norm results (ppl for perplexity)

Network	L^2 Batch/Layer norm	L^1 Batch/Layer norm
ResNet56 (Cifar10)	93.03%	93.07%
ResNet18 (ImageNet)	69.8%	69.74%
ResNet50 (ImageNet)	75.3%	75.32%
Transformer (WMT14)	5.1 ppl	5.2 ppl

References

[1] Gitman, I. and Ginsburg, B. Comparison of batch normalization and weight normalization algorithms for the large-scale image classification. *CoRR*, abs/1709.08145, 2017.

[2] Salimans, T. and Kingma, D. P. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems*, pp. 901–909, 2016.

[3] Simon, M. K. Probability distributions involving gaussian random variables: A handbook for engineers and scientists. 2007.

[4] Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Chapter 7

Additional results - Spatial Contrasting

Work by Elad Hoffer, Itay Hubara, Nir Ailon

Abstract

Convolutional networks have marked their place over the last few years as the best performing model for various visual tasks. They are, however, most suited for supervised learning from large amounts of labeled data. Previous attempts have been made to use unlabeled data to improve model performance by applying unsupervised techniques. These attempts require different architectures and training methods. In this work we present a novel approach for unsupervised training of Convolutional networks that is based on contrasting between spatial regions within images. This criterion can be employed within conventional neural networks and trained using standard techniques such as SGD and back-propagation, thus complementing supervised methods.

7.1 Introduction

For the past few years convolutional networks (ConvNets, CNNs) [LBBH98] have proven themselves as a successful model for vision related tasks [KSH12a] [MKS+15] [PCD15] [RASC14]. A convolutional network is composed of multiple convolutional and pooling layers, followed by a fully-connected affine transformations. As with other neural network models, each layer is typically followed by a non-linearity transformation such as a rectified-linear unit (ReLU).
A convolutional layer is applied by cross correlating an image with a trainable weight filter. This stems from the assumption of stationarity in natural images, which means that features learned for one local region in an image can be shared for other regions and images.

Deep learning models, including convolutional networks, are usually trained in a su-

pervised manner, requiring large amounts of labeled data (ranging between thousands to millions of examples per-class for classification tasks) in almost all modern applications. These models are optimized a variant of stochastic-gradient-descent (SGD) over batches of images sampled from the whole training dataset and their ground truth-labels. Gradient estimation for each one of the optimized parameters is done by back propagating the objective error from the final layer towards the input. This is commonly known as "backpropagation" [RHW].

One early well known usage of unsupervised training of deep architectures was as part of a pre-training procedure used for obtaining an effective initial state of the model. The network was later fine-tuned in a supervised manner as displayed by [Hin07]. Such unsupervised pre-training procedures were later abandoned, since they provided no apparent benefit over other initialization heuristics in more careful fully supervised training regimes. This led to the de-facto almost exclusive usage of neural networks in supervised environments.

In this work we will present a novel unsupervised learning criterion for convolutional network based on comparison of features extracted from regions within images. Our experiments indicate that by using this criterion to pre-train networks we can improve their performance and achieve state-of-the-art results.

7.2 Previous works

Using unsupervised methods to improve performance have been the holy grail of deep learning for the last couple of years and vast research have been focused on that. We here by give a short overview of the most popular and recent methods that tried to tackle this problem.

AutoEncoders and reconstruction loss Probably the most popular application for unsupervised learning using neural networks, and ConvNets in particular. Autoencoders are NN which aim to transform inputs into outputs with the least possible amount of distortion. Autoencoder is constructed using an encoder $G(x; w_1)$ that maps an input to a hidden latent representation, followed by a decoder $F(y; w_2)$, that maps the representation back into the input space. Mathematically, this can be written in the following general form:

$$\hat{x} = F(G(x; w_1); w_2)$$

The underlying encoder and decoder contain a set of trainable parameters that can be tied together and optimized for a predefined criterion. The encoder and decoder can have different architectures, including fully-connected neural networks, ConvNets and others. The criterion used for training is the reconstruction loss, usually the mean

squared error (MSE) between the original input and it's reconstruction [ZKTF10]

$$min\|x - \hat{x}\|^2$$

This allows an efficient training procedure using the aforementioned backpropagation and SGD techniques. Over the years autoencoders gain fundamental role in unsupervised learning and many modification to the classic architecture were made. [Ng11] regularized the latent representation to be sparse, [VLBM08] substituted the input with a noisy version thereof, requiring the model to denoise while reconstructing. Kingma et al. (2014) obtained very promising results with variational autoencoders (VAE). Variational autoencoder models inherit autoencoder architecture, but make strong assumptions concerning the distribution of latent variables. They use variational approach for latent representation learning, which results in an additional loss component and specific training algorithm called Stochastic Gradient Variational Bayes (SGVB). VAE assumes that the data is generated by a directed graphical model $p(x|z)$ and require the encoder to learn an approximation $q_{w_1}(z|x)$ to the posterior distribution $p_{w_2}(z|x)$ where w_1 and w_2 denote the parameters of the encoder and decoder. The objective of the variational autoencoder in this case has the following form:

$$\mathcal{L}(w_1, w_2, x) = -D_{KL}(q_{w_1}(z|x)||p_{w_2}(z)) + \mathbb{E}_{q_{w_1}(z|x)}\big(\log p_{w_2}(x|z)\big)$$

Recently stacked set of denoising autoencoders architectures showed promising results in both semi-supervised and unsupervised tasks. Stacked what-where autoencoder by [ZMGL15a] computes a set of complementary variables that enable reconstruction whenever a layer implements a many-to-one mapping. Ladder networks by [RBH$^+$15] - use lateral connections to allow higher levels of an autoencoder to focus on invariant abstract features by applying layer-wise cost function.

Exemplar Networks: unsupervised method introduced by[DSRB14] takes a different approach to this task and train the network to discriminate between a set of pseudo-classes. Each pseudo-class is formed by applying multiple transformations to a randomly sampled image patch. The number of pseudo-classes can be as big as the size of the input samples. This criterion ensure that different input samples would be distinguished while providing robustness to the applied transformations.

Adversarial Generative Models: A recently introduced model that can be used in an unsupervised fashion [GPAM$^+$14]. Adversarial Generative Models Uses a set of networks, one trained to discriminate between data sampled from the true underlying distribution (e.g., a set of images), and a separate generative network trained to be

an adversary trying to confuse the first network. By propagating the gradient through the paired networks, the model learns to generate samples that are distributed similarly to the source data. As shown by [RMC15],this model can create useful latent representations for subsequent classification tasks as demonstrated

Sampling Methods: Methods for training models to discriminate between a very large number of classes often use a *noise contrasting criterion*. In these methods, roughly speaking, the posterior probability $P(t|y_t)$ of the ground-truth target t given the model output on an input sampled from the true distribution $y_t = F(x)$ is maximized, while the probability $P(t|y_n)$ given a noise measurement $y = F(n)$ is minimized. This was successfully used in a language domain to learn unsupervised representation of words. The most noteworthy case is the word2vec model introduced by [MSC+13]. When using this setting in language applications, a natural contrasting noise is a smooth approximation of the Unigram distribution. A suitable contrasting distribution is less obvious when data points are sampled from a high dimensional continuous space, such as in the case of patches of images.

7.2.1 Problems with Current Approaches

Only recently the potential of ConvNets in an unsupervised environment began to bear fruit, still we believe it is not fully uncovered.

The majority of unsupervised optimization criteria currently used are based on variations of reconstruction losses. One limitation of this fact is that a pixel level reconstruction is non-compliant with the idea of a discriminative objective, which is expected to be agnostic to low level information in the input. In addition, it is evident that MSE is not best suited as a measurement to compare images, for example, viewing the possibly large square-error between an image and a single pixel shifted copy of it. Another problem with recent approaches such as [RBH+15, ZKTF10] is their need to extensively modify the original convolutional network model. This leads to a gap between unsupervised method and the state-of-the-art, supervised, models for classification - which can hurt future attempt to reconcile them in a unified framework, and also to efficiently leverage unlabeled data with otherwise supervised regimes.

7.3 Learning by Comparisons

The most common way to train NN is by defining a loss function between the target values and the network output. Learning by comparison approaches the supervised task from a different angle. The main idea is to use distance comparisons between samples to learn useful representations. For example, we consider relative and qualitative examples of the form "X_1 is closer to X_2 than X_1 is to X_3. Using a comparative measure with neural network to learn embedding space was introduced in the "Siamese network"

framework by [BBB+93] and later used in the works of [CHL05]. One use for this methods is when the number of classes is too large or expected to vary over time, as in the case of face verification, where a face contained in an image has to compared against another image of a face. This problem was recently tackled by [SKP15] for training a convolutional network model on triplets of examples. There, one image served as an *anchor* x, and an additional pair of images served as a positive example x_+ (containing an instance of the face of the same person) together with a negative example x_-, containing a face of a different person. The training objective was on the embedded distance of the input faces, where the distance between the anchor and positive example is adjusted to be smaller by at least some constant α from the negative distance. More precisely, the loss function used in this case was defined as

$$L(x, x_+, x_-) = \max\{\|F(x) - F(x_+)\|_2 - \|F(x) - F(x_-)\|_2 + \alpha, 0\} \tag{7.1}$$

where $F(x)$ is the embedding (the output of a convolutional neural network), and α is a predefined margin constant. Another similar model used by [HA15a] with triplets comparisons for classification, where examples from the same class were trained to have a lower embedded distance than that of two images from distinct classes. This work introduced a concept of a distance ratio loss, where the defined measure amounted to:

$$L(x, x_+, x_-) = \frac{e^{\|F(x)-F(x_+)\|_2}}{e^{\|F(x)-F(x_+)\|_2} + e^{\|F(x)-F(x_-)\|_2}} \tag{7.2}$$

This loss has a flavor of a probability of a biased coin flip. By 'pushing' this probability to zero, we express the objective that pairs of samples coming from distinct classes should be less similar to each other, compared to pairs of samples coming from the same class. It was shown empirical by [BJTM16] to provide better feature embeddings than the margin based distance loss 7.1

7.4 Our Contribution: Spatial Contrasting

One implicit assumption in convolutional networks, is that features are gradually learned hierarchically, each level in the hierarchy corresponding to a layer in the network. Each spatial location within a layer corresponds to a region in the original image. It is empirically observed that deeper layers tend to contain more 'abstract' information from the image. Intuitively, features describing different regions within the same image are likely to be semantically similar (e.g. different parts of an animal), and indeed the corresponding deep representations tend to be similar. Conversely, regions from two probably unrelated images (say, two images chosen at random) tend to be far from each other in the deep representation. This logic is commonly used in modern deep networks such as [SLJ+15] [LCY13a] [HZRS15a], where a global average pooling is used to aggregate spatial features in the final layer used for classification.

Our suggestion is that this property, often observed as a side effect of supervised applications, can be used as a desired objective when learning deep representations in an unsupervised task. Later, the resulting representation can be used, as typically done, as a starting point or a supervised learning task. We call this idea which we formalize below *Spatial contrasting*. The spatial contrasting criterion is similar to noise contrasting estimation [GH10] [MK13], in trying to train a model by maximizing the expected probability on desired inputs, while minimizing it on contrasting sampled measurements.

7.4.1 Formulation

We will concern ourselves with samples of images patches $\tilde{x}^{(m)}$ taken from an image x. Our convolutional network model, denoted by $F(x)$, extracts spatial features f so that $f^{(m)} = F(\tilde{x}^{(m)})$ for an image patch $\tilde{x}^{(m)}$. We wish to optimize our model such that for two features representing patches taken from the same image $\tilde{x}_i^{(1)}, \tilde{x}_i^{(2)} \in x_i$ for which $f_i^{(1)} = F(\tilde{x}_i^{(1)})$ and $f_i^{(2)} = F(\tilde{x}_i^{(2)})$, the conditional probability $P(f_i^{(1)}|f_i^{(2)})$ will be maximized.
This means that features from a patch taken from a specific image can effectively predict, under our model, features extracted from other patches in the same image. Conversely, we want our model to minimize $P(f_i|f_j)$ for i, j being two patches taken from distinct images. Following the logic presented before, we will need to sample *contrasting patch* $\tilde{x}_j^{(1)}$ from a different image x_j such that $P(f_i^{(1)}|f_i^{(2)}) > P(f_j^{(1)}|f_i^{(2)})$, where $f_j^{(1)} = F(\tilde{x}_j^{(1)})$. In order to obtain contrasting samples, we use regions from two random images in the training set. We will use a distance ratio, described earlier 7.2 for the supervised case, to represent the probability two feature vectors were taken from the same image.

The resulting training loss for a pair of images will be defined as

$$L_{SC}(x_1, x_2) = -\log \frac{e^{-\|f_1^{(1)} - f_1^{(2)}\|_2}}{e^{-\|f_1^{(1)} - f_1^{(2)}\|_2} + e^{-\|f_1^{(1)} - f_2^{(1)}\|_2}} \tag{7.3}$$

Effectively minimizing a log-probability under the SoftMax measure. This formulation is portrayed in figure **??**. Since we sample our contrasting sample from the same underlying distribution, we can evaluate this loss considering the image patch as both patch compared (anchor) and contrast symmetrically. The final loss will be the average between these estimations:

$$\hat{L_{SC}}(x_1, x_2) = \frac{1}{2}\left[L_{SC}(x_1, x_2) + L_{SC}(x_2, x_1)\right]$$

7.4.2 Method

Since training convolutional network is done in batches of images, we can use the multiple samples in each batch to train our model. Each image serves as a source for

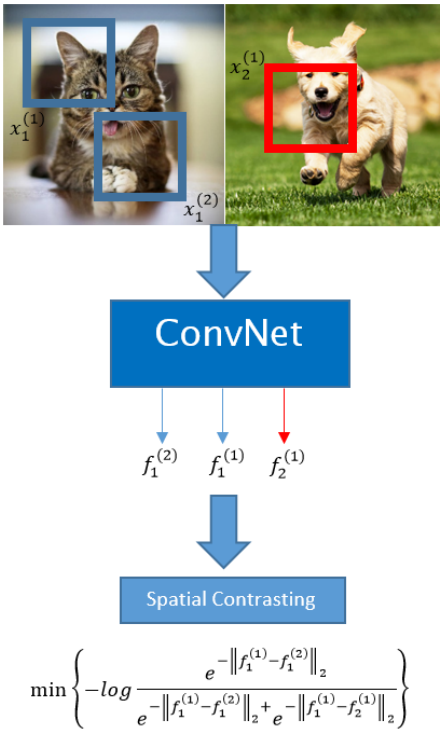


Figure 7.1: Spatial contrasting depiction.

both an anchor and positive patches, for which the corresponding features should be closer, and also a source for contrasting samples for all the other images in that batch. For a batch of N images, two samples from each image are taken, and N^2 different distance comparisons are made. The final loss is the average distance ratio for images in the batch:

$$\bar{L_{SC}}(\{x\}_{i=1}^{N}) = \frac{1}{N}\sum_{i=1}^{N} L_{SC}(x_i, \{x\}_{j\neq i}) = -\frac{1}{N}\sum_{i=1}^{N} \log \frac{e^{-\|f_i^{(1)}-f_i^{(2)}\|_2}}{\sum_{j=1}^{N} e^{-\|f_i^{(1)}-f_j^{(2)}\|_2}} \tag{7.4}$$

Since the criterion is differentiable with respect to its inputs, it is fully compliant with standard methods for training convolutional network and specifically using backpropagation and gradient descent. Furthermore, SC can be applied to any layer in the network hierarchy. In fact, SC can be used at multiple layers within the same convolutional network. The spatial properties of the features means that we can also sample from feature space $\tilde{f}^{(m)} \in f$ instead of from the original image, which we use to simplify implementation. The complete algorithm for batch training is described in 7.1. This algorithm is also related to the batch normalization layer [IS15], a recent usage for batch statistics in neural networks. Spatial contrasting also uses the batch statistics, but to sample contrasting patches.

Algorithm 7.1 Calculation the spatial contrasting loss

Require: $X = \{x\}_{i=1}^{N}$ # Training on batches of images

Get the spatial features for the whole batch of images
Size: $N \times W_f \times H_f \times C$
$\{f\}_{i=1}^{N} \leftarrow \texttt{ConvNet}(X)$

# Sample spatial features and calculate embedded distance between all pairs of images
for i = 1 **to** N **do**
  $\tilde{f}_i^{(1)} \leftarrow \texttt{sample}(f_i)$
  for j = 1 **to** N **do**
    $\tilde{f}_j^{(2)} \leftarrow \texttt{sample}(f_j)$
    $Dist(i,j) \leftarrow \|\tilde{f}_i^{(1)} - \tilde{f}_j^{(2)}\|_2$
  end for
end for

Calculate log SoftMax normalized distances
$d_i \leftarrow -\log \frac{e^{-Dist(i,i)}}{\sum_{k=1}^{N} e^{-Dist(i,k)}}$

Spatial contrasting loss is the mean of distance ratios
return $\frac{1}{N}\sum_{i=1}^{N} d_i$

7.5 Experiments

In this section we report empirical results showing that using SC loss as an unsupervised pretraining procedure can improve state-of-the-art performance on subsequent classification. We experimented with MNIST, CIFAR-10 and STL10 datasets. We used modified versions of well studied networks such as those of [LCY13a] [RBH+15]. A detailed description of our architecture can be found in Appendix A.

In each one of the experiments, we used the spatial contrasting criterion to train the network on the unlabeled images. Training was done by using SGD with an initial learning rate of 0.1 that was decreased by a factor of 10 whenever the measured loss stopped decreasing. After convergence, we used the trained model as an initialization for a supervised training on the complete labeled dataset. The supervised training was done following the same regime, only starting with a lower initial learning rate of 0.01. We used mild data augmentations, such as small translations and horizontal mirroring. The datasets we used are:

- STL10 ([CNL11]). This dataset consists of $100,000$ 96×96 colored, unlabeled images, together with another set of $5,000$ labeled training images and $8,000$ test images . The label space consists of 10 object classes.

- Cifar10 ([KH09]). The well known CIFAR-10 is an image classification benchmark dataset containing $50,000$ training images and $10,000$ test images. The image

Table 7.1: State of the art results on STL-10 dataset

Model	**STL-10 test accuracy**
Zero-bias Convnets - [PKHH14]	70.2%
Triplet network - [HA15a]	70.7%
Exemplar Convnets - [DSRB14]	72.8%
Target Coding - [YLL$^+$15]	73.15%
Stacked what-where AE - [ZMGL15a]	74.33%
Spatial contrasting initialization (this work)	$81.34\% \pm 0.1$
The same model without initialization	$72.6\% \pm 0.1$

sizes 32×32 pixels, with color. The classes are airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships and trucks.

- MNIST ([LBBH98]). The MNIST database of handwritten digits is one of the most studied dataset benchmark for image classification. The dataset contains 60,000 examples of handwritten digits from 0 to 9 for training and 10,000 additional examples for testing. Each sample is a 28 x 28 pixel gray level image.

7.5.1 Results on STL10

Since STL10 is comprised of mostly unlabeled data, it is the most suitable to highlight the benefits of the spatial contrasting criterion. The initial training was unsupervised, as described earlier, using the entire set of $105,000$ samples (union of the original unlabeled set and labeled training set). The representation outputted by the training, was used to initialize supervised training on the $5,000$ labeled images. Evaluation was done on a separate test set of $8,000$ samples. Comparing with state of the art results 7.1 we see an improvement of 7% in test accuracy over the best model by [ZMGL15a], setting the SC as best model at 81.3% test classification accuracy. We also compare with the same network, but without SC initialization, which achieves a lower classification of 72.6%. This is an indication that indeed SC managed to leverage unlabeled examples to provide a better initialization point for the supervised model.

7.5.2 Results on Cifar10

For Cifar10, we used a previously used setting [CN12] [Hui13] [DSRB14] to test a model's ability to learn from unlabeled images. In this setting, only $4,000$ samples from the available $50,000$ are used with their label annotation, but the entire dataset is used for unsupervised learning. The final test accuracy is measured on the entire $10,000$ test set.
In our experiments, we trained our model using SC criterion on the entire dataset, and then used only 400 labeled samples per class (for a total of 4000) in a supervised regime over the initialized network. The results are compared with previous efforts in table

Table 7.2: State of the art results on Cifar10 dataset with only 4000 labeled samples

Model	**Cifar10 test accuracy**
Convolutional K-means Network - [CN12]	70.7%
View-Invariant K-means - [Hui13]	72.6%
DCGAN - [RMC15]	73.8%
Exemplar Convnets - [DSRB14]	76.6%
Ladder networks - [RBH$^+$15]	79.6%
Spatial contrasting initialization (this work)	$79.2\% \pm 0.3$
The same model without initialization	$72.4\% \pm 0.1$

Table 7.3: results on MNIST dataset

Model	**MNIST test error**
Stacked what-where AE - [ZMGL15a]	0.71%
Triplet network - [HA15a]	0.56%
[JKRL09]	0.53%
Ladder networks - [RBH$^+$15]	0.36%
DropConnect - [WZZ$^+$13]	0.21%
Spatial contrasting initialization (this work)	$0.34\% \pm 0.02$
The same model without initialization	$0.63\% \pm 0.02$

7.2. Using the SC criterion allowed an improvement of 6.8% over a non-initialized model, and achieved a final test accuracy of 79.2%. This is a competitive result with current state-of-the-art model of [RBH$^+$15].

7.5.3 Results on MNIST

The MNIST dataset is very different in nature from the Cifar10 and STL10, we experimented earlier. The biggest difference, relevant to this work, is that spatial regions sampled from MNIST images usually provide very little, or no information. Because of this fact, SC is much less suited for use with MNIST, and was conjured to have little benefit. We still, however, experimented with initializing a model with SC criterion and continuing with a fully-supervised regime over all labeled examples. We found again that this provided benefit over training the same network without pre-initialization, improving results from 0.63% to 0.34% error on test set. The results, compared with previous attempts are included in 7.3.

7.6 Conclusions and future work

In this work we presented spatial contrasting - a novel unsupervised criterion for training convolutional networks on unlabeled data. Its is based on comparison between spatial features sampled from a number of images. We've shown empirically that using

spatial contrasting as a pretraining technique to initialize a ConvNet, can improve its performance on a subsequent supervised training. In cases where a lot of unlabeled data is available, such as the STL10 dataset, this translates to state-of-the-art classification accuracy in the final model.

Since the spatial contrasting loss is a differentiable estimation that can be computed within a network in parallel to supervised losses, future work will attempt to embed it as a semi-supervised model. This usage will allow to create models that can leverage both labeled an unlabeled data, and can be compared to similar semi-supervised models such as the ladder network [RBH$^+$15]. It is is also apparent that contrasting can occur in dimensions other than the spatial, the most straightforward is the temporal one. This suggests that similar training procedure can be applied on segments of sequences to learn useful representation without explicit supervision.

Chapter 8

Additional results - Semi-supervised learning by metric embedding

Work by Elad Hoffer, Nir Ailon

Abstract

Deep networks are successfully used as classification models yielding state-of-the-art results when trained on a large number of labeled samples. These models, however, are usually much less suited for semi-supervised problems because of their tendency to overfit easily when trained on small amounts of data. In this work we will explore a new training objective that is targeting a semi-supervised regime with only a small subset of labeled data. This criterion is based on a deep metric embedding over distance relations within the set of labeled samples, together with constraints over the embeddings of the unlabeled set. The final learned representations are discriminative in euclidean space, and hence can be used with subsequent nearest-neighbor classification using the labeled samples.

8.1 Introduction

Deep neural networks have been shown to perform very well on various classification problems, often yielding state-of-the-art results. Key motivation for the use of these models, is the assumption of hierarchical nature of the underlying problem. This assumption is reflected in the structure of NNs, composed of multiple stacked layers of linear transformations followed by non-linear activation functions. The NN final layer is usually a softmax activated linear transformation indicating the likelihood of each class, which can be trained by cross-entropy using the known target of each sample, and back-propagated to previous layers. The hierarchical property of NNs has been ob-

served to yield high-quality, discriminative representations of the input in intermediate layers. These representative features, although not explicitly part of the training objective, were shown to be useful in subsequent tasks in the same domain as demonstrated by [RASC14]. One serious problem occurring in neural network is their susceptibility to overfit over the training data. Due to this fact, a considerable part of modern neural network research is devoted to regularization techniques and heuristics such as [SHK+14, IS15, WZZ+13, SVI+15], to allow the networks to generalize to unseen data samples. The tendency to overfit is most apparent with problems having a very small number of training examples per class, and these are considered ill-suited to solve with neural network models. Because of this property, semi-supervised regimes in which most data is unlabeled, are considered hard to learn and generalize with NNs.

In this work we will consider a new training criterion designed to be used with deep neural networks in semi-supervised regimes over datasets with a small subset of labeled samples. Instead of a usual cross-entropy between the labeled samples and the ground truth class indicators, we will use the labeled examples as targets for a metric embedding. Under this embedding, which is the mapping of a parameterized deep network, the features of labeled examples will be grouped together in euclidean space. In addition, we will use these learned embeddings to separate the unlabeled examples to belong each to a distinct cluster formed by the labeled samples. We will show this constraint translates to a minimum entropy criterion over the embedded distances. Finally, because of the use of euclidean space interpretation of the learned features, we are able to use a subsequent nearest-neighbor classifier to achieve state-of-the-art results on problems with small number of labeled examples.

8.2 Related work

8.2.1 Learning metric embedding

Previous works have shown the possible use of neural networks to learn useful metric embedding. One kind of such metric embedding is the "Siamese network" framework introduced by [BBB+93] and later used in the works of [CHL05]. One use for this methods is when the number of classes is too large or expected to vary over time, as in the case of face verification, where a face contained in an image has to compared against another image of a face. This problem was recently tackled by [SKP15] for training a convolutional network model on triplets of examples. Learning features by metric embedding was also shown by [HA15b] to provide competitive classification accuracy compare to conventional cross-entropy regression. This work is also related to [RPDB15], who introduced Magnet loss - a metric embedding approach for fine-grained classification. The Magnet loss is based on learning the distribution of distances for each sample, from K clusters assigned for each classified class. It then uses an intermediate k-means clustering, to reposition the different assigned clusters. This proved to allow

better accuracy than both margin-based Triplet loss, and softmax regression. Using metric embedding with neural network was also specifically shown to provide good results in the semi-supervised learning setting as seen in [WRMC12].

8.2.2 Semi-supervised learning by adversarial regularization

As stated before, a key approach to generalize from a small training set, is by regularizing the learned model. Regularization techniques can often be interpreted as prior over model parameters or structure, such as L_p regularization over the network weights or activations. More recently, neural network specific regularizations that induce noise within the training process such as [SHK+14, WZZ+13, SVI+15] proved to be highly beneficial to avoid overfitting. Another recent observation by [GSS15] is that training on adversarial examples, inputs that were found to be misclassified under small perturbation, can improve generalization. This fact was explored by [FZK+16] and found to provide notable improvements to the semi supervised regime by [MMK+15].

8.2.3 Semi-supervised learning by auxiliary reconstruction loss

Recently, a stacked set of denoising auto-encoders architectures showed promising results in both semi-supervised and unsupervised tasks. A stacked what-where autoencoder by [ZMGL15b] computes a set of complementary variables that enable reconstruction whenever a layer implements a many-to-one mapping. Ladder networks by [RBH+15] - use lateral connections to allow higher levels of an auto-encoder to focus on invariant abstract features by applying a layer-wise cost function.

Generative adversarial network (GAN) is a recently introduced model that can be used in an unsupervised fashion [GPAM+14]. Adversarial Generative Models use a set of networks, one trained to discriminate between data sampled from the true underlying distribution (e.g., a set of images), and a separate generative network trained to be an adversary trying to confuse the first network. By propagating the gradient through the paired networks, the model learns to generate samples that are distributed similarly to the source data. As shown by [RMC15], this model can create useful latent representations for subsequent classification tasks. The usage for these models for semi-supervised learning was further developed by [Spr16] and [SGZ+16], by adding a $N+1$ way classifier (number of classes + and additional "fake" class) to the discriminator. This proved to allow excellent accuracy with only a small subset of labeled examples.

8.2.4 Semi-supervised learning by entropy minimization

Another technique for semi-supervised learning introduced by [GB04] is concerned with minimizing the entropy over expected class distribution for unlabeled examples. Regularizing for minimum entropy can be seen as a prior which prefers minimum overlap between observed classes. This can also be seen as a generalization of the "self-training" wrapper method described by [TGH15], in which unlabeled examples are re-introduced

after being labeled with the previous classification of the model. This is also related to the "Transductive suport vector machines" (TSVM) [VV98] which introduces a maximum margin objective over both labeled and unlabeled examples.

Comparing with previous works such as [CHL05] and [WRMC12], this work uses a novel objective which is composed of a distance ratio measure (unlike the contrastive, hinge based loss used before), and an entropy minimization on the distance measure to labeled samples. Although distance ratio loss ([HA15b]) and entropy minimization ([GB04]) are not new, this is the first attempt to our knowledge, of combining these ideas for semi-supervised metric learning.

8.3 Our contribution: Neighbor embedding for semi-supervised learning

In this work we are concerned with a semi-supervised setting, in which learning is done on data of which only a small subset is labeled. Given observed sets of labeled data $X_L = \{(x, y)\}_{i=1}^{l}$ and unlabeled data $X_U = \{x\}_{i=l+1}^{n}$ where $x \in \mathcal{X}$, $y \in \mathcal{C}$, we wish to learn a classifier $f : \mathcal{X} \to \mathcal{C}$ to have a minimum expected error on some unseen test data X_{test}.

We will make a couple of assumptions regarding the given data:

- The number of labeled examples is small compared to the whole observed set $l \ll n$.
- Structure assumption - samples within the same structure (such as a cluster or manifold) are more likely to share the same label. This assumption is shared with many other semi-supervised approaches as discussed in [CSZ09],[WRMC12].

Using these assumptions, we are motivated to learn a metric embedding that forms clusters such that samples can be classified by their L_2 distance to the labeled examples in a nearest-neighbor procedure.

We will now define our learning setting on the semi-labeled data, using a neural network model denoted as $F(x; \theta)$ where x is the input fed into the network, and θ are the optimized parameters (dropped henceforward for convenience). The output of the network for each sample is a vector of features of D dimensions $F(x) \in \mathbb{R}^D$ which will be used to represent the input.

Our two training objectives which we aim to train our embedding networks by are:

(i) Create feature representation that form clusters from the labeled examples $\{(x, y)\} \in X_L$ such that two examples x_1, x_2 sharing the same label $y_1 = y_2$ will have a smaller embedded distance than any third example x_3 with a different label $y_1 \neq y_3$

$$\|F(x_1) - F(x_2)\|_2 < \|F(x_1) - F(x_3)\|_2$$

(ii) For each unlabeled example, its feature embedding will be close to the embeddings of one specific label occurring in L:

For all $x \in X_U$, $z \in X_L$, there exists a specific class $l \in \mathcal{C}$ such that

$$\|F(x) - F(z_l)\|_2 \ll \|F(x) - F(z_k)\|_2$$

where z_l is any labeled example of class l and z_k is any example from class $k \in \mathcal{C} \setminus \{l\}$.

As the defined objectives create embeddings that target a nearest-neighbor classification with regard to the labeled set, we will refer to it as "Neighbor embedding".

8.4 Learning by distance comparisons

We will define a discrete distribution for the embedded distance between a sample $x \in \mathcal{X}$, and c labeled examples $z_1, ..., z_c \in X_L$ each belonging to a different class:

$$P(x; z_1, ..., z_c)_i = \frac{e^{-\|F(x)-F(z_i)\|^2}}{\sum_{j=1}^{c} e^{-\|F(x)-F(z_j)\|^2}}, i \in \{1...c\} \tag{8.1}$$

This definition assigns a probability $P(x; z_1, ..., z_c)_i$ for sample x to be classified into class i, under a 1-nn classification rule, when $z_1, ..., z_c$ neighbors are given. It is similar to the stochastic-nearest-neighbors formulation of [GHRS04], and will allow us to state the two underlying objectives as measures over this distribution.

8.4.1 Distance ratio criterion

Addressing objective (i), we will use a sample $x_l \in X_L$ from the labeled set belonging to class $k \in \mathcal{C}$, and another set of sampled labeled examples $z_1, ..., z_c \in X_L$. In this work we will sample in uniform over all available samples for each class.

Defining the class-indicator $I(x)$ as

$$I(x_l)_i = \begin{cases} 1 & \text{if } i = k \\ 0 & \text{otherwise} \end{cases}$$

we will minimize the cross-entropy between $I(x_l)$ and the distance-distribution of x with respect to $z_1, ..., z_c$8.1:

$$L(x_l, z_1, ..., z_c)_L = H\left(I(x_l), P(x_l; z_1, ..., z_c)\right) \tag{8.2}$$

This is in fact a slightly modified version of distance ratio loss introduced in [HA15b].

$$L(x_l, z_1, ..., z_c)_L = -\log \frac{e^{-\|F(x_l)-F(z_k)\|^2}}{\sum_{i=1}^{c} e^{-\|F(x_l)-F(z_i)\|^2}} \tag{8.3}$$

This loss is aimed to ensure that samples belonging to the same class will be mapped to have a small embedded distance compared to samples from different classes.

8.4.2 Minimum entropy criterion

Another part of the optimized criterion, inspired by [GB04], is designed to reduce the overlap between the different classes of the unlabeled samples.

We will promote this objective by minimizing the entropy of the underlying distance distribution of x, again with respect to labeled samples $z_1, ..., z_c$8.1:

$$L(x, z_1, ..., z_c)_U = H(P(x; z_1, ..., z_c)) \tag{8.4}$$

which is defined as

$$L(x, z_1, ..., z_c)_U = -\sum_{i=1}^{c} \frac{e^{-\|F(x)-F(z_i)\|^2}}{\sum_{j=1}^{c} e^{-\|F(x)-F(z_j)\|^2}} \cdot \log \frac{e^{-\|F(x)-F(z_i)\|^2}}{\sum_{j=1}^{c} e^{-\|F(x)-F(z_j)\|^2}} \tag{8.5}$$

We note that entropy is lower if the distribution 8.1 is sparse, and higher if the distribution is dense, and this intuition is compatible with our objectives.

Our final objective will use a sampled set of labeled examples, where each class is represented $\{z_1, ..., z_c\}$ and additional labeled x_l and unlabeled x_u examples, combining a weighted sum of both 8.3 and 8.5 to form:

$$L(x_l, x_u, \{z_1, ..., z_c\}) = \lambda_L L(x_l, z_1, ..., z_c)_L + \lambda_U L(x_u, z_1, ..., z_c)_U \tag{8.6}$$

Where $\lambda_L, \lambda_U \in [0, 1]$ are used to determine the weight assigned to each criterion.

This loss is differentiable and hence can be used for gradient-based training of deep models by existing optimization approaches and back-propagation ([RHW]) through the embedding neural network. The optimization can further be accelerated computationally by using mini-batches of both labeled and unlabeled examples.

8.5 Qualities of neighbor embedding

We will now discuss some observed properties of neighbor embeddings, and their usefulness to semi-supervised regimes using neural network models.

8.5.1 Reducing overfit

Usually, when using NNs for classification, a cross-entropy loss minimization is employed by using a fixed one-hot indicator (similar to 8.2) as target for each labeled example, thus maximizing a log-likelihood of the correct label. This form of optimization over a fixed target tend to cause an overfitting of the neural-network, especially on small labeled sets. This was lately discussed and addressed by [SVI+15] using added random noise to the targets by sampling uniformly from the set of classes, effectively smoothing the cross-entropy target distribution. This regularization technique was shown empirically to yield better generalization by reducing the overfitting over the training set.
Training on distance ratio comparisons, as shown in our work, provides a natural alternative to this problem. By setting the optimization target to be the embeddings of labeled examples, we create a continuously moving target that is dependent on the current model parameters. We speculate that this reduces the model's ability to overfit easily on the training data, allowing very small labeled datasets to be exploited.

8.5.2 Embedding into euclidean space

By training the model to create feature embedding that are discriminative with respect to their distance in euclidean space, we can achieve good classification accuracy using a simple nearest-neighbor classifier. This embedding allows an interpretation of semantic relation in euclidean space, which can be useful for various tasks such as information retrieval, or transfer learning.

8.5.3 Combining supervised and unsupervised objectives

Neighbor embedding is composed of both supervised 8.3 and unsupervised 8.5 objectives that are weighted by λ_L, λ_U coefficients. This can be used to balance or possibly anneal over time ([ZMMAP16]) to adapt for the availability of labeled samples. This form of balancing was found previously to allow for better representation learning using unlabeled data.

8.5.4 Incorporating prior knowledge

We also note that prior knowledge about a problem at hand can be incorporated into the expected measures with respect to the distance distribution 8.1. E.g, knowledge of relative distance between classes can be used to replace $I(x)$ as target distribution in eq. 8.3 and knowledge concerning overlap between classes can be used to relax the constraint in eq. 8.5.

8.6 Experiments

All experiments were conducted using the Torch7 framework by [CKF11]. Code reproducing these results will by available at `https://github.com/eladhoffer/SemiSupContrast`. For every experiment we chose a small random subset of examples, with a balanced number from each class and denoted by X_L. The remaining training images are used without their labeled to form X_U. Finally, we test our final accuracy with a disjoint set of examples X_{test}. No data augmentation was applied to the training sets.

In each iteration we sampled uniformly a set of labeled examples $z_1, ...z_{|\mathcal{C}|} \in X_L$. In addition, batches of uniformly sampled examples were also sampled again from the labeled set X_L, and the unlabeled set X_U.

A batch-size of $b = 32$ was used for all experiments, totaling a sampled set of $2 \cdot b + |\mathcal{C}|$ examples for each iteration, where $|\mathcal{C}| = 10$ for both datasets. We used 8.6 as optimization criterion, where $\lambda_L = \lambda_U = 1$. Optimization was done using the Accelerated-gradient method by [Nes83] with an initial learning rate of $lr_0 = 0.1$ which was decreased by a factor of 10 after every 30 epochs. Both datasets were trained on for a total of 90 epochs. Final test accuracy results was achieved by using a k-NN classifier with best results out of $k = \{1, 3, 5\}$. These results were average over 10 random subsets of labeled data. The choices for λ and k-NN parameters were made using a validation set. We did not found any substantial difference between the values we explored, so they were usually left as the default value for simplicity.

As the embedding model was chosen to be a convolutional network, the spatial properties of input space are crucial. We thus omit results on permutation-invariant versions of these problems, noting they usually tend to achieve worse classification accuracies. We also note that the neural network models themselves are very simple to ensure performance achieved is due to the proposed objective, and not the network architecture.

8.6.1 Results on MNIST

The MNIST database of handwritten digits introduced by [LBBH98] is one of the most studied dataset benchmark for image classification. The dataset contains 60,000 examples of handwritten digits from 0 to 9 for training and 10,000 additional examples for testing, where each sample is a 28 x 28 pixel gray level image.

We followed previous works ([WRMC12],[ZMGL15b],[RBH$^+$15]) and used semi-supervised regime in which only 100 samples (10 for each class) were used as X_L along with their labels. For the embedding network, we used a convolutional network with 5-convolutional layers, where each layer is followed by a ReLU non-linearity and batch-normalization layer [IS15]. The full network structure is described in Appendix table 1. Results are displayed in table 8.1 and reflect that our approach yields state-of-the-art results in this regime.

We also attempted to visualize the outcome of using this method, by training an

Table 8.1: Results for MNIST. Using 100 labeled examples, no data-augmentation.

Model	Test error %
EmbedCNN [WRMC12]	7.75
SWWAE [ZMGL15b]	9.17
Ladder network [RBH+15]	0.89 (± 0.50)
Conv-CatGAN [Spr16]	1.39 (± 0.28)
Ours	**0.78 (± 0.3)**

(a) (b)

Figure 8.1: MNIST 2d visualization before (a) and after (b) training. 100 colored labeled samples, unlabeled samples marked in gray

additional model with a final 2-dimensional embedding. Figure 8.1b shows the final embeddings, where labeled examples are marked in color with their respective class, and unlabeled examples are marked in gray. We can see that, in accordance with our objectives, the labeled examples formed clusters in euclidean space separate by their labels, while unlabeled examples were largely grouped to belong each to one of these clusters.

8.6.2 Results on Cifar-10

Cifar-10 introduced by [KH09] is an image classification benchmark dataset containing $50,000$ training images and $10,000$ test images. The image sizes 32×32 pixels, with color. The classes are airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships and trucks.

Following a commonly used regime, we trained on 4000 randomly picked samples (400 for each class). As the convolutional embedding network, we used a network similar to that of [LCY13b] which is described in table 1. The test error results are

Table 8.2: Results for Cifar-10. Using 4000 labeled samples, no data-augmentation.

Model	Test error %
Spike-and-Slab Sparse Coding [GCB12]	31.9
View-Invariant k-means [Hui13]	27.4 (± 0.7)
Exemplar-CNN [DSRB14]	23.4 (± 0.2)
Ladder network [RBH$^+$15]	20.04 (± 0.47)
Conv-CatGan [Spr16]	19.58 (± 0.58)
ImprovedGan [SGZ$^+$16]	**18.63 (± 2.32)**
Ours	20.3 (± 0.5)

brought in table 8.2.

As can be observed, we achieve competitive results with state-of-the-art in this regime. We also note that current best results are from generative models such as [Spr16] and [SGZ$^+$16] that follow an elaborate and computationally heavy training procedure compared with our approach.

8.7 Conclusions

In this work we have shown how neural networks can be used to learn in a semi-supervised setting using small sets of labeled data, by replacing the classification objective with a metric embedding one. We introduced an objective for semi-supervised learning formulated as minimization of entropy over a distance encoding distribution. This objective is compliant with standard techniques of training deep neural network and requires no modification of the embedding model. Using the method in this work, we were able to achieve state-of-the-art results on MNIST with only 100 labeled examples and competitive results on Cifar10 dataset. We speculate that this form of learning is beneficial to neural network models by decreasing their tendency to overfit over small sets of training data. The objectives formulated here can potentially leverage prior knowledge on the distribution of classes or samples, as well as incorporating this knowledge in the training process. For example, utilizing the learned embedded distance, we speculate that a better sampling can be done instead of a uniform one over the entire set.

Further exploration is needed to apply this method to large scale problems, spanning a large number of available classes, which we leave to future work.

Chapter 9

Discussion

9.1 Summary

9.1.1 Deep metric learning

In chapter 3, we've demonstrated how deep models can be used to learn explicit representations by training on metric embedding rather than explicit classification. We've introduced the "Triplet network", a model that uses three inputs, of which one serves as an anchor and the others are negative and positive examples. Training is done on the objective that the two similar examples must have a lower embedded euclidean distance than two dissimilar examples. We also formulated a novel loss function, that acts on the *ratio* of the embedded distances, rather than on their difference. We thus allowed the model to learn complex and interpretable embedding, without the need for additional predetermined hyper-parameter.

This method was later used to solve a variety of tasks where class space is too large or cannot be determined such as face identification [LWY+17] and image descriptors [KCR+16, BJTM16]. Furthermore, we showed in chapter 7 how the notion of metric embedding in the context of deep models can leverage new sources of information to enable self-supervised learning in which ground-truth labels are not available. Using spatial proximity as our distance measure, we were able to train a convolutional network without any classification labels.

Similarly, we were also able to benefit from this approach in a semi-supervised setting, where only a small subset of labels is available as demonstrated in chapter 8. Using the distance between classes as a "soft" nearest-neighbor classification probability allowed us to both classify and regularize the entropy of the class assignment. This use of metric embedding proved to create a very strong baseline for semi-supervised learning in several tasks, with a much simpler method compared to contemporary works.
We suggest that deep metric embedding may continue to prove useful in the future, as

comparative measures are easier to obtain compared to hard-labels.

9.1.2 Training with large batches

In chapter 4, we've seen how training deep networks is affected by the batch size used. We revealed that there is no inherent "generalization gap" when training in a large batch setting. Instead, we suggested that the leading cause for the performance degradation stems from the decreased number of optimization iterations when training for the same amount of epochs. Consequently, the training optimization regime requires modifications according to the batch size used, such as the learning rate, the application of batch-norm and the number of steps performed.

This discovery paved the way to many contemporary efforts aimed at leveraging the computational benefits of large batch training while keeping the generalization performance [GDG+17, YKC+18]. Insights from our work also enabled new approaches, that balance between the batch size used and accompanying hyper-parameters used for training [SKYL17]. Breaking the barrier of batch size will allow future research and application to scale by data-parallelism over many devices and computational units.

Another interesting aspect this work illuminates is the fact the generalization performance of deep models is not disjoint from the optimization procedure we use. This connection complies with recent findings [SHN+18] showing that the optimization procedure itself serves as an implicit bias to the performance of the final model.

9.1.3 The marginal value of the last classification layer

In chapter 5, we've inspected the fully-connected classifier that is prevalent in almost all classification models and discovered that it has a very mild effect on the performance of these networks. By fixing the last layer to be a random orthogonal matrix, we showed that the layers preceding the final classifier can be accounted for most of the model performance. We further leveraged the use of a fixed classifier and suggested using a Hadamard matrix as its replacement, allowing memory and computational savings at both training and inference time.
Our discovery, beyond allowing computational benefits from keeping the last layer fixed, also sheds light on these models as a whole. It allowed succeeding research to focus on layers preceding the classifier [JG18], as well as to adopt new methods for transfer learning [MSZH].

A strong connection can also be drawn from our work on metric learning we described in Chapter 3 and that of the fixed classifier, as both endeavors leave the representation learning entirely at the hands of the convolutional layers, with similar results. This connection implies that we should continue to search for alternative model struc-

tures and training procedures, that will replace the apparent redundancy in current choices.

9.1.4 The role of batch-normalization

In chapter 6, we've examined the popular batch normalization layer. We suggested that batch-norm is not a statistical tool to reduce "internal covariate shift", as originally thought. Instead, we viewed batch-norm as regularization, functioning by introducing an invariant to the norm of the network's weights. This alternative view implied that weight-decay, also commonly used in training, is stripped from its original purpose to control the norm of the weights and instead serves only as a learning-rate modulation factor.

Additionally, understanding that batch-norm is constricting the network by the L^2 norm allowed us to offer alternative normalization schemes. Using L^1 and L^∞ norms instead made it possible to train models to the same accuracy using fewer computations and in reduced numerical precision. Furthermore, we were able to improve weight-normalization, a significantly more computationally appealing approach, which prior to our work displayed inferior performance to that of batch-norm.

Insights from this work continue to make a substantial impact, by allowing to replace batch-norm with other forms of training [ZDM19] and enabling the use of low-precision computations [BHHS18].

9.2 Open questions

9.2.1 How optimization affects generalization?

One apparent issue, appearing in our works and of contemporary others, is the visible tension between the generalization capabilities of neural networks and the capability of optimizing them to fit a desired training objective. For example, it was noted that better optimization algorithms devised lately have improved optimization speed and reduced final objective loss, but frequently resulted in diminished generalization capability [WRS+17, KS17]. As such, the use or misuse of optimization methods such as SGD should be considered as an integral part of the inductive bias of the trained model.

This connection should be explored further, to uncover the cause and properties of the implicit bias inflicted by optimization methods. Understanding these may allow us to both improve the speed and sample complexity needed to train deep models, as well as to increase generalization by introducing explicit regularization.

9.2.2 Can we decouple expressiveness from structure?

Another apparent question that arises from our works is that of expressiveness versus structure in deep models. We observed that in many cases the number of model parameters has negligible importance compared to the structure of these models. This property was most apparent in our work targeting the final classifier, where it accounted for a considerable part of models' parameters, but removing it resulted in only a marginal accuracy decrease. Recent trends in neural network structures such as residual connections [HZRS16], separable convolutions [HZC+17] and attention [BCB14] also illustrate the fact that structure, rather than the sheer amount of parameters, is more crucial to the performance of deep models. We believe that an essential yet not explored question is whether we can decouple the structure of networks from their use of trained parameters. This decoupling will allow us to investigate the root causes that differentiate one deep model from another and to design better models with less computational redundancy.

9.2.3 How do model attributes affect one another?

Throughout our research, we've witnessed several model and algorithm attributes conflict and influence each other. For example, in chapter 4 we've seen that the size of the batch used in training may require a modification of the learning rate, the duration of training and the use of batch-norm. Similarly, in 6, we've seen that there is a relationship between the use of batch-norm, weight decay and the learning rate used with a direct consequence to their function and necessity.
We conjecture that similar symbiosis resides between other hyper-parameters and methods used in modern deep networks. For example, we've seen in chapter 5 that the temperature used in the softmax layer has a determinant effect of the convergence speed and accuracy of the final model. The dynamics of this temperature is, in turn, affected by the learning rate used, as well as regularization and architecture choices (such as the use of residual layers or batch-norm).

These influence between co-existing design choices of deep learning models can be elusive to spot and hard to untangle once observed. The use of hyper-parameter search methods and heuristics further complicates this phenomenon, while possibly having a considerable impact on the models we use. We believe that more connections will be made in the future and might render some of the practices today irrelevant.

9.2.4 What more are we missing?

Finally, we bring the readers attention to the fact that many of our findings were in contradiction, at the time of their publication, with common practices and perceptions of the function of deep networks. The fast-paced nature of deep learning research, along

with a still lacking theoretical understanding, leads to reliance on heuristics and past efforts by others. Despite the impressive progress done in the past several years, we feel that substantial foundational work is yet to be done. Furthermore, in continuation with our work, we believe current practices and design choices of deep networks and their training need to be questioned and examined more thoroughly.

Appendix

Models used in Spatial contrasting

Table 1: Convolutional models - (feature-maps, kernel, stride, padding) for each layer. Convolutional layers are each followed by ReLU and Batch-norm.

MNIST	Cifar-10
Input: 28×28 monochrome	Input: 32×32 RGB
Conv-ReLU-BN (16, 5x5, 1x1, 1x1)	Conv-ReLU-BN (192, 5x5, 1x1, 2x2)
Max-Pooling (2x2, 2x2)	Conv-ReLU-BN (160, 1x1, 1x1)
Conv-ReLU-BN (32, 3x3, 1x1, 1x1)	Conv-ReLU-BN (96, 1x1, 1x1)
Conv-ReLU-BN (64, 3x3, 1x1, 1x1)	Max-Pooling (3x3, 2x2)
Conv-ReLU-BN (64, 3x3, 1x1, 1x1)	Conv-ReLU-BN (96, 5x5, 1x1, 2x2)
Max-Pooling (2x2, 2x2)	Conv-ReLU-BN (192, 1x1, 1x1)
Conv-ReLU-BN (128, 3x3, 1x1, 1x1)	Conv-ReLU-BN (192, 1x1, 1x1)
Avg-Pooling (6x6, 1x1)	Max-Pooling (3x3, 2x2)
	Conv-ReLU-BN (192, 3x3, 1x1, 1x1)
	Conv-ReLU-BN (192, 1x1, 1x1)
	Avg-Pooling (7x7, 1x1)

Bibliography

[ASE17] Antreas Antoniou, Amos Storkey, and Harrison Edwards. Data augmentation generative adversarial networks. *arXiv preprint arXiv:1711.04340*, 2017.

[BBB+93] Jane Bromley, James W Bentz, Léon Bottou, Isabelle Guyon, Yann LeCun, Cliff Moore, Eduard Säckinger, and Roopak Shah. Signature verification using a "siamese" time delay neural network. *International Journal of Pattern Recognition and Artificial Intelligence*, 7(04):669–688, 1993.

[BCB14] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[BHHS18] Ron Banner, Itay Hubara, Elad Hoffer, and Daniel Soudry. Scalable methods for 8-bit training of neural networks. 2018.

[BJTM16] Vassileios Balntas, Edward Johns, Lilian Tang, and Krystian Mikolajczyk. Pn-net: Conjoined triple deep network for learning local image descriptors. *arXiv preprint arXiv:1601.05030*, 2016.

[BKH16] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[CHL05] Sumit Chopra, Raia Hadsell, and Yann LeCun. Learning a similarity metric discriminatively, with application to face verification. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 539–546. IEEE, 2005.

[CJLV16] William Chan, Navdeep Jaitly, Quoc Le, and Oriol Vinyals. Listen, attend and spell: A neural network for large vocabulary conversational speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pages 4960–4964. IEEE, 2016.

[CKF11] Ronan Collobert, Koray Kavukcuoglu, and Clément Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, number EPFL-CONF-192376, 2011.

[CN12] Adam Coates and Andrew Y Ng. Learning feature representations with k-means. In *Neural Networks: Tricks of the Trade*, pages 561–580. Springer, 2012.

[CNL11] Adam Coates, Andrew Y Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In *International Conference on Artificial Intelligence and Statistics*, pages 215–223, 2011.

[CS16] Nadav Cohen and Amnon Shashua. Inductive bias of deep convolutional networks through pooling geometry. *arXiv preprint arXiv:1605.06743*, 2016.

[CSS16] Nadav Cohen, Or Sharir, and Amnon Shashua. On the expressive power of deep learning: A tensor analysis. In *Conference on Learning Theory*, pages 698–728, 2016.

[CSZ09] Olivier Chapelle, Bernhard Scholkopf, and Alexander Zien. Semi-supervised learning (chapelle, o. et al., eds.; 2006)[book reviews]. *IEEE Transactions on Neural Networks*, 20(3):542–542, 2009.

[CV95] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[CZM+18] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation policies from data. *arXiv preprint arXiv:1805.09501*, 2018.

[DCLT18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[DSRB14] Alexey Dosovitskiy, Jost Tobias Springenberg, Martin Riedmiller, and Thomas Brox. Discriminative unsupervised feature learning with convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 766–774, 2014.

[DT17] Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.

[EBKS+96] Jeff Elman, Elizabeth Bates, Annette Karmiloff-Smith, Mark Johnson, Domenico Parisi, and Kim Plunkett. *Rethinking Innateness: A Connectionist Perspective on Development*. MIT Press, Cambridge, MA, 1996.

[FZK+16] Jiashi Feng, Tom Zahavy, Bingyi Kang, Huan Xu, and Shie Mannor. Ensemble robustness of deep learning algorithms. *arXiv preprint arXiv:1602.02389*, 2016.

[GAG+17] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional sequence to sequence learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1243–1252. JMLR. org, 2017.

[GB04] Yves Grandvalet and Yoshua Bengio. Semi-supervised learning by entropy minimization. In *Advances in neural information processing systems*, pages 529–536, 2004.

[GCB12] Ian J. Goodfellow, Aaron Courville, and Yoshua Bengio. Large-scale feature learning with spike-and-slab sparse coding. 2012.

[GDG+17] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

[GH10] Michael Gutmann and Aapo Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *International Conference on Artificial Intelligence and Statistics*, pages 297–304, 2010.

[GHRS04] Jacob Goldberger, Geoffrey E Hinton, Sam T Roweis, and Ruslan Salakhutdinov. Neighbourhood components analysis. In *Advances in neural information processing systems*, pages 513–520, 2004.

[GLL18] Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V Le. Dropblock: A regularization method for convolutional networks. In *Advances in Neural Information Processing Systems*, pages 10750–10760, 2018.

[GPAM+14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2672–2680, 2014.

[GPSW17] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *International Conference on Machine Learning*, pages 1321–1330, 2017.

[GSS15] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. 2015.

[HA15a] Elad Hoffer and Nir Ailon. Deep metric learning using triplet network. In *Similarity-Based Pattern Recognition*, pages 84–92. Springer, 2015.

[HA15b] Elad Hoffer and Nir Ailon. Deep metric learning using triplet network. In *Similarity-Based Pattern Recognition*, pages 84–92. Springer, 2015.

[Hin07] Geoffrey E Hinton. To recognize shapes, first learn to generate images. *Progress in brain research*, 165:535–547, 2007.

[Hoc91] Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, 91(1), 1991.

[Hor91] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[Hui13] Ka Y Hui. Direct modeling of complex invariances for visual object features. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 352–360, 2013.

[HW62] David H Hubel and Torsten N Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160(1):106–154, 1962.

[HZC+17] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[HZRS15a] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[HZRS15b] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[IS15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of The 32nd International Conference on Machine Learning*, pages 448–456, 2015.

[JG18] Daniel Jakubovitz and Raja Giryes. Improving dnn robustness to adversarial attacks using jacobian regularization. *arXiv preprint arXiv:1803.08680*, 2018.

[JKRL09] Kevin Jarrett, Koray Kavukcuoglu, Marc'Aurelio Ranzato, and Yann LeCun. What is the best multi-stage architecture for object recognition? In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 2146–2153. IEEE, 2009.

[KB14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[KCR+16] BG Kumar, Gustavo Carneiro, Ian Reid, et al. Learning local image descriptors with deep siamese and triplet convolutional networks by minimising global loss functions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5385–5394, 2016.

[KH09] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. *Computer Science Department, University of Toronto, Tech. Rep*, 2009.

[KMK+16] David Krueger, Tegan Maharaj, János Kramár, Mohammad Pezeshki, Nicolas Ballas, Nan Rosemary Ke, Anirudh Goyal, Yoshua Bengio, Aaron Courville, and Chris Pal. Zoneout: Regularizing rnns by randomly preserving hidden activations. *arXiv preprint arXiv:1606.01305*, 2016.

[KS17] Nitish Shirish Keskar and Richard Socher. Improving generalization performance by switching from adam to sgd. *arXiv preprint arXiv:1712.07628*, 2017.

[KSH12a] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. *Advances In Neural Information Processing Systems*, pages 1–9, 2012.

[KSH12b] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[LBBH98] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[LBD+89] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[LBH15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[LCY13a] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

[LCY13b] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *ICLR2014*, 2013.

[LPM15] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

[LSD15] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[LWK17] Christos Louizos, Max Welling, and Diederik P Kingma. Learning sparse neural networks through l_0 regularization. *arXiv preprint arXiv:1712.01312*, 2017.

[LWY+17] Weiyang Liu, Yandong Wen, Zhiding Yu, Ming Li, Bhiksha Raj, and Le Song. Sphereface: Deep hypersphere embedding for face recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, page 1, 2017.

[MG15] James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, pages 2408–2417, 2015.

[MK13] Andriy Mnih and Koray Kavukcuoglu. Learning word embeddings efficiently with noise-contrastive estimation. In *Advances in Neural Information Processing Systems*, pages 2265–2273, 2013.

[MKS+15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[MMK+15] Takeru Miyato, Shin-ichi Maeda, Masanori Koyama, Ken Nakae, and Shin Ishii. Distributional smoothing by virtual adversarial examples. *stat*, 1050:2, 2015.

[MP43] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[MSC+13] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[MSZH] Pramod Kaushik Mudrarkarta, Mark Sandler, Andrey Zhmoginov, and Andrew Howard. Parameter-efficient transfer and multitask learning.

[Nes83] Yurii Nesterov. A method of solving a convex programming problem with convergence rate o (1/k2). 1983.

[Ng11] Andrew Ng. Sparse autoencoder. 2011.

[NVL+15] Arvind Neelakantan, Luke Vilnis, Quoc V Le, Ilya Sutskever, Lukasz Kaiser, Karol Kurach, and James Martens. Adding gradient noise improves learning for very deep networks. *arXiv preprint arXiv:1511.06807*, 2015.

[OAGR18] Myle Ott, Michael Auli, David Granger, and Marc'Aurelio Ranzato. Analyzing uncertainty in neural machine translation. *arXiv preprint arXiv:1803.00047*, 2018.

[PCD15] Pedro O Pinheiro, Ronan Collobert, and Piotr Dollar. Learning to segment object candidates. In *Advances in Neural Information Processing Systems*, pages 1981–1989, 2015.

[PKHH14] Tom Le Paine, Pooya Khorrami, Wei Han, and Thomas S Huang. An analysis of unsupervised pre-training in light of recent advances. *arXiv preprint arXiv:1412.6597*, 2014.

[PTC+17] Gabriel Pereyra, George Tucker, Jan Chorowski, Łukasz Kaiser, and Geoffrey Hinton. Regularizing neural networks by penalizing confident output distributions. *arXiv preprint arXiv:1701.06548*, 2017.

[RASC14] Ali Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. Cnn features off-the-shelf: an astounding baseline for recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 806–813, 2014.

[RBH+15] Antti Rasmus, Mathias Berglund, Mikko Honkala, Harri Valpola, and Tapani Raiko. Semi-supervised learning with ladder networks. In *Advances in Neural Information Processing Systems*, pages 3532–3540, 2015.

[RDS+15] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[RHW] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1.

[RHW86] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors, 1986.

[RM51] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[RMC15] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[RPDB15] Oren Rippel, Manohar Paluri, Piotr Dollar, and Lubomir Bourdev. Metric learning with adaptive density discrimination. *stat*, 1050:18, 2015.

[RPK+17] Maithra Raghu, Ben Poole, Jon Kleinberg, Surya Ganguli, and Jascha Sohl Dickstein. On the expressive power of deep neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2847–2854. JMLR. org, 2017.

[SGZ+16] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. *arXiv preprint arXiv:1606.03498*, 2016.

[SHK+14] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[SHN+18] Daniel Soudry, Elad Hoffer, Mor Shpigel Nacson, Suriya Gunasekar, and Nathan Srebro. The implicit bias of gradient descent on separable data. *The Journal of Machine Learning Research*, 19(1):2822–2878, 2018.

[Sie95] Hava T Siegelmann. Computation beyond the turing limit. *Science*, 268(5210):545–548, 1995.

[SK16] Tim Salimans and Durk P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems*, pages 901–909, 2016.

[SKP15] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 815–823, 2015.

[SKYL17] Samuel L Smith, Pieter-Jan Kindermans, Chris Ying, and Quoc V Le. Don't decay the learning rate, increase the batch size. *arXiv preprint arXiv:1711.00489*, 2017.

[SLJ+15] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[Spr16] Jost Tobias Springenberg. Unsupervised and semi-supervised learning with categorical generative adversarial networks. In *International Conference on Learning Representations (ICLR)*. 2016.

[SPW+18] Jonathan Shen, Ruoming Pang, Ron J Weiss, Mike Schuster, Navdeep Jaitly, Zongheng Yang, Zhifeng Chen, Yu Zhang, Yuxuan Wang, Rj Skerrv-Ryan, et al. Natural tts synthesis by conditioning wavenet on mel spectrogram predictions. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4779–4783. IEEE, 2018.

[SSS+17] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

[STE13] Christian Szegedy, Alexander Toshev, and Dumitru Erhan. Deep neural networks for object detection. In *Advances in neural information processing systems*, pages 2553–2561, 2013.

[SVI+15] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *arXiv preprint arXiv:1512.00567*, 2015.

[SVI+16] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[SWL18] Leon Sixt, Benjamin Wild, and Tim Landgraf. Rendergan: Generating realistic labeled data. *Frontiers in Robotics and AI*, 5:66, 2018.

[TGH15] Isaac Triguero, Salvador García, and Francisco Herrera. Self-labeled techniques for semi-supervised learning: taxonomy, software and empirical study. *Knowledge and Information Systems*, 42(2):245–284, 2015.

[TPC+17] Toan Tran, Trung Pham, Gustavo Carneiro, Lyle Palmer, and Ian Reid. A bayesian data augmentation approach for learning deep models. In *Advances in Neural Information Processing Systems*, pages 2797–2806, 2017.

[UVL18] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Deep image prior. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9446–9454, 2018.

[Vap92] Vladimir Vapnik. Principles of risk minimization for learning theory. In *Advances in neural information processing systems*, pages 831–838, 1992.

[vdODZ+] Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. In *9th ISCA Speech Synthesis Workshop*, pages 125–125.

[VLBM08] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM, 2008.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

[VV98] Vladimir Naumovich Vapnik and Vlamimir Vapnik. *Statistical learning theory*, volume 1. Wiley New York, 1998.

[WH18] Yuxin Wu and Kaiming He. Group normalization. *arXiv preprint arXiv:1803.08494*, 2018.

[WRMC12] Jason Weston, Frédéric Ratle, Hossein Mobahi, and Ronan Collobert. Deep learning via semi-supervised embedding. In *Neural Networks: Tricks of the Trade*, pages 639–655. Springer, 2012.

[WRS+17] Ashia C Wilson, Rebecca Roelofs, Mitchell Stern, Nati Srebro, and Benjamin Recht. The marginal value of adaptive gradient methods in machine learning. In *Advances in Neural Information Processing Systems*, pages 4148–4158, 2017.

[WSC+16] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus

Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[WZZ+13] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *International Conference on Machine Learning*, pages 1058–1066, 2013.

[XGD+17] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1492–1500, 2017.

[YKC+18] Chris Ying, Sameer Kumar, Dehao Chen, Tao Wang, and Youlong Cheng. Image classification at supercomputer scale. *arXiv preprint arXiv:1811.06992*, 2018.

[YLL+15] Shuo Yang, Ping Luo, Chen Change Loy, Kenneth W Shum, and Xiaoou Tang. Deep representation learning with target coding. 2015.

[Zag16] Komodakis Zagoruyko. Wide residual networks. In *BMVC*, 2016.

[ZCDLP17] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017.

[ZDM19] Hongyi Zhang, Yann N. Dauphin, and Tengyu Ma. Residual learning without normalization via better initialization. In *International Conference on Learning Representations*, 2019.

[ZKTF10] Matthew D Zeiler, Dilip Krishnan, Graham W Taylor, and Rob Fergus. Deconvolutional networks. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2528–2535. IEEE, 2010.

[ZMGL15a] Junbo Zhao, Michael Mathieu, Ross Goroshin, and Yann Lecun. Stacked what-where auto-encoders. *arXiv preprint arXiv:1506.02351*, 2015.

[ZMGL15b] Junbo Zhao, Michael Mathieu, Ross Goroshin, and Yann Lecun. Stacked what-where auto-encoders. *arXiv preprint arXiv:1506.02351*, 2015.

[ZMMAP16] Francisco Zamora-Martínez, Javier Muñoz-Almaraz, and Juan Pardo. Integration of unsupervised and supervised criteria for deep neural networks training. In *International Conference on Artificial Neural Networks*, pages 55–62. Springer, 2016.

[ZZL15] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, pages 649–657, 2015.

פרמטרים נלמדים, אשר תחליף את המבנה הנפוץ. נראה איך שימוש בשכבה קבועה מאפשר להגיע לביצועי הכללה שקולים, עם חיסכון משמעותי בגודלו הסופי של המודל. בנוסף, נציע מבנה נוסף של מסווג המבוסס על מטריצת האדאמרד, שבה רק האיברים ± 1 ותאפשר חיסכון נוסף בחישובים הדרושים לסיווג ברשת.

מבנה נפוץ ברשתות עמוקות מודרניות הוא "נרמול מקבץ" (batch-normalization) אשר תפקידו לנרמל את הערכים בכל שכבה כך שיהיו בעלי ממוצע 0 ושונות 1. השימוש במבנה זה אפשר שימוש ברשתות עמוקות יותר מבעבר, ולשיפור משמעותי בזמן האימון וכן בביצועיי ההכללה של רשת לאחר שאומנה. במקור, נטען כי מקור השיפור של נרמול זה הוא בכך שהוא מאפשר לרשת להתמודד בצורה טובה עם השינוי בפילוג של הערכים בין שכבות הרשת, אשר קורה במהלך תהליך האימון. בעבודה זו נציע סיבה חלופית לשיפור המתקבל מ"נרמול מקבץ", אשר מתאפיינת בהגבלת המשקולות ברשת כך שמוצא כל שכבה יישמר ללא השפעה של הנורמה האוקלידית שלהן. נראה שהשימוש בנרמול המקבץ גורם לכך ששיטות רגולריזציה שתפקידן להגביל את נורמת המשקולות משנות את תפקידן המקורי, ומהוות רק אפנון של קצב הלמידה של הרשת.

מכיוון שתחת הפרשנות החלופית שהצענו, הנרמול מבצע הגבלה לפי נורמה אוקלידית, נראה כי ניתן להשתמש בנורמות אחרות במקומה. שימוש בנורמות אחרות יאפשר אימון בדיוק נומרי נמוך יותר, וכן יאפשר חסכון במשאבי מחשוב וזכרון. בנוסף, נציע שיפור לשיטת נרמול אחרת – "נרמול משקולות" (weight-normalization) אשר לה יתרון חישובי משמעותי בכך שאין תלויה בגודל המקבץ בו נעשה שימוש. באמצעות הגבלת הנורמה, נראה כי הפער המשמעותי בינה ובין נרמול המקבץ מצטמצם בצורה משמעותית.

רגולרזיציה – אשר באמצעות אילוץ מפורש, הוספה של רעש לתהליך האימון או באמצעים אחרים, מאפשרות לשפר את יכולת ההכללה של הרשת.

מבנים מסויימים של רשתות ידועים במיוחד ביכולת הלמידה ובביצועים שלהם על בעיות נתונות, ע"י התאמה מובנית לתחום הנלמד. הדוגמא הבולטת למודל מסוג זה הוא "רשתות קונבולוציה" (Convolutional networks) אשר מורכב משכבות הפועלות בצורה מקומית על מרחב הקלט, תוך שיתוף המשקולות ע"פ אזורים שונים. רשתות אלו מותאמות במיוחד למידע בעל אופי סטציונרי כתמונות וקול. בעבודה זו נציג מחקר במספר מוקדים שמאתגרים ומוסיפים הבנה מחודשת של שיטות נפצות במודלים של למידה עמוקה. ממצאי המחקר ידגימו כי מספר שיטות והיורסטיקות קיימות לתכנון, אימון והסקה באמצעות רשתות עמוקות הן שגויות, ויוצעו עבורן פתרונות חלופיים.

רשתות עמוקות ידועות ביכולתן ללמוד ייצוגים של מידע, אך לרוב ייצוג זה אינו נלמד באופן מפורש, אלא כתוצר משני של אימון לסיווג דוגמאות למחלקות. אנו נראה שניתן לאמן רשתות באופן מפורש לייצר ייצוגים של מידע ע"י דרישה לשימור מרחק - כך שדוגמאות הרחוקות יותר מבחינה סמנטית, יהיו רחוקות יותר גם במרחב האוקלידי שבמוצא הרשת.
לצורך זה נציג את "רשת השלשות" ("Triplet Network") ופונקציית מטרה חדשה, אשר יאפשרו ללמוד מרחקים יחסיים בין דוגמאות המוזנות לרשת. נציג אמפירית כיצד המודל שלמדנו לומד ייצוגים הניתנים לפרשנות מבחינה סמנטית, וזאת גם עם מידע מוצומצם לגבי מרחב הבעיה והדמיון הצפוי בין מחלקות שונות בסט האימון. נדגים גם איך שיטה זו מאפשרת למידה גם במקרים בהם אין תיוג מפורש של הדוגמאות (למידה שאינה מפוקחת), ובמקרים בהם תיוג זה דליל מאוד (למידה מפוקחת למחצה).

בתהליך האימון של רשתות עמוקות ישנו לרוב שימוש במקבץ של דוגמאות (batch) בכל צעד אופטימיזציה שמבוצע. שימוש במקבץ זה מאפשר האצה של משך האימון על-ידי מקבול החישוב על-פני מספר גדול של יחידות עיבוד, ועל כן חשובה מאוד ליכולת לעשות שימוש במודלים גדולים ובמקורות מידע המתרחבים משנה לשנה. למרות הרצון להגדיל את גודל המקבץ, ובכך לשפר את ביצועיי האימון של הרשת, פעולה זה לוותה בעבר בפגיעה ביכולת ההכללה של הרשת – הדיוק המתקבל על דוגמאות שלא נראו בסט האימון. פגיעה זו, שכונתה "פער ההכללה", נתפסה בעבר כתכונה מובנית של אימון בשימוש במקבץ גדול, ולכן היוותה מכשול עיקרי ביכולת להגדיל את גודל המודלים וכמות המידע בו משתמשים לאמן. בעבודה זו נראה באמצעות טיעונים איכותיים וכמותיים כי "פער ההכללה" אינו נובע מתכונות המודל, אלא מאופן האימון שלו. נדגים כי מקור הפער הוא בכך שבאימון עם מקבץ גדול מספר צעדי האופטימיזציה קטן בהתאם. בנוסף, משיקולים סטטסיטיים נציע מספר שינויים בתהליך האימון ובמשך האימון אשר יגשרו על הפער ואף יובילו למודלים עם ביצועי הכללה משופרים.

רשתות עמוקות שתפקידן לסווג למספר ידוע של מחלקות בנויות כך שבסופן שכבת סיווג לינארית. שכבה זו מבצעת הטלה לינארית בין ייצוג הרשת לוקטור אשר מימדו הוא מספר המחלקות המסווגות. משום כך, מספר הפרמטרים אשר שכבה זו עושה בהם שימוש גדל ביחס ישר למספר המחלקות, ולעיתים קרובות מהווה חלק גדול ממספר הפרמטרים הכולל ברשת וכמות החישובים הנדרשים בכל סיווג.
בעבודה זו נטען כי לשכבה זו יש יכולת ייצוג מוגבלת ואף זניחה ביחס ליתר השכבות ברשת, ואת על אף כמות הפרמטרים הגדולה בה היא עושה שימוש. נדגים זאת ע"י שימוש בשכבה קבועה, ללא

תקציר

השימוש ברשתות עמוקת ללמידת מכונה מחולל בשנים האחרונות שינוי מהותי במספר רב של תחומים כגון ראייה ממוחשבת, זיהוי-קול, והבנת שפה טבעית. במרכז המהפכה נמצאות רשתות עצביות מלאכותיות, מבנים שנחקרו כבר באמצע המאה שעברה, אך עם השיפור במשאבי החישוב והמידע הנגישים, חזרו לקדמת תחום הלמידה.

רשתות עמוקות הן מבנים המורכבים מרצף של שכבות המופרדות באמצעות פונקציות לא-לינאריות. לכל שכבה סט של פרמטרים נלמדים (משקולות) אשר קביעתם מאפשרת הוצאת מידע רצוי בעבור הקלט המתקבל לרשת. אימון רשתות אלו מתבצע ע"י למידה מקצה-אל-קצה של פונקציית המטרה, כאשר בכל פעם מוזנות דוגמאות אל הרשת, והשגיאה בסופן מחושבת עבור כל המשקולות באמצעות הפעלה של כלל השרשרת בתהליך הידוע כ"פעפוע לאחור" (Back-propagation). לאחר מכן, עדכון המשקולות מתבצע ע"י צעד בכיוון הנגזרת ביחס לשגיאה.

על אף ההצלחה האמפירית של הרשתות העמוקות, שאלות רבות נותרו פתוחות בנוגע ליסודות התיאורטיים עליהן הן מבוססות. את שאלות אלו ניתן לקטלג לשלוש קטגוריות עיקריות:

- יכולת הייצוג של רשתות – ידוע כי רשתות עמוקות הן "מקרב אוניברסלי" ומסוגלות לקרב כל פונקצייה עד לדיוק רצוי, תחת מגבלות מסויימות. למרות זאת, לרוב לא ידוע מה מבנה הרשת הדרוש לקירוב פונקצייה נתונה בצורה אופטימלית.
- אופטימיזציה של רשתות – היכולת לכוון את המשקולות של רשת נתונה תחת פונקציית מטרה כלשהי כך שתושג שיטת אימון נמוכה תוך זמן קצר ככל האפשר דגש מיוחד מושם לעובדה שרשתות יחד עם פונקציית המטרה הן לרוב פונקציות לא-קמורות, ולכן לא ניתן להבטיח את קצב התכנסות או היתכנתו.
- יכולת ההכללה של רשתות – היכולות של רשתות להפגין ביצועים טובים גם על דוגמאות שלא נראו במהלך האימון. תכונה זו בולטת במיוחד בשימוש ברשתות ל"העתקת למידה" (transfer learning) שבו רשת שאומנה על משימה אחת משמשת לפתור משימה אחרת בעלת מאפיינים משותפים.

מנקודת המבט של למידה סטטיסטית, מבנה רשת מגדיר מחלקה של היפותזות, אשר אימון הרשת מביא לבחירה של היפותזה אחת מתוכן ע"י התאמת הפרמטרים הנתונים. רשתות מודרניות הן לרוב בעלות מספר גדול של פרמטרים – לעיתים סדרי גודל יותר ממספר הדוגמאות עליהן הן מאומנות, ועל כן ישנו צורך להגביל בצורה אחרת את מרחב ההיפותזות הקיים. הגבלה זו מתבצעת ע"י שיטות

המחקר בוצע בפקולטה להנדסת חשמל בהנחייתו של פרופסור דניאל סודרי ופרופסור ניר אילון מהפקולטה למדעי המחשב.

חלק מן התוצאות בחיבור זה פורסמו כמאמרים מאת המחבר ושותפיו למחקר בכנסים ובכתבי-עת במהלך תקופת מחקר הדוקטורט של המחבר, אשר גרסאותיהם העדכניות ביותר הינן:

Elad Hoffer and Nir Ailon. Deep metric learning using triplet network. In *International Workshop on Similarity-Based Pattern Recognition*, pages 84–92. Springer, 2015.

Elad Hoffer, Ron Banner, Itay Golan, and Daniel Soudry. Norm matters: efficient and accurate normalization schemes in deep networks. In *Advances in Neural Information Processing Systems*, 2018.

Elad Hoffer, Itay Hubara, and Daniel Soudry. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. In *Advances in Neural Information Processing Systems*, pages 1731–1741, 2017.

Elad Hoffer, Itay Hubara, and Daniel Soudry. Fix your classifier: the marginal value of training the last weight layer. *International Conference For Learning Representations*, 2018.

תודות

אני רוצה להודות למנחים שלי, פרופ׳ דניאל סודרי ופרופ׳ ניר אילון, על הכוונתם ותרומתם האישית והמקצועית להצלחתי. תודה לשותפיי בכתיבת המאמרים: איתי חוברה, רון באנר ואיתי גולן, על שיתוף פעולה פורה ומהנה.
אני רוצה להודות למשפחתי היקרה: להורי, אורי ואביבה, שהעניקו לי את הרצון ללמוד וליצור, ותמכו בי בכל שפניתי. תמיד תהיו עבורי דוגמא לחיקוי. לאחיי ברק ונועה, שמהווים תמיד מקור לגאווה ואני שמח שהם חלק מחיי. לסבי וסבותיי: מיקי ופלביה, חנה והרי ,שלכל אחד מהם חלק בהישג זה.

לבסוף, אבקש להודות לאשתי האהובה נירית, על תמיכתה ושותפותה יחד עם בניי כרמל ושקד, שהצטרפו לחיינו במהלך לימודיי.

אני מודה לטכניון על התמיכה הכספית הנדיבה בהשתלמותי.

למידה עמוקה: בחינה מחודשת של שיטות

חיבור על מחקר

לשם מילוי חלקי של הדרישות לקבלת התואר
דוקטור לפילוסופיה

אלעד הופר

הוגש לסנט הטכניון — מכון טכנולוגי לישראל
ניסן התשע"ט חיפה אפריל 2019

למידה עמוקה: בחינה מחודשת של שיטות

אלעד הופר