

Weight and Gradient Centralization in Deep Neural Networks

Wolfgang Fuhl
University of Tübingen
Sand 14, 72076 Tübingen, Germany
wolfgang.fuhl@uni-tuebingen.de

Enkelejda Kasneci
University of Tübingen
Sand 14, 72076 Tübingen, Germany
enkelejda.kasneci@uni-tuebingen.de

Abstract

Batch normalization is currently the most widely used variant of internal normalization for deep neural networks. Further work has shown that the normalization of weights and additional conditioning as well as the normalization of gradients further improve the generalization. In this work we combine several of these methods and thereby further increase the generalization of the networks. The advantage of the newer methods compared to the batch normalization is not only the increased generalization but also that they only have to be applied during the training and therefore do not influence the running time during use.

1. Introduction

Deep neural networks (DNN) [21] are currently the most successful machine learning method and owe their recent progress to the steadily growing data sets [19], improvements in massively parallel architectures [17], high-speed bus systems such as PCIe, optimization methods [28, 15], new training techniques [7, 16], and the steadily growing fields of application. These advances in technology make it possible to train deep neural networks on huge datasets like ImageNet [19], however, further techniques had to be introduced to prevent the gradients from becoming too small [10] and the normalization of the data [13] has a huge impact on the generalization of large networks. Generalization alone is not the only quality feature of a good learning process of neural networks. Another important point is the acceleration of the learning process and the resource-saving use of the techniques. This is due to the fact that the most successful architectures already have a high resource requirement per se and additional techniques to improve generalization can therefore only use a small amount of additional resources. This can be seen very clearly when comparing the optimization techniques themselves. The most popular methods are Stochastic Gradient

Decent (SGD) with momentum [28] and Adam [15] which introduces a second momentum. There are many other optimization algorithms [28, 15, 2, 4] but SGD and Adam are the most popular. Both methods allow batch based learning and require only a constant multiple of the gradient (for the momentum) as additional memory. Comparing this with the Levenberg-Marquardt algorithm (LM) [22, 24], which was the most popular method for training neural networks for quite some time, it is noticeable that the memory consumption in the case of LM grows quadratic to the weights. This is due to the fact that the LM algorithm calculates the exact derivatives for each weight over the whole network and not only local derivatives as it is the case with backpropagation. Further procedures like the residual layers [10], weight initialization strategy [6, 9], activation functions [25], gradient clipping [26, 27], algorithms for adaptive learning rate optimization [28, 15], and many more have been introduced and are subject to the same conditions of generalization improvement, training stabilization, and resource conservation.

In neural networks themselves, statistics are also collected and used to balance the forward and backward flow of data and errors. The best known method used directly on the activation of neurons is the Batch Normalization (BN) [13]. Other procedures that work on the activation functions are the instance normalization (IN) [34, 12], layer normalization (LN) [1] and group normalization (GN) [35]. These procedures smooth the optimization landscape [32] and lead to an improvement of the generalization. The disadvantages of the BN are that it continues to process the data in the neural network as an independent layer even after training and that it must be applied to a relatively large batch size. To avoid these disadvantages, weight normalization (WN) [31, 11] and weight standardization (WS) [29] were introduced. These must only be applied during training and are independent of the batch size. WN limits the weight vectors via different standards whereas WS normalizes the weight vectors via the mean and standard deviation. A newer technique that works only on the gradient is

the Gradient Centralization (GC) [36] which subtracts the mean value from the gradient. All these advanced techniques smooth the error space and lead to a faster and usually better generalization of neural networks.

In this work we deal with these extended techniques and try to find good combinations of the methods. In our evaluation it has been shown that the combination of the filter mean subtraction and the gradient mean subtraction in combination is very effective for different networks. We have also tested other combinations and found that many of them also depend on batch normalization. Our main contributions are as follows:

- 1 The combination of mean gradient and mean filter subtraction
- 2 Publicly available CUDA implementations
- 3 Description of the integration into the back propagation algorithm
- 4 A comprehensive comparison with advanced techniques

2. Related Work

In this section we describe the related work based on three groups. The first group is the manipulation of the data after the activation functions, which has the disadvantage that they have to be executed in the later application of the model. The second group is the manipulation of the weights during the training. Here the weights can be standardized or otherwise restricted. The last group is the manipulation of the gradients. Here, after each back propagation, normalizations and restrictions are applied to the gradients before they are used for the change of the weights.

2.1. Manipulation of the output of the activations

This type of normalization is the most common use of internal manipulation in DNNs today. In batch normalization (BN) [13], the mean value and standard deviation are calculated over several batches and used for normalization. This gives the output of neurons after the activation layer a mean value of zero and uniform variance. With the group normalization GN [35] groups are formed over which the normalization is performed instead of normalizing over the number of copies in a batch as with the BN. This eliminates the need for large batches, which is the case with the BN. Other alternatives are the instance normalization IN [34, 12] and the layer normalization LN [1]. For IN, each specimen is used individually for the calculation of the mean and standard deviation, and for LN, the individual layers. The IN and LN have been successfully used for recurrent neural networks (RNN) [33]. However, all these methods have the disadvantage that the normalization has to be applied even after training.

2.2. Manipulation of the weights of the model

In weight normalization (WN) [31, 11], the weights of the neural network are multiplied by a constant divided by the Euclidean distance of the weight vector of a neuron. This decouples the weights with respect to their length, thereby accelerating the training. An extension of this method is the weight standardization (WS) [29] which does not require a constant but calculates the mean and standard deviation and thereby normalizes the weights. Like the previous manipulation methods, this one smooths the error landscape, which speeds up the training and smooths the generalization of the final model. An advantage of these methods is that they only have to be applied during training but not in the final model. However, these methods have a limitation and that is the fine-tuning of neural networks. If the original model was not trained with a weight normalization, these methods cannot be used for fine-tuning without creating a high initial error of the model. This is due to the fact that the restrictions and norms for weights of the original model most likely do not apply.

2.3. Manipulation of the gradient after back propagation

Another very common technique is gradient manipulation over the first [28] and second moment [15]. This gradient impulse allows neural networks to be trained in a stable way without the gradients exploding, which is interpreted as a damped oscillation. The second momentum leads in most cases to a faster generalization and the final performance of the model is usually slightly worse compared to training with only the first momentum. These moments are moving averages which are formed over the calculated gradients and represent a pre-determined portion of the next weight adjustment. An advanced method in this area is gradient clipping [26, 27] where randomly selected gradients are set to zero or a small random value is added to each gradient. Another technique is to project the gradients onto subspaces [8, 20, 3]. Here, for example, the Riemannian approach is used to map the gradients onto a Riemannian manifold. After the mapping, the gradients are used to adjust the weights. In addition, a very simple procedure was presented, which subtracts the current mean value of the gradients in addition to the moments [36].

3. Method

Since our approach is a combination of several previously published approaches (Weight mean subtraction and gradient centralization), we proceed as follows in this section. We describe the already published methods formally and introduce a naming convention, which we use later in the evaluation. This should make it easier for the reader to evaluate the effectiveness of different methods. In the

108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215

following we will refer to operations on the data in forward propagation as $F_{s,c,y,x}$ where s is the sample, c is the channel and y, x is the spatial position in the data. For the weights we use $W_{out,in,y,x}$. Where out is the output channel, in is the input channel and y, x is the spatial position for fully connected layers. In the case of convolution layers y, x is the position in the two-dimensional convolution mask, which together with in defines the convolution tensor. To manipulate the gradient we use $\Delta W_{out,in,y,x}$ with the same indices as we used for the weights ($W_{out,in,y,x}$). Since a normalization can be applied not only to the data and the gradient but also to the back propagated error, the error is denoted by $E_{s,c,y,x}$ where the indices are the same as the indices of the forward propagated data ($F_{s,c,y,x}$).

3.1. Weight normalization

In this section the equations used for weight normalization are presented. In all equations, j represents the axis to which the normalization was performed orthogonally. This means that we have calculated a separate mean value, standard deviation or Euclidean distance for each index of j .

$$W_{j,in,y,x} = W_{j,in,y,x} * \frac{k}{\|W_{j,in,y,x}\|} \quad (1)$$

In Equation 1 the weight normalization [31, 11] is described (WN). This normalizes each weight in a tensor with the ratio of a constant k (In our experiments 1) divided by the Euclidean distance of the tensor.

$$W_{j,in,y,x} = W_{j,in,y,x} - \overline{W}_{j,in,y,x} \quad (2)$$

Since the pure normalization over the mean value of the tensor of the weights has no separate designation, we use WC in our work. WC is defined in Equation 2 and calculates a separate mean value for each weight tensor and subtracts it from each weight.

$$W_{j,in,y,x} = \frac{W_{j,in,y,x} - \overline{W}_{j,in,y,x}}{std(W_{j,in,y,x})} \quad (3)$$

The final normalization of the weights is the weight standardization [29] which is defined in Equation 3. Here, as in WC, the mean value is subtracted and each weight of a tensor is also divided by the standard deviation.

3.2. Gradient normalization

In this section the gradient normalization is introduced. Since the modern optimizers use the moving averaging already with the momentum [28, 15]. We also think, that the authors from gradient centralization [36] already tried different approaches like the standardization we only present the recently published approach here. Also as in the weight normalization section, j corresponds to j against the axis along which orthogonal normalization is performed.

$$\Delta W_{j,in,y,x} = \Delta W_{j,in,y,x} - \overline{\Delta W}_{j,in,y,x} \quad (4)$$

As can be seen in Equation 4, the mean value is subtracted from each gradient tensor. The mean value is recalculated for each output layer.

3.3. Data normalization

In this section, we briefly describe the different data normalizations. In our analysis, we only used the batch normalization [13]. In this section j as well as j_1 and j_2 (in case of the instance normalization) stand for the axis or plane to which the normalization is orthogonal. Since scale and shift is learned in data manipulation we denote them with γ and β respectively.

$$F_{s,j,y,x} = \gamma * \left(\frac{F_{s,j,y,x} - \overline{F}_{s,j,y,x}}{std(F_{s,j,y,x})} \right) + \beta \quad (5)$$

Equation 5 describes the batch normalization[13]. As mentioned above, γ and β are the scale and shift parameters which are learned during training. Since j is on the second index, each channel has its own average and standard deviation.

$$F_{j,c,y,x} = \gamma * \left(\frac{F_{j,c,y,x} - \overline{F}_{j,c,y,x}}{std(F_{j,c,y,x})} \right) + \beta \quad (6)$$

Equation 6 is the layer normalization [1]. Compared to batch normalization [13], the layer normalization is the normalization along the samples in the batch. This means that each sample has its own average and standard deviation.

$$F_{j_1,j_2,y,x} = \gamma * \left(\frac{F_{j_1,j_2,y,x} - \overline{F}_{j_1,j_2,y,x}}{std(F_{j_1,j_2,y,x})} \right) + \beta \quad (7)$$

In the case of the instance normalization [34, 12], each sample is normalized on its own. Equation 7 describes this procedure. It does not normalize along an axis like the other methods, but each sample and each channel separately.

The only approach still missing is the group normalization [35]. Here groups are formed between the individual instances, which have their own mean values and standard deviations. Since we cannot simply describe this with our annotation, the equation for the group normalization [35] is not included in this paper.

3.4. Error normalization

Inspired by the data normalization, we have also done some small evaluations regarding the error normalization as a separate normalization approach. For this purpose we evaluated the standardization as well as the simple mean value subtraction. The simple mean subtraction is based on the fact that in the case of weight normalization, the simple mean has proven to be very effective. In our simple

216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323

Table 1. The used naming convention for our evaluation.

Name	WN	WC	WS	GC	BN	LN	IN	EBN	ELN	EB	EL
Eq.	1	2	3	4	5	6	7	8	9	10	11

implementations, we did not use the scale and shift (γ , β) parameters and applied the normalization directly.

$$E_{s,j,y,x} = \frac{E_{s,j,y,x} - \bar{E}_{s,j,y,x}}{\text{std}(E_{s,j,y,x})} \quad (8)$$

$$E_{j,c,y,x} = \frac{E_{j,c,y,x} - \bar{E}_{j,c,y,x}}{\text{std}(E_{j,c,y,x})} \quad (9)$$

The Equations 8 and 9 describe the error normalization along the channels and samples. The procedure is the same as for the batch normalization [13] and the layer normalization [1]. As you can see in the equations, we have omitted the learned γ and β parameters and the rest of the equations is the same. Thus the standard deviation and the mean value are calculated in each iteration and normalization is performed by subtracting the mean value and dividing by the standard deviation.

$$E_{s,j,y,x} = E_{s,j,y,x} - \bar{E}_{s,j,y,x} \quad (10)$$

$$E_{j,c,y,x} = E_{j,c,y,x} - \bar{E}_{j,c,y,x} \quad (11)$$

For the two other Equations 10, 11 we calculated only the average value over the samples or the channels and subtracted it. This was recalculated just like that in each iteration. An overview of the abbreviations used in the rest of the document is shown in Table 1.

One way to include the normalizations is to add them to the back propagation workflow. This is shown in algorithm 1 where each normalization is placed in either the forward, backward, or gradient computation flow. Since the batch normalization [13] is a separate layer and learns the scaling and shifting, it was not inserted.

As you can see in Algorithm 1, the filter normalization is applied before the use in the forward path and the gradient normalization is applied immediately after the gradient calculation. This is because the filters must be adjusted first, otherwise the gradient will not match the weights and the weights would have no influence on the forward pass. For gradient normalization, it is applied after the calculation so that the gradients are correctly available for the weight update in the optimizer. In case of the back propagated error, the error is normalized after the calculation of the back propagation whereby it would of course also be possible to normalize the input error. But since this is normalized in the previous layer, it is already normalized.

In this paper we present the combination of GC [36] and WC, whereas WC without the standard deviation (Division with the standard deviation) has, to our knowledge, never

Data: Data,Weights

Result: Output

Function Forward is

Weights=Normalize(Weights);

Output=cuDNNFWD(Weights,Data);

end

Data: ErrorIn,Weights

Result: ErrorOut

Function Backward is

ErrorOut=cuDNNBWD(Weights,ErrorIn);

ErrorOut=Normalize(ErrorOut);

end

Data: ErrorIn,Data

Result: Grad

Function CompGrads is

Grad=cuDNNCompGrad(Data,ErrorIn);

Grad=Normalize(Grad);

end

Algorithm 1: Algorithmic description of the function placement for the normalizations. Since batch normalization is implemented as a own layer with learned scaling and shifting, it is not considered in this illustration. It could be placed functionally in the forward propagation and would normalize the output.

been published independently. GC [36] and WC can also be integrated into the optimizer itself. In the following we give two examples. One for SGD [2] with momentum [28] and the other for ADAM [15].

Data: $W_{j,in,y,x}^t, \Delta W_{j,in,y,x}^t, \alpha, \psi, M_{j,in,y,x}^t$

Result: $W_{j,in,y,x}^{t+1}$

Function SGD is

$\Delta W_{j,in,y,x}^t = \Delta W_{j,in,y,x}^t - \overline{\Delta W}_{j,in,y,x}^t$;

$M_{j,in,y,x}^t = \psi * M_{j,in,y,x}^t + (1-\psi) * \Delta W_{j,in,y,x}^t$;

$W_{j,in,y,x}^{t+1} = W_{j,in,y,x}^t - \alpha * M_{j,in,y,x}^t$;

$W_{j,in,y,x}^{t+1} = W_{j,in,y,x}^{t+1} - \overline{W}_{j,in,y,x}^{t+1}$;

end

Algorithm 2: Integration of the GC and WC normalization into the stochastic gradient decent optimization with momentum. The variables are Weights $W_{j,in,y,x}^t$, gradients $\Delta W_{j,in,y,x}^t$, learning rate α , momentum factor ψ , and momentum $M_{j,in,y,x}^t$. j again is the index of the normalization.

In Algorithm 2 the integration of the GC and the WC normalization in combination into SGD is shown. Here the first line ($\Delta W_{j,in,y,x}^t = \Delta W_{j,in,y,x}^t - \overline{\Delta W}_{j,in,y,x}^t$) is the gradient centralization. Afterwards the momentum is combined with the gradients by the factor ψ . In the next step the weights are adjusted using the learning rate α and the

324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377

378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431

mean value is subtracted from the final weights. This last step is the weight centralization ($W_{j,in,y,x}^{t+1} = W_{j,in,y,x}^{t+1} - \overline{W}_{j,in,y,x}^{t+1}$).

Data: $W_{j,in,y,x}^t, \Delta W_{j,in,y,x}^t, \alpha, \psi_1, \psi_2, M_{j,in,y,x}^t, V_{j,in,y,x}^t$

Result: $W_{j,in,y,x}^{t+1}$

Function ADAM is

$$\begin{aligned} \Delta W_{j,in,y,x}^t &= \Delta W_{j,in,y,x}^t - \overline{\Delta W}_{j,in,y,x}^t; \\ M_{j,in,y,x}^t &= \psi_1 * M_{j,in,y,x}^t; \\ M_{j,in,y,x}^t &+ = (1 - \psi_1) * \Delta W_{j,in,y,x}^t; \\ V_{j,in,y,x}^t &= \psi_2 * V_{j,in,y,x}^t; \\ V_{j,in,y,x}^t &+ = (1 - \psi_2) * \Delta W_{j,in,y,x}^t \odot \Delta W_{j,in,y,x}^t; \\ \hat{M}_{j,in,y,x}^t &= \frac{M_{j,in,y,x}^t}{1 - \psi_1^t}; \\ \hat{V}_{j,in,y,x}^t &= \frac{\psi_2 * V_{j,in,y,x}^t}{1 - \psi_2^t}; \\ W_{j,in,y,x}^{t+1} &= W_{j,in,y,x}^t - \alpha * \frac{\hat{M}_{j,in,y,x}^t}{\sqrt{\hat{V}_{j,in,y,x}^t + \epsilon}}; \\ W_{j,in,y,x}^{t+1} &= W_{j,in,y,x}^{t+1} - \overline{W}_{j,in,y,x}^{t+1}; \end{aligned}$$

end

Algorithm 3: Integration of the GC and WC normalization into the ADAM optimization. The variables are Weights $W_{j,in,y,x}^t$, gradients $\Delta W_{j,in,y,x}^t$, learning rate α , first order momentum factor ψ_1 , second order momentum factor ψ_2 , first order momentum $M_{j,in,y,x}^t$, and second order momentum $V_{j,in,y,x}^t$. j again is the index of the normalization.

In Algorithm 3 shows the integration of GC and WC in the ADAM optimization. For this, as with SGD, GC is applied first ($\Delta W_{j,in,y,x}^t = \Delta W_{j,in,y,x}^t - \overline{\Delta W}_{j,in,y,x}^t$). Then the new first order momentum is calculated in the following two lines using the factor ψ_1 ($M_{j,in,y,x}^t = \psi_1 * M_{j,in,y,x}^t + (1 - \psi_1) * \Delta W_{j,in,y,x}^t$). Subsequently the second order momentum is calculated with the factor ψ_2 ($V_{j,in,y,x}^t = \psi_2 * V_{j,in,y,x}^t + (1 - \psi_2) * \Delta W_{j,in,y,x}^t \odot \Delta W_{j,in,y,x}^t$). In the penultimate step the weights are adjusted with the learning rate α as well as the two moments ($W_{j,in,y,x}^{t+1} = W_{j,in,y,x}^t - \alpha * \frac{\hat{M}_{j,in,y,x}^t}{\sqrt{\hat{V}_{j,in,y,x}^t + \epsilon}}$). The last step is then again the weight centralization WC ($W_{j,in,y,x}^{t+1} = W_{j,in,y,x}^{t+1} - \overline{W}_{j,in,y,x}^{t+1}$).

4. Neural Network Models

Figure 1 shows the architectures used in our experimental evaluation. The first model (Figure 1 a)) is a small model with batch normalization. We used this model to show the impact of the different normalization approaches on small models with and without batch normalization. The second model (Figure 1 b)) is a ResNet-34 and a commonly used larger deep neural network. We used it with and without batch normalization during our experiments to show the im-

port of the normalization approaches on residual networks. The third model (Figure 1 c)) is a classical architecture for neural networks without batch normalization. This model was used to show the impact of the normalization to classical neural network architectures. The last model (Figure 1 d)) is a fully convolutional neural network [23]. It uses the U-connections [30] to improve the result for semantic segmentation. We used this network together with the VOC2012 [5] data set in the semantic segmentation task to show the impact of the normalizations. For training and evaluation we used the DLIB [14] library for deep neural networks. In this library we have also integrated our normalization and the state of the art approaches against which we compare ourselves.

5. Data sets

In this section we present all used data sets, describe the used training parameters as well as the optimization techniques and the data augmentation. For a simplified reproducibility we have limited ourselves to a minimum of data manipulation and used only public data sets. The batch size, input resolution as well as the random weight initialization is given too.

CIFAR10 [18] has of 60,000 colour images with a resolution of 32×32 . The public data set has ten different classes. For training, 50,000 images are provided with 5,000 examples per class. The validation set consists of 10,000 images with 1,000 examples for each class. The task in this data set is to classify a given image to one of the ten categories.

Training: We used a batch size of 50 and an initial learning rate of 10^{-3} . As optimizer, we used ADAM [15] with weight decay of $5 * 10^{-5}$, momentum one with 0.9 and momentum two with 0.999. As random weight initialization we used formula 16 from [6] and all bias terms are set to 0. For data augmentation, we cropped a 32×32 region from a 40×40 image with a zero padding of the original image at the borders. In addition, we used a constant mean subtraction (mean-red 122.782, mean-green 117.001, mean-blue 104.298) and division by 256.0 for the input image. The training itself was conducted for 300 epochs, whereby the learning rate was decreased by 10^{-1} after each 50 epochs.

CIFAR100 [18] is a more difficult but similar public data set like CIFAR10 and consists of color images with a resolution of 32×32 . The task here is as in CIFAR10 to classify the given image to one of the one hundred given classes. The training set consists of 500 examples per class and the validation set has 100 examples per class. This means that CIFAR100 has the same amount of images as CIFAR10 for training and validation but one hundred instead of ten classes.

Training: We used a batch size of 50 and an initial learning rate of 10^{-1} . As optimizer we used SGD with mo-

486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539

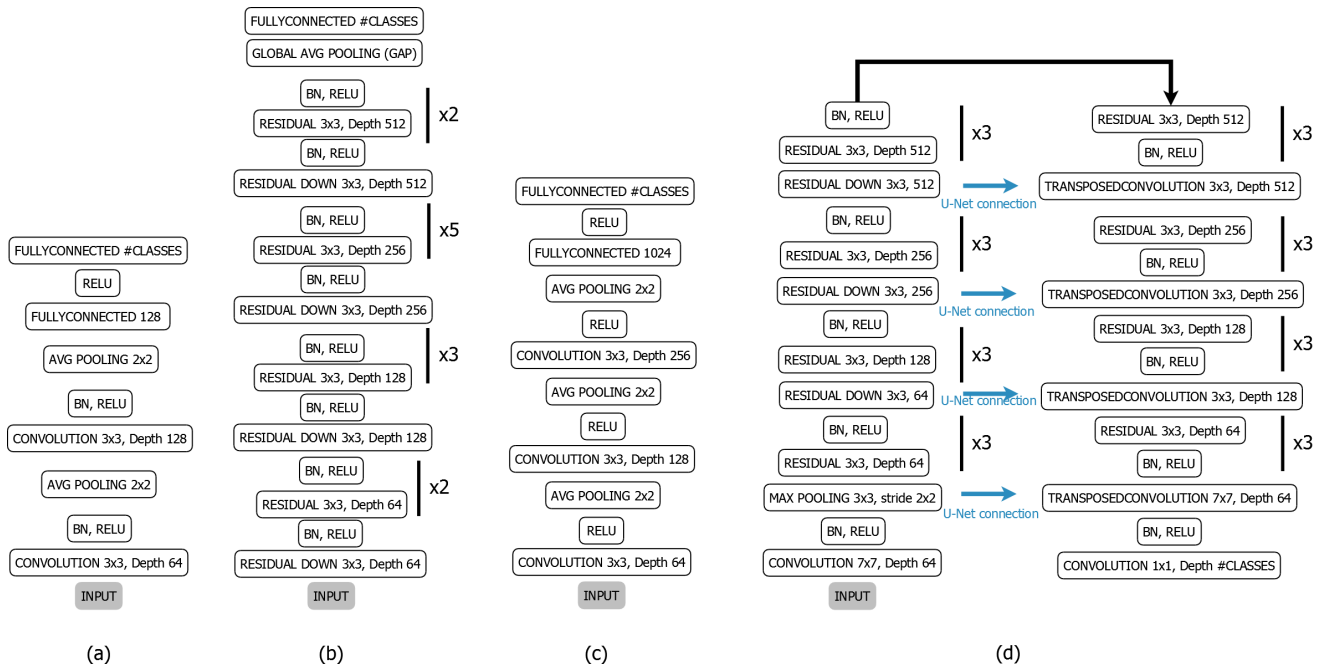


Figure 1. All used architectures in our experimental evaluation. (a) is a small neural network model with batch normalization. (b) is a ResNet-34 architecture. (c) is a small model without batch normalization. (d) is a residual network using the interconnections from U-Net [30] for semantic image segmentation.

mentum [28] 0.9 and a weight decay of $(5 * 10^{-4})$. For data augmentation, we cropped a 32×32 region from a 40×40 image with a zero padding of the original image at the borders. In addition, we used a constant mean subtraction (mean-red 122.782, mean-green 117.001, mean-blue 104.298) and division by 256.0 for the input image. The training itself was conducted for 300 epochs, whereby the learning rate was decreased by 10^{-1} after each 50 epochs. For weight initialization we used formula 16 from [6] and all bias terms are set to 0.

VOC2012 [5] is a public available data set which can be used for detection, classification and semantic segmentation. In our experiments we only used the semantic segmentation annotations as well as the semantic segmentation task. For the semantic segmentation task a class is assigned for each output pixel. This data set has twenty different classes and each image can contain different object classes and different amounts of the same object. This also means that not every object is present in every image. The training set consists of 1,464 images with 3,507 segmented objects and the validation set has 1,449 images with a total of 3,422 segmented objects on it. The amount of objects in this data set is not balanced which increases the challenge additionally. There is also a third data set which does not contain any annotations and can be used initially in an unsupervised fashion to have a good weight initialization. We did not use the third data set in our training nor in our evaluation.

Training: The initial learning rate was set to 10^{-1} with

a constant batch size of ten. As optimizer we used SGD with momentum [28] set to 0.9 and additionally weight decay of $1 * 10^{-4}$. For weight initialization we used formula 16 from [6] and all bias terms are set to 0. The data was augmented by cropping 227×227 regions out of the input image. In addition, we used random color offset and left right flipping of the image. Before the image was processed we subtracted a constant mean (mean-red 122.782, mean-green 117.001, mean-blue 104.298) and divided each value by 256.0. We trained each model for 800 epochs and reduced the learning rate by 10^{-1} after each 200 epochs.

6. Evaluation

In this section we show the results on CIFAR10, CIFAR100 and VOC2012. We use the models from Figure 1 and apply the training parameters and procedures from Section 5. In the first experiment we show over which areas in the data the mean value subtraction can be used most effectively. In the following three experiments we compare the combination of GC and WC with the state of the art.

Table 2 shows the evaluation of mean subtraction on different areas of weights, gradients and back propagated errors. As can be seen, the mean subtraction on the error propagated back is not very effective, because it significantly worsens the generalization of the deep neural network. This shows that the error normalization without the data normalization as it happens in batch normalization [13] only brings disadvantages. For the weights and the gradients, a convo-

540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593

594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647

Table 2. The results for the mean subtraction normalization on CIFAR10 on different target areas for mean computation. The used model was c) from Figure 1.

Reference area	Target	Accuracy
Baseline	non	84.14%
Global	Weight	84.91%
Tensor	Weight (WC)	85.95%
Channel	Weight	85.63%
Instance	Weight	84.37%
Global	Gradient	84.01%
Tensor	Gradient (GC [36])	84.89%
Channel	Gradient	84.38%
Instance	Gradient	83.36%
Global	ERROR	79.03%
Sample	ERROR (EL)	72.15%
Channel	ERROR (EB)	75.40%
Instance	ERROR	69.73%

lutional tensor seems to be most effective for normalization. This means that each tensor is used for the mean calculation and this mean is subtracted only from this tensor. It is also clearly seen that weight normalization gives better results independent of gradient normalization except for instance based normalization. In the instance based normalization an average value is calculated for every two dimensional mask and this average value is subtracted from the mask. Based on these results we decided to define WC on the tensor and to discard the normalization of the back propagated error for further evaluations.

In Table 3 the results on CIFAR10 of different normalizations and also the baseline, which is CNN without normalization, are shown. As you can see, the combination WC and GC can be effectively applied to all convolutions and additionally to the penultimate fully connected layer (Indicated by the keyword *fully*). This can be seen in model a) and c) from Figure 1. In model b) there is only one fully connected layer in which normalization is not effective because it generates the output. For model a) and b) we have also performed the evaluations with and without batch normalization. As you can see, the combination WC and GC works even better without batch normalization for model a). This is due the result is the best of all for this model. Especially together with the normalization in the penultimate fully connected layer. In case of model b), the additional use of batch normalization is much better, which is due to the residual blocks. Therefore, for the further evaluations, all residual blocks are evaluated with batch normalization only. Since model c) does not have an integrated batch normalization, we have only evaluated without batch normalization.

The combination of WS and GC has not proven to be advantageous for all models, which is why we will not use it in the further evaluations. In general, the best normaliza-

Table 3. Classification accuracy on the CIFAR10 data set. The first column specifies the methods, the second column the used model from Figure 1 and the third column is the classification accuracy. Models a) and b) were evaluated with and without batch normalization. For the models a) and c) we also used the normalization in the penultimate fully connected layer which is specified with the keyword *fully*.

Method	Model	Accuracy
Baseline	a	81.87%
WN [31, 11] $k = 1$	a	71.74%
WC	a	85.01%
WS [29]	a	73.00%
GC [36]	a	81.42%
WS [29], GC [36]	a	74.51%
WC, GC [36]	a	85.75%
WC, GC [36], fully	a	87.07%
Baseline, BN [13]	a	84.67%
WN [31, 11] $k = 1$, BN	a	82.95%
WC, BN [13]	a	85.38%
WS [29], BN [13]	a	79.65%
GC [36], BN [13]	a	84.01%
WS [29], GC [36], BN [13]	a	81.01%
WC, GC [36], BN [13]	a	85.48%
WC, GC [36], BN [13], fully	a	85.95%
Baseline	b	88.35%
WN [31, 11] $k = 1$	b	58.77%
WC	b	80.15%
WS [29]	b	nan
GC [36]	b	69.85%
WC, GC [36]	b	89.61%
Baseline, BN [13]	b	91.00%
WN [31, 11] $k = 1$, BN [13]	b	61.02%
WC, BN [13]	b	92.50%
WS [29], BN [13]	b	79.83%
GC [36], BN [13]	b	92.01%
WS [29], GC [36], BN [13]	b	79.71%
WC, GC [36], BN [13]	b	92.68%
Baseline	c	84.14%
WN [31, 11] $k = 1$	c	83.73%
WC	c	85.95%
WC, fully	c	86.64%
WS [29]	c	10.05%
GC [36]	c	84.89%
GC [36], fully	c	85.37%
WS [29], GC [36]	c	10.72%
WC, GC [36]	c	87.46%
WC, GC [36], fully	c	87.62%

tion across all evaluations on CIFAR10 is the combination of WC and GC. For the residual blocks the batch normalization is added. For the consideration of the normalizations individually without batch normalization, WC is clearly the best, with GC coming directly after.

702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755

Table 4. Classification accuracy on the CIFAR100 data set. The first column specifies the methods, the second column the used model from Figure 1 and the third column is the classification accuracy. Model a) was evaluated with and without batch normalization. For the models a) and c) we also used the normalization in the penultimate fully connected layer which is specified with the keyword *fully*.

Method	Model	Accuracy
Baseline	a	46.31%
WN [31, 11] $k = 1$	a	45.93%
WC	a	50.19%
WS [29]	a	41.19%
GC [36]	a	45.55%
WC, GC [36]	a	50.99%
WC, GC [36], fully	a	52.03%
Baseline, BN [13]	a	54.04%
WN [31, 11] $k = 1$, BN	a	44.26%
WC, BN [13]	a	55.01%
WS [29], BN [13]	a	48.99%
GC [36], BN [13]	a	53.59%
WC, GC [36], BN [13]	a	56.45%
WC, GC [36], BN [13], fully	a	56.78%
Baseline, BN [13]	b	68.99%
WN [31, 11] $k = 1$, BN [13]	b	52.97%
WC, BN [13]	b	69.89%
WS [29], BN [13]	b	63.52%
GC [36], BN [13]	b	69.34%
WC, GC [36], BN [13]	b	70.24%
Baseline	c	46.31%
WN [31, 11] $k = 1$	c	10.31%
WC	c	52.05%
WS [29]	c	34.65%
GC [36]	c	47.95%
WC, GC [36]	c	53.16%
WC, GC [36], fully	c	53.90%

Table 4 shows the results of models a), b), and c) of Figure 1 on the CIFAR100 data set. As you can see, again the combination WC and GC is the most effective. As with CIFAR10 (Table 3), this applies in particular to the additional use of normalization in the last fully connected layer (Indicated by the keyword *fully*). Like CIFAR10 (Table ??), the normalization WC always delivers better results in comparison to GC, if both normalizations are evaluated alone. However, there is a difference for the batch normalization for model a). The additional batch normalization is much more effective than model a) without batch normalization. In all evaluations in Tables 3 and 4 one also sees that the normalizations WS and WN have worsened the generalization of the model. In one case WS even led to a nan result.

Table 5 shows the evaluation of different normalization methods on the VOC2012 data set with model d) from Figure 1. Normalization was used in all layers except the final

Table 5. The average pixel accuracy classification results for different normalization methods on the VOC2012 validation set using model d) from Figure 1. We applied the normalization specified in column one to all layers except for the last convolution.

Method	Average Pixel Accuracy
Baseline, BN [13]	85.15%
WS [29], BN [13]	81.23%
WN [31, 11] $k = 1$	75.76%
GC [36], BN [13]	85.91%
WC, BN [13]	86.92%
WC, GC [36], BN [13]	88.98%

convolution before output. As you can see, both GC and WC improve the result significantly. In combination with the batch normalization, the result is improved by more than 3%. This clearly shows that the combination of WC and GC can be used very effectively together with batch normalization for residual blocks. For the methods WS and WN, however, the generalization of the deep neural network is even worse.

7. Limitations

A disadvantage of WC and GC is that for residual blocks without the batch normalization the results are also worse. This can be seen in Table 3 for model b) from Figure 1. Here you can see in the evaluations that for both as a single normalization without the batch normalization the results are significantly worse. In combination, however, they work better than the model without normalization and without batch normalization. An advantage of the combination of WC and GC compared to batch normalization is that they only need to be used in training (see Algorithm 2 and 3). For batch normalization, however, it is necessary to apply the mean subtraction, division by the standard deviation, scaling, and shift at runtime. However, since this can be calculated with a complexity linear to the input, it hardly affects the runtime.

8. Conclusion

In this work we have shown that weight centralization is a very effective normalization method. Together with the gradient centralization and for residual networks additionally with the batch normalization, this combination exceeds the state of the art. We have also shown over which area the mean subtraction is most effective. Our results were generated with four different nets on three public data sets and clearly show that the additional use of weight centralization is effective and improves the generalization of deep neural networks.

810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863

References

- [1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016. 1, 2, 3, 4
- [2] Léon Bottou. Stochastic gradient learning in neural networks. *Proceedings of Neuro-Nimes*, 91(8):12, 1991. 1, 4
- [3] Minhyung Cho and Jaehyung Lee. Riemannian approach to batch normalization. In *Advances in Neural Information Processing Systems*, pages 5225–5235, 2017. 2
- [4] John Duchi, Elad Hazan, and Yoram Singer. Adaptive sub-gradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011. 1
- [5] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. <http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html>. 5, 6
- [6] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010. 1, 5, 6
- [7] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014. 1
- [8] Harshit Gupta, Kyong Hwan Jin, Ha Q Nguyen, Michael T McCann, and Michael Unser. Cnn-based projected gradient descent for consistent ct image reconstruction. *IEEE transactions on medical imaging*, 37(6):1440–1453, 2018. 2
- [9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015. 1
- [10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 1
- [11] Lei Huang, Xianglong Liu, Yang Liu, Bo Lang, and Dacheng Tao. Centered weight normalization in accelerating training of deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2803–2811, 2017. 1, 2, 3, 7, 8
- [12] Xun Huang and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1501–1510, 2017. 1, 2, 3
- [13] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015. 1, 2, 3, 4, 6, 7, 8
- [14] Davis E King. Dlib-ml: A machine learning toolkit. *Journal of Machine Learning Research*, 10(Jul):1755–1758, 2009. 5
- [15] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 1, 2, 3, 4, 5
- [16] Durk P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. In *Advances in neural information processing systems*, pages 2575–2583, 2015. 1
- [17] David Kirk et al. Nvidia cuda software and gpu parallel computing architecture. In *ISMM*, volume 7, pages 103–104, 2007. 1
- [18] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 5
- [19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 1
- [20] Måns Larsson, Anurag Arnab, Fredrik Kahl, Shuai Zheng, and Philip Torr. A projected gradient descent method for crf inference allowing end-to-end training of arbitrary pairwise potentials. In *International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition*, pages 564–579. Springer, 2017. 2
- [21] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 1
- [22] K Levenberg. A method for the solution of certain problems in least squares, 1944. 1
- [23] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015. 5
- [24] Donald W Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the society for Industrial and Applied Mathematics*, 11(2):431–441, 1963. 1
- [25] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010. 1
- [26] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. Understanding the exploding gradient problem. *CoRR, abs/1211.5063*, 2:417, 2012. 1, 2
- [27] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318, 2013. 1, 2
- [28] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1):145–151, 1999. 1, 2, 3, 4, 6
- [29] Siyuan Qiao, Huiyu Wang, Chenxi Liu, Wei Shen, and Alan Yuille. Weight standardization. *arXiv preprint arXiv:1903.10520*, 2019. 1, 2, 3, 7, 8
- [30] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015. 5, 6

972	[31] Tim Salimans and Durk P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In <i>Advances in neural information processing systems</i> , pages 901–909, 2016. 1, 2, 3, 7, 8	1026
973		1027
974		1028
975		1029
976	[32] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? In <i>Advances in Neural Information Processing Systems</i> , pages 2483–2493, 2018. 1	1030
977		1031
978		1032
979		1033
980	[33] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. <i>IEEE transactions on Signal Processing</i> , 45(11):2673–2681, 1997. 2	1034
981		1035
982		1036
983	[34] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. <i>arXiv preprint arXiv:1607.08022</i> , 2016. 1, 2, 3	1037
984		1038
985		1039
986	[35] Yuxin Wu and Kaiming He. Group normalization. In <i>Proceedings of the European conference on computer vision (ECCV)</i> , pages 3–19, 2018. 1, 2, 3	1040
987		1041
988		1042
989	[36] Hongwei Yong, Jianqiang Huang, Xiansheng Hua, and Lei Zhang. Gradient centralization: A new optimization technique for deep neural networks. <i>arXiv preprint arXiv:2004.01461</i> , 2020. 2, 3, 4, 7, 8	1043
990		1044
991		1045
992		1046
993		1047
994		1048
995		1049
996		1050
997		1051
998		1052
999		1053
1000		1054
1001		1055
1002		1056
1003		1057
1004		1058
1005		1059
1006		1060
1007		1061
1008		1062
1009		1063
1010		1064
1011		1065
1012		1066
1013		1067
1014		1068
1015		1069
1016		1070
1017		1071
1018		1072
1019		1073
1020		1074
1021		1075
1022		1076
1023		1077
1024		1078
1025		1079