UNIVERSITY OF TÜBINGEN

MASTER THESIS

Dissecting BatchNorm: An Ablation Study on the Core Components

Author:
Vanessa TSINGUNIDIS

Supervisor:
Andrés FERNÁNDEZ

Examiner:
Prof. Dr. Philipp HENNIG

Second Examiner:
Prof. Dr. Andreas GEIGER

A thesis submitted in fulfillment of the requirements for the degree of Master of Science (M.Sc.)

in the

Methods of Machine Learning Group
Department of Computer Science

October 30, 2023

Declaration of Authorship

I, Vanessa TSINGUNIDIS, declare that this thesis titled, "Dissecting BatchNorm: An Ablation Study on the Core Components" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

UNIVERSITY OF TÜBINGEN

Abstract

Department of Computer Science

Master of Science (M.Sc.)

Dissecting BatchNorm: An Ablation Study on the Core Components

by Vanessa TSINGUNIDIS

Successfully training Deep Neural Networks (DNNs) is a complex task, primarily due to their inherent intricacy and the multitude of factors that influence their performance. Various methodologies have been introduced to aid this training process, among which Batch Normalization (BatchNorm) has gained considerable attention. While the advantages of BatchNorm are recognized, a clear consensus on the specific mechanisms driving these effects is still a subject of discussion, particularly concerning the role and necessity of the learnable parameters, γ, and $\boldsymbol{\beta}$.
This thesis seeks to shed light on the contributions of these parameters within the BatchNorm method. Through a targeted ablation study, this work examines the distinct contributions of the learnable parameters in conjunction with the non-adaptive normalization. This entails evaluating their impact on model performance along with analyzing the evolution of γ and $\boldsymbol{\beta}$ throughout the training process across different layers.
The empirical investigations in this work indicate that the non-adaptive normalization in BatchNorm primarily improves model performance. The learnable parameters could not significantly influence the training performance in the experiments with the smaller convolutional architecture. However, in the context of the deeper Convolutional Neural Netowk (CNN), both γ and $\boldsymbol{\beta}$ were observed to affect the convergence speed and model accuracy significantly. Moreover, γ and β show changes in their relative contribution to the training performance, depending on the depth of the architecture and the position of the BatchNorm layers in the network. Our results suggest that the success of BatchNorm cannot be attributed solely to the non-adaptive normalization, or the learnable parameters, γ and β.

Acknowledgements

Firstly, I would like to thank my supervisor, Andrés Fernández, for his guidance, patience, and consistent availability for many spontaneous chats, especially toward the end. Also, I'd like to express my gratitude for your deep engagement in the topic and for always providing ideas.
In addition, I want to thank Frank Schneider for the brainstorming sessions and constructive feedback during the process.
Also, a big thanks to Philipp Hennig and the people in his lab, in particular, for fostering a comforting and supportive environment, which is not at all given but a result of their personal efforts.
Lastly, (although they probably won't read this) infinite thanks to my friends and family for your emotional support throughout writing this thesis and during the whole period of completing this master's degree.

Contents

List of Figures

List of Tables

List of Abbreviations

List of Symbols

$f_{\boldsymbol{\theta}}$	model prediction function
$\boldsymbol{\theta}$	model weights
Θ	weights space
$\mathbf{x}_i$	instance of input features
X	random variable for input features
$\mathcal{X}$	input space
$\mathbf{y}_i$	instance of label for input features
$\hat{\mathbf{y}}$	predicted output
$\dot{\mathbf{y}}_i$	instance of label one-hot encoding
Y	random variable for output labels
$\mathcal{Y}$	output space
$\mathbf{b}$	bias term
$\mathbf{h}$	hidden layer activation
ϕ	activation function
$(\mathbf{x}_i, \mathbf{y}_i)$	sample of input and label pair
$\mathcal{D}$	training distribution
C	number of classes
D	training dataset
$\mathcal{L}$	loss term
$\mathcal{L}_{CE}$	cross-entropy loss term
η	learning rate
$R_{\text{erm}}(f_{\boldsymbol{\theta}})$	empirical model risk
$O_{i,j}$	feature map in a convolutional layer
K	kernel in a convolutional layer
n_{in}	input units of a fully connected layer
n_{out}	output units of a fully connected layer
U	uniform distribution
N	normal distribution
$\mathbf{d}$	binary mask vector
$\mathbf{H}$	Hessian of the loss
$\boldsymbol{\gamma}$	gamma parameter
$\boldsymbol{\beta}$	beta parameter
$BN_{\gamma\beta}$	batch normalization transformation
BN	non-adaptive batch normalization only
BN_{β}	batch normalization without $\boldsymbol{\gamma}$
BN_{γ}	batch normalization without $\boldsymbol{\beta}$
$\mathcal{B}$	mini-batch
m	size of the mini-batch
$\mu_{\mathcal{B}}$	mean of a mini-batch
$\sigma_{\mathcal{B}}^2$	variance of a mini-batch
$\tilde{\mathbf{x}}_i$	instance of normalized features
$\overline{\mu}$	running mean

$\overline{\sigma^2}$	running variance
$\mathbf{v}$	vector for the direction of $\boldsymbol{\theta}$
$\|\|\mathbf{v}\|\|$	Euclidean (ℓ_2) norm of $\mathbf{v}$
g	magnitude of $\boldsymbol{\theta}$
$\mathcal{S}_i$	group in group normalization layer
G	group number in group normalization

Chapter 1

Introduction

DNNs offer exceptional capabilities across a wide range of applications and tasks (Goodfellow et al., 2016). However, successfully training DNNs poses a complex task. The field of Deep Learning (DL) has proposed various techniques to overcome training difficulties via improved optimization (Dauphin et al., 2014; Sainath et al., 2013), weight initialization (Glorot and Bengio, 2010; He et al., 2016) and regularization methods (Srivastava et al., 2014). A significant stride was the introduction of the BatchNorm technique by Ioffe et al. (2015). BatchNorm has quickly been adopted as a standard component for training DNNs and found its way into numerous state-of-the-art models (Szegedy, Vanhoucke, et al., 2016; Amodei et al., 2016; Szegedy, Ioffe, et al., 2017; Dosovitskiy et al., 2021). While the internal computations are well known, and the research community agrees on the beneficial effects of BatchNorm (Bjorck et al., 2018; Szegedy, Vanhoucke, et al., 2016), there is no consensus regarding the precise underlying mechanisms of these effects (Santurkar et al., 2019; Cai et al., 2018; Frankle et al., 2020; Awais et al., 2020; Peerthum, 2023).

At its core, BatchNorm normalizes the output or activation of a layer to have zero mean and unit variance. This is achieved by computing the mean and the variance over a mini-batch from the training data and applying those statistics for the normalization of activations. In the context of BatchNorm, this transformation can be described as a non-adaptive normalization, which is followed by an affine transformation with two learnable parameters γ and β. During training, both parameters are adjusted by the optimizer, along with the weights of the network. The first parameter γ scales the activations, while β applies a shift on the activations (See Section 2.2.1).

BatchNorm was initially introduced with the objective of tackling the internal shift of hidden layer activations during training, an effect Ioffe et al. defined as the Internal Covariance Shift (ICS). However, subsequent studies have strongly questioned this assumption (Santurkar et al., 2019; Xu et al., 2019; Davis et al., 2021; Lewkowycz et al., 2020). Numerous other effects are attributed to the usage of BatchNorm ranging from improved convergence rates, to reduced sensitivity, towards parameter initialization (Bjorck et al., 2018; Luo et al., 2018), and potential regularization effects that reduce the need for classical regularization methods such as dropout (Ghorbani et al., 2019). Another aspect of investigation in this context, is the contribution of the respective components of BatchNorm, namely the non-adaptive normalization and the learnable parameters γ and β. The original paper introduces the learnable component under the rationale to equip the BatchNorm method with the flexibility to undo the non-adaptive normalization if it serves the optimization process. The authors assume that this is necessary if the activation function after BatchNorm has saturating regions, as is the case for the sigmoid activation function (Ioffe et al., 2015). However, the paper did not delve further into the specifics of the two learnable parameters, leaving a gap in the comprehension of their full impact.

Research has examined the role of BatchNorm's learnable parameters, with varying outcomes. Frankle et al. (2020) observed, that freezing all other weights and solely training γ and β still yields competitive performance. Training an equivalent set of randomly selected parameters elsewhere in the network failed to achieve similar outcomes. In contrast, findings by Davis et al. (2021) show that γ and β tend to remain near their initialization, indicating the parameters influence the model performance to a negligible extent. Peerthum (2023) investigate the respective contributions of the non-adaptive normalization and the learnable component on varying Residual Neural Network (ResNet) architectures. Their results show that, for ResNets using basic blocks, the non-adaptive normalization is sufficient on its own. ResNets that utilize bottleneck blocks heavily rely on the learnable parameters in BatchNorm. Peerthum (2023) argue that in these cases, γ and β introduce additional degrees of freedom for the optimizer to train the network.

The above explorations do not provide consensual findings on the role of the learnable parameters in BatchNorm. This motivates further research on the BatchNorm components in general and their impact on the beneficial effects of this method. Our work seeks to shed light on the learnable parameters γ and β in BatchNorm and their behavior over the training process. By empirically evaluating each component on different DL architectures, we aim to provide a comprehensive understanding of their individual and collective impacts on model performance. In particular, we set up an ablation study with several modifications to the original BatchNorm formula by freezing the learnable parameters γ and β respectively (Section 3.1). In addition, we design experiments that allow for a detailed investigation of γ and β at each BatchNorm layer (Section 3.3) in order to determine trends in relation to their performance contribution.

Our experiments provide a detailed analysis of the performance contribution of the individual components, showing the context-dependent importance of γ and β, based on the network architecture and the positioning of BatchNorm layers. We further provide insights into the behavior of the two parameters at each BatchNorm layer over the training process. While our experiments could not find a solid relation between the trajectory of the learnable parameters and their performance contribution, we can identify trends in the training behavior specific to the position of a BatchNorm layer in the network and in relation to other embedded BatchNorm layers. Extending from the comprehensive analysis conducted in this thesis, we outline the key contributions and research findings in the following:

- **Examination of different model architectures and datasets** to discern the varying impacts of learnable parameters on training performance. In most cases, the non-adaptive normalization is the component that primarily induces the beneficial effects of BatchNorm on the training performance. However, there are cases in which the contribution of the learnable parameters can be crucial for the method to be beneficial at all.
- **The success of BatchNorm cannot be solely attributed to one component.** The Experiments, with BatchNorm at the last layer, reveal the learnable parameters γ and β can compensate for cases in which the non-adaptive normalization hinders model performance.
- **Visualization of the trajectory of γ and β over the training process.** For both parameters γ and β, the pattern exhibited is primarily characterized by the position of the BatchNorm layer within the network architecture, irrespective of its eventual performance impact.

Chapter 2

Background

This chapter briefly introduces foundational aspects of neural network training along with architectures and methods utilized in the context of this work. The focus of this chapter is set on the BatchNorm method (Section 2.2). After providing details on the BatchNorm computations (Section 2.2.1), we delve into the discussion on the effects attributed to BatchNorm in Section 2.2.3 and the contribution of the learnable component Section 2.2.4. Lastly, this chapter provides an insight into alternative normalization techniques along with their advantages and drawbacks in comparison to BatchNorm (Section 2.2.5).

2.1 Neural Network Training

A neural network aims to learn a mapping $f_{\boldsymbol{\theta}} : \mathcal{X} \to \mathcal{Y}$ from an input $\mathbf{x} \in \mathcal{X}$ to the corresponding label or ground truth $\mathbf{y} \in \mathcal{Y}$. In simple feed-forward networks, such as the Multilayer Perceptron (MLP), this is achieved by processing the input through a sequence of layers to produce a prediction $\hat{\mathbf{y}}$. Each layer consists of multiple weights $\boldsymbol{\theta} \in \Theta$ that apply a linear transformation to the input, followed by a non-linear activation function ϕ, allowing the network to learn complex, non-linear mappings. For clarity, $\mathbf{h}$ represents the output activation of an intermediate layer, while $\hat{\mathbf{y}}$ represents the final prediction after the last layer:

$$\mathbf{h} = \phi(\boldsymbol{\theta}\mathbf{x} + \mathbf{b}) \tag{2.1}$$

with a bias $\mathbf{b}$ at each layer to introduce a learnable offset to aid the network training. The last layer of a neural network serves a specific function, namely to transform the internal representation of the input into a prediction $\hat{\mathbf{y}}_i$. For multi-class classification tasks with classes denoted by C, it is common to employ the softmax activation function. Softmax transforms the raw layer output into one-hot encoded class probabilities $\hat{\mathbf{y}}_c$, meaning only one unit of the vector is activated. The probabilities over the classes C lie in the range $[0, 1]$ such that the entire vector sums to one, which ensures the predictions $\hat{\mathbf{y}}$ align with the intended classification task and are interpretable as probabilities for decision-making and model evaluation.

In supervised learning, a set of training data $D = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{N}$ comes with pairs of inputs and corresponding labels $(\mathbf{x}_i, \mathbf{y}_i)$ to aid the decision making process. Typically, the data pairs are drawn independently and identically distributed (i.i.d.) from the underlying data distribution $\mathcal{D}$ such that the network receives representative samples of the entire data space $\mathcal{X} \times \mathcal{Y}$. The labels $\mathbf{y}$ serve as the ground truth that the model strives to predict. The difference between the ground truth $\mathbf{y}$ and the prediction $\hat{\mathbf{y}}$ is measured by the loss $\mathcal{L}(\hat{\mathbf{y}}, \mathbf{y}) \in \mathbb{R}_{\geq} 0$. By training the neural network, we seek to minimize the empirical risk over the dataset:

$$R_{\text{erm}}(f_{\boldsymbol{\theta}}) = \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}(f_{\boldsymbol{\theta}}(\mathbf{x}_i), \mathbf{y}_i) \tag{2.2}$$

Among various loss functions, the Cross-Entropy (CE) loss is a commonly used metric for multi-class classification tasks (Janocha et al., 2017) to measure the difference between the predicted probability distribution and the ground truth distribution. The CE loss is given by:

$$\mathcal{L}_{CE}(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_{c=1}^{C}\mathbf{y}_c \log(\hat{\mathbf{y}}_c) \tag{2.3}$$

where $\mathbf{y}_c$ is the ground truth probability for class $c \in C$, and $\hat{\mathbf{y}}_c$ denotes the predicted probability for a class. Minimizing this loss pushes the output of the neural network towards the ground truth probabilities.

Gradient-based optimization methods, such as Stochastic Gradient Descent (SGD), are employed to minimize the empirical risk by iteratively optimizing θ. Rather than using the entire dataset for each update, which can be computationally intensive, SGD commonly employs mini-batches, subsets of the training data. SGD updates the weights $\boldsymbol{\theta}$ and the bias $\mathbf{b}$ with a learning rate η that determines the size of the update at each iteration t.

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta\nabla_{\boldsymbol{\theta}}\mathcal{L}((f_{\boldsymbol{\theta}_t}(\mathbf{x}_i), \mathbf{y}_i)) \tag{2.4}$$

The term $\nabla_{\boldsymbol{\theta}}\mathcal{L}$ represents the gradient of $\mathcal{L}$ with respect to the parameters, including the weights $\boldsymbol{\theta}$ and the biases b. In each iteration, $\nabla_{\theta}\mathcal{L}$ points in the direction of the steepest ascent in the loss landscape. By moving into the opposite direction $-\nabla_{\theta}$, the loss can be locally minimized. With numerous iterations, this method allows a neural network to train and generalize well when subjected to unseen input data.

2.1.1 Convolutional Neural Networks

CNNs (LeCun, Bottou, Bengio, et al., 1998), have become a foundational architecture in DL, especially for images (Springenberg et al., 2015). A primary distinction of CNNs compared to traditional fully connected neural networks is their incorporation of convolutional layers to learn spatial hierarchies of features from the input. The convolutional layer operates by applying a kernel K of size q, a tensor usually small in spatial dimensionality but spreads along the entirety of the depth d of the input. The kernel is convolved across the spatial dimensionality of the input with a stride s denoting the number of pixels by which the kernel moves. The result is a 2D activation map O of size $(o \times o)$, which reduces the number of weights in the model compared to fully connected architectures. The produced activation maps are stacked along the depth dimension, each representing locations and strengths of detected features. The convolution operation transforms an input $\mathbf{x}_i$ with height h and width w of size $(h \times w \times d)$ into a size of $(o \times o \times q)$.

Although the method is technically a cross-correlation operation, it is referred to as convolution in DL. This spatial locality of convolutions makes them particularly suitable for image analysis, among others. While at the initial layers, kernels typically capture rudimentary patterns such as edges or textures, deeper layers recognize more abstract features, from shapes to intricate objects (Springenberg et al., 2015).

After the convolution operation, CNNs often employ a pooling layer to reduce the spatial dimensions of the activations, thereby further decreasing the required amount of computations and weights. Max pooling, for example, works by selecting the maximum value from a window in the activation map, thereby preserving dominant features and making the representation more robust to small translations.

2.1.2 ReLU Activation

The Rectified Linear Unit (ReLU) (Fukushima, 1975) activation function is utilized for various types of neural network architectures, including CNNs and MLPs (Dubey et al., 2022). The ReLU function is defined as:

$$\phi(x) = \max(0, x) \tag{2.5}$$

ReLU offers computational efficiency due to its simplicity. Evaluating ReLU only requires a threshold at zero, which comes without much computational and memory overhead. Unlike saturating functions like sigmoid or tanh, whose gradients tend toward zero for extreme values of $\mathbf{x}$, ReLU's gradient is either zero (for $\mathbf{x} < 0$) or one (for $\mathbf{x} > 0$), ensuring more robust and more consistent gradient signals during optimization, especially in deep networks (Glorot, Bordes, et al., 2011). However, if too many input values are below zero, ReLU leads to units that never activate, known as the "dying ReLU" problem.

2.1.3 Parameter Initialization

The choice of weight and bias initialization can have a great impact on the training of DNNs. Proper initialization ensures balanced activations across layers, aiding in faster convergence and avoiding issues like vanishing or exploding gradients during the early stages of training (Kumar, 2017). Conversely, improperly initialized weights and biases can lead to saturated activation functions at the beginning of the training, hindering the model's ability to learn.

Xavier (Glorot) Initialization. The core principle of Xavier initialization is to maintain the variance of activations consistent across layers. For a fully connected layer with n_{in} input units and n_{out} output units, the parameters are initialized according to a uniform distribution,

$$\boldsymbol{\theta} \sim U\left(-\sqrt{\frac{6}{n_{in} + n_{out}}}, \sqrt{\frac{6}{n_{in} + n_{out}}}\right) \tag{2.6}$$

or a normal distribution with zero mean and variance $\frac{2}{n_{in}+n_{out}}$. Xavier initialization is primarily developed for the sigmoid and tanh activation functions.

He Initialization: An approach tailored for the ReLU activation function and similar activation functions (Clevert et al., 2016; Mu et al., 2020). The initialization keeps the variance of activations consistent across layers. For a layer with n_{in} input units, the weights are initialized from a distribution with zero mean and variance $\frac{2}{n_{in}}$:

$$\boldsymbol{\theta} \sim N\left(0, \sqrt{\frac{2}{n_{in}}}\right) \tag{2.7}$$

or from a uniform distribution in the range $\left[-\sqrt{\frac{6}{n_{in}}}, \sqrt{\frac{6}{n_{in}}}\right]$.

2.1.4 Regularization Methods

Regularization techniques are essential for preventing overfitting in DNNs, a phenomenon that occurs when, instead of capturing the underlying patterns, the model memorizes the training set, leading to high performance on training data but poor performance otherwise. This section presents the methods that are relevant in the context of this work. However, there exist various other methods that have a regularizing effect.

ℓ_2 Regularization or weight decay, adds a penalty term to the loss function. This prevents the weights from reaching large values. This effectively constrains the updates optimization process, forcing the model to find a solution in a smaller, constrained weight space. The regularized loss function is defined as:

$$\mathcal{L}_{\ell_2} = \mathcal{L}(\boldsymbol{\theta}) + \frac{\lambda}{2} \sum_i \boldsymbol{\theta}_i^2 \tag{2.8}$$

Dropout. Srivastava et al. (2014) introduced dropout, a regularization technique that randomly sets a fraction of the activations $\mathbf{h}$ for a given layer to zero at each update in the training process, which mitigates the risk of overfitting. The dropout operation for the output $\mathbf{h}$ of a layer is defined by,

$$\mathbf{h}_{dropout} = \mathbf{h} \odot \mathbf{d} \tag{2.9}$$

Here, $\odot$ denotes element-wise multiplication, and $\mathbf{d}_i$ is a binary mask vector drawn from a Bernoulli distribution:

$$\mathbf{d}_i \sim \text{Bernoulli}(p) \tag{2.10}$$

where p is the probability of each element being independently set to zero. At test time, all units in the layer stay active, but the weights are scaled by p.

2.1.5 Normalization in Deep Neural Networks

Normalizing the inputs to a model is a common pre-processing technique in machine learning (Friedman, 1987; Goodfellow et al., 2016). Given input data $\mathbf{x}$, normalization is generally defined as:

$$\hat{\mathbf{x}} = \frac{\mathbf{x} - \mu}{\sigma^2} \tag{2.11}$$

In this equation, the scalars, μ and σ^2, are the mean and variance of $\mathbf{x}$, so that $\hat{\mathbf{x}}$ has zero mean and unit variance after the normalization. It ensures that input data is on the same scale, preventing the optimization process from being disproportionately influenced by a single input feature.

LeCun, Bottou, Orr, et al. (1998) investigate the beneficial effects of pre-processing input data by mean subtraction and variance scaling. With this transformation, the authors aim to regulate the sensitivity of the output to changes in the input distribution. LeCun, Bottou, Orr, et al. (1998) can show that the normalization affects the eigenvalues of the Hessian matrix $\mathbf{H}$, a second-order derivative matrix capturing the curvature of the loss function with respect to the model parameters.

$$\mathbf{H} = \nabla_{\boldsymbol{\theta}}^2 \mathcal{L}(\hat{\mathbf{y}}, \mathbf{y}) \tag{2.12}$$

Normalizing the input features can alter the distribution of eigenvalues of the Hessian matrix, potentially fostering more favorable conditions for optimization.

This principle can be extended to hidden layers. However, applying normalizations to inputs of intermediate layers is not as straightforward as normalizing static input data and comes with a greater computational expense: The activations constantly change over the training time, and directly normalizing those requires continuous computation of statistics over the entire training set for each training step.

2.2 Batch Normalization

One fundamental challenge of training DNNs stems from the nature of the parameter updates. The weights and the bias of a layer, $\boldsymbol{\theta}$ and b, are updated with the underlying assumption that the activations of the preceding layers remain unchanged. However, during the training process, weights in all layers are concurrently adjusted, leading to alterations in their output distributions. This alteration in output distributions of a layer, commonly termed "internal covariate shift", means that subsequent layers are faced with a changing input distribution at each iteration. This can result in inconsistent weight updates as each layer attempts to learn from a continuously changing distribution, slowing down the optimization procedure (Goodfellow et al., 2016).

Aiming to address this issue, Ioffe et al. (2015) introduce BatchNorm with the objective of normalizing the activations of a layer. BatchNorm can be viewed as an additional layer that normalizes each feature dimension of an input $\mathbf{x}_i = \mathbf{x}_i^{(1)}...\mathbf{x}_i^{(d)}$. At each feature dimension, the distribution is normalized to have zero mean and a standard deviation of one. In addition, Ioffe et al. (2015) introduce the two learnable parameters γ and $\boldsymbol{\beta}$. The first parameter is a vector trained to approximate the optimal scaling factor for each feature dimension, allowing the network to modulate the magnitude of the features based on their importance. Similarly, $\boldsymbol{\beta}$ is a vector that shifts the activations, enabling the network to adjust the normalized values. For each feature dimension of the input $\mathbf{x}_i^{(k)}$ with m values in the mini-batch $\mathcal{B}_{\mathbf{x}_{1...m}}$, the transformation $BN_{\gamma,\beta} : \mathbf{x}_{1...m} \rightarrow \mathbf{y}_{1...m}$ is defined as,

$$BN_{\gamma,\beta}(\mathbf{x}_i^{(k)}) \equiv \gamma^{(k)} \frac{\mathbf{x}_i^{(k)} - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} + \boldsymbol{\beta}^{(k)} = \tilde{\mathbf{y}}^{(k)} \tag{2.13}$$

where $\epsilon \in \mathbb{R}_{>0}$ is a small constant added to the variance estimate to ensure no division by zero, thus preventing numerical instability. Both learnable parameters can be defined such that the transformations applied by BatchNorm can recover the identity transform. If ϵ is neglected, the original activations can be restored by defining $\gamma = \sigma_{\mathcal{B}}^2$, and $\boldsymbol{\beta} = \mu_{\mathcal{B}}$.

2.2.1 Details on the Internal Computations

Training Time. In the forward pass, the mean and the variance of the current mini-batch $\mathcal{B}$ are computed. Specifically, the variances are computed per feature dimension of the input rather than as joint covariances. This is necessary to handle cases where the batch size is smaller than the number of activations, resulting in singular covariance matrices (Ioffe et al., 2015). For clarity, the feature dimension of the input (k) is omitted in the following equations.

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}\mathbf{x}_i \tag{2.14}$$

$$\sigma_{\mathcal{B}}^{2(k)} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(\mathbf{x}_i - \mu_{\mathcal{B}}) \tag{2.15}$$

Calculated mean and variance are then applied to the input $\mathbf{x}_i$, such that the distribution of each feature dimension is normalized to have zero mean and unit variance if ϵ is neglected. In the context of this work, we refer to this transformation as non-adaptive normalization. The effect on the distributions is illustrated in Figure 2.1 (b).

$$\tilde{\mathbf{x}}_i = \frac{\mathbf{x}_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \tag{2.16}$$

The non-adaptive normalization is followed by the application of the learnable parameters:

$$\tilde{\mathbf{y}}_i = \gamma\tilde{\mathbf{x}}_i + \boldsymbol{\beta} \tag{2.17}$$

After the transformation by γ and $\boldsymbol{\beta}$, the activations can be shifted to a different mean and scaled to another standard deviation, as follows:

$$\mu[\tilde{\mathbf{x}}\gamma + \boldsymbol{\beta}] = \mu[\tilde{\mathbf{x}}\gamma] + \boldsymbol{\beta} = \gamma\mu[\tilde{\mathbf{x}}] + \boldsymbol{\beta} = \boldsymbol{\beta} \tag{2.18}$$

$$\sigma^2[\tilde{\mathbf{x}}\gamma + \boldsymbol{\beta})] = \sigma^2[\tilde{\mathbf{x}}\gamma] = \gamma^2\sigma^2[\tilde{\mathbf{x}}] = \gamma^2 \tag{2.19}$$

As a result, the distribution of the activations is not necessarily normalized to have zero mean and unit variance anymore, but instead, γ adjusts the variance, and $\boldsymbol{\beta}$ shifts the mean of the distributions. Figure 2.1 illustrates the change in the distributions through the BatchNorm transformation.

In the backward pass, the gradient of each input $\mathbf{x}_i$ is affected by the entire mini-batch statistics. The gradient of the loss ℓ through this transformation has to be computed with respect to the learnable parameters γ and $\boldsymbol{\beta}$. Ioffe et al. (2015) provide the backward pass of the computations in the original paper:

$$\frac{\partial\ell}{\partial\tilde{\mathbf{x}}_i} = \frac{\partial\ell}{\partial\tilde{\mathbf{y}}_i}\cdot\gamma \tag{2.20}$$

$$\frac{\partial\ell}{\partial\sigma_{\mathcal{B}}^2} = \sum_{i=1}^{m}\frac{\partial\ell}{\partial\tilde{\mathbf{x}}_i}\cdot(\mathbf{x}_i - \mu_{\mathcal{B}})\cdot -\frac{1}{2}\cdot(\sigma_{\mathcal{B}} + \epsilon)^{-\frac{3}{2}} \tag{2.21}$$

$$\frac{\partial\ell}{\partial\mu_{\mathcal{B}}} = \left(\sum_{i=1}^{m}\frac{\partial\ell}{\partial\tilde{\mathbf{x}}_i}\cdot\frac{-1}{\sqrt{\sigma_{\mathcal{B}} + \epsilon}}\right) + \frac{\partial\ell}{\partial\sigma_{\mathcal{B}}^2}\cdot\frac{\sum_{i=1}^{\mathrm{m}} -2(\mathbf{x}_i - \mu_{\mathcal{B}})}{\mathrm{m}} \tag{2.22}$$

$$\frac{\partial\ell}{\partial\mathbf{x}_i} = \frac{\partial\ell}{\partial\tilde{\mathbf{x}}_i}\cdot\frac{1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} + \frac{\partial\ell}{\partial\sigma_{\mathcal{B}}^2}\cdot\frac{2\cdot(\mathbf{x}_i - \mu_{\mathcal{B}})}{\mathrm{m}} + \frac{\partial\ell}{\partial\mu_{\mathcal{B}}}\cdot\frac{1}{\mathrm{m}} \tag{2.23}$$

$$\frac{\partial\ell}{\partial\gamma} = \sum_{i=1}^{m}\frac{\partial\ell}{\partial\tilde{\mathbf{y}}_i}\cdot\tilde{\mathbf{x}}_i \tag{2.24}$$

$$\frac{\partial\ell}{\partial\boldsymbol{\beta}} = \sum_{i=1}^{m}\frac{\partial\ell}{\partial\tilde{\mathbf{y}}_i} \tag{2.25}$$

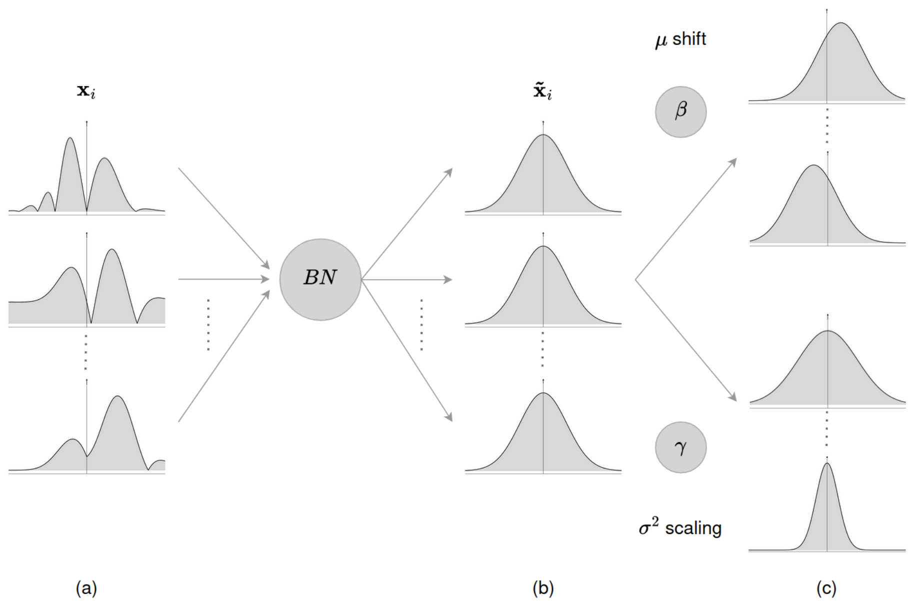


FIGURE 2.1: Illustration of the change of distributions across the feature dimensions of an input $\mathbf{x}_i$ through the BatchNorm transformation. (a) illustrates distributions of the input before BatchNorm, (b) the normalized distributions after the non-adaptive normalization with $\mu_{\mathcal{B}}$ and $\sigma^2_{\mathcal{B}}$, and (c) the change in the distributions that can be induced by either γ or $\boldsymbol{\beta}$.

Inference Time. In contrast to the training process, the output should be deterministic during inference, relying solely on the input rather than the entire mini-batch $\mathcal{B}$. Hence, instead of calculating the mini-batch statistics, the authors from the BatchNorm paper suggest using the population statistics by processing multiple training mini-batches $\mathcal{B}$ of size m and averaging over them. Again, for clarity, (k) is omitted.

$$\mu[\mathbf{x}] = \mu_{\mathcal{B}} \tag{2.26}$$

$$\sigma^2[\mathbf{x}] = \frac{m}{m-1} \cdot \mu_{\mathcal{B}}[\sigma^2_{\mathcal{B}}] \tag{2.27}$$

The transformation $\tilde{\mathbf{y}}_i = BN_{\gamma,\beta}(\mathbf{x}_i)$ is replaced by:

$$\tilde{\mathbf{y}}_i = \frac{\gamma \cdot \mathbf{x}_i}{\sqrt{\sigma^2[\mathbf{x}_i] + \epsilon}} + \left(\boldsymbol{\beta} - \frac{\gamma \cdot \mu[\mathbf{x}_i]}{\sqrt{\sigma^2[\mathbf{x}_i] + \epsilon}} \right) \tag{2.28}$$

In practice, during the validation phase, it is not possible to utilize the population averages as these are still evolving. Hence, employing running averages of the statistics becomes more appropriate for evaluating performance on the validation set. The running mean $\overline{\mu}$ and running variance $\overline{\sigma}^2$ are gathered over the training to be utilized at inference.

Popular machine learning frameworks, such as PyTorch (Paszke et al., 2019) and TensorFlow (Abadi et al., 2015), use a parameter named momentum term to weigh the update for the running statistics. Momentum in BatchNorm serves a unique purpose distinct from its conventional use in optimization algorithms such as SGD (Robbins et al., 1951). Here, it controls the Exponential Moving Average (EMA) update for the running statistics by stabilizing and smoothening the estimations over training time.

In the Tensorflow implementation, with setting the Momentum to zero, the running mean and running variance come from the last seen batch, and if the Momentum is one, the running averages come from the first mini-batch (Abadi et al., 2015).

$$\overline{\mu} = \text{momentum} \cdot \overline{\mu} + (1 - \text{momentum}) \cdot \mu_i \tag{2.29}$$

$$\overline{\sigma}^2 = \text{momentum} \cdot \overline{\sigma}^2 + (1 - \text{momentum}) \cdot \sigma_i^2 \tag{2.30}$$

The PyTorch implementation uses a different definition, such that the value of momentum equals (1-momentum) in Tensorflow (Paszke et al., 2019).

$$\overline{\mu} = (1 - \text{momentum}) \cdot \overline{\mu} + \text{momentum} \cdot \mu_i \tag{2.31}$$

$$\overline{\sigma}^2 = (1 - \text{momentum}) \cdot \overline{\sigma}^2 + \text{momentum} \cdot \sigma_i^2 \tag{2.32}$$

2.2.2 BatchNorm Layers

The implementation of BatchNorm within neural networks varies depending on the type of layer to which it is applied and the shape of the layer output. In the initial paper by Ioffe et al. (2015), BatchNorm is suggested to be placed right before the activation function within fully connected layers. Then, the BatchNorm transformation in a fully connected layer can be represented as:

$$\phi(BN_{\gamma,\beta}(\boldsymbol{\theta}\mathbf{x}_i + \mathbf{b})) \tag{2.33}$$

Relation between Beta and Bias. Ioffe et al. (2015) mention that with the implementation of BatchNorm, the effect of the bias is canceled out by the subsequent mean subtraction and suggest ignoring it and, further, subsume the bias by $\boldsymbol{\beta}$. A comparison of the gradient computation underpins the assumption (Goodfellow et al., 2016). The computation of the gradient of the loss in BatchNorm parallels how gradients would be calculated for a bias term. Thus, the parameter $\boldsymbol{\beta}$ in a BatchNorm layer can be seen as analogous to the bias term in a traditional neural network layer, and the above transformation can be expressed as:

$$\phi(BN_{\gamma,\beta}(\boldsymbol{\theta}\mathbf{x}_i)) \tag{2.34}$$

$BN_{\gamma,\beta}$ is applied independently to each dimension of $\mathbf{x}$ and with a separate pair of trainable parameters $\gamma^{(k)}, \boldsymbol{\beta}^{(k)}$ per feature.

Convolutional Layers. The properties of convolutional layers require an adjusted application of BatchNorm compared to fully connected layers. Here, the BatchNorm layer normalizes all activations over all locations of the input jointly. The mean $\mu_{\mathcal{B}}$ and the variance $\sigma_{\mathcal{B}}$ are computed over all values in a feature map across the mini-batch and the spatial locations, visualized in Figure 2.2. In addition, instead of learning $\gamma^{(k)}, \boldsymbol{\beta}^{(k)}$ per activation, there is a separate pair of parameters per feature map. The computations at inference time are similarly adjusted, such that BatchNorm uses the same linear transformation for each activation within one feature map O.

2.2.3 On the Effects of BatchNorm

In the original paper, the authors introduce BatchNorm with the primary reason to reduce the ICS (Ioffe et al., 2015). While they argued that BatchNorm stabilizes the distribution of layer activations and facilitates training, the only experiment in the paper that takes the ICS into account shows the mean of the activations at the last hidden layer. Ioffe et al. (2015) utilize a simple network with three fully connected hidden layers, trained on MNIST. Without BatchNorm, the activations fluctuate significantly in the early phases of training. In contrast, the curve of the activations appears significantly smoother with BatchNorm. However, this does not provide substantial proof for the claim that BatchNorm reduces the ICS or that reducing the ICS is actually beneficial for training DNNs.

The studies by Santurkar et al. (2019) have particularly challenged this view and proposed alternative explanations for the beneficial effects of BatchNorm. By visualizing the distribution of activations at different layers, the authors found that the performance gain of BatchNorm does not seem to be linked to the reduction of ICS and might not even reduce it at all. Santurkar et al. (2019) induced an additional ICS to show that in scenarios where the ICS is inherent, BatchNorm still improves the performance of the model. In contrast, Awais et al. (2020) propose results that rather underpin the importance of the reduction of the ICS and the role of BatchNorm in this issue. In their experiments, the authors found the learnable parameters to have little effect on the performance of a model, but the non-adaptive normalization alone could reach competitive performance. The authors suggest this indicates that the reduction of the ICS is the primary reason for BatchNorm's beneficial effects.

In the original BatchNorm paper, additional effects are awarded to the method besides the reduction of the ICS (Ioffe et al., 2015). These include enhanced convergence, preventing exploding or vanishing gradients, allowing for higher learning rates, and reducing sensitivity to initialization of the weights. The following section provides a collection of the main research directions in the literature on the effects of BatchNorm. Although the effects are widely accepted and tested in practice (Smith et al., 2017; Bjorck et al., 2018), they rather represent characteristics of the method than reveal fundamental reasons behind the effectiveness of BatchNorm.

Smoothness of the Loss Landscape. Along with questioning the reduction of the ICS, Santurkar et al. (2019) propose BatchNorm leads to a smoother optimization landscape. They show that under natural conditions, the Lipschitzness of the loss and the gradients is decreased, which is related to the smoothness of the loss surface (Gouk et al., 2021). Santurkar et al. (2019) observe that with BatchNorm, the loss changes are smaller, and the magnitudes of the gradients are reduced. In addition, BatchNorm appears to reduce the Lipschitzness of the gradients of the loss, as well. A function f is L-Lipschitz, if

$$||f(\mathbf{x}_1) - f(\mathbf{x}_2)|| \leq L||\mathbf{x}_1 - \mathbf{x}_2|| \tag{2.35}$$

holds, for all choices of $\mathbf{x}_1$ and $\mathbf{x}_2$, with $L \in \mathbb{R}_{>0}$. A smoother loss surface means reduced variations and irregularities in the landscape, such as high curvature or narrow valleys. The stabilization of gradients helps to guide the optimization process effectively and prevents the gradients from exploding or vanishing. The authors claim this effect enables an increase in the learning rate without destabilizing the training process (Santurkar et al., 2019). This makes a strong case for the fundamental mechanisms of BatchNorm. However, it does not reveal which aspects of the method lead to the improved Lipschitzness.

In addition to the work by Santurkar et al. (2019), more recent literature suggests the improvement of the Lipschitzness, or the smoothness of the loss surface, as a general effect of regularization methods, that enables DNNs to train stably (Gouk et al., 2021; Kim et al., 2022).

BatchNorm as a Regularization method. Subsequent research literature investigates the regularization effects of BatchNorm on DNNs (Teye et al., 2018; Luo et al., 2018). Ghorbani et al. (2019) show that if BatchNorm is trained with the true aggregated mean and variance, it induces poor conditioning. On the other hand, BatchNorm seems to rely on these aggregated statistics to reveal its full potential and outperform other normalization methods (Rao et al., 2020). This suggests that, specifically, the mean and variance for each mini-batch introduce slight variability in the normalized activations, serving as a form of regularizer that accelerates the ability of the model to generalize well on unseen data and prevent overfitting. Furthermore, Luo et al. (2018) show that the regularization effect is not only observable for scenarios in which ICS is significant. BatchNorm also improves the model's generalization performance in cases where the shift is minimal.

Beneficial effects, such as faster training and less overfitting by utilizing regularization methods, are a recurrent theme in deep learning. However, Kohler et al. (2018) suggest the effects of BatchNorm go beyond that of other regularization techniques. Their work shows that the transformations imposed by BatchNorm lead the magnitude of the weight vector to be independent of its direction. The authors argue that BatchNorm can exploit those properties for the optimization process, which makes it superior to other regularization methods. Still, Kohler et al. (2018) state that decoupling the length and direction of the parameters does not fully solve the questions around the reasons for BatchNorm's success.

Control over Large Hessian Eigenvalues. In their paper, Ghorbani et al. (2019) could show that the fractions of eigenvalues of the Hessian of $\mathcal{L}(\hat{\mathbf{y}}, \mathbf{y})$ are significantly reduced with BatchNorm. Not using BatchNorm led to a rapid appearance of large isolated eigenvalues, and the gradients tended to concentrate in the corresponding eigenspaces, which was shown to slow down optimization. BatchNorm appears to suppress those outliers in the eigenvalues and thereby speed up training. They argue that lowering the eigenvalues of the covariance matrix allows for higher learning rates. Rao et al. (2020) show this effect is not unique to the BatchNorm method but appears to happen in many other normalization methods (Wu et al., 2018; Ulyanov et al., 2016).

Position in the Architecture Matters. Laurent et al. (2015) demonstrated that although BatchNorm is effective in deep feed-forward networks, the optimal positioning within Recurrent Neural Networks (RNNs) appears to be more complex. The authors observed that only if BatchNorm is applied to input-to-hidden transitions, it yields beneficial results in RNNs. Otherwise, convergence speed did not improve. Conversely, Cooijmans et al. (2016) proposed that, with appropriate reparameterization, BatchNorm could also enhance the performance of hidden-to-hidden transitions within RNNs. This suggests the application of BatchNorm layers could be sensitive to the network architecture and positioning of the layers, requiring additional adaptations to unlock the full potential of BatchNorm.

2.2.4 On the Learnable Component in BatchNorm

In the broader discourse on BatchNorm, a critical area of investigation is the learnable component. The authors justify the introduction of γ and β with the ability of the BatchNorm layer to undo the non-adaptive normalization if that is the optimal adjustment for the optimization problem Ioffe et al. (2015). However, as both parameters are trained by the optimizer, there is no direct control over how γ and β adjust the mean and the variance of the activations. Besides the note that β could subsume the bias term (See Section 2.2.2), no further details are provided in the original paper.

Frankle et al. (2020) investigate the expressive power of the trainable component alone by isolating the contribution of γ and β from that of the learned weights. The authors freeze all the weights at their random initialization and only train the affine transformation, reaching a competitive performance regarding the limitations of the model capacity in this setting. Surprisingly, training an equivalent number of randomly chosen learnable parameters elsewhere in the network does not yield nearly the same results.

In contrast to these findings, Davis et al. (2021) have investigated that γ and β tend to stay close to their initial values, which is zero for β and one for γ, but strongly question that those parameters are optimal near identity settings. Instead, the authors suggest the affine transformation is either unnecessary, which aligns with the suggestions by Awais et al. (2020), or something in the optimization process hinders the parameters from reaching their optimal values. The work by Davis et al. (2021) then focuses on the initialization and updates γ, showing that besides reaching a better performance with other initial values for γ, for all cases, both parameters stay close to their initial values. Thus, the authors suggest modifying the update step of the parameters by adjusting the learning rate respectively. However, this paper aims to improve the performance of BatchNorm but does not seek to understand the role of the learnable parameters. In addition, a formal explanation is missing for the claim that the optimal solution of γ and β cannot be close to their initial values.

Peerthum (2023) aim to separate the relative effects of the two components in BatchNorm, the non-adaptive normalization, and the affine transformation. They propose two modifications of the technique, one that applies only the non-adaptive normalization step and one that applies only the affine transformation. Those modifications are then compared on several ResNets, with the original BatchNorm and a version without BatchNorm at all. Peerthum (2023) can show that the role of the affine transformation varies across several ResNet architectures. In their work, the ResNet18 and ResNet34, which use basic blocks, the non-adaptive normalization step alone reaches comparable performance to the original BatchNorm version. On the contrary, the affine transformation is critical for the ResNet50 and ResNet101 models, which make use of bottleneck blocks.

2.2.5 Alternative Normalization Techniques

Despite its huge success, the application of BatchNorm has its drawbacks. For instance, it requires additional overhead and complexity in the training process, as the statistics for test time have to be accumulated for each batch. Another consideration is that BatchNorm can be less advantageous in scenarios with small batch sizes and σ^2 undefined for a batch size of one. The need for efficient optimization with small batch sizes arises as machine learning systems increasingly adopt distributed architectures. Even though the batch size might be relatively large in these architectures, it often gets divided across multiple machines, sometimes resulting in single or dual sample batches on each device (Qiao et al., 2019).

Consequently, the choice of normalization technique should be carefully considered, particularly when data availability is constrained. Several alternatives exist aside from BatchNorm, especially for settings where it may not be the most suitable (Xu et al., 2019; Clevert et al., 2016; Xiang et al., 2017).

Weight Normalization. Salimans et al. (2016) propose a method that normalizes the weights, a reparameterization of the weight vectors in a neural network that decouples the length of those weight vectors from their direction. Weight normalization re-parameterizes the d-dimensional weight vector $\boldsymbol{\theta}$:

$$\boldsymbol{\theta} = \frac{g}{||\mathbf{v}||}\mathbf{v} \tag{2.36}$$

Here, $\mathbf{v}$ is a d-dimensional vector, $||\mathbf{v}||$ denotes the Euclidean norm of $\mathbf{v}$ and g is a scalar. Gradient descent performs the updates with respect to those parameters instead, with the effect of fixing the Euclidean norm of the weight vector to $||\boldsymbol{\theta}|| = g$, independent of $\mathbf{v}$. Unlike BatchNorm, weight normalization is deterministic and does not make use of a learnable component. In addition, Rao et al. (2020) can show that weight normalization does not affect the conditioning of the Hessian as opposed to BatchNorm while reaching comparable performance. Despite its simplicity, the experiments show it can reach much of the speed improvements of BatchNorm. However, weight normalization highly depends on the input data and can be unstable during training (Gitman et al., 2017).

Layer Normalization. At the same time, Ba et al. (2016) introduce a technique that directly estimates the normalization statistics from the layer inputs to fix the mean and the variance of inputs within a layer and thereby aim to reduce the ICS. The layer normalization statistics are computed for each sample across all channels d:

$$\mu = \frac{1}{d}\sum_{k=1}^{d}\mathbf{x}_k \tag{2.37}$$

$$\sigma_i = \frac{1}{d}\sum_{k=1}^{d}(\mathbf{x}_k - \mu)^2 \tag{2.38}$$

Figure 2.2 shows a visualization of that operation. The calculation of statistics eliminates dependencies between the layers and similarly applies the learnable parameters γ and $\boldsymbol{\beta}$ after the non-adaptive normalization. Layer normalization has been shown to work successfully in RNNs and transformer-based models (Ba et al., 2016; Vaswani et al., 2017). This success underpins the notion that a reduction of the ICS is unlikely to be the only reason for BatchNorm's success.

Instance Normalization can be considered a variation of above layer normalization (Ulyanov et al., 2016), which prevents instance-specific mean and covariance shifts. It computes the mean and variance for each channel and each sample, visualized in Figure 2.2. This technique could be shown to be especially successful in training generative adversarial networks (Xu et al., 2019), for which BatchNorm is not the most suitable (Xiang et al., 2017).

Group Normalization. Another approach independent of batch sizes is group normalization (Wu et al., 2018). This technique divides the input channels into groups and normalizes the features within each group before applying the trainable parameters γ and $\boldsymbol{\beta}$. The group number G is a pre-defined hyperparameter containing groups $\mathcal{S}_i$ with $i \in N$, denoting the index of the channels.

$$\mu_i = \frac{1}{m}\sum_{r\in\mathcal{S}_i}\mathbf{x}_r \tag{2.39}$$

$$\sigma_i^2 = \frac{1}{m}\sum_{r\in\mathcal{S}_i}(\mathbf{x}_r - \mu_i)^2 \tag{2.40}$$

If the number of groups is set to one, group normalization is equivalent to the above instance normalization (See Figure 2.2).

Weight Standardization. Qiao et al. (2019) found inspiration from the findings that BatchNorm improves the Lipschitzness of the loss and the gradients (Santurkar et al., 2019). Qiao et al. (2019) propose weight standardization, a method developed to utilize this effect further. Instead of normalizing the activations, weight standardization re-parametrizes the weights in convolutional layers.

$$\hat{\boldsymbol{\theta}} = [\hat{\boldsymbol{\theta}}_{i,j} | \hat{\boldsymbol{\theta}}_{i,j} = \frac{\boldsymbol{\theta}_{i,j} - \mu_{\boldsymbol{\theta}_i}}{\sigma_{\boldsymbol{\theta}_i} + \epsilon}] \tag{2.41}$$

With $\mu_{\boldsymbol{\theta}_i}$ and $\sigma_{\boldsymbol{\theta}_i}$ are the mean and the variance over the weights of each channel in the convolutional layer. Experiments from the original paper show that weight standardization can, even more, smoothen the loss landscape of the optimization problem (Qiao et al., 2019). Different from BatchNorm, this method does not apply any learnable component because the authors suggest additional normalization layers can be used along with weight standardization.

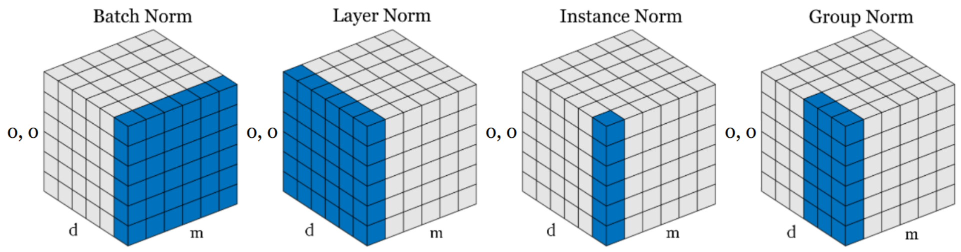


FIGURE 2.2: **Normalization methods.** Each subplot shows a feature map tensor with m as the batch axis, d as the channel axis and (q, w) as the spatial axes. The pixels in blue are normalized by the same mean and variance, computed by aggregating the values of these pixels. Illustration adapted from Wu et al. (2018).

2.3 Summary

The literature presented throughout this chapter emphasizes the complexity of the task to train DNNs and the importance of the BatchNorm technique in the realm of training DNNs. This method has been empirically shown to offer benefits such as faster convergence, enhanced stability, and improved performance. Furthermore, its role as a regularizer is well-established and extensively utilized across a spectrum of DL applications. However, the application of BatchNorm encloses drawbacks regarding the computational overload and the inability to handle batch sizes of one. Alternatives to BatchNorm exist, yet none of them could be proven to be generally superior.

While researchers agree on the beneficial effects of BatchNorm, the underlying mechanisms are still a subject of discussion. Until today, BatchNorm is an active area of research with various theories and approaches (Hoedt et al., 2022). None of the theories or investigated effects can fully serve as the solution to why BatchNorm is of such great benefit. An area of particular intrigue is the role of the affine transformation in BatchNorm, characterized by the trainable parameters gamma and beta. This aspect has been underrepresented in current research narratives despite its centrality to the BatchNorm process. Research findings show disparate results on the impact of the affine transformation of BatchNorm, which motivates further investigation.

Chapter 3

Methods

In order to investigate the contribution of the respective components in BatchNorm, we designed an ablation study with BatchNorm in Section 3.1, introducing the modifications on the formula in Section 3.1.1 along with the experimental setup (Section 3.1.2). This includes the network architectures and the implementation of the BatchNorm modifications, along with additional details considered for subsequent analysis. Section 3.2 provides insights on the study of performance contributions, and Section 3.3 provides the methodology for the detailed investigation of the learnable parameters during training [1].

3.1 On the Design of the Ablation Study

For the investigation of the behavior of γ and $\boldsymbol{\beta}$ during training, we want to be able to relate the behavior of the two parameters to their impact on model performance. This is important because we first want to identify whether the non-adaptive normalization in BatchNorm is the only component contributing to the performance gain. If that is the case for all models, the behavior of the trainable parameters in these experiments would be irrelevant for the effects induced by BatchNorm. In contrast, if the two parameters significantly affect the model performance, it allows for a meaningful investigation of the behavior of γ and $\boldsymbol{\beta}$ in relation to their performance contribution.

We design an ablation study by modifying the BatchNorm formula in different ways, such that the learnable parameters are removed from the equation individually or together. The modifications are designed to investigate the contribution of the γ and $\boldsymbol{\beta}$ upon the non-adaptive normalization step. This allows us to clarify if the learnable parameters are of benefit or even necessary for the BatchNorm method.

3.1.1 Modifications of BatchNorm

The original BatchNorm formula is modified in three ways for the investigation. Together with the original BatchNorm formula and the Vanilla model without any BatchNorm layers, this provides us with five models to compare in relation to each other (See Section 2.2.1 for details).

Original BatchNorm. $BN_{\gamma,\beta}$ with the non-adaptive normalization and the two learnable parameters γ and $\boldsymbol{\beta}$. The feature dimension (k) is omitted for clarity.

$$BN_{\gamma,\beta}(\mathbf{x}_i) \equiv \gamma \frac{\mathbf{x}_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} + \boldsymbol{\beta} \tag{3.1}$$

Non-adaptive Normalization Only. The second modification BN is only utilizes the non-adaptive component of BatchNorm.

$$BN(\mathbf{x}_i) \equiv \frac{\mathbf{x}_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \tag{3.2}$$

[1] Implementation can be found here: `https://github.com/Vanessa-Ts/DissectBatchNorm`

Non-adaptive Normalization with Learnable $\boldsymbol{\beta}$. BN_β encompasses the learnable shift post the non-adaptive normalization. As above, γ is set to one.

$$BN_\beta(\mathbf{x}_i) \equiv \frac{\mathbf{x}_i - \mu_\mathcal{B}}{\sqrt{\sigma^2_\mathcal{B} + \epsilon}} + \boldsymbol{\beta} \tag{3.3}$$

Non-adaptive Normalization with Learnable $\boldsymbol{\gamma}$. BN_γ trains the γ parameter, while $\boldsymbol{\beta}$ is set to zero. This way, the activations are scaled but not shifted after the non-adaptive normalization.

$$BN_\gamma(\mathbf{x}_i) \equiv \gamma \frac{\mathbf{x}_i - \mu_\mathcal{B}}{\sqrt{\sigma^2_\mathcal{B} + \epsilon}} \tag{3.4}$$

3.1.2 Experimental Setup

For the comparison of performance contributions of the respective BatchNorm components, we consider a standardized experimental setup, where our purpose is not to achieve state-of-art results but to understand how the individual components of BatchNorm contribute to the beneficial effects in the training process, in a typical DL setup. To allow for better-supported conclusions on the role of the learnable parameters in BatchNorm, we seek varying complexity in terms of datasets and network architectures, relating the impact of the respective components and the training behavior of γ and $\boldsymbol{\beta}$ to the data set and the architecture.

The network architectures and the training details used in the experiments build upon the work by Schneider et al. (2019). The authors introduce a Deep Learning Optimizer Benchmark Suite (DeepOBS)[2] that enables the empirical comparison of an optimizer across various datasets, network architectures and other components of DNN training in a unified setting to improve comparability and reproducibility of results. These properties and the provided performance targets without BatchNorm make it a suitable basis for our empirical study.

Selection of Datasets and Network Architectures. In our investigation, we have selected the CIFAR-10 and CIFAR-100 datasets for image classification (Krizhevsky, 2009) to provide a variation in the number of classes in the training data (See Appendix B for details). These datasets are widely recognized and utilized in the machine learning community, presenting a standard benchmark for evaluating and comparing the performance of different network architectures and optimization strategies (Goodfellow et al., 2016; Thakkar et al., 2018; Schneider et al., 2019).

In order to compare the performance contributions of the respective modifications, a model is needed that performs well without BatchNorm layers but can be improved by the application of such. DeepOBS provides several architectures implemented without any BatchNorm layers that can be extended for our work. Regarding the activation function, Hasani et al. (2019) could show that the placement of BatchNorm layers before or after a ReLU activation function does not significantly affect the model performance. Hence, it makes sense to select networks in which the activation functions employed are all ReLUs in order to mitigate possible effects on performance induced by the position of the BatchNorm layer in relation to the activation function.

The first network 3C3D consists of three convolutional layers with ReLU activation functions, followed by max-pooling. After this sequence, the architecture includes three fully connected layers, often called dense or linear layers. The first two linear layers have ReLu activation functions, while the third linear layer uses a softmax function, defined in Section 2.1, as the last layer in the network. For all layers, the weight matrices are initialized using the Xavier initialization (See Section 2.1.3), and all biases are initialized to zero. The network additionally uses ℓ_2 regularization, introduced in Section 2.1.4, on the weights but not the biases.

[2]Github repository: https://github.com/fsschneider/DeepOBS

The second architecture resembles the ALL-CNN-C network introduced in the paper by Springenberg et al. (2015). The DeepOBS implementation of the network is composed of nine convolutional layers, each with a ReLU activation function. Additionally, the network utilizes dropout regularization before a sequence of three convolutional layers. The original paper does not provide information on the initialization of weights and learnable parameters, but the DeepOBS implementation suggests the weights are initialized with Xavier, and the biases are initialized at a value of 0.1. We adopt the initialization of the weights for our experiments but set the bias values **b** to zero to support equal conditions regarding the initialization of $\boldsymbol{\beta}$, noting that this change did not prevent the DNN from learning the task.

Implementation of BatchNorm Layers. In both network architectures, the BatchNorm layers are placed between the convolutional or linear layer and the ReLU activation function, as proposed in the original paper (Ioffe et al., 2015). We initialized $\boldsymbol{\beta}$ to zero, similar to the bias in the Vanilla version of each model, and the scale parameter γ to one. For each modification in Section 3.1.1, we adjust the BatchNorm layers accordingly by freezing γ and $\boldsymbol{\beta}$ to one and zero, respectively. Figure 3.1 illustrates the structure of a convolutional layer from our network architectures with BatchNorm. Our analysis in Section 3.2.3 takes into account that the position and amount of BatchNorm layers can affect the performance contribution, especially when BatchNorm is placed at the last layer of the network.

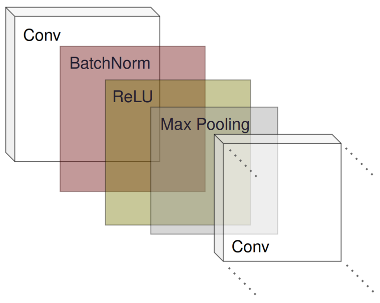


FIGURE 3.1: **BatchNorm Layer Implementation.** Illustration of a convolutional layer with BatchNorm. Max pooling is only applied in the 3C3D. Apart from that, the structure is similar for both architectures.

Characteristics of the Training Setup. In all experiments, we employed the CE loss suitable for the multi-class image classification task and SGD as the optimization algorithm of choice, both defined in Section 2.1. The hyperparameters were set with a focus on comparing the performance contributions rather than achieving state-of-the-art performance for each modification. For the CIFAR10 3C3D problem, we adopted the hyperparameter setting provided in the baselines from DeepOBS for the model without BatchNorm layers, referred to as the Vanilla version of a model in this work. The hyperparameter setting for the CIFAR100 ALL-CNN-C model was adapted from the original paper (Springenberg et al., 2015). DeepOBS does not provide baselines for the CIFAR100 3C3D model. We apply the hyperparameter setting from the CIFAR100 ALL-CNN-C since it also leads to acceptable performance for the Vanilla model.

The analysis in Section 3.2 includes a heuristic assessment of the performance contributions across a wide range of learning rates to address the issue of observing a behavior only specific to one hyperparameter setting. The corresponding results in Section 4.1.2 show that tuning the learning rate is not necessarily required for our comparative analysis. Moreover, individual tuning of the learning rate and batch size for each model with and without BatchNorm layers would, in turn, require tuning each modified version of BatchNorm from Section 3.1.1. Yet, this can be considered neglectable as we focus on comparing performance differences between the modifications for each model, and individual tuning might induce additional effects on the performance not directly related to the application of BatchNorm. Table 3.1 provides an overview of the models and hyperparameters for the training process.

Though not directly considered a hyperparameter, it plays a crucial role in the fair and empirical comparison to set the seed to the same value for all experiments, providing a controlled environment that minimizes the influence of random initialization on the observed behaviors.

Dataset	Architecture	Hyperparameter	Epochs
CIFAR10	3C3D	batch size: 128, lr: 0.022	100
CIFAR100	3C3D	batch size: 256, lr: 0.166	350
CIFAR100	ALL-CNN-C	batch size: 256, lr: 0.166	350

TABLE 3.1: Dataset, network architecture, and hyperparameter setting for the experiments, including the batch size and the learning rate (lr).

3.2 Analysis of Performance Contributions

To assess the impact of the respective components of BatchNorm, we focus on the training performance of the introduced modifications in relation to each other. Key indicators of this contribution are discerned by observing differences in training loss and training accuracy to identify any potential disparities in the training dynamics that could be attributed to γ and $\boldsymbol{\beta}$, separately and jointly.

3.2.1 Performance Implications of BatchNorm Modifications

To provide a clear context for our observations, we use the performance of both the Vanilla version of the neural network and the $BN_{\gamma,\beta}$ as baselines. The performance of the Vanilla version serves as the lower baseline, indicating whether a modification of BatchNorm is beneficial or potentially detrimental to the model performance. $BN_{\gamma,\beta}$ serves as the upper baseline in the experiments. Relating the performance of the modifications with that of $BN_{\gamma,\beta}$ allows us to identify whether the non-adaptive normalization alone is primarily the beneficial component of BatchNorm, or the learnable parameters, in addition, lead to the success of BatchNorm.

For our analysis, we suggest that if the count of the trainable parameters is reduced, there is a potential decrease in the performance enhancement brought by BatchNorm. Specifically, we anticipate the highest performance from the original BatchNorm, followed by either BN_γ or BN_β. A superior performance of the original BatchNorm method indicates the success of BatchNorm cannot solely be attributed to the non-adaptive normalization component. The combination with the learnable parameters then plays a pivotal role, likely due to the optimizer's ability to train two additional parameters.

If BN exhibits performance similar to $BN_{\gamma,\beta}$ across all models, it clearly indicates that the learnable parameters do not have an effect on the model performance and can be neglected. We note that removing both trainable parameters leaves the model with fewer learnable parameters because the effect of the bias is zeroed out by the non-adaptive normalization in BatchNorm. A parameter that adjusts the mean of the activations and thereby allows the optimizer to control how many activations are set to zero by the subsequent ReLU operation. If the learnable parameters do not affect performance, it indicates the network is robust to small changes in the number of activations that are deactivated by the ReLU function. If the adaptive BN_β demonstrates increased performance to the non-adaptive BatchNorm version, the model benefits from the trainable shift operation introduced at every layer. Conversely, if BN_γ outperforms BN and BN_β, it would suggest that a learnable scaling term, adjusting the variance of the normalized activations is more beneficial than shifting the mean.

3.2.2 Robustness to Hyperparameter Setting

In the context of evaluating learning rates for optimization algorithms, an order of magnitude test offers a systematic method to gauge the performance induced by different learning rates. This comparison is conducted by examining different orders of magnitude. For our work, we utilize this test not to determine the optimal learning rate but to ensure that the respective performance contributions are not specific to a particular value of the learning rate. Therefore, we added experiments for learning rates ten times larger and ten times smaller than the original one as heuristics to showcase the robustness of the observed behavior across a wide range of learning rates. The results are presented in Section 4.1.2.

3.2.3 BatchNorm in the Last Layer

For the implementation, we have refrained from using BatchNorm at the last convolutional or linear layer in each network, as illustrated in Figure 3.1. The reason for this lies in the function of the last layer to produce a set of predictions or probability distributions in a classification task. Intuitively, the normalization by BatchNorm can potentially compromise the quality of the predictions. The normalization introduces an additional layer of stochasticity due to the dependence of BatchNorm on the mini-batch statistics, which might not be desirable in the final decision-making layer. Furthermore, good predictions are not required to follow normalized distributions. Hence, it is likely that γ and β show a different effect on the training performance if BatchNorm is placed at this position in the network. We have considered this intuition in our analysis and added experiments to investigate if such an effect appears in the last layer (Section 3.2.3).

To ensure the observed effects can be attributed to the positioning of BatchNorm in the last layer, we extend this analysis to explore whether the amount and position of BatchNorm layers can further change the impact of the respective components on the training performance (Appendix A).

3.3 Behavioral Analysis of the Learnable Parameters

We proceed with an analysis of γ and β, based on the performance contributions documented in (Section 4.1.1). The core objective is to investigate whether discernible trajectories of γ and β during training can provide new insights and further reveal information about the model performance.

3.3.1 Activity of Learnable Parameters.

In the context of this work, we use the term "activity" to describe the extent to which γ and β differ from their neutral states - one and zero, respectively.For γ, low activity means that the values stay very close to one, as scaling activations with a value of one has no effect. A value close to zero means the activations are reduced, while activations are amplified for every value over one. With γ, the optimizer can adjust the variance of the normalized activations, as shown in Figure 2.1.

For β, this means that if the values stay very close to zero, it has a low activity. While a high activity implies that the values differ significantly from zero in either the negative or positive direction in the value space. Strong positive values of β shift the mean of the activations towards positive values. As, in our setting, each BatchNorm layer is followed by a ReLU, a strong positive shift indicates more features are activated, and a strong negative shift indicates the sparsity of the activations is increased.

The sparsity of activations refers to the proportion of activations that are zero and has been shown to aid DNNs in prioritizing salient features in the data, potentially offering robustness against noise and unessential patterns (Zhou et al., 2016). Specifically, by adjusting β, the optimizer can regulate the number of activations, thereby inducing sparsity within the network. In this light, the beta parameter in BatchNorm might act as a regularizer, comparable to dropout.

3.3.2 Visualization of Learnable BatchNorm Parameters

To visualize the dynamics of the learnable parameters across training, we use contour plots, which can represent a 3D surface by plotting constant z slices, called contours, on a 2D format. In our case, this property pertains to the activity of the learnable parameter, γ or β. For each BatchNorm layer in the model, a plot is created that displays the trajectory of either γ and β, as shown in Figure 3.2. The vertical axis denotes the training steps, and the horizontal axis represents the dimensions of the respective layer.

The color intensity in these plots is denoted by the range, set by the maximum and minimum values of the respective parameter across all BatchNorm layers of a model, which is crucial for our analysis. Centralized scales, with zero for $\boldsymbol{\beta}$ and one for γ, allow us to identify the magnitude and direction of shifts applied to the activations throughout the layers of the model.

Leveraging the fact that most trajectories are monotonic, we sorted the feature dimensions in each plot by their converged value to improve the visualization, which has no loss of generality since DNNs are considered invariant to permutations of linear features or convolutional channels (Hoefler et al., 2021).

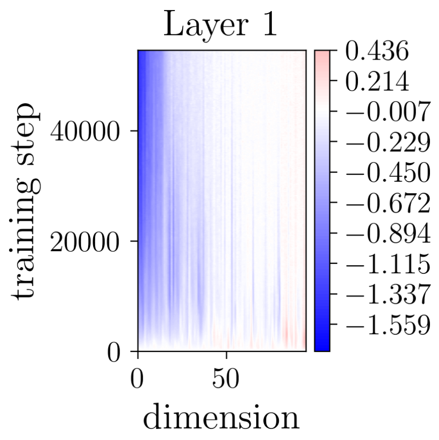


FIGURE 3.2: **Parameter Visualization.** Example of a contour plot for analyzing the learnable parameters of BatchNorm at a specific layer. The vertical axis denotes the training steps, the horizontal axis the feature dimensions, and the values of the respective parameter are denoted by the color scale.

Chapter 4

Experiments

This chapter outlines the experiments in order to identify the performance contribution of each model from Table 3.1, in Section 4.1. The results are then used in the investigation of γ and β, allowing for an analysis of the relation between the behavior of the learnable parameters and their performance contribution in a DL model (Section 4.2). Due to the variety of experiments and results in this chapter, we summarize our findings in Section 4.2.3.

4.1 Results from the Ablation Study

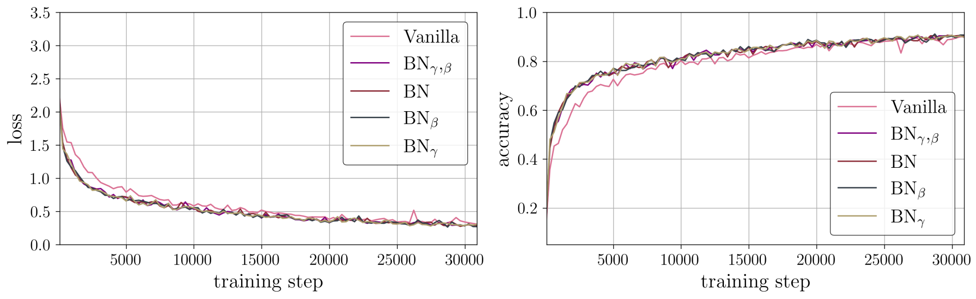
The experiments from the ablation study on BatchNorm, with the methodologies established in Chapter 3, show the training loss and accuracy of the models from Table 3.1. Each model is trained with all BatchNorm modifications respectively and without BatchNorm layers at all, as described in Section 3.1. In all experiments, the Vanilla version of the model and the original BatchNorm version serve as lower and upper baselines, respectively (SeeSection 3.1.1 for details). The performance contribution of the respective BatchNorm components is analyzed across models (Section 4.1.1), learning rates (Section 4.1.2), and positions (Section 4.1.3).

4.1.1 The Role of the Learnable Parameters

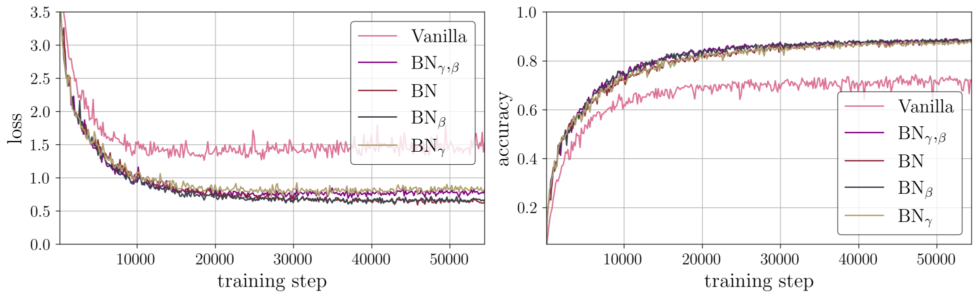
For the CIFAR-10 3C3D and CIFAR-100 3C3D, γ and β do not show any significant effects on the loss and accuracy curvature over training time (Figure 4.1a and Figure 4.1b). However, the CIFAR-100 ALL-CNN-C can significantly benefit from applying the trainable parameters, γ and β (Figure 4.1c).

Specifically for the CIFAR-10 3C3D model, considered less complex in the context of this work, the application of the original BatchNorm formula led to a faster convergence of loss and accuracy in early training phases (Figure 4.1a). Yet, the model converges to a similar loss and accuracy at the end of training. In this scenario, in which the benefit induced by BatchNorm is merely faster convergence, we observe BN to be similarly successful as $BN_{\gamma,\beta}$, indicating γ and β have no significant effect on the training performance. The CIFAR-100 3C3D benefits from the application of BatchNorm with faster convergence and higher accuracy at the end of training (Figure 4.1b). Here, again, the different modifications do not show significant performance differences up to stochastic training noise.

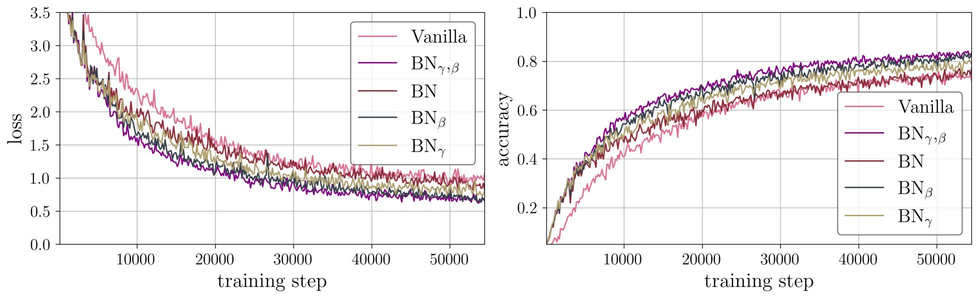
The deeper CIFAR-100 ALL-CNN-C model likewise converges faster and to improved accuracy with the application of BatchNorm (Figure 4.1c). Contrary to the above findings, the ablation of the learnable parameters from the BatchNorm formula showed, although not huge, significant effects on the training performance of the model. Here, BN introduces the least benefits to the model performance, as expected (See Section 3.2.1). It does lead to faster convergence but converges to a similar training loss and accuracy as the Vanilla model. The BN_{γ} can slightly accelerate loss and accuracy, but BN_{β} overtakes this improvement. The most beneficial effect can be observed if the model has the ability to train both γ and β.



(A) CIFAR-10 3C3D model trained for 100 epochs with SGD, a learning rate of 0.023, and a batch size of 128.



(B) CIFAR-100 3C3D trained for 350 epochs with SGD, a learning rate of 0.166, and a batch size of 256.



(C) CIFAR-100 ALL-CNN-C trained for 350 epochs with SGD, a learning rate of 0.166, and a batch size of 256.

FIGURE 4.1: Training loss and accuracy for all three models and all modifications of BatchNorm along with the Vanilla version ● and original BatchNorm ● serving as baselines.

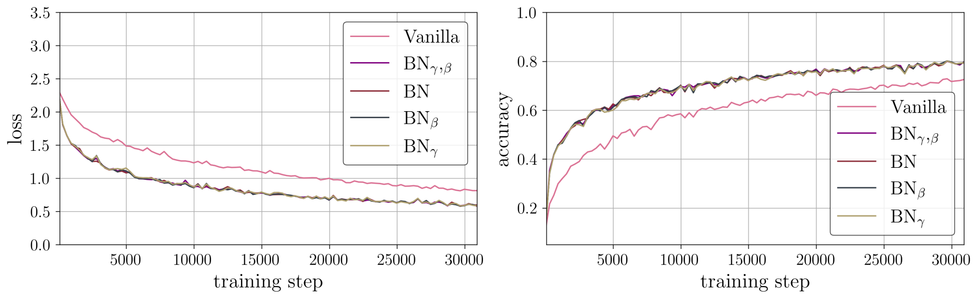
4.1.2 Similar Contribution Across Learning Rates

For each model (CIFAR-10 3C3D, CIFAR-100 3C3D and CIFAR-100 ALL-CNN-C), an order of magnitude test was performed on the learning rates. This ensures the observed performance contribution of the BatchNorm components is not specific to the value of the learning rate (See Section 3.2). Each learning rate is adjusted by an order of magnitude smaller ($\times 10^{-1}$) and larger ($\times 10^{1}$) as in Figure 4.1. For each model, the figure shows the training loss and accuracy across the varying learning rates. The findings below support the idea that the performance contribution is not specific to the hyperparameter setting, as outlined in Section 3.2.2.

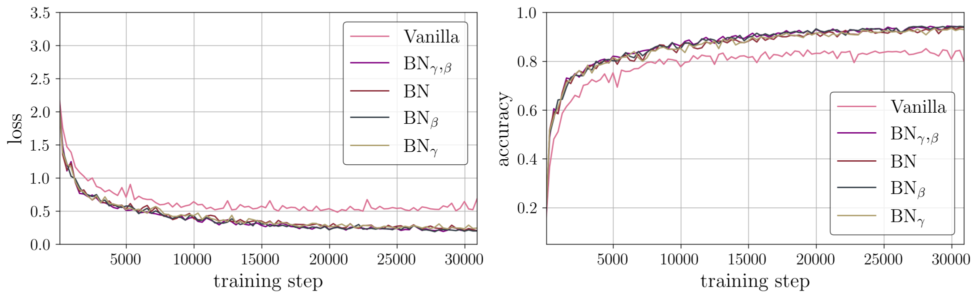
CIFAR-10 3C3D. For both learning rates (0.022×10^{-1} and 0.022×10^{1}), all modifications of BatchNorm show to improve the model performance (Figure 4.2). However, there is no difference in the training performances regardless of the number of learnable parameters of the BatchNorm modification. Together with the results in Figure 4.1a, this allows for the confident conclusion that for the CIFAR-10 3C3D model, the non-adaptive normalization alone is the beneficial component of BatchNorm.

CIFAR-100 3C3D. Figure 4.3 shows all modifications of BatchNorm consistently performed well across both learning rates (0.166×10^{-1} and 0.166×10^{1}) with the CIFAR-100 3C3D model. In contrast, the Vanilla model struggled, diverging right at the start with the larger learning rate. Our findings suggest that even without the learnable parameters γ and β, BatchNorm is effective for this model. However, we did observe minor variations in the data, likely resulting from noise. While it wasn't our main focus, Figure 4.3b is an example of BatchNorm effectively supporting larger learning rates, thereby leading to improved training outcomes.

CIFAR-100 ALL-CNN-C. In our analysis of the CIFAR-100 ALL-CNN-C model presented in Figure 4.4, we found that all BatchNorm modifications improved the training performance for both learning rates (0.166×10^{-1} and 0.166×10^{1}). However, the performance differences between these modifications are still evident, consistent with what is shown in Figure 4.1c. Specifically, $BN_{\gamma,\beta}$ marginally outperforms the other modifications, while the standard BN shows the least effect. With the larger learning rate, BN_{β} enhances convergence speed more than BN_{γ} and the basic BN, a trend also seen in Figure 4.1c. It should be noted that the Vanilla model diverges when trained with the larger learning rate (0.022×10^{1}).

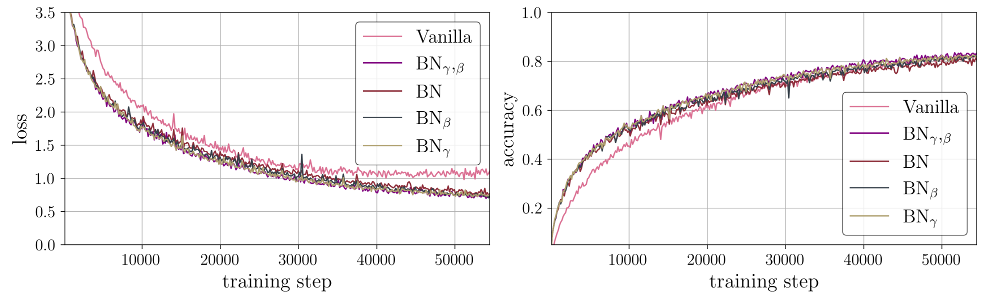


(A) CIFAR-10 3C3D model trained with SGD and a learning rate of 0.022×10^{-1}.

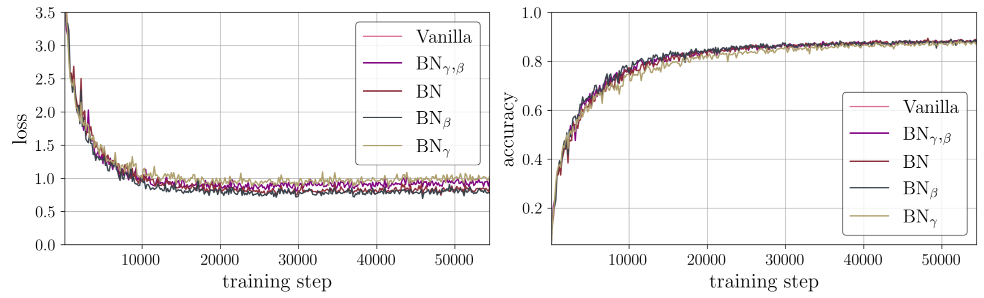


(B) CIFAR-10 3C3D trained with SGD and a learning rate of 0.022×10^{1}.

FIGURE 4.2: Training loss and accuracy for the CIFAR-10 3C3D model trained for 100 epochs, a batch size of 128, and learning rates varying in the order of magnitude (0.022×10^{-1} and 0.022×10^{1}).

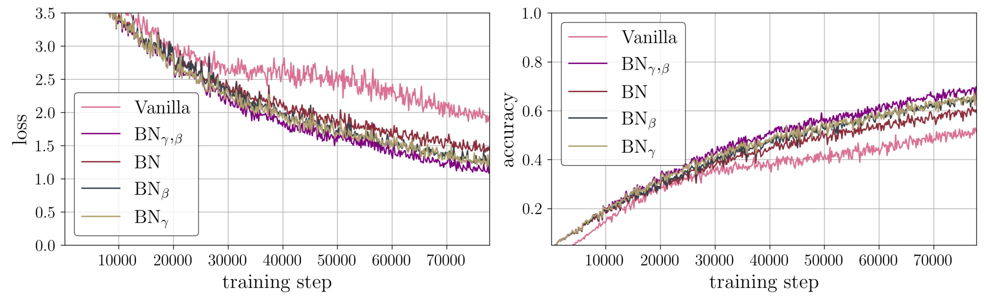


(A) CIFAR-100 3C3D model trained with SGD and a learning rate of 0.166×10^{-1}.

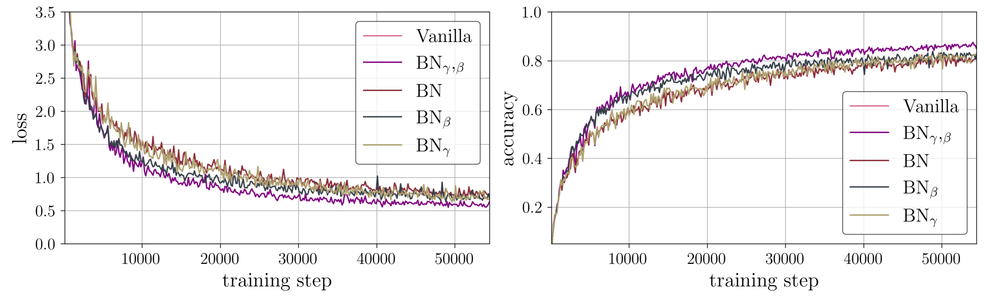


(B) CIFAR-100 3C3D model trained with SGD and a learning rate of 0.166×10^{1}. With this learning rate, the Vanilla model ● could not be trained at all.

FIGURE 4.3: Training loss and accuracy for the CIFAR-100 3C3D model trained for 350 epochs with a batch size of 256, and learning rates vary in the order of magnitude (0.166×10^{-1} and 0.166×10^{1}).



(A) CIFAR-100 ALL-CNN-C model trained with SGD and a learning rate of 0.166×10^{-1}.



(B) CIFAR-100 ALL-CNN-C trained with SGD and a learning rate of 0.166×10^{1}. With this learning rate, the Vanilla model ● could not be trained at all.

FIGURE 4.4: Training loss and accuracy for the CIFAR-100 ALL-CNN-C model trained for 350 epochs with a batch size of 256 and learning rates varying in the order of magnitude (0.166×10^{-1} and 0.166×10^{1}).

4.1.3 The Last Layer is a Special Case

For the investigation focused on the difference of performance contributions if BatchNorm is placed at the final layer, we added a BatchNorm layer after the last convolutional layer in the CIFAR-100 ALL-CNN-C model, which means in this experiment the model has one BatchNorm layer more compared to the original CIFAR-100 ALL-CNN-C in Section 4.1.1. The experiments show that the contributions of the respective BatchNorm components to the model performance varied significantly from prior experiments (Figure 4.5). Notably, the use of non-adaptive normalization in isolation resulted in a pronounced drop in accuracy and an increase in loss compared to the Vanilla model. Surprisingly, γ alone but even more both learnable parameters together can compensate for the negative effect of the non-adaptive normalization and make the application of BatchNorm beneficial for the model.

In comparison to the findings in Figure 4.1, we contend that the role of the learnable parameters in this setting is adjusted due to the idiosyncrasies of the last layer. Moreover, we tested if the change in the impact of the BatchNorm can be observed for other positionings and varying numbers of BatchNorm layers (See Appendix A). However, none of the experiments showed a comparable change in the performance contribution, except if there is a BatchNorm layer placed after the last convolutional layer.

Nevertheless, it is noteworthy that the training performance of the original BatchNorm $BN_{\gamma,\beta}$ here is similar to the performance of $BN_{\gamma,\beta}$ from Figure 4.1c, which does not have a BatchNorm layer after the last convolution. Hence, placing an additional BatchNorm at the last layer in a DNN does not necessarily interfere with model performance but changes the role of the learnable parameters γ and $\boldsymbol{\beta}$.

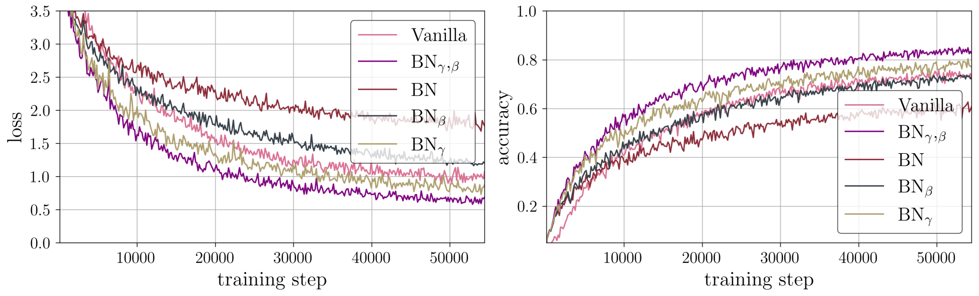


FIGURE 4.5: Training loss and accuracy for the CIFAR-100 ALL-CNN-C model with BatchNorm **at the last layer**, for all modifications of BatchNorm trained with SGD for 350 epochs with a batch size of 256 and a learning rate of 0.166. The Vanilla version of the model ● and the original BatchNorm version ● serve as lower and upper baselines for the comparison of performance contributions, respectively.

4.2 Results from the Activity Analysis

In our endeavor to analyze the training behavior of learnable parameters, $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$, our primary focus is on their activity, which we characterized in Section 3.3.1. In Section 4.2.1, we present the behavior of the learnable parameters for cases in which those affect the training performance. Since the experiments in Section 3.1 have shown this only applies for the CIFAR-100 ALL-CNN-C model, we analyze the activity of $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$ on that model first. We do this for both the experiments with the general setup of BatchNorm layers (Section 4.1), described in Section 3.1.2 and the experiments with an additional BatchNorm layer at the end of the CIFAR-100 ALL-CNN-C from Section 4.1.3. Moreover, we set the findings from Section 4.2.1 in contrast to the behavior of $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$ for cases in which those do not affect the training performance (Section 4.2.2).

To effectively monitor the trajectory of the parameters over training, we opted to measure the values of the $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$ vectors across various layers every five steps. This strategy was chosen as a balance, allowing us to mitigate computational overhead while ensuring that we capture sufficient and pertinent data.

4.2.1 The Learnable Component Matters

CIFAR-100 ALL-CNN-C Without BatchNorm at the Last Layer. For both modifications that train the $\boldsymbol{\beta}$ parameter ($BN_{\gamma,\beta}$ and BN_{β}) a clear pattern of negative and positive shifts can be observed, according to the position of a BatchNorm layer in the network (Figure 4.6). At the first layer, $\boldsymbol{\beta}$ induces positive and negative shifts across the feature dimensions, while at the intermediate layers, $\boldsymbol{\beta}$ is negative or zero across all feature dimensions. At the last layer, $\boldsymbol{\beta}$ shows a trend towards positive values. In Figure 4.6b, the overall scale of the $\boldsymbol{\beta}$ values is shifted towards the positive value range compared to the $\boldsymbol{\beta}$ values of the $BN_{\gamma,\beta}$ (Figure 4.6a). The maximum negative shift at the intermediate layers is halved, and the positive shift at the last BatchNorm layer is approximately twice as strong.

The $\boldsymbol{\gamma}$ parameter as well shows similar patterns across modifications, here $BN_{\gamma,\beta}$ and BN_{γ} (See Figure 4.7). In the first layers, $\boldsymbol{\gamma}$ converges to values that amplify and dampen the activations across the feature dimensions. Especially in the intermediate layers, $\boldsymbol{\gamma}$ shows little activity in relation to the activity in the last layer. Here, $\boldsymbol{\gamma}$ converges to values amplifying the activations across all channels.

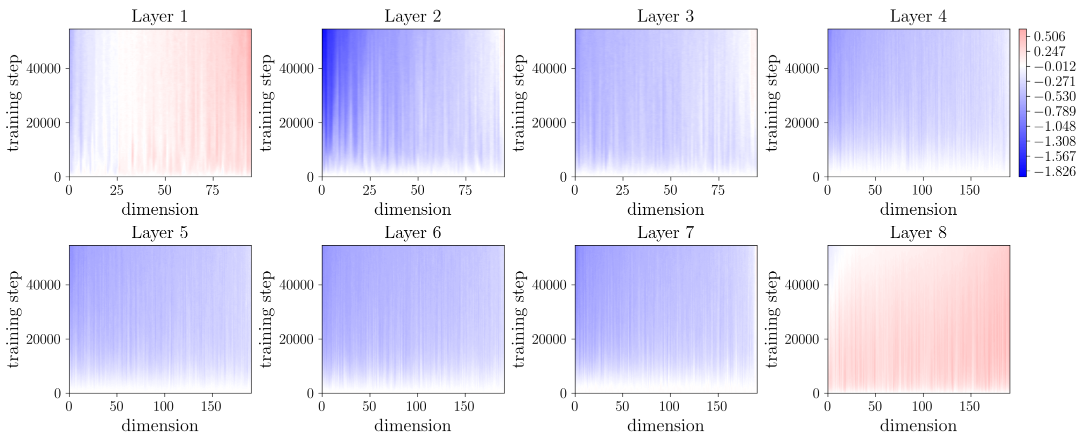
In contrast, with the modification BN_{β} the scale of the $\boldsymbol{\gamma}$ values across layers does not significantly change. Considering the performance contribution of $\boldsymbol{\gamma}$ in Figure 4.1c, the impact of the parameter in this setting might not be significant enough in order for the optimizer to change the trajectory.

CIFAR-100 ALL-CNN-C With BatchNorm at the Last Layer. In Section 4.1.3, we observe a drastic change in the performance contribution of $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ when BatchNorm is placed at the last layer of the CIFAR-100 ALL-CNN-C. Figure 4.8 shows the behavior of $\boldsymbol{\beta}$ for the modifications $BN_{\gamma,\beta}$ and BN_{β} across all layers. Similarly, Figure 4.9 provides the visualization of the $\boldsymbol{\gamma}$ parameter for $BN_{\gamma,\beta}$ and BN_{γ}. In general, we can observe a similar pattern compared to the CIFAR-100 ALL-CNN-C model without BatchNorm at the last layer (See Figure 4.6 and Figure 4.7) regarding the individual trends of the learnable parameters in the first, intermediate, and last BatchNorm layers.

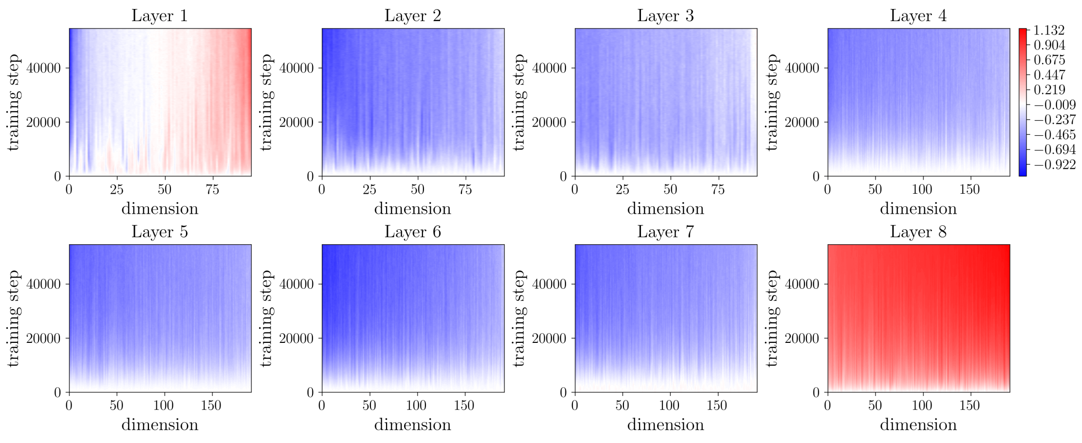
However, we could observe the scale of the $\boldsymbol{\beta}$ values of BN_{β} on the positive range is reduced almost by half (Figure 4.8b). Considering that BN_{β} performs worse than the Vanilla model, for this setting (See Figure 4.5), the reduction of the $\boldsymbol{\beta}$ activity could be an indication for the decreased performance. For a confident assessment, additional experiments revealing the same relative performance contributions of the learnable parameters in BatchNorm would be necessary.

In contrast, γ shows a drastic increase regarding the scale of the values across layers. The activation of γ at the last BatchNorm layer is approximately doubled for $BN_{\gamma,\beta}$ and BN_γ compared to the same model without BatchNorm at the last layer in the network (Figure 4.7). Considering that BN_γ can improve model performance in Figure 4.5, the high activity of γ could be an indication of the beneficial effects of that parameter in the training process. Same as with β, this trend requires additional experiments. The comparison between the two modifications, $BN_{\gamma,\beta}$ and BN_γ shows the only difference lies in the small shift of the value scale towards the positive direction with BN_γ.

It is conspicuous that for this positioning of BatchNorm layers, γ stays very close to one across all layers, except for the last layer, meaning the optimizer only adjusts γ at the last BatchNorm layer in the network. The results suggest that for the CIFAR-100 ALL-CNN-C with an additional BatchNorm layer at the end, γ can be fixed to one at all layers and only trained for the last BatchNorm layer without experiencing a decrease in performance.

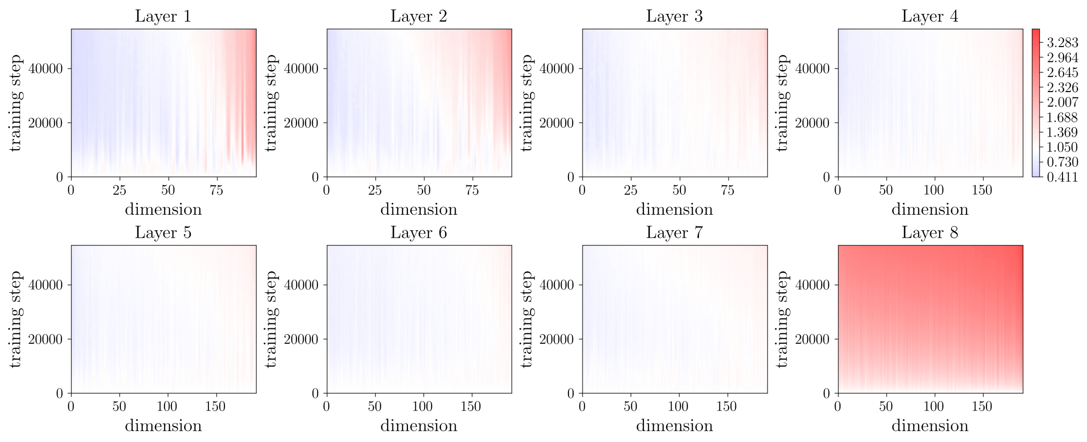


(A) β values of the CIFAR-100 ALL-CNN-C model, trained with **original BatchNorm $BN_{\gamma,\beta}$**.

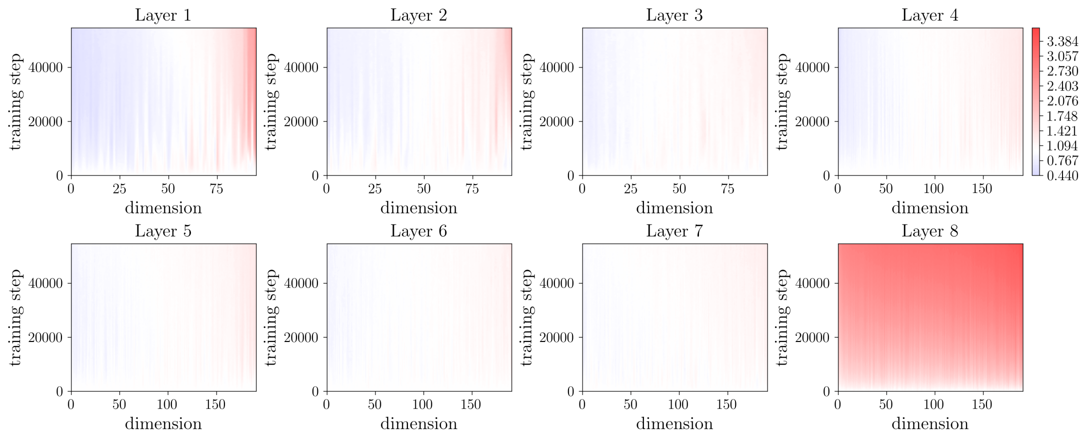


(B) β values of the CIFAR-100 ALL-CNN-C model, trained with the **BN_β BatchNorm modification**.

FIGURE 4.6: Course of β values across all eight BatchNorm layers of the CIFAR-100 ALL-CNN-C model, trained for 350 epochs with a learning rate of 0.166 and a batch size of 256. Note that (A) and (B) have different colorscales.

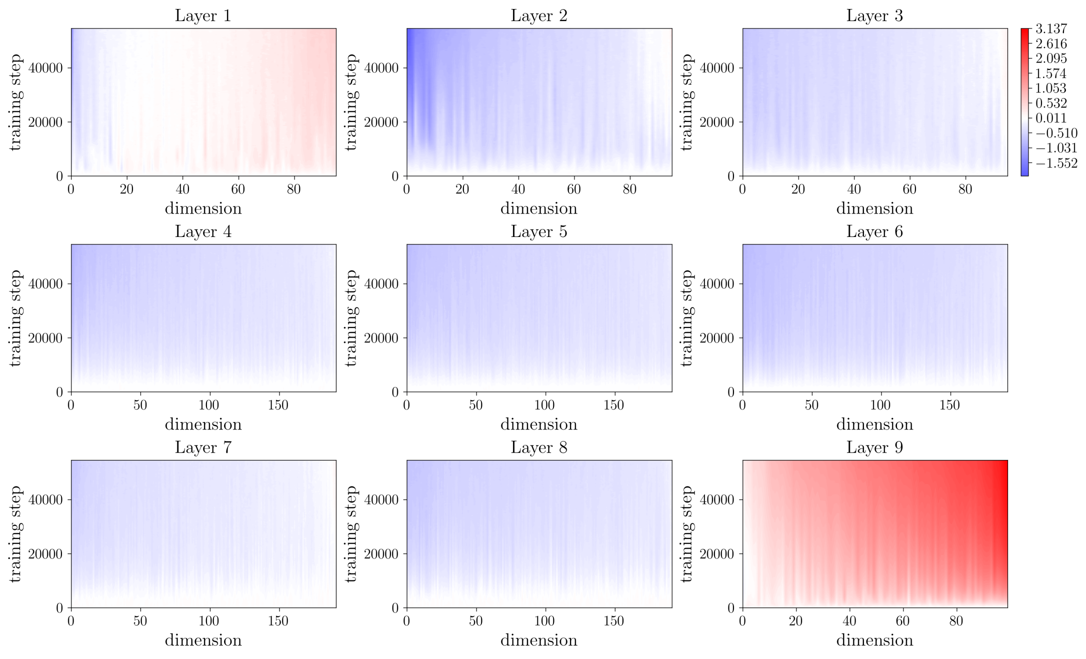


(A) γ values of the CIFAR-100 ALL-CNN-C model, trained with **original BatchNorm $BN_{\gamma,\beta}$**.

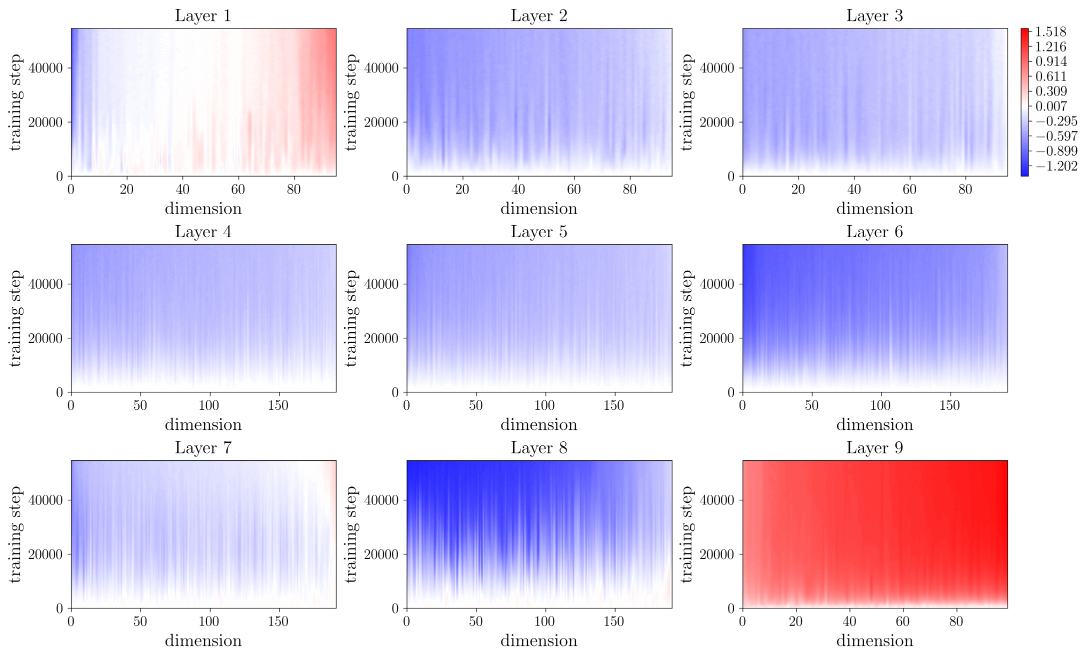


(B) γ values of the CIFAR-100 ALL-CNN-C model, trained with the **BN_{γ} BatchNorm modification**.

FIGURE 4.7: Course of γ values across all eight BatchNorm layers of the CIFAR-100 ALL-CNN-C model, trained for 350 epochs with a learning rate of 0.166 and a batch size of 256.. Note that (A) and (B) have different colorscales.

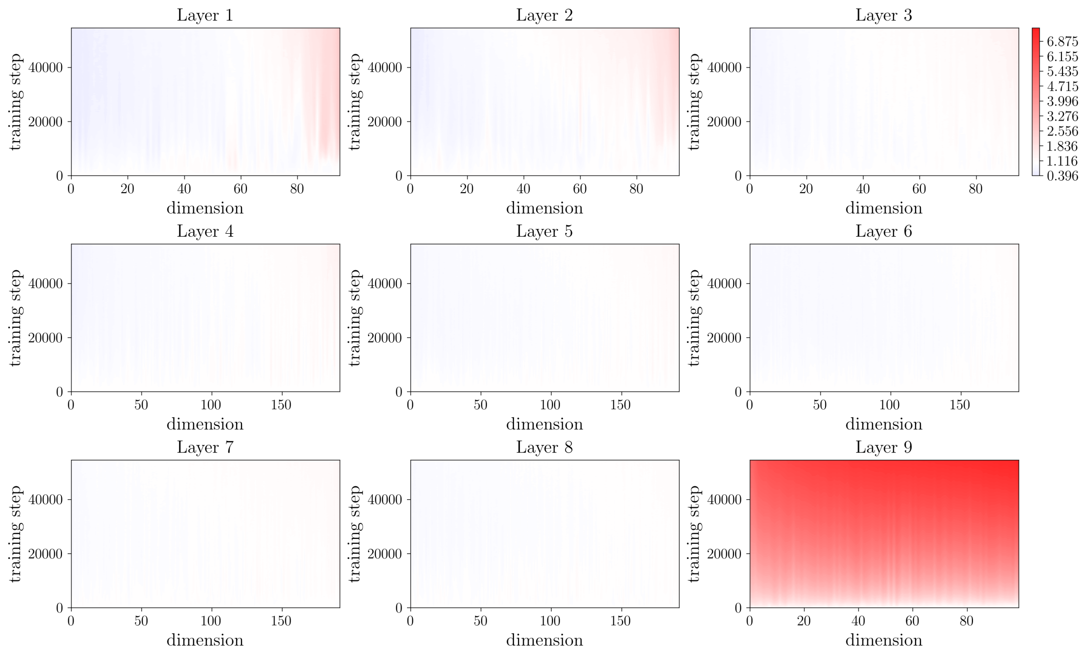


(A) β values of the CIFAR-100 ALL-CNN-C model, trained with **original BatchNorm** $BN_{\gamma,\beta}$.

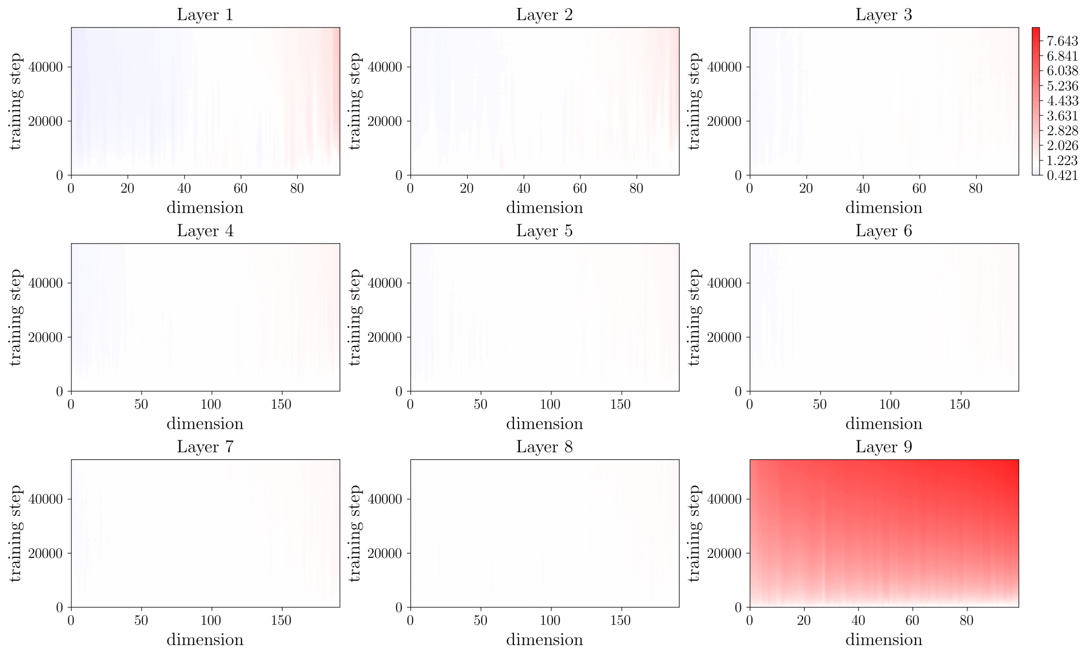


(B) β values of the CIFAR-100 ALL-CNN-C model, trained with the BN_{β} **BatchNorm modification**.

FIGURE 4.8: Course of β values of all BatchNorm layers of the CIFAR-100 ALL-CNN-C model, with BatchNorm **at the last layer**, trained for 350 epochs with a learning rate of 0.166 and a batch size of 256.



(A) γ values of the CIFAR-100 ALL-CNN-C model, trained with **original BatchNorm** $BN_{\gamma,\beta}$.



(B) γ values of the CIFAR-100 ALL-CNN-C model, trained with the BN_{β} **BatchNorm modification**.

FIGURE 4.9: Course of γ values across all bn layers of the CIFAR-100 ALL-CNN-C model, with BatchNorm **at the last layer**, trained for 350 epochs with a learning rate of 0.166 and a batch size of 256.

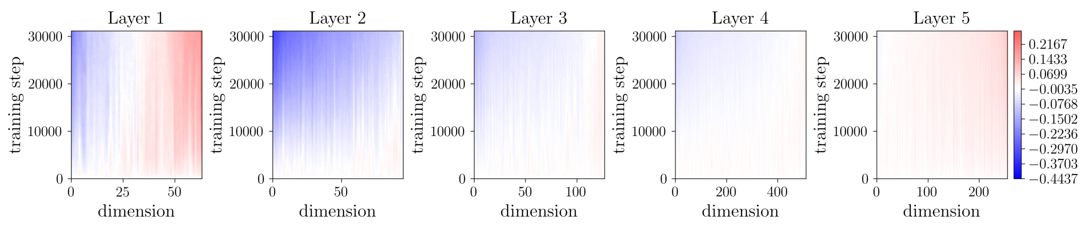
4.2.2 The Learnable Component Does Not Matter

For both models, CIFAR-10 3C3D and CIFAR-100 3C3D, the learnable parameters do not show any significant effect on the training performance in Section 4.1.1. However, the behavior of the parameters can still be used to contrast it to behavior from scenarios in which γ and β have a significant impact.

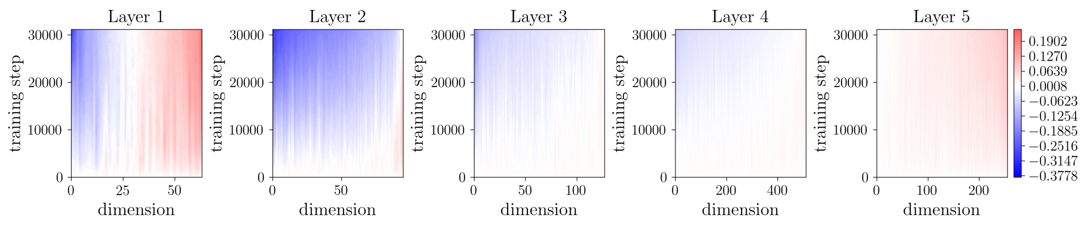
CIFAR-10 3C3D. Although not affecting training performance, the β parameter is adjusted at the first two layers for both modifications of BatchNorm, $BN_{\gamma,\beta}$ and BN_{β} (Figure 4.10). Overall, β shows a marginal trend towards negative values at intermediate layers and positive values at the last BatchNorm layer. However, γ, as expected, shows little activity across all layers and training steps, even for the last layer (Figure 4.11). For both parameters, the scale of the values across layers is marginally shifted towards the positive direction if the other parameter is ablated from the equation.

CIFAR-100 3C3D. Figure 4.12 shows, that γ and β are active across all BatchNorm layers, and both modifications. Although the activity is stronger for the CIFAR-100 ALL-CNN-C model, it is surprising that the optimizer is adjusting the learnable parameters at all. This could indicate that the slight performance differences in Figure 4.1b are not due to noise. However, this claim would require additional experiments, for example, with a tuned hyperparameter setting for each modification, left for future work.

Across layers, β shows a similar pattern compared to the other two models. Again, the parameter induces negative and positive shifts in the first layer, negatively shifts the activations at the intermediate layers, and applies a positive shift on the activations in the last layer. The amplification induced by γ of the last BatchNorm layer is significantly stronger compared to all other layers. Regarding the scale of values across layers, we can observe a shift towards the positive value space for both parameters. This effect aligns with the observations from the other experiments with the CIFAR-100 ALL-CNN-C (Figure 4.7 and Figure 4.9) and the CIFAR-10 3C3D (Figure 4.11, such that it cannot be attributed to the performance contribution of the learnable parameters.

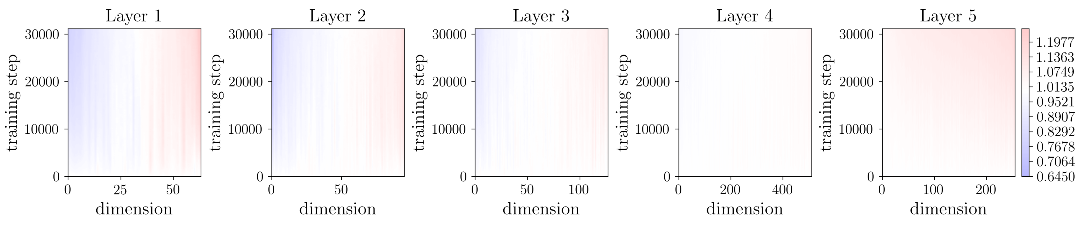


(A) β values of **original BatchNorm** $BN_{\gamma,\beta}$.

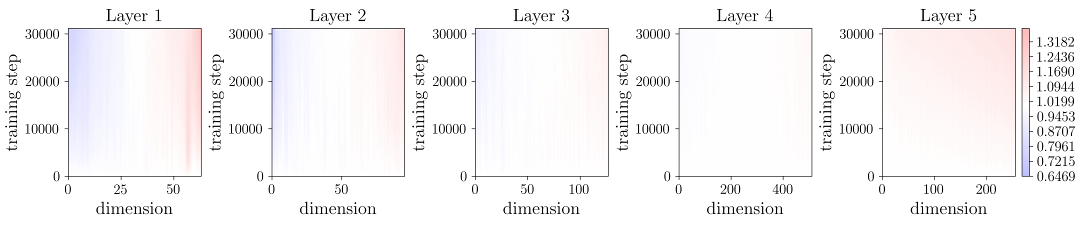


(B) β values of the BN_{β} **BatchNorm modification**.

FIGURE 4.10: Course of β values across all five BatchNorm layers of the CIFAR-10 3C3D model, trained for 100 epochs with a learning rate of 0.022 and a batch size of 128. Note that (A) and (B) have different colorscales.

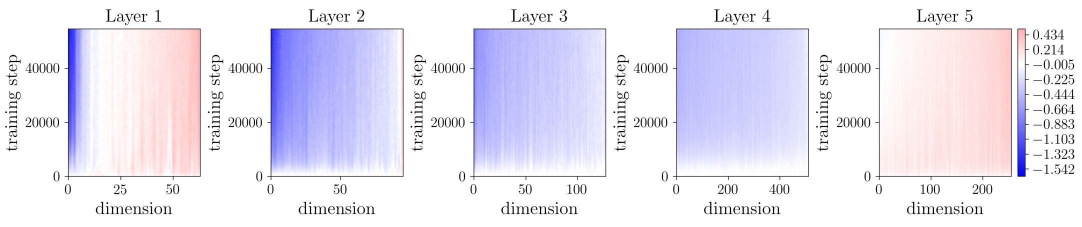


(A) γ values of **original** BatchNorm $BN_{\gamma,\beta}$.

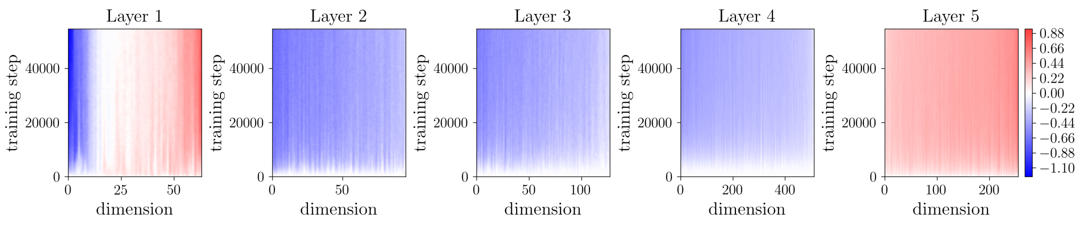


(B) γ values of the BN_{γ} **BatchNorm modification**.

FIGURE 4.11: Course of γ values across all five BatchNorm layers of the CIFAR-10 3C3D model, trained for 100 epochs with a learning rate of 0.022 and a batch size of 128. Note that (A) and (B) have different colorscales.



(A) β values of the **original** BatchNorm $BN_{\gamma,\beta}$.

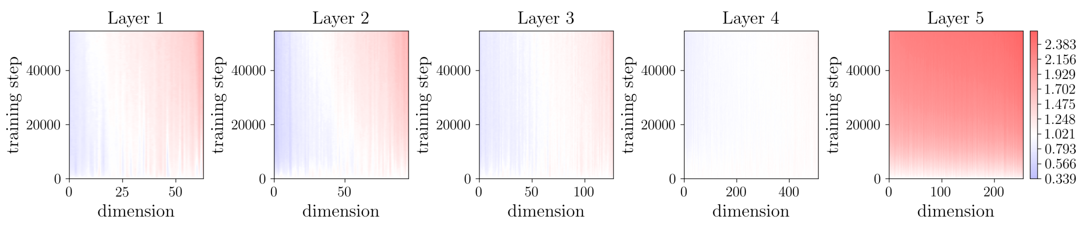


(B) β values of the BN_{β} **BatchNorm modification**.

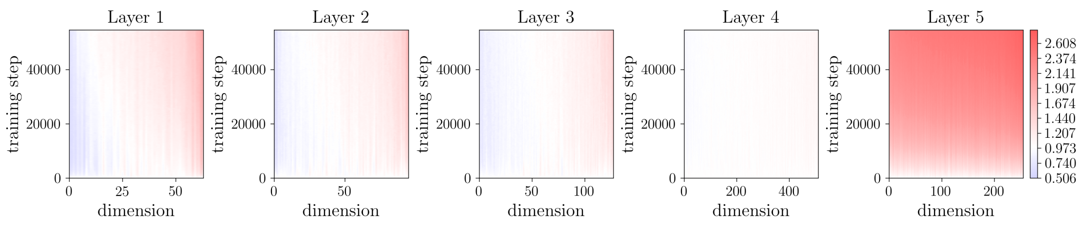
FIGURE 4.12: Course of β values across all five BatchNorm layers of the CIFAR-100 3C3D model trained for 350 epochs with a learning rate of 0.166 and a batch size of 256. Note that (A) and (B) have different colorscales.

4.2.3 Summary of Results

Throughout our empirical study examining the training behavior of the learnable parameters of BatchNorm across various network configurations, several consistent patterns emerged. For both parameters γ and β, the pattern exhibited is primarily characterized by the position of the BatchNorm layer within the network architecture, irrespective of its eventual performance impact. In the first BatchNorm layer, β converges to both positive and negative values across channels. As the input progresses to intermediate layers, there is a discernible negative shift in activations across all dimensions. Though the degree of this shift varies, each feature dimension consistently remains non-positive.



(A) γ values of the **original BatchNorm** $BN_{\gamma,\beta}$.



(B) γ values of the BN_{γ} **BatchNorm modification**.

FIGURE 4.13: Course of γ values across all eight BatchNorm layers of the CIFAR-100 3C3D model trained for 350 epochs with a learning rate of 0.166 and a batch size of 256. Note that (A) and (B) have different colorscales.

However, in the final BatchNorm layers, β demonstrates a propensity to shift activations positively across all feature dimensions. This pattern suggests that during the processing in intermediate BatchNorm layers, a considerable fraction of activations are zeroed by the ReLU function, owing to their negative values, similar to the functionality of dropout (Defined in Section 2.1.4).

On the other hand, γ displayed a more reserved behavior. Its activity was especially limited during the initial phases of training and within intermediate BatchNorm layers. Yet, there was a noticeable trend for γ to amplify activations in the last BatchNorm layer. This amplifying trait was particularly evident in the CIFAR-100 ALL-CNN-C and CIFAR-100 3C3D models, whereas it was notably absent in the CIFAR-10 3C3D configuration. Considering that in the latter two models, γ did not discernibly affect training performance, it is challenging to correlate γ's activity directly with its impact on performance.

Across all the models, another consistent observation was the trend of the learnable parameters to apply their most substantial positive shifts and amplifications on activations, specifically in the last layer. This recurrent pattern across models suggests that there might be inherent benefits in training performance by minimizing activations in intermediate layers and conserving them in the final BatchNorm layers.

Chapter 5

Discussion and Outlook

This study delved into the specifics of BatchNorm, especially emphasizing the roles and impacts of its respective components. Our approach involved creating modifications of BatchNorm to discern the relative contribution of the non-adaptive normalization and the learnable parameters to training performance. We ensured the contributions were consistent across multiple learning rates and different DL setups.

For the smaller 3C3D network, we demonstrated that the non-adaptive normalization in BatchNorm solely contributes to the performance improvement. The learnable parameters γ and β do not offer any significant effect. In contrast, with the deeper CIFAR-100 ALL-CNN-C model, both learnable parameters are necessary to achieve the best performance improvements. Beyond that, the non-adaptive normalization is the component that primarily induces the beneficial effects of BatchNorm on the training performance in all cases.

A distinct observation of performance contributions only emerged when placing BatchNorm in the network's last layer. Here, relying solely on non-adaptive normalization adversely impacts training performance. Only with both learnable parameters together the model performance can be improved in comparison to the Vanilla version of that model. Experiments with the last layer have shown that the success of BatchNorm cannot be attributed solely to one of the components. While in most cases, the non-adaptive normalization induces the major effects, depending on the network architecture and the position of the BatchNorm layers, the contribution of the learnable parameters can be crucial for the method to be beneficial at all.

Further, we extended the experiments by visualizing the behavior of the learnable parameters γ and β. We contrasted the behavior of the learnable parameters for cases in which those matter to cases in which the behavior of the γ and β did not matter. Our experiments did not show behavior patterns that align with the performance contribution of the learnable parameters. The performance contribution of the learnable parameters did not show in the way those are optimized over the training process. Nevertheless, our observations can provide insights into the general behavior of γ and β. The trajectory of the two parameters is denoted by the position of a BatchNorm layer in the network. γ and β show a specific trend at all first layers, intermediate layers, and the last BatchNorm layer, respectively. These findings suggest one of the reasons for the success of BatchNorm could lie in the way γ and β affect the number of activations that are zeroed by the subsequent ReLU function.

Our study has limitations related to its empirical nature. The 3C3D network used in our investigations might have ample capacity to tackle the classification task in our experiments, limiting the scope of improvement via BatchNorm. Another aspect regards the visual comparison method employed to understand the behavior of the learnable parameters. While the qualitative inspection allowed us to identify clear trends and propose new ideas, incorporating quantitative metrics could reinforce the findings and potentially reveal additional information. Potential metrics include the mean or median, magnitude, and number of zero-crossings for the trajectories of γ and β.

The findings from our empirical analysis provide a source of motivation for future works on the comprehensive exploration of BatchNorm and various initial approaches for further experiments. A natural extension to this work would be incorporating additional networks and datasets, allowing for a more systematic exploration of the effects of network depth and dataset complexity on the efficacy and behavior of the different BatchNorm components. In addition, our findings also underscore the significance of the learnable parameters when BatchNorm is added to the last layer of DNN. Experiments in which γ and β remain static in initial BatchNorm layers and become trainable in the last layers could offer computational advantages while maintaining performance improvement. Another intriguing notion is the operational similarity between BatchNorm and the ReLU activation function to dropout mechanisms. An in-depth exploration of how γ and β affect the number of activations that get zeroed out by the subsequent activation function and how this is related to the model performance could provide valuable findings on training DNNs in general. Our exploration has provided novel insights into the mechanisms of BatchNorm and its learnable parameters. The outlined research directions above promise a deeper understanding and more refined applications of this technique in future neural network designs.

Bibliography

Abadi, Martín et al. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org.

Amodei, Dario et al. (June 2016). "Deep Speech 2 : End-to-End Speech Recognition in English and Mandarin". In: *Proceedings of The 33rd International Conference on Machine Learning*. Ed. by Maria Florina Balcan and Kilian Q. Weinberger. Vol. 48. Proceedings of Machine Learning Research. New York, New York, USA: PMLR, pp. 173–182.

Awais, Muhammad, Md Tauhid Iqbal, and Sung-Ho Bae (Sept. 2020). "Revisiting Internal Covariant Shift for Batch Normalization". In: *IEEE Transactions on Neural Networks and Learning Systems* PP. DOI: 10.1109/TNNLS.2020.3026784.

Ba, Lei Jimmy, Jamie Ryan Kiros, and Geoffrey E. Hinton (2016). "Layer Normalization". In: *CoRR* abs/1607.06450. arXiv: 1607.06450.

Bjorck, Johan, Carla P. Gomes, and Bart Selman (2018). "Understanding Batch Normalization". In: *CoRR* abs/1806.02375. arXiv: 1806.02375.

Cai, Yongqiang, Qianxiao Li, and Zuowei Shen (Sept. 2018). "A Quantitative Analysis of the Effect of Batch Normalization on Gradient Descent". In: *arXiv e-prints*, arXiv:1810.00122, arXiv:1810.00122. DOI: 10.48550/arXiv.1810.00122. arXiv: 1810.00122 [cs.LG].

Clevert, Djork-Arné, Thomas Unterthiner, and Sepp Hochreiter (2016). "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)". In: *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun.

Cooijmans, Tim et al. (2016). "Recurrent Batch Normalization". In: *CoRR* abs/1603.09025. arXiv: 1603.09025.

Dauphin, Yann N. et al. (2014). "Identifying and Attacking the Saddle Point Problem in High-Dimensional Non-Convex Optimization". In: *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*. NIPS'14. Montreal, Canada: MIT Press, pp. 2933–2941.

Davis, Jim and Logan Frank (Oct. 2021). "Revisiting Batch Norm Initialization". In: DOI: 10.48550/ARXIV.2110.13989. arXiv: 2110.13989 [cs.CV].

Dosovitskiy, Alexey et al. (2021). "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *International Conference on Learning Representations*.

Dubey, Shiv Ram, Satish Kumar Singh, and Bidyut Baran Chaudhuri (2022). "Activation functions in deep learning: A comprehensive survey and benchmark". In: *Neurocomputing* 503, pp. 92–108. DOI: 10.1016/j.neucom.2022.06.111.

Frankle, Jonathan, David J. Schwab, and Ari S. Morcos (2020). "Training BatchNorm and Only BatchNorm: On the Expressive Power of Random Features in CNNs". In: *CoRR* abs/2003.00152. arXiv: 2003.00152.

Friedman, Jerome H. (1987). "Exploratory Projection Pursuit". In: *Journal of the American Statistical Association* 82.397, pp. 249–266. ISSN: 01621459.

Fukushima, Kunihiko (1975). "Cognitron: A self-organizing multilayered neural network". In: *Biological Cybernetics* 20, pp. 121–136.

Ghorbani, Behrooz, Shankar Krishnan, and Ying Xiao (2019). “An Investigation into Neural Net Optimization via Hessian Eigenvalue Density”. In: *CoRR* abs/1901.10159. arXiv: 1901.10159.

Gitman, Igor and Boris Ginsburg (2017). “Comparison of Batch Normalization and Weight Normalization Algorithms for the Large-scale Image Classification”. In: *ArXiv* abs/1709.08145.

Glorot, Xavier and Yoshua Bengio (2010). “Understanding the difficulty of training deep feedforward neural networks”. In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2010, Chia Laguna Resort, Sardinia, Italy, May 13-15, 2010*. Ed. by Yee Whye Teh and D. Mike Titterington. Vol. 9. JMLR Proceedings. JMLR.org, pp. 249–256.

Glorot, Xavier, Antoine Bordes, and Yoshua Bengio (Apr. 2011). “Deep Sparse Rectifier Neural Networks”. In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Geoffrey Gordon, David Dunson, and Miroslav Dudík. Vol. 15. Proceedings of Machine Learning Research. Fort Lauderdale, FL, USA: PMLR, pp. 315–323.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep learning*. English. Adapt. Comput. Mach. Learn. Cambridge, MA: MIT Press. ISBN: 978-0-262-03561-3.

Gouk, Henry et al. (Feb. 2021). “Regularisation of Neural Networks by Enforcing Lipschitz Continuity”. In: *Mach. Learn.* 110.2, pp. 393–416. ISSN: 0885-6125. DOI: 10.1007/s10994-020-05929-w.

Hasani, Moein and Hassan Khotanlou (2019). *An Empirical Study on Position of the Batch Normalization Layer in Convolutional Neural Networks*. DOI: 10.1109/icspis48872.2019.9066113.

He, Kaiming et al. (2016). “Deep Residual Learning for Image Recognition”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

Hoedt, Pieter-Jan, Sepp Hochreiter, and Günter Klambauer (2022). “Normalization is dead, long live normalization!” In: *ICLR Blog Track*.

Hoefler, Torsten et al. (2021). “Sparsity in Deep Learning: Pruning and growth for efficient inference and training in neural networks”. In: *Journal of Machine Learning Research* 22.241, pp. 1–124.

Ioffe, Sergey and Christian Szegedy (2015). “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *CoRR* abs/1502.03167. arXiv: 1502.03167.

Janocha, Katarzyna and Wojciech Marian Czarnecki (2017). “On Loss Functions for Deep Neural Networks in Classification”. In: *CoRR* abs/1702.05659. arXiv: 1702.05659.

Kim, Bum Jun et al. (Oct. 2022). “Smooth Momentum: Improving Lipschitzness in Gradient Descent”. In: *Applied Intelligence* 53.11, pp. 14233–14248. ISSN: 0924-669X. DOI: 10.1007/s10489-022-04207-7.

Kohler, Jonas et al. (May 2018). “Exponential convergence rates for Batch Normalization: The power of length-direction decoupling in non-convex optimization”. In: *arXiv e-prints*, arXiv:1805.10694, arXiv:1805.10694. DOI: 10.48550/arXiv.1805.10694. arXiv: 1805.10694 [stat.ML].

Krizhevsky, Alex (2009). *Learning multiple layers of features from tiny images*. Tech. rep. Toronto: University of Toronto.

Kumar, Siddharth Krishna (2017). “On weight initialization in deep neural networks”. In: *CoRR* abs/1704.08863. arXiv: 1704.08863.

Laurent, César et al. (2015). "Batch Normalized Recurrent Neural Networks". In: *CoRR* abs/1510.01378. arXiv: 1510.01378.

LeCun, Yann, Leon Bottou, Yoshua Bengio, et al. (1998). "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11, pp. 2278–2324. DOI: 10.1109/5.726791.

LeCun, Yann, Leon Bottou, Genevieve B. Orr, et al. (1998). *Efficient BackProp*. DOI: 10.1007/3-540-49430-8_2.

Lewkowycz, Aitor et al. (2020). "The large learning rate phase of deep learning: the catapult mechanism". In: *CoRR* abs/2003.02218. arXiv: 2003.02218.

Luo, Ping et al. (2018). "Towards Understanding Regularization in Batch Normalization". In: *CoRR* abs/1809.00846. arXiv: 1809.00846.

Mu, Dongliang et al. (2020). "RENN: Efficient Reverse Execution with Neural-Network-Assisted Alias Analysis". In: *Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering*. ASE '19. San Diego, California: IEEE Press, pp. 924–935. ISBN: 9781728125084. DOI: 10.1109/ASE.2019.00090.

Paszke, Adam et al. (2019). "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc.

Peerthum, Yashna (Mar. 2023). *Empirical Evaluation of the Shift and Scale Parameters in Batch Normalization*. DOI: 10.31979/ARXIV.2303.12818.

Qiao, Siyuan et al. (Mar. 2019). "Micro-Batch Training with Batch-Channel Normalization and Weight Standardization". In: *arXiv e-prints*, arXiv:1903.10520, arXiv:1903.10520. DOI: 10.48550/arXiv.1903.10520. arXiv: 1903.10520 [cs.CV].

Rao, Vinay and Jascha Sohl-Dickstein (2020). "Is Batch Norm unique? An empirical investigation and prescription to emulate the best properties of common normalizers without batch dependence". In: *CoRR* abs/2010.10687. arXiv: 2010.10687.

Robbins, Herbert and Sutton Monro (1951). "A Stochastic Approximation Method". In: 22, pp. 400–407. ISSN: 0003-4851. DOI: 10.1214/aoms/1177729586.

Sainath, Tara N. et al. (2013). "Improving training time of Hessian-free optimization for deep neural networks using preconditioning and sampling". In: *CoRR* abs/1309.1508. arXiv: 1309.1508.

Salimans, Tim and Diederik P. Kingma (2016). "Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks". In: *CoRR* abs/1602.07868. arXiv: 1602.07868.

Santurkar, Shibani et al. (2019). *How Does Batch Normalization Help Optimization?* arXiv: 1805.11604 [stat.ML].

Schneider, Frank, Lukas Balles, and Philipp Hennig (2019). "DeepOBS: A Deep Learning Optimizer Benchmark Suite". In: *CoRR* abs/1903.05499. arXiv: 1903.05499.

Smith, Samuel L., Pieter-Jan Kindermans, and Quoc V. Le (2017). "Don't Decay the Learning Rate, Increase the Batch Size". In: *CoRR* abs/1711.00489. arXiv: 1711.00489.

Springenberg, Jost Tobias et al. (2015). "Striving for Simplicity: The All Convolutional Net". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Workshop Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun.

Srivastava, Nitish et al. (2014). "Dropout: a simple way to prevent neural networks from overfitting". In: *J. Mach. Learn. Res.* 15.1, pp. 1929–1958. DOI: 10.5555/2627435.2670313.

Szegedy, Christian, Sergey Ioffe, et al. (2017). "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning". In: *Proceedings of the Thirty-First*

AAAI Conference on Artificial Intelligence. AAAI'17. San Francisco, California, USA: AAAI Press, pp. 4278–4284.

Szegedy, Christian, Vincent Vanhoucke, et al. (June 2016). "Rethinking the Inception Architecture for Computer Vision". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Teye, Mattias, Hossein Azizpour, and Kevin Smith (July 2018). "Bayesian Uncertainty Estimation for Batch Normalized Deep Networks". In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, pp. 4907–4916.

Thakkar, Vignesh, S. Tewary, and C. Chakraborty (2018). "Batch Normalization in Convolutional Neural Networks — A comparative study with CIFAR-10 data". In: *2018 Fifth International Conference on Emerging Applications of Information Technology (EAIT)*, pp. 1–5. DOI: 10.1109/EAIT.2018.8470438.

Ulyanov, Dmitry, Andrea Vedaldi, and Victor S. Lempitsky (2016). "Instance Normalization: The Missing Ingredient for Fast Stylization". In: *CoRR* abs/1607.08022. arXiv: 1607.08022.

Vaswani, Ashish et al. (2017). "Attention is All You Need". In.

Wu, Yuxin and Kaiming He (2018). "Group Normalization". In: *International Journal of Computer Vision* 128, pp. 742–755.

Xiang, Sitao and Hao Li (2017). "On the effect of Batch Normalization and Weight Normalization in Generative Adversarial Networks". In: *CoRR* abs/1704.03971. arXiv: 1704.03971.

Xu, Jingjing et al. (2019). "Understanding and Improving Layer Normalization". In: *CoRR* abs/1911.07013. arXiv: 1911.07013.

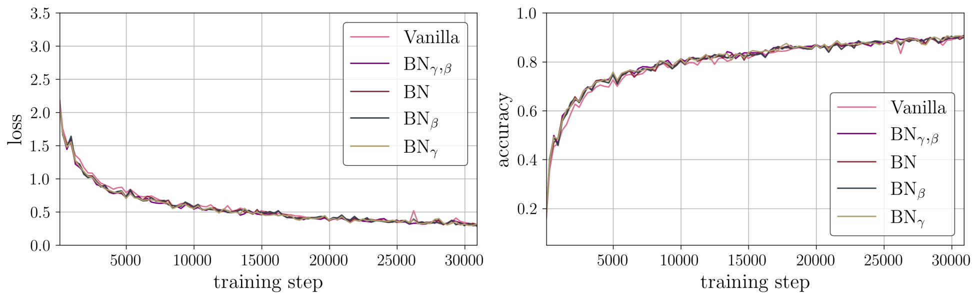
Zhou, Bolei et al. (2016). "Learning Deep Features for Discriminative Localization". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2921–2929. DOI: 10.1109/CVPR.2016.319.

Appendix A

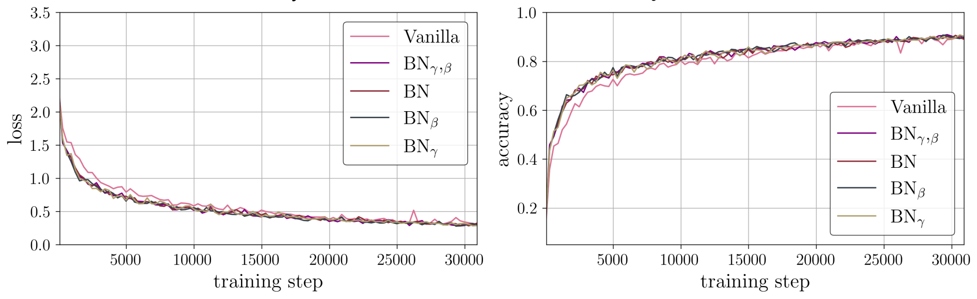
Additional Experiments

A.1 Varying Number and Positioning of BatchNorm Layers

A.1.1 Cifar10 3C3D

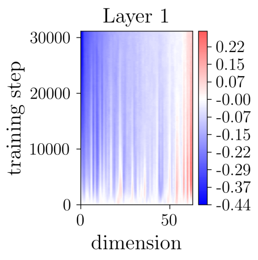


(A) One BatchNorm layer after the **first convolutional layer** in the Cifar10 3C3D.

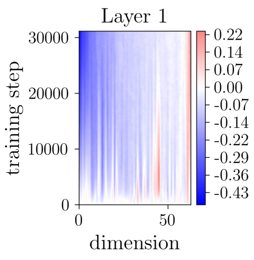


(B) One BatchNorm layer at the **penultimate layer** in the Cifar10 3C3D

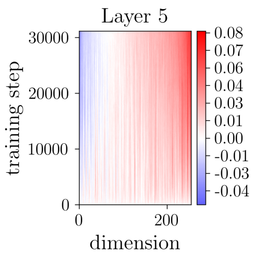
FIGURE A.1: Training loss and accuracy for the Cifar10 3C3D model with **one BatchNorm layer**, trained for 100 epochs with a batch size of 128, and a learning rate of 0.022 from Table 3.1.



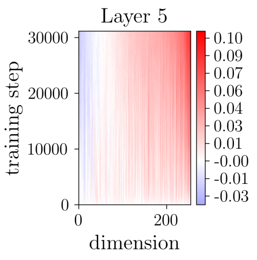
(A) β's of $BN_{\gamma,\beta}$



(B) β's of BN_β

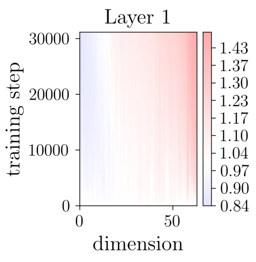


(C) β's of $BN_{\gamma,\beta}$

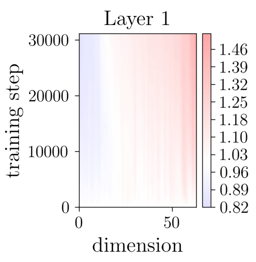


(D) β's of BN_β

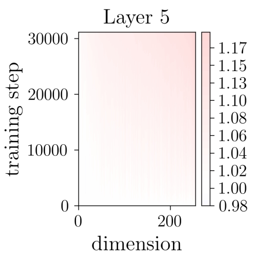
FIGURE A.2: β of the Cifar10 3C3D model with **one BatchNorm layer**. In (A) and (B), BatchNorm is placed after the first layer. In (C) and (D), BatchNorm is placed at the penultimate layer.



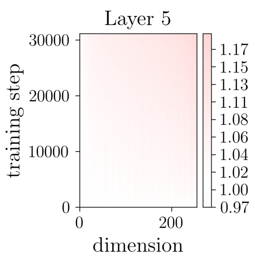
(A) γ's of $BN_{\gamma,\beta}$



(B) γ's of BN_{γ}

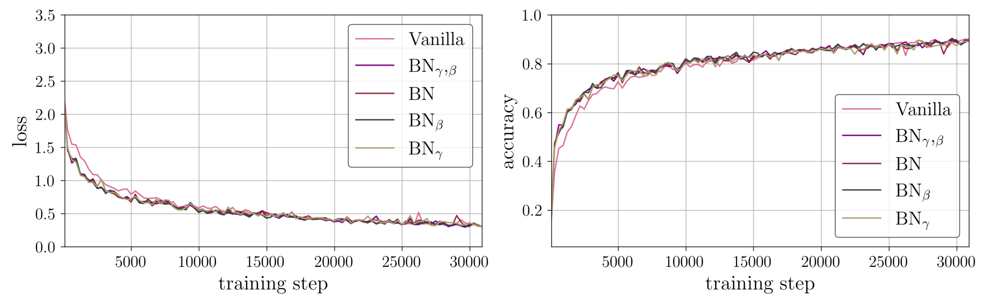


(C) γ's of $BN_{\gamma,\beta}$

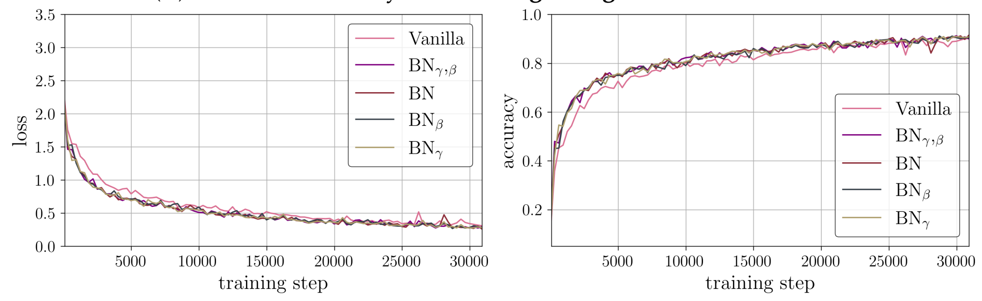


(D) γ's of BN_{γ}

FIGURE A.3: γ of the Cifar10 3C3D model with **one BatchNorm layer**. In (A) and (B), BatchNorm is placed after the first layer. In (C) and (D), BatchNorm is placed at the penultimate layer.

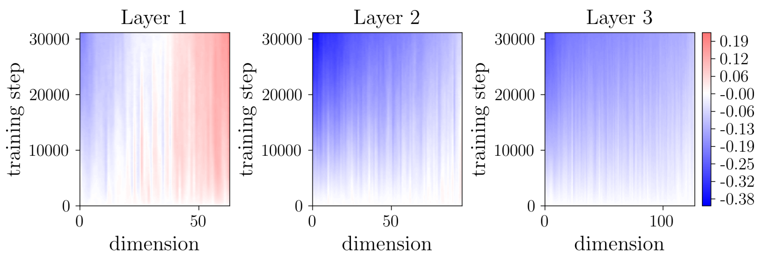


(A) Three BatchNorm layers **at the beginning of** the Cifar10 3C3D.

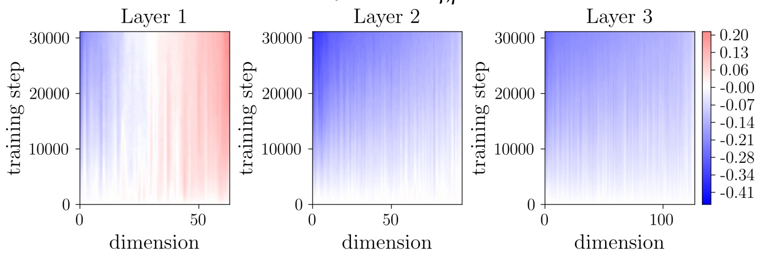


(B) Three BatchNorm layers **at the end of** the Cifar10 3C3D

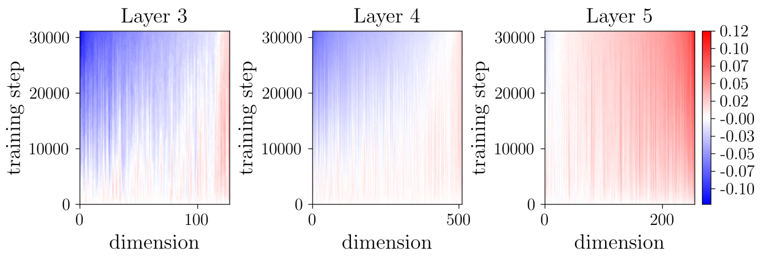
FIGURE A.4: Training loss and accuracy for the Cifar10 3C3D model with **three BatchNorm layers**, trained for 100 epochs with a batch size of 128, and a learning rate of 0.022.



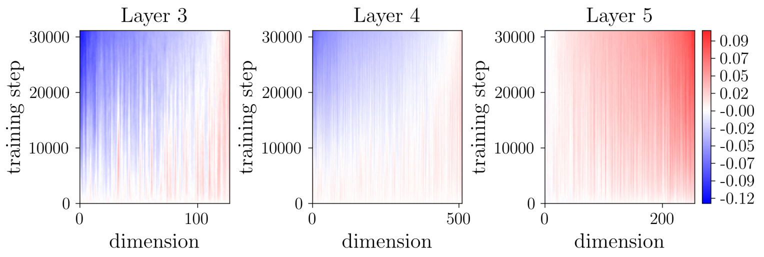
(A) β's of $BN_{\gamma,\beta}$



(B) β's of BN_{β}

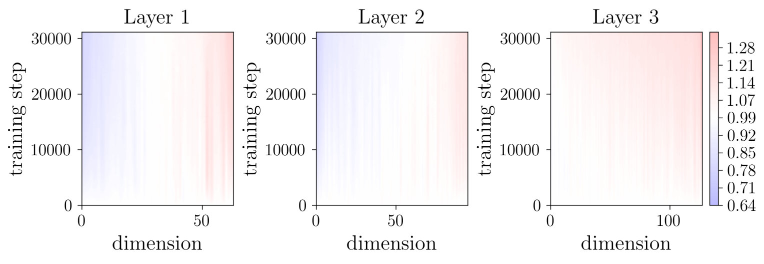


(C) β's of $BN_{\gamma,\beta}$

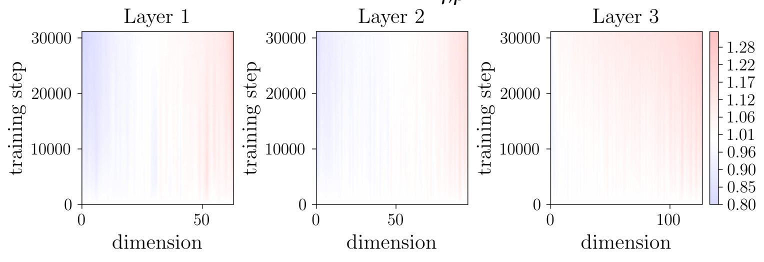


(D) β's of BN_{β}

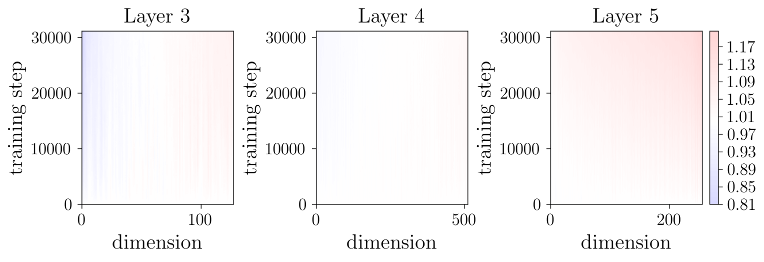
FIGURE A.5: β of the Cifar10 3C3D model with **three BatchNorm layers**. In (A) and (B), BatchNorm is placed after the first layer. In (C) and (D), BatchNorm is placed at the penultimate layer.



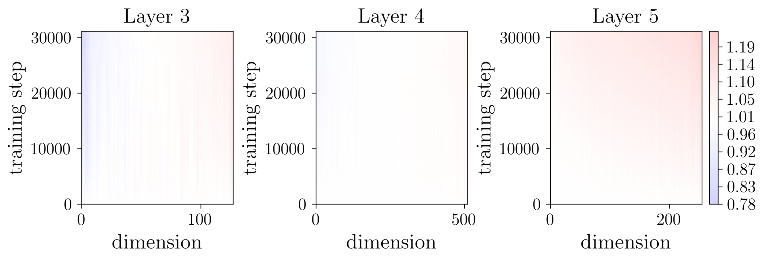
(A) γ's of $BN_{\gamma,\beta}$



(B) γ's of BN_{γ}



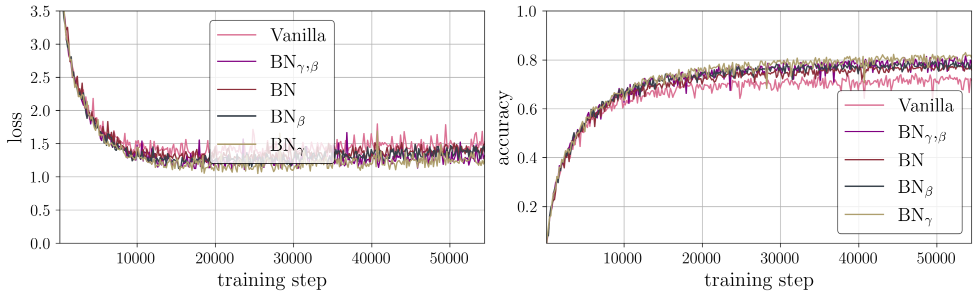
(C) γ's of $BN_{\gamma,\beta}$



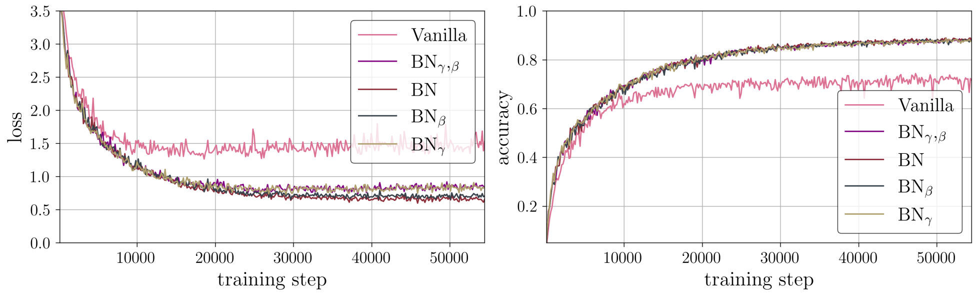
(D) γ's of BN_{γ}

FIGURE A.6: γ of the Cifar10 3C3D model with **three BatchNorm layers**. In (A) and (B), BatchNorm is placed after the first layer. In (C) and (D), BatchNorm is placed at the penultimate layer.

A.1.2 Cifar100 3C3D

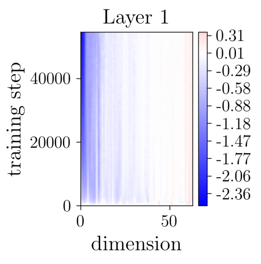


(A) One BatchNorm layer after the **first convolutional layer** in the Cifar100 3C3D.

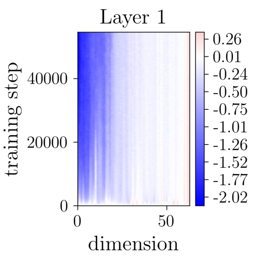


(B) One BatchNorm layer at the **penultimate layer** in the Cifar100 3C3D

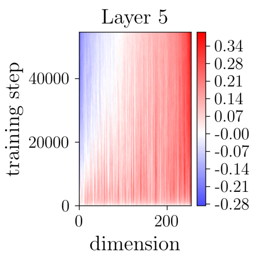
FIGURE A.7: Training loss and accuracy for the Cifar100 3C3D model with **one BatchNorm layer**, trained for 350 epochs with a batch size of 256, and a learning rate of 0.166.



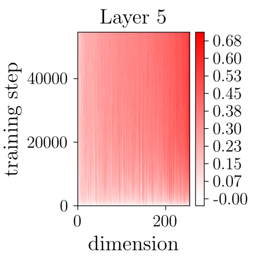
(A) β's of $BN_{\gamma,\beta}$



(B) β's of BN_β

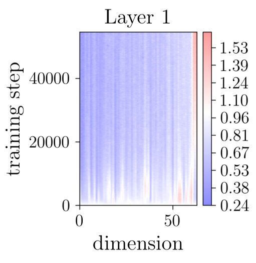


(C) β's of $BN_{\gamma,\beta}$

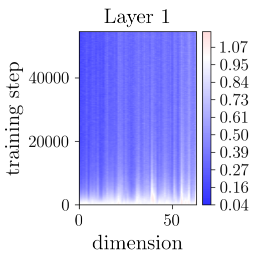


(D) β's of BN_β

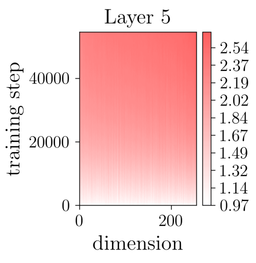
FIGURE A.8: β of the Cifar100 3C3D model with **one BatchNorm layer**. In (A) and (B), BatchNorm is placed after the first layer. In (C) and (D), BatchNorm is placed at the penultimate layer.



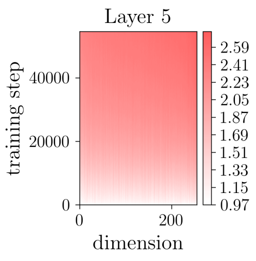
(A) γ's of $BN_{\gamma,\beta}$



(B) γ's of BN_{γ}

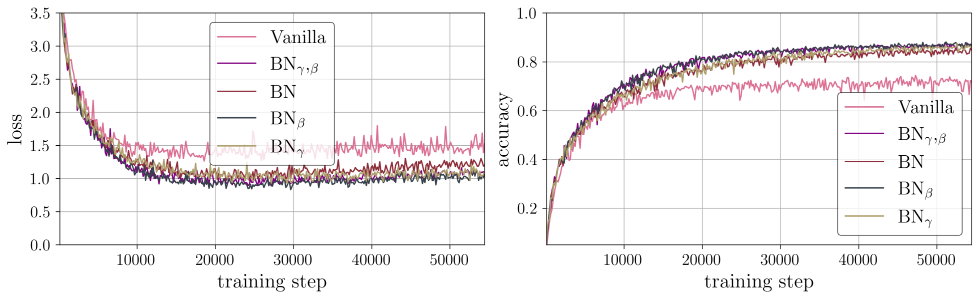


(C) γ's of $BN_{\gamma,\beta}$

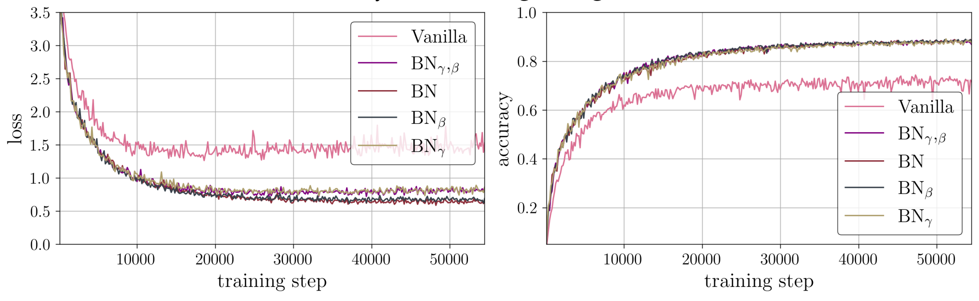


(D) γ's of BN_{γ}

FIGURE A.9: γ of the Cifar100 3C3D model with **one BatchNorm layer**. In (A) and (B), BatchNorm is placed after the first layer. In (C) and (D), BatchNorm is placed at the penultimate layer.

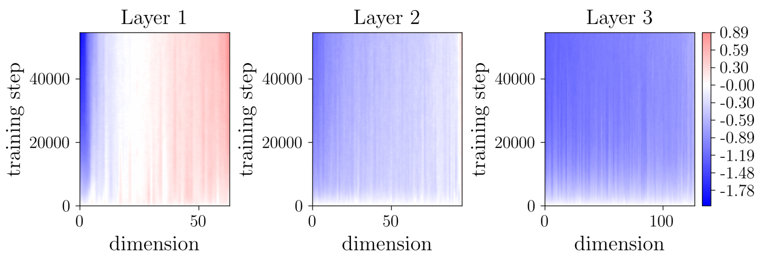


(A) Three BatchNorm layers **at the beginning of** the Cifar100 3C3D.

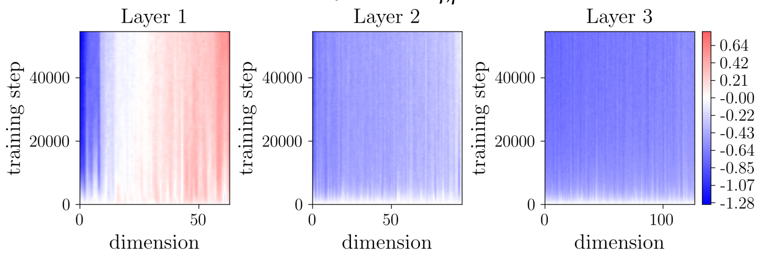


(B) Three BatchNorm layers **at the end of** the Cifar100 3C3D

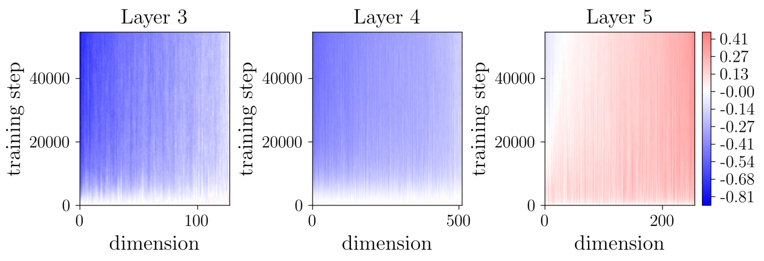
FIGURE A.10: Training loss and accuracy for the Cifar100 3C3D model with **three BatchNorm layers**, trained for 350 epochs with a batch size of 256, and a learning rate of 0.166.



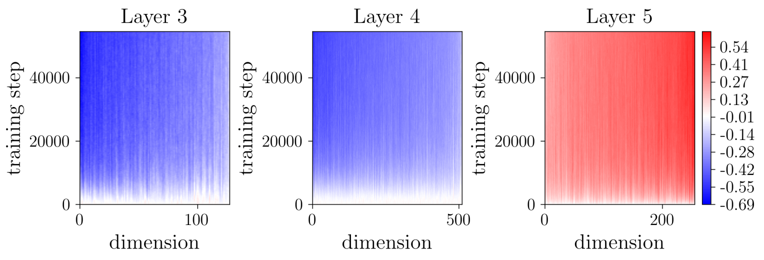
(A) β's of $BN_{\gamma,\beta}$



(B) β's of BN_{β}

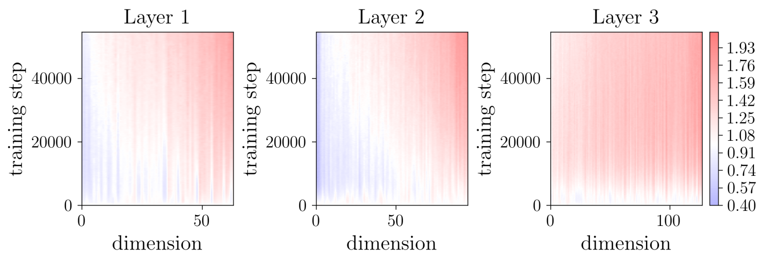


(C) β's of $BN_{\gamma,\beta}$

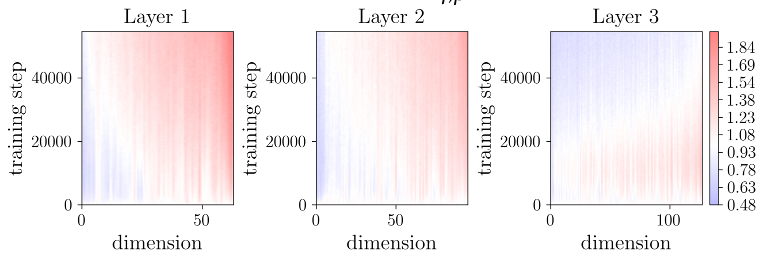


(D) β's of BN_{β}

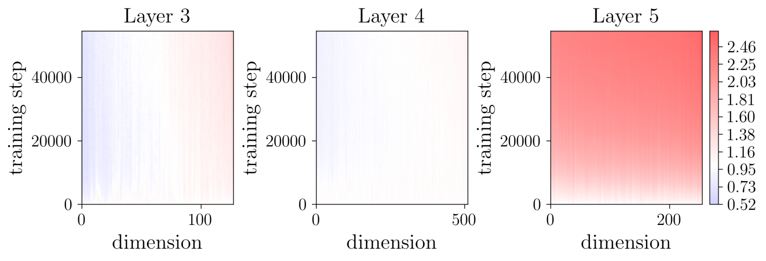
FIGURE A.11: β of the Cifar100 3C3D model with **three BatchNorm layers**. In (A) and (B), BatchNorm is placed after the first layer. In (C) and (D), BatchNorm is placed at the penultimate layer.



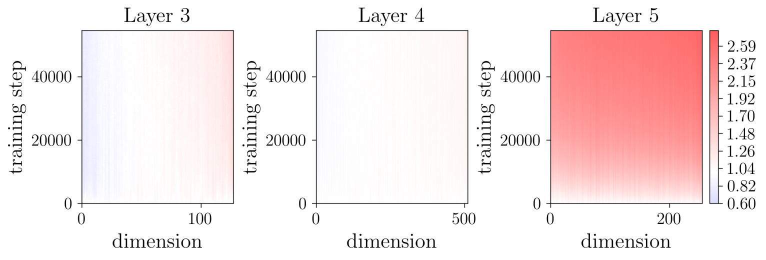
(A) γ's of $BN_{\gamma,\beta}$



(B) γ's of BN_{γ}



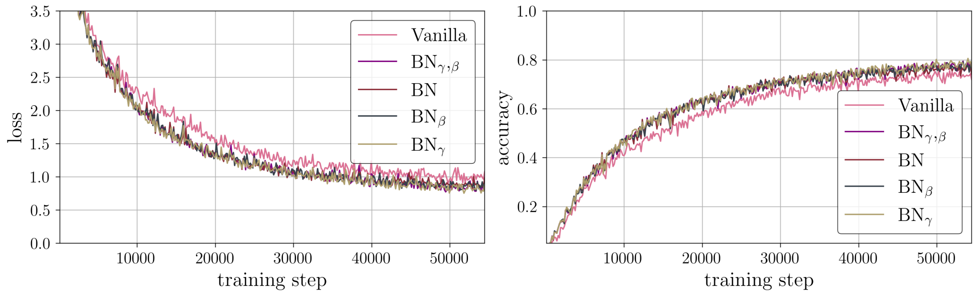
(C) γ's of $BN_{\gamma,\beta}$



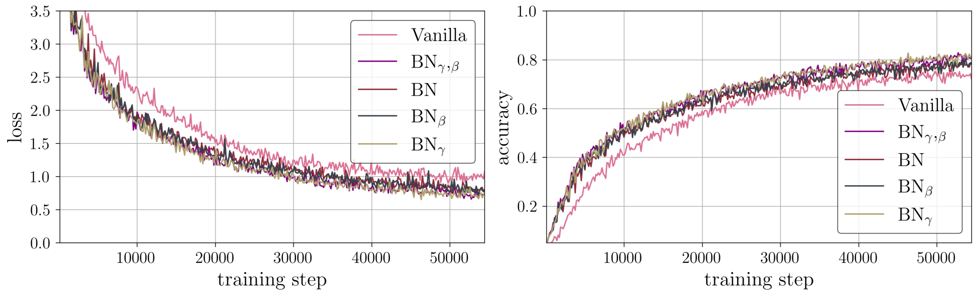
(D) γ's of BN_{γ}

FIGURE A.12: γ of the Cifar100 3C3D model with **three BatchNorm layers**. In (A) and (B), BatchNorm is placed after the first layer. In (C) and (D), BatchNorm is placed at the penultimate layer.

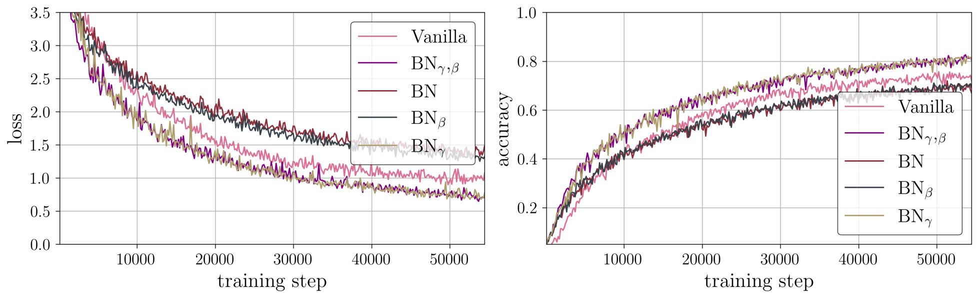
A.1.3 Cifar100 ALL-CNN-C



(A) One BatchNorm layer after the **first convolutional layer** in the ALL-CNN-C.

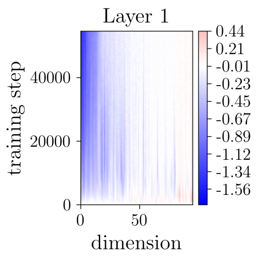


(B) One BatchNorm layer after the **penultimate convolutional layer** in the ALL-CNN-C.

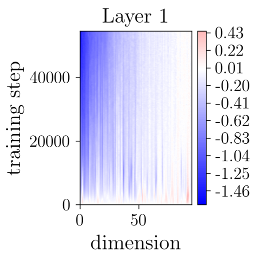


(C) One BatchNorm layer after the **last convolutional layer** in the ALL-CNN-C.

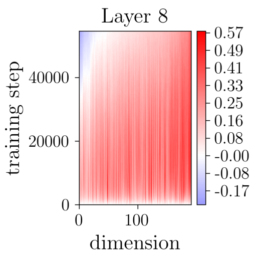
FIGURE A.13: Training loss and accuracy for the Cifar100 ALL-CNN-C model with **one BatchNorm layer** at different positions in the network. All models are trained for 350 epochs with SGD and a learning rate of 0.166.



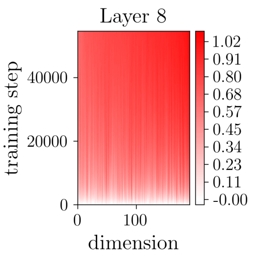
(A) β's of $BN_{\gamma,\beta}$



(B) β's of BN_{β}

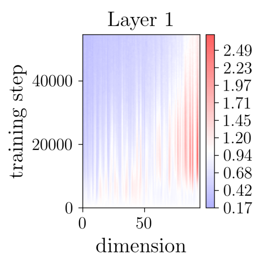


(C) β's of $BN_{\gamma,\beta}$

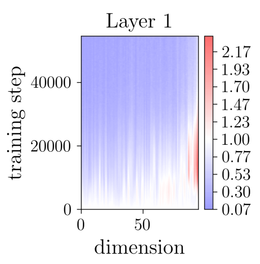


(D) β's of BN_{β}

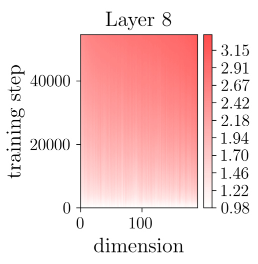
FIGURE A.14: β of the Cifar100 ALL-CNN-C model with **one BatchNorm layer**. In (A) and (B), BatchNorm is placed after the first layer. In (C) and (D), BatchNorm is placed at the penultimate layer.



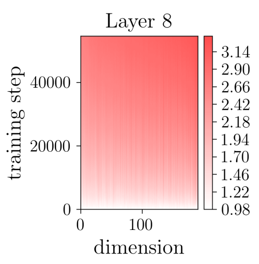
(A) γ's of $BN_{\gamma,\beta}$



(B) γ's of BN_{γ}

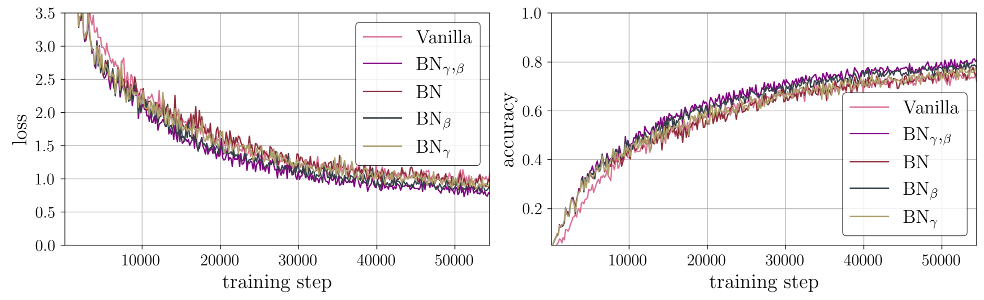


(C) γ's of $BN_{\gamma,\beta}$

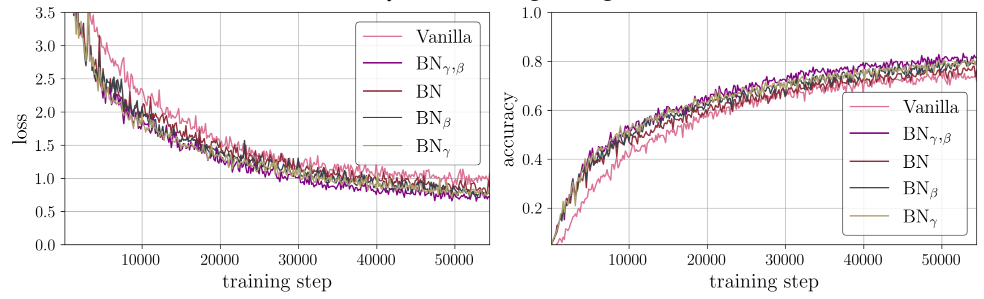


(D) γ's of BN_{γ}

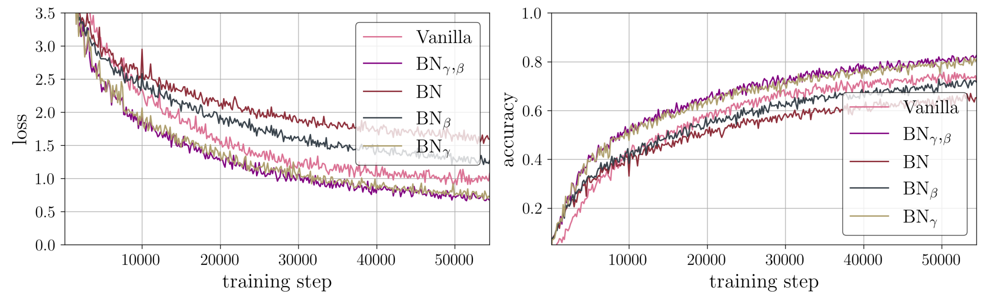
FIGURE A.15: γ of the Cifar100 ALL-CNN-C model with **one BatchNorm layer**. In (A) and (B), BatchNorm is placed after the first layer. In (C) and (D), BatchNorm is placed at the penultimate layer.



(A) Four BatchNorm layers at the **beginning of** the ALL-CNN-C.

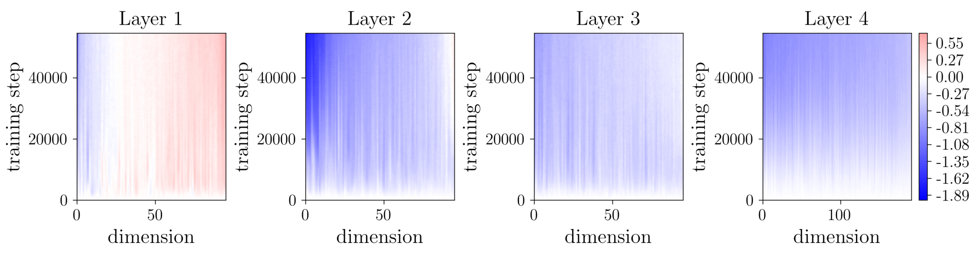


(B) Four BatchNorm layers **at the end of** the ALL-CNN-C (the last layer is left out).

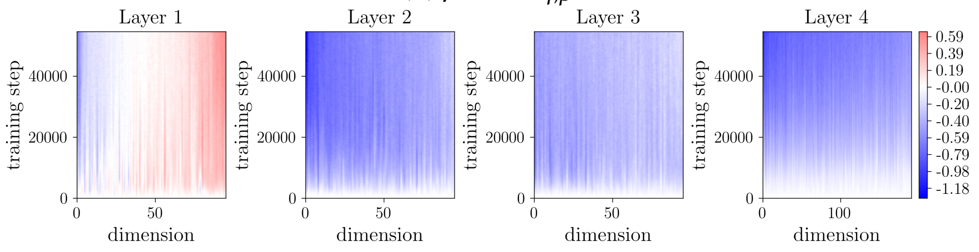


(C) Four BatchNorm layers **at the end of** the ALL-CNN-C (last layer included).

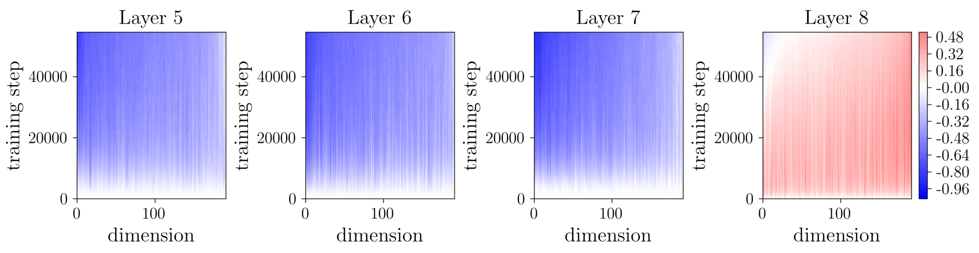
FIGURE A.16: Training loss and accuracy for the Cifar100 ALL-CNN-C model with **four BatchNorm layers** at different positions in the network. All models are trained for 350 epochs with SGD and a learning rate of 0.166.



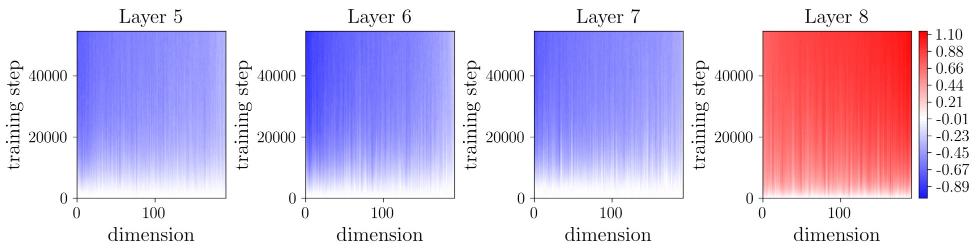
(A) β's of $BN_{\gamma,\beta}$



(B) β's of BN_{β}

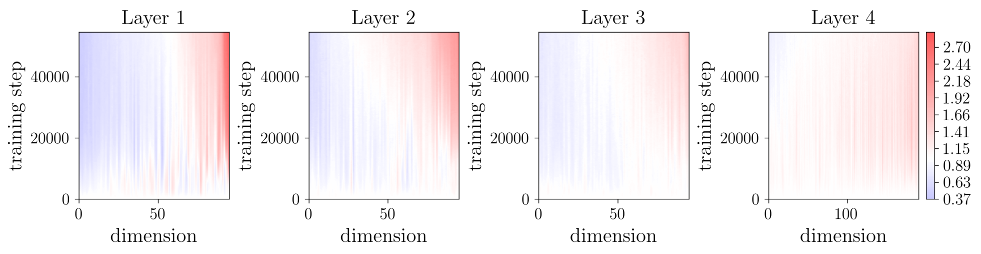


(C) β's of $BN_{\gamma,\beta}$

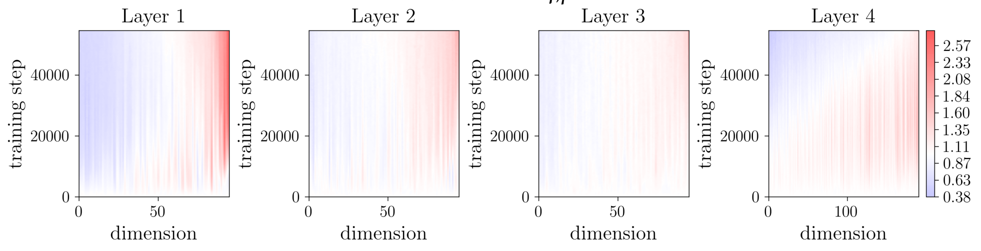


(D) β's of BN_{β}

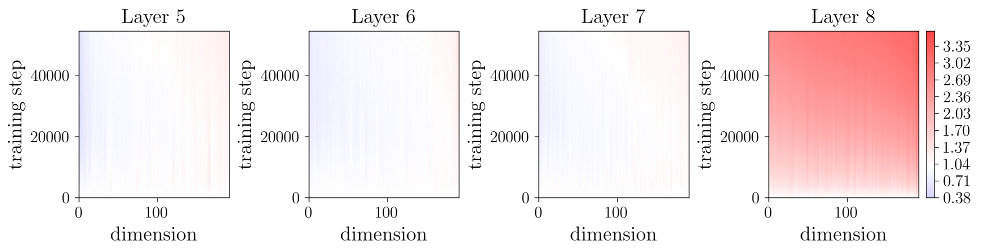
FIGURE A.17: β of the Cifar100 ALL-CNN-C model with **four BatchNorm layers**. In (A) and (B), BatchNorm is placed after the first layer. In (C) and (D), BatchNorm is placed at the penultimate layer.



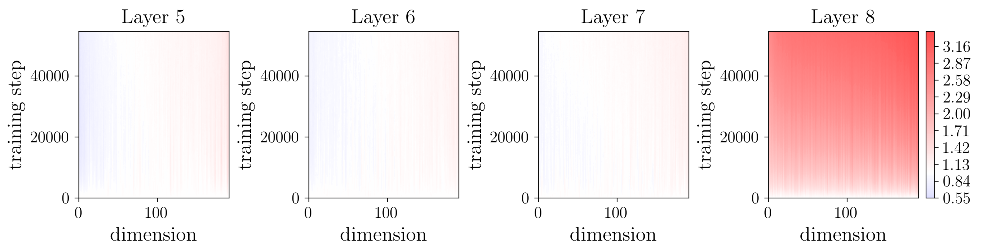
(A) γ's of $BN_{\gamma,\beta}$



(B) γ's of BN_{γ}



(C) γ's of $BN_{\gamma,\beta}$



(D) γ's of BN_{γ}

FIGURE A.18: γ of the Cifar100 ALL-CNN-C model with **four BatchNorm layers**. In (A) and (B), BatchNorm is placed after the first layer. In (C) and (D), BatchNorm is placed at the penultimate layer.

Appendix B

Network Architectures

B.1 Network Architectures

The **3c3d** network consists of:

- Three conv layers with ReLUs, each followed by max-pooling
- Two fully connected layers with 512 and 256 units and ReLU activation
- 100-unit output fully connected layer followed by a softmax activation function
- CE loss
- ℓ_2 regularization on the weights (but not the biases) with a default factor of 0.002
- The weight matrices are initialized using Xavier initialization, and the biases are initialized to 0.

The **all-cnn-c** network consists of:

- Nine convolutional layers, each followed by a ReLU function
- Average pooling layer after last convolutional layer with kernel size 6.
- CE loss
- ℓ_2 regularization is used on the weights (but not the biases) which defaults to $5e-4$.
- The paper does not comment on initialization. Here, we use Xavier initialization for convolutions and constant initialization of 0 for biases.

More details on the 3C3D and the ALL-CNN-C architectures can be found in DeepOBS (Schneider et al., 2019) [1] and in the original ALL-CNN-C paper (Springenberg et al., 2015).

[1]Github repository: `https://github.com/fsschneider/DeepOBS`