

The Hidden Side of Deep Learning

Keesiu Wong

Technical University of Munich

Matriculation number: 03643161

keesiu.wong@tum.de

Abstract

In the recent years, deep learning research focused a lot on architectural advances for specific use cases, often neglecting other, more general techniques. This paper aims to shed light to this "hidden side" of deep learning, in order to support researchers and practitioners with their development. Therefore, multiple, promising methods are presented for the initialization, normalization, and regularization of deep neural networks, structured in a comprising taxonomy. Finally, a practical guideline is provided, suggesting possible solutions for common deep learning problems.

1. Introduction

Modern deep learning models are continuously evolving, their architectures are becoming larger, increasingly complicated and more and more tailored for certain use cases. Besides those highly specialized, structural designs of the neural networks, there are many ways to improve the learning process and performance of those models in practice. However, those generally applicable tactics receive significantly less research attention compared to the architectural design decisions and thus are less understood.

Therefore, this paper is a review to shed light on that hidden side of deep learning, by summarizing different ideas to improve the efficiency and effectiveness of neural network training, surveying those different approaches in terms of how well and why they work, as well as suggesting which techniques to use when.

The content is structured as follows: In Section 2, techniques are discussed that aims at supporting the underlying optimization process. Concretely, subsection 2.1 presents the two most important initialization methods in detail, namely Xavier and He Initialization, while section 2.2 focuses on Batch, Weight, Layer, Instance, and Group Normalization. Regularization methods are covered in section 3, together with a respective taxonomy. More precisely, subsection 3.1 deals with regularization via an additional term to the loss function, for example L^2 or L^1 regulariza-

tion. In contrast, subsection 3.2 shows regularization techniques via the network architecture, like Parameter Sharing, Ensemble Methods or Dropout. Furthermore, subsection 3.3 points out how to regularize effectively via data, especially with carefully tuned data augmentation and its meta-learning variants like AutoAugment. Finally, section 4 concludes with a holistic guideline, summarizing possible solutions for common problems during development of fully-connected, convolutional and recurrent neural networks.

Consequently, this review provides three distinct research contributions: firstly, a taxonomy to structure optimization- and regularization-based strategies, secondly an overview of the most promising initialization, normalization and regularization techniques, as well as thirdly, a practical guideline which tweak to use in which situation.

2. Optimization

Optimization techniques are an integral tool of machine learning, and one way to enhance the training of those models are to improve the underlying optimization mechanism. However, in deep learning second-order methods are often not applicable, since it is too costly to evaluate the Hessian. Thus, the training of neural networks often relies on first-order, gradient-based methods like Stochastic Gradient Descent or Adam.

Besides those two well-known and widely used optimizers, many alternatives were proposed in the past. Also, advancements like task-specific loss formulations or new optimization strategies like learning rate decay were developed to improve the optimization process within deep learning. However, for the purpose of this paper, those topics are not within its scope.

Rather, this section focuses on two other, often neglected ways to ease the training of neural networks in the following. On the one hand, good initialization strategies aim to begin the optimization process from more promising starting points, in the sense that it provides meaningful and stable gradients during back-propagation. On the other hand, normalization techniques are also crucial to stabilize the activations and enabling larger learning rates for fast training.

2.1. Initialization

In deep learning, the initialization of the weights can have a tremendous impact on the convergence behavior. In fact, until a few years ago, it was a significant bottleneck to the depth of neural networks. [13] showed that even for a small, five-layer neural network with hyperbolic tangent as activation functions, a naive, uniformly-distributed initialization of the weights $w_i \sim U[-\frac{1}{\sqrt{n_{in}}}, \frac{1}{\sqrt{n_{in}}}]$, n_{in} being the size of the previous layer, can lead to quick saturation of the activations, as well as the back-propagated gradients, see Appendix A.1.

To understand the reason for these issues of vanishing and exploding gradients, the key is to look at the variance of a output unit y (with omitted bias for simplicity):

$$\text{Var}(y) = \text{Var}\left(\sum_i^{n_{in}} w_i x_i\right) \quad (1)$$

$$= \sum_i^{n_{in}} \text{Var}(w_i x_i) \quad (2)$$

$$= \sum_i^{n_{in}} [\mathbb{E}(w_i)]^2 \text{Var}(x_i) + [\mathbb{E}(x_i)]^2 \text{Var}(w_i) + \text{Var}(w_i) \text{Var}(x_i) \quad (3)$$

$$= \sum_i^{n_{in}} \text{Var}(w_i) \text{Var}(x_i) \quad (4)$$

$$= n_{in} \text{Var}(w_i) \text{Var}(x_i). \quad (5)$$

Here, we have assumed a mutual independence of the w_i and x_i in equation (2), zero-mean of w_i and x_i in equation (4) and finally identical distribution of all w_i and x_i in equation (5). Thus, the variance of a output unit y gets multiplied by the number of input units n_{in} .

However, in order to avoid saturation of the activations, a good initialization strategy should avoid reducing or magnifying the magnitudes of the variance, and rather keep the variance of the output the same as the input. Thus, the weight initialization should be chosen such that

$$\forall i, \text{Var}(w_i) := \frac{1}{n_{in}}. \quad (6)$$

Furthermore, from a back-propagation perspective, we also want to avoid vanishing or exploding gradients. Thus, with a similar calculation, an additional condition can be derived:

$$\forall i, \text{Var}(w_i) := \frac{1}{n_{out}}, \quad (7)$$

with n_{out} being the size of the next layer. Note, that both equations (6) and (7) cannot be fulfilled at the same time, unless the weight matrix is quadratic.

2.1.1 Xavier Initialization

In order to compromise between these two constraints (6) and (7), [13] proposed to rely on their harmonic mean:

$$\text{Var}(w_i) := \frac{2}{n_{in} + n_{out}}. \quad (8)$$

Given that the weights are sampled from a Uniform distribution, this led to their proposed, normalized initialization method, nowadays known as *Xavier or Glorot Initialization*:

$$w_i \sim U\left[-\frac{\sqrt{6}}{\sqrt{n_{in} + n_{out}}}, \frac{\sqrt{6}}{\sqrt{n_{in} + n_{out}}}\right]. \quad (9)$$

Computing the variance $\text{Var}(w_i) = \frac{1}{12} \left(\frac{2\sqrt{6}}{\sqrt{n_{in} + n_{out}}}\right)^2 = \frac{2}{n_{in} + n_{out}}$ equals the desired harmonic mean. This leads to much more stable and diverse activations and gradients, see Appendix A.2. Note, that if one samples from a Normal distribution instead of a Uniform distribution, one can directly set the variance to equation (8) to archive the same behavior.

2.1.2 He Initialization

The derivation of the Xavier Initialization is based on the simplification that activation functions are linear, which is approximative in the case of hyperbolic tangent. However, modern deep learning architectures rely on highly non-linear activation functions like the Rectified Linear Unit (ReLU), for which this assumption is invalid. This gave rise to the so-called *He or Kaiming Initialization* [18]. Here, the variance is calculated as follows:

$$\text{Var}(y) = \text{Var}\left(\sum_i^{n_{in}} w_i x_i\right) \quad (10)$$

$$= \sum_i^{n_{in}} \text{Var}(w_i x_i) \quad (11)$$

$$= n_{in} \text{Var}(w_i x_i) \quad (12)$$

$$= n_{in} [\mathbb{E}(w_i)]^2 \text{Var}(x_i) + [\mathbb{E}(x_i)]^2 \text{Var}(w_i) + \text{Var}(w_i) \text{Var}(x_i) \quad (13)$$

$$= n_{in} [\mathbb{E}(x_i)]^2 \text{Var}(w_i) + \text{Var}(w_i) \text{Var}(x_i) \quad (14)$$

$$= n_{in} \text{Var}(w_i) [\mathbb{E}(x_i)]^2 + \text{Var}(x_i) \quad (15)$$

$$= n_{in} \text{Var}(w_i) \mathbb{E}(x_i^2). \quad (16)$$

Again, we have assumed mutual independence of the w_i and x_i in equation (11), identical distribution of w_i and x_i in equation (12), and zero-mean of w_i in equation (14). However, in contrast to the previous case, we cannot assume zero mean for x_i due to the ReLU activation, which is why $\mathbb{E}(x_i^2)$ cannot be reduced to $\text{Var}(x_i)$.

Instead, the expectation is computed as

$$\mathbb{E}(x_i^2) = \int_{-\infty}^{+\infty} \max(0, y')^2 \mathbb{P}(y') dy' \quad (17)$$

$$= \int_0^{+\infty} \max(0, y')^2 \mathbb{P}(y') dy' \quad (18)$$

$$= \int_0^{+\infty} y'^2 \mathbb{P}(y') dy' \quad (19)$$

$$= \frac{1}{2} \int_{-\infty}^{+\infty} y'^2 \mathbb{P}(y') dy' \quad (20)$$

$$= \frac{1}{2} \int_{-\infty}^{+\infty} [y' - \mathbb{E}(y')]^2 \mathbb{P}(y') dy' \quad (21)$$

$$= \frac{1}{2} \mathbb{E}[[y' - \mathbb{E}(y')]^2] \quad (22)$$

$$= \frac{1}{2} \text{Var}(y'), \quad (23)$$

where y' denotes the pre-activations of the previous layer. Note, that in equation (18) we used the fact that the integrand is zero for negative integration variables, in equation (20) we assumed symmetry of $\mathbb{P}(y')$ around zero, and in equation (21) we assumed the pre-activations to be zero-mean. Together with the previous result, this leads to:

$$\text{Var}(y) = \frac{1}{2} n_{in} \text{Var}(w_i) \text{Var}(y'). \quad (24)$$

This can be interpreted as that taking the ReLU activation functions into account leads to killing half of the pre-activations, resulting in halving the the variance of the activations during the forward pass from one layer to the next.

Again, to achieve stable activations, the variance needs to be re-scaled such that the magnitudes of the variance remain constant (e.g. 1):

$$\forall i, \text{Var}(w_i) := \frac{2}{n_{in}}. \quad (25)$$

That means, in order to compensate for ReLU, the variance of the weights only needs to be doubled compared to equation (6). Additionally, [16] showed that only a variance set like (25) can avoid exponential growth in the length of the activation vectors. It is even sufficient to only scale the forward signal properly, since the backward signal is then automatically stabilized, and vice versa [18]. The superiority of He Initialization over Xavier Initialization for ReLU-based architectures is also shown experimentally, see Appendix A.2.1. Furthermore, [29] showed mathematically, that if one takes the non-linear activation functions into consideration, Xavier Initialization actually exhibits exponentially smaller variance of the inputs with increasingly deeper layers, while He Initialization is provably stable.

2.1.3 Fixup Initialization

Fixup Initialization [64] was designed to circumvent the need for normalization in deep Residual Networks (ResNets) [17]. Usually, standard initialization methods like Xavier or He initialization cannot properly account for the skip-connections in ResNets without normalization techniques like Batch Normalization, leading to exploding gradients. However, Fixup Initialization proposed the following method to train ResNets with m layers and L residual branches:

1. Initialize the classification layer and the last layer of each residual branch to 0.
2. Initialize every other layer using a standard method (e.g. He Initialization), and scale only the weight layers inside residual branches by $L^{-\frac{1}{2m-2}}$.
3. Add a scalar multiplier (initialized at 1) in every branch and a scalar bias (initialized at 0) before each convolution, linear, and element-wise activation layer.

This stabilizes the training of deep residual networks, even for networks with 10,000 layers. Furthermore, Fixup enabled ResNets to achieved state-of-the-art performance in image classification and machine translation, both without normalization.

2.1.4 Further Initialization Alternatives

Besides the presented methods, multiple other initialization strategies were developed in the past. A selection is presented in the following:

- *LSUV* (Layer-Sequential-Unit-Variance) Initialization [34] suggests to first pre-initialize the weights with orthonormal matrices, and then normalizing the variance of the outputs to one. Thus, it can be seen as a Batch Normalization on layer output done before the start of the training. Empirical results indicate that it outperforms Xavier and He Initialization with maxout or ReLU activations on image classification tasks.
- [1] improved the LSUV by introducing a *active fraction* hyperparameter in order to control how likely it is for a ReLU unit to produce a non-zero value.
- [26] proposed a *Data-Dependent Initialization* for CNNs. The idea is to set the weights such that all units train at a similar rate to avoid vanishing or exploding gradients, by using either PCA or k-means. Both approaches outperforms Xavier and He Initialization on image recognition and object detection problems.
- For RNNs with ReLU activations, [50] initializes weights via a normalized positive-definite matrix, by scaling an identity matrix with Gaussian noise with its largest eigenvalue.

2.2. Normalization

Normalization is often a crucial technique to dramatically improve the training efficiency of neural networks. In general, it follows the idea to stabilize the training by normalizing the distribution of layer inputs and thus allowing larger learning rates.

2.2.1 Batch Normalization

Batch Normalization [24] is arguably the most adopted normalization method. For input mini-batch $B = \{x_1, \dots, x_m\}$, learnable parameters γ, β and output $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$, it is defined as:

$$\begin{aligned} \mu_B &\leftarrow \frac{1}{m} \sum_{i=1}^m x_i && // \text{ mini-batch mean} \\ \sigma_B^2 &\leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_B)^2 && // \text{ mini-batch variance} \\ \hat{x}_i &\leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} && // \text{ normalize} \\ y_i &\leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) && // \text{ scale and shift} \end{aligned}$$

Therefore, it inserts an additional layer on top of the pre-activations of a given layer, that normalizes them across the mini-batch to zero-mean and unit standard deviation by subtracting the mini-batch-wise mean and dividing by its mini-batch-wise standard-deviation per feature. The strategy is to limit the so-called *Internal Covariate Shift*, meaning the "change in the distribution of of network activations due to the change in network parameters during training" [24].

However, despite the effectiveness of Batch Normalization in practice, the Internal Covariate Shift might not be the main reasoning for its success. Rather, the reparametrization of the underlying optimization problem might have a smoothing effect on the loss landscape, thus enabling more stable gradients and faster training [45].

2.2.2 Weight Normalization

Weight Normalization [44] is an alternative to standard Batch Normalization, where the normalization is performed on the weights of the layers directly, instead on their pre-activations. By reparametrizing the weight vector \mathbf{w} of each neuron $y = \phi(\mathbf{w} \cdot \mathbf{x} + b)$ into $\mathbf{w} = g \cdot \mathbf{v} / \|\mathbf{v}\|$, its Euclidean norm can be directly trained via the scalar $g = \|\mathbf{w}\|$, independent from its direction \mathbf{v} .

Hence, Weight Normalization results in a similar speed up of the optimization process like Batch Normalization while requiring 16% lower computational overhead in the case of Convolutional Neural Networks (CNNs), because they usually have much fewer weights than input dimensions [44].

2.2.3 Layer Normalization

Layer Normalization [2] is similar to Batch Normalization. Each pre-activation is normalized by subtracting a mean and dividing by a standard deviation. However, in Layer Normalization, those statistics are computed across the layer and separately for each sample, instead across the mini-batch. Concretely, given the outputs of the l^{th} hidden layer h^l , the incoming weights of the i^{th} hidden units w_i^l , and pre-activations $a_i^l = w_i^{l\top} h^l$, the activation of the i^{th} unit of the l^{th} hidden layer is computed as $h_i^l = f(\frac{g_i}{\sigma_i}(a_i - \mu_i) + b_i)$, with layer-wise mean $\mu^l = \frac{1}{H} \sum_{i=1}^H a_i^l$, layer-wise standard deviation $\sigma^l = \sqrt{\frac{1}{H} \sum_{i=1}^H (a_i^l - \mu^l)^2}$, non-linear activation function f , as well as the bias b_i and the gain g_i as learnable parameters.

Contrary to Batch Normalization, Layer Normalization is also straightforward to implement for Recurrent Neural Networks (RNNs), because it only depends on the inputs to a layer at the current time-step. Experimental results with a Long-Short-Term-Memory (LSTM) architecture indicate, that Layer Normalization was not only able to converge much faster than Batch Normalization, but also converges to a lower validation error, see Appendix A.3.

2.2.4 Instance Normalization

Instance Normalization [53] is a proposed normalization method for Style Transfer problems. Similarly to Layer Normalization, it is performed over a single sample instead of the whole mini-batch. However, the normalization statistics are also calculated per channel, which removes instance-specific contrast information. This supports the generator network and accelerates the stylization process of a 512x512 image from several minutes to real-time [53].

2.2.5 Group Normalization

Group Normalization [60] is another normalization method. As the name suggests, the normalization statistics are computed by partitioning each image along the channel dimension into equally-sized groups. Thus this method can be seen as a middle ground between Layer and Instance Normalization, as Figure 1 illustrates. The rationale behind his grouping is to leverage the correlations within close channels, while allowing more flexibility than Layer Normalization.

For image classification problems, this results in a similar performance to Batch Normalization, and in particular outperforms Layer and Instance Normalization, see Appendix A.4. But in contrast to Batch Normalization, Group Normalization remains stable with smaller batch sizes, see Appendix A.5.

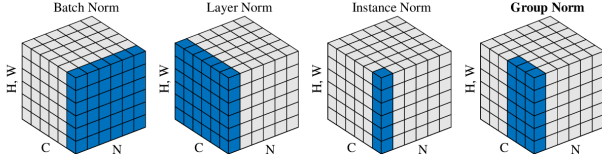


Figure 1. Different normalization methods. [60]

2.2.6 Further Normalization Alternatives

Besides the presented normalization methods, many more alternatives to standard Batch Normalization came up in recent years, designed for different purposes. However, many of those variants are not well-known. An exhaustive overview would require extensive literature review and goes beyond the scope of this paper. Nonetheless, some promising methods are pointed out in the following:

- *Batch Re-Normalization* [23] extends Batch Normalization with an affine transformations per dimension to work better on small or not independent identically distributed mini-batches.
- *Batch-Instance Normalization* [39] interpolates Batch and Instance Normalization via a trainable parameter, thus enables adaptive learning on how much style information to use.
- *Switchable Normalization* [33] takes this idea even further and learns a weighted average between Batch, Layer, and Instance Normalization, resulting in better performance at various computer vision problems.
- *Spectral Normalization* [35] controls the Lipschitz constant of the discriminator function to stabilize the training of Generative Adversarial Networks (GANs), by normalizing the spectral norm of the weight matrix such that satisfies a Lipschitz constraint.
- *ScaleNorm* [40] uses a scaled Euclidean normalization in order to replace Layer Normalization within Transformer models for even faster training and an additional regularization effect.
- *Recurrent Batch Normalization* [6] translates standard Batch Normalization to the LSTM architecture in both the input-to-hidden and the hidden-to-hidden transformations.
- *GraphNorm* [4] is a recent normalization technique designed to leverage the specific structure of Graph Neural Networks (GNNs) and yield higher efficiency and improved generalization.

3. Regularization

Regularization is one of the most important pillars in machine learning, trying to get models to perform well also on new, unseen input, instead of just on the training data. This goal is what makes machine learning actually a learning problem, and fundamentally harder than a pure optimization problem. Concretely, regularization is defined as "any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error" [15].

However, there are many regularization strategies to achieve that, ranging from encoding specific prior knowledge to penalizing complexity in favor for simple models. Furthermore, those regularization can be explicit, meaning techniques which reduce the representational capacity of the model, or implicit, i.e. affect the effective, achievable model capacity [20].

In order to classify different regularization techniques, [28] proposes a taxonomy of regularization strategies based on the underlying empirical risk minimization problem:

$$\operatorname{argmin}_w \frac{1}{|\mathcal{D}|} \sum_{(x_i, t_i) \in \mathcal{D}} E(f_w(x_i), t_i) + R(\dots).$$

Considering the elements of this formula, five different directions of regularization strategies can be distinguished:

1. R : via a regularization term (subsection 3.1),
2. f : via the network architecture (subsection 3.2),
3. \mathcal{D} : via data (subsection 3.3),
4. E : via the error function or
5. argmin : via the optimization itself.

Based on this taxonomy, the most important regularization strategies based on regularization terms, network architecture or data are presented in the following. Regularization techniques based on changing of the error functions (e.g. mean-squared-error or cross-entropy) or modifying the optimization process (e.g. early stopping) are not within the scope of this paper. In practice, multiple of such regularization techniques might be combined. For a broader overview, also confer [28].

3.1. Regularization via Regularization Term

The classical approach of explicitly regularizing a machine learning model via a regularization term stems from its application within (Linear) Regression models. Usually, a regularizer R is added to the loss function, in order to penalize the parameter norm and to drive the weights closer to zero. In contrast to the error function E , the regularizer R is independent of the targets and rather encodes other desired properties, thus provides inductive bias to the model [28].

3.1.1 L^2 Regularization

L^2 Regularization [15], also known as *Weight Decay*, is the most popular regularization term: $R(w) = \lambda \frac{1}{2} \|w\|_2^2$. Here, λ is a hyperparameter to control the strength of the regularization. Note, that Weight Decay induces bias to the model corresponding to a symmetric multivariate normal distribution prior on the weights [28]. Recent research also indicates that the effectiveness of Weight Decay highly relies on the timing of the regularization, i.e. only plays a critical role during the initial transient period, but not for convergence or the generalization afterwards [14].

3.1.2 L^1 Regularization

L^1 Regularization [15] uses the absolute-value norm instead of the Euclidean norm: $R(w) = \lambda \|w\|_1$. From a Bayesian perspective, this is equivalent to a log-prior term that is maximized by MAP Bayesian inference when the prior is an isotropic Laplace distribution over w [15]. Consequently, L^1 Regularization leads to a more selective weight distribution and thus to a more sparse model.

3.1.3 Elastic Net Regularization

Elastic Net Regularization [68] was proposed to combine the strength of L^2 and L^1 regularization. Especially in a regression setting, the Elastic Net approach improves the prediction performance via a bias-variance trade-off similarly to Ridge regression, but also incorporates the variable selection effect from Lasso regression, leading to a more parsimonious and sparse model.

3.1.4 Contractive Penalty

Contractive Penalty [3] aims to control the smoothness of the learned function, by penalizing large derivatives: $R(f_w, x) = \|J_{f_w}(x)\|_F^2$. Here, $\|\cdot\|_F$ denotes the Frobenius norm and J_{f_w} the Jacobian matrix of f_w . It was originally designed for Contractive Autoencoders [43].

3.1.5 Parameter Tying

In a similar way like L^2 encourages weights to be close to zero, *Parameter Tying* [15] is another way to focus the search space of model parameters to be close to some other, promising values. For example, one can use a parameter norm penalty of the form $R(w^{(A)}, w^{(B)}) = \|w^{(A)} - w^{(B)}\|_2^2$ to tie the weights $w^{(A)}$ of a model A to the weights $w^{(B)}$ of another model B , which was trained on a similar task. Furthermore, Parameter Tying can be also used to tie the weights of a supervised model to the weights of an unsupervised model on the same dataset, in order to capture the distribution of the observed input data [30].

3.2. Regularization via Network Architecture

Another way to regularize a neural network is to alter its architecture to have certain properties, matching certain domain-specific assumptions. One might even argue that any modification to a Fully-Connected Neural Network encodes some form of knowledge. However, additional constraints usually also make the optimization task harder, in exchange for achieving a regularization effect.

3.2.1 Parameter Sharing

Parameter Sharing [15] means to force sets of parameters to be equal - thus can be seen as an extreme form of Parameter Tying. However, those equality constraints are usually incorporated within the network architecture f itself, instead of a regularization term R like in the Parameter Tying case. One of the most prominent use cases of Parameter Sharing are CNNs. Those convolutional layers encode prior knowledge about shift-equivariance and locality of feature extraction [28]. Furthermore, they also reduce the number of weights that need to be learned by summarizing them into convolutional kernels, thus also speeds up the optimization process significantly. In a similar way, RNNs use Parameter Sharing to reflect the time-equivariance of input and hidden units.

3.2.2 Ensemble Methods and Bagging

Ensemble Methods [15] use different models to get multiple predictions per input sample. Those models are then averaged, meaning they democratically vote for the most likely output. The rationale behind this approach is that different models usually don't make the same errors on the test set.

Bagging (Bootstrap Aggregating) [15] is one form of ensemble methods. It creates multiple different models by training them on different sub-datasets, each sampled from the original training set with replacement. Since not every sample is present in every sub-dataset, outliers don't affect all models, resulting in a more robust model average.

3.2.3 Dropout

For neural networks, Bagging is often not computationally feasible. Instead, one of the alternatives is to use *Dropout* [49]. This introduces stochasticity into the model architecture by randomly dropping units from the network during training, usually each with 50% probability. Effectively, this is like training an ensemble of multiple "thinned" sub-networks with Parameter Sharing. At test time, the whole, unthinned network is then again used to approximate the ensemble vote, with a down-scaling to correct the expected value. Dropout brings not just an explicit regularization by modifying the expected objective, but also an implicit one due to the stochasticity in the updates [56].

3.2.4 Further Dropout Alternatives

Besides standard Dropout, many alternatives and improvements were developed in the last years. The main variants are point out in the following:

- *MC-Dropout* [11] suggests to use Dropout as Bayesian approximation by performing multiple stochastic forward passes through a network and averaging the results, allowing to assess model uncertainty.
- *DropConnect* [54] generalizes Dropout, that sets a randomly selected subset of weights to zero, instead of a subset of activations. Empirical results show, that DropConnect often outperforms standard Dropout.
- *MC-DropConnect* [36] combines the Bayesian approach of MC-Dropout with DropConnect, resulting in better uncertainty estimations, see Appendix A.6.
- *Spatial Dropout* [51] tries to overcome the problem of strong spatial correlations in CNNs by applying Dropout per feature map.
- *Max-Pooling Dropout* [58] leverages Dropout for convolutional layers by applying it directly to the max-pooling layer before the pooling operation.
- *RNN-Drop* [37] brings the advantage of Dropout to LSTMs, by applying the Dropout mask on the internal cell values and use the same mask at every timestep.
- *Variational RNN Dropout* [12] brings the Bayesian MC-Dropout approach into the RNN-domain by applying the same dropout mask at each time step for inputs, outputs, and recurrent layers. In contrast to RNN-Drop, the masks are applied to the weights instead of the LSTM-internal cell values, thus can be applied to general RNNs and GRUs.
- *Recurrent Dropout* [46] applies Dropout to the cell update vector, which seems to outperform RNN-Drop and Variational RNN Dropout. For an illustration and comparison of those approaches, see Appendix A.7.
- *Curriculum Dropout* [38] proposes to use a time scheduling for the dropout intensity instead of a fixed dropout probability, in order to start easy and adaptively increasing the difficulty of the learning problem. Empirical results show that it can significantly improve standard Dropout, see Appendix B.1.
- *Shakeout* [25] randomly chooses to enhance or reverse each unit's contribution to its next layer, instead of randomly discarding units as in Dropout. Therefore, Shakeout adaptively combines L^0 , L^1 and L^2 regularization terms, leading to improved sparsity and stability during the training process.

3.3. Regularization via Data

The generalization performance of deep neural networks depends on three key factors: the complexity of the underlying problem, a suitable model capacity for sufficient expressiveness, and enough data to represent the problem well. However in practice, the most limiting factor is often the available training data. This is why regularization by increasing the dataset size is usually one of the most effective ways to improve the validation accuracy of a model.

This can be done in two ways: On the one hand, more representative samples can be collected from the real world, preferably from the same data distribution. Even in the extreme case, the model can be pre-trained with a significantly larger, but remotely related dataset for regularization, leveraging so-called *Transfer Learning*. On the other hand, the dataset can be also increased via *Data Augmentation*. In a broader sense, this refers to the general strategy to make the training data larger and more representative by applying certain transformations. Both methods are highly popular and effective, implicit regularizers in practice. Similarly to Weight Decay, the regularization via Data Augmentation is especially important for the early phases of the training process [14]. Recent research even suggests to completely replace Weight Decay and Dropout with it [20].

However, especially Data Augmentation requires domain-specific knowledge and understanding of the specific dataset, since it fundamentally depends on the type of the given data, e.g. images, text, audio, or tabular data. Furthermore, those transformations can be either done in a representation-preserving fashion, i.e. simulating the same ground truth data distribution, or in a representation-modifying way, meaning by adjusting the underlying data distribution or feature space to simplify the learning problem [28].

For the purposes of this paper, a selection over the most important Data Augmentation techniques for image classification problems is presented in the following. A more extensive overview can be also found in [48].

3.3.1 Data Augmentation

Data Augmentation [15] in the traditional sense means to create additional fake data by transforming the inputs x_i of the dataset \mathcal{D} in such a way, that it provides new, meaningful samples. This works especially well for image classification problems in computer vision: geometric transformations like flipping, cropping, rotation, translation, perspective transforms or color space manipulations like brightness, contrast saturation or hue adjustments as well as kernel filters like sharpening or blurring are all typical ways to increase the variety of the training data. Indeed, those kinds of transformations are probably the most well-known regularization technique via data.

3.3.2 Noise Injection

Even injecting random noise, either on the input [15] or on the hidden units [9], is a common strategy to improve the generalization capabilities and the robustness of the model. This universal strategy can be also applied to various data types like audio or tabular data in a natural way. For images, another alternative would be to randomly add some black and white pixels, adding so-called "salt and pepper"-noise.

3.3.3 Cutout and Random Erasing

Cutout [10] takes this idea of adding noise even a step further, by completely removing a randomly chosen square from an image. The rationale is to force the model to take the whole context into account, instead of possibly only focusing on prominent features, and making it more robust against object occlusion. *Random Erasing* [66] follows a very similar approach, and additionally suggests to focus the cutout preferably close to object.

3.3.4 MixUp and SamplePairing

MixUp [63] follows the approach to construct virtual training data by combining multiple training samples, as the name suggests. Concretely, two randomly drawn samples are aggregated to a convex combination, as well as their respective labels, via a Beta-distributed weight parameter. This approach incorporates the prior knowledge, that linear interpolations of feature vectors should lead to the same linear interpolation of the associated targets. [22] proposed a similar, but slightly simpler concept called *SamplePairing*. While it only uses the label of one sample and just relies on a simple average, *SamplePairing* achieves comparable results like *MixUp* (see Appendix A.8).

3.3.5 CutMix

CutMix [62] was designed to combine the strengths of *Cutout* with those of *MixUp*. Instead of simply erasing certain areas and leaving a uninformative and unrealistic region, *CutMix* adds the patch of another image and also mixes the ground truth labels proportionally. Thus, it leverages the regional Dropout feature of *Cutout* while improving the training efficiency and is less locally ambiguous and unnatural compared to *MixUp*. Empirical results show it is significantly outperforming both, see Appendix B.2.

3.3.6 AugMix

Many data augmentation techniques improve accuracy at the cost of robustness or uncertainty. *AugMix* [19] is a recent technique developed to overcome this trade-off. It

uses stochasticity to sample and chain multiple basic transformations, a formulation to mix multiple augmented images into convex combinations, and a Jensen-Shannon Divergence consistency loss to ensure smoother neural network responses. Empirical results indicate an improvement of the robustness and uncertainty estimates, while simultaneously outperforming *Cutout*, *MixUp* and *CutMix*.

3.3.7 Smart Augmentation

Traditional data augmentation relies on a hand-crafted combination of transformations with carefully set parameters. This gives rise to the question if one can learn to optimize the design of such transformation pipelines, that is using meta-learning for data augmentation. *Smart Augmentation* [31] is one of the early attempts to learn data augmentation on a meta-level. It relies on a second neural network, which takes multiple input images and uses convolutional layers to generate a new image for training the classification network.

3.3.8 AutoAugment

AutoAugment [7] was designed to automatically search for the best augmentation policy via Reinforcement Learning. In detail, each policy contains five sub-policies, which again are concatenations of two elemental image operations. Those sub-policies are randomly applied to the images within a mini-batch. For the Reinforcement Learning framework, a RNN controller predicts a augmentation policy, which then is used to train a child network. The achieved accuracy is forwarded as a reward with a policy gradient method to update the controller. Empirical results show that *AutoAugment* consistently achieves state-of-the-art results on various image recognition tasks, outperforming simple methods like *Cutout*, see Appendix B.3.

3.3.9 Fast AutoAugment

However, *AutoAugment* is quite compute-intensive, since for each update step on the meta-level, a whole child network has to be trained. To overcome this challenge and to find good augmentation policies efficiently, *Fast AutoAugment* [32] was proposed. It relies on density matching, treating augmented data as missing data points of the training distribution, and recovering them by the exploitation and exploration of inference-time augmentations via Bayesian hyperparameter optimization. Therefore, it reduces computing hours by three orders of magnitudes, while achieving comparable results like *AutoAugment*.

3.3.10 Population Based Augmentation

Population Based Augmentation [21] is another alternative to *AutoAugment*, also addressing the issue of high com-

putational costs. Instead of finding a fixed augmentation policy via hyperparameter search on a meta level, it aims at learning a schedule of policies. This is done by leveraging Population Based Training, a search algorithm that optimizes the parameters of a network jointly with their hyperparameters. Thus, Population Based Augmentation achieves comparable results like AutoAugment and similar performance gains like Fast AutoAugment.

3.3.11 Adversarial AutoAugment

Adversarial AutoAugment [65] is similar to Population Based Augmentation in that sense, that it replaces a fixed augmentation policy with a dynamic schedule of augmentation policies along with the training process. However, as the name suggests, Adversarial AutoAugment utilizes an adversarial framework to jointly optimize the target network and the augmentation policy search by reformulating it into a min-max-game. In fact, a policy network constantly aims to maximize the training loss through generating adversarial policies, which improves the robustness of the model. Adversarial AutoAugment is not just more efficient than AutoAugment due to using only one target network for policy evaluation, but also significantly outperforms AutoAugment, see Appendix B.4.

3.3.12 RandAugment

RandAugment [8] completely removes the need for a separate, expensive and possibly sub-optimal search phase on a proxy task. Rather, the parameters for the data augmentation are fold into the hyperparameters of the model. Concretely, two new hyperparameters are introduced. One selects how many transformations should be sampled from a uniformly distributed, fixed set of elementary transformations. The second scales the strength of each transformations. These two additional hyperparameters incur minimal computational cost, while the predictive performance gain of RandAugment is competitive compared with AutoAugment, see Appendix B.5.

3.3.13 Uncertainty-based Sampling

Recently, [59] proposed a *Uncertainty-based Sampling* scheme similar to RandAugment. However, instead of averaging the effect of all transformations like RandAugment, the idea is to better select the transformations with strong performance. It is performed by uniformly sampling a number of transformations at random, applying them to the data and training the network on those data points with the highest losses. This approach achieves similar test accuracy like Adversarial AutoAugment, but is conceptually simpler and computationally more efficient, since it does not require training an adversarial network, see Appendix B.6.

3.3.14 Further Data Augmentation Alternatives

Besides these presented, image-centric techniques, some others are presented in the following, focusing on different or more general problem settings:

- *Fancy PCA* [27] adds principal components to the inputs proportional to the corresponding eigenvalues, reflecting the invariance of object identities under changes of illuminations.
- [9] suggests to use simple transformations like noise, interpolation or extrapolation in the feature space instead of the input space.
- *ISDA* (Implicit Semantic Data Augmentation) [55] augments the training set by applying semantic transformations like "make-bespectacled" or "change-view-angle" on deep features.
- *GMCEM* (Generalized Monte Carlo Expectation Maximization) [52] uses a Bayesian approach to data augmentation, by sampling from a distribution learned from the training set.
- *TANDA* (Transformation Adversarial Networks for Data Augmentation) [42] uses Generative Adversarial Networks (GANs) to generate sequences of data augmentation operations for text and image tasks.
- [41] uses a CycleGAN to perform data augmentation via Style Transfer, i.e. combining the content of an image with the style of a second.
- [67] translates AutoAugment from the image classification to the object detection problem.
- *UDA* (Unsupervised Data Augmentation) [61] used advanced data augmentation techniques like RandAugment or Back-Translation to improve semi-supervised learning on vision and language tasks.
- *SMOTE* (Synthetic Minority Over-Sampling Technique) [5] was designed to address the class imbalance of classification tasks, by generating convex combinations in the feature space of the nearest neighbors within the minority class.
- *EDA* (Easy Data Augmentation Techniques) [57] are a set of four operations for augmenting textual data: synonym replacement, random insertion, random swap and random deletion.
- *Back Translation* [47] is a text augmentation technique to paraphrase a text while retaining its meaning. It leverages machine translation by translating a text into another language, and then back to the source language, which usually alters the source text.

4. Guideline

As one saw in the previous sections, there is at great number of different techniques to improve deep learning models in terms of optimization and regularization. However, many of those techniques are not wide-spreaded, neither in research nor in practice. This not only leads to a lack of variety and over-usage of well-known techniques, but also to inferior design choices for the network architecture as well as sub-optimal models in production.

In order to help researchers and practitioners to overcome some problems during deep learning development, multiple problem- and architecture-specific recommendations shall be given. Note, that still each problem instance highly depends on the specific architecture, data and task and thus requires individual investigation and treatment. However, the guideline on the bottom of this page summarizes possible starting points.

5. Conclusion

Despite the recent advances in deep learning, much is still not well understood and many areas remain an active field of research. Especially, the majority of deep learning research focuses on new neural network architectures, often leaving more general optimization and regularization techniques comparatively understudied.

To shed light on this hidden side of deep learning, an overview of various initialization, normalization and regularization methods was given in this paper. Furthermore, a guideline was presented to overcome problems that may arise during deep learning development.

The author hopes that this paper provides valuable insights over modern deep learning techniques, supports researchers and practitioners to speed-up their development processes and improves their models, as well as sparks more research interest towards this exciting side of deep learning.

Problem	Solution	Fully-Connected NN	Convolutional NN	Recurrent NN
Vanishing / Exploding Gradient	Initialization	He Initialization for ReLU based NN	Fixup Initialization for ResNets (otherwise as in Krähenbühl 2016)	As in Talathi 2016 for ReLU based RNN
Learning takes too long	Normalization	Batch Normalization	Group Normalization (Weight Normalization if time complexity bad)	Layer Normalization
Too little data	Data Augmentation	Noise Injection for numeric data (+SMOTE if classification task)	Uncertainty-based Sampling	Back-Translation for textual data, Noise Injection for other sequential data
Overfitting	Regularization	Standard or Curriculum Dropout	Spatial or Max-Pooling Dropout	Recurrent Dropout
High uncertainty	Robustness	MC-DropConnect	AugMix	Variational RNN Dropout

References

- [1] D. Aguirre and O. Fuentes. Improving weight initialization of relu and output layers. In *ICANN*, 2019.
- [2] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization, 2016.
- [3] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives, 2014.
- [4] T. Cai, S. Luo, K. Xu, D. He, T. yan Liu, and L. Wang. Graphnorm: A principled approach to accelerating graph neural network training, 2020.
- [5] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. Smote: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321357, Jun 2002.
- [6] T. Coorjans, N. Ballas, C. Laurent, alar Glehre, and A. Courville. Recurrent batch normalization, 2017.
- [7] E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le. Autoaugment: Learning augmentation policies from data, 2019.
- [8] E. D. Cubuk, B. Zoph, J. Shlens, and Q. V. Le. Randaugment: Practical automated data augmentation with a reduced search space, 2019.
- [9] T. DeVries and G. W. Taylor. Dataset augmentation in feature space, 2017.
- [10] T. DeVries and G. W. Taylor. Improved regularization of convolutional neural networks with cutout, 2017.
- [11] Y. Gal and Z. Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning, 2016.
- [12] Y. Gal and Z. Ghahramani. A theoretically grounded application of dropout in recurrent neural networks, 2016.
- [13] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In Y. W. Teh and M. Titterton, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. JMLR Workshop and Conference Proceedings.
- [14] A. Gohatkar, A. Achille, and S. Soatto. Time matters in regularizing deep networks: Weight decay and data augmentation affect early learning dynamics, matter little near convergence, 2019.
- [15] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [16] B. Hanin and D. Rolnick. How to start training: The effect of initialization and architecture, 2018.
- [17] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition, 2015.
- [18] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, 2015.
- [19] D. Hendrycks, N. Mu, E. D. Cubuk, B. Zoph, J. Gilmer, and B. Lakshminarayanan. Augmix: A simple data processing method to improve robustness and uncertainty, 2020.
- [20] A. Hernandez-Garcia and P. König. Data augmentation instead of explicit regularization, 2020.
- [21] D. Ho, E. Liang, I. Stoica, P. Abbeel, and X. Chen. Population based augmentation: Efficient learning of augmentation policy schedules, 2019.
- [22] H. Inoue. Data augmentation by pairing samples for images classification, 2018.
- [23] S. Ioffe. Batch renormalization: Towards reducing minibatch dependence in batch-normalized models, 2017.
- [24] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.
- [25] G. Kang, J. Li, and D. Tao. Shakeout: A new approach to regularized deep neural network training, 2019.
- [26] P. Krähenbühl, C. Doersch, J. Donahue, and T. Darrell. Data-dependent initializations of convolutional neural networks, 2016.
- [27] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25, pages 1097–1105. Curran Associates, Inc., 2012.
- [28] J. Kukaka, V. Golkov, and D. Cremers. Regularization for deep learning: A taxonomy, 2017.
- [29] S. K. Kumar. On weight initialization in deep neural networks, 2017.
- [30] J. Lasserre, C. Bishop, and T. Minka. Principled hybrids of generative and discriminative models. volume 1, pages 87–94, 07 2006.
- [31] J. Lemley, S. Bazrafkan, and P. Corcoran. Smart augmentation learning an optimal data augmentation strategy. *IEEE Access*, 5:58585869, 2017.
- [32] S. Lim, I. Kim, T. Kim, C. Kim, and S. Kim. Fast autoaugment, 2019.
- [33] P. Luo, J. Ren, Z. Peng, R. Zhang, and J. Li. Differentiable learning-to-normalize via switchable normalization, 2019.
- [34] D. Mishkin and J. Matas. All you need is a good init, 2016.
- [35] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida. Spectral normalization for generative adversarial networks, 2018.
- [36] A. Mobiny, H. V. Nguyen, S. Moulik, N. Garg, and C. C. Wu. Dropconnect is effective in modeling uncertainty of bayesian deep networks, 2019.
- [37] T. Moon, H. Choi, H. Lee, and I. Song. Rndrop: A novel dropout for rnns in asr. In *2015 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, pages 65–70, 2015.
- [38] P. Morerio, J. Cavazza, R. Volpi, R. Vidal, and V. Murino. Curriculum dropout, 2017.
- [39] H. Nam and H.-E. Kim. Batch-instance normalization for adaptively style-invariant neural networks, 2019.
- [40] T. Q. Nguyen and J. Salazar. Transformers without tears: Improving the normalization of self-attention, 2019.
- [41] L. Perez and J. Wang. The effectiveness of data augmentation in image classification using deep learning, 2017.
- [42] A. J. Ratner, H. R. Ehrenberg, Z. Hussain, J. Dunmon, and C. R. Learning to compose domain-specific transformations for data augmentation, 2017.

- [43] S. Rifai, P. Vincent, X. Muller, X. Glorot, and Y. Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. In *ICML*, 2011.
- [44] T. Salimans and D. P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks, 2016.
- [45] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry. How does batch normalization help optimization?, 2019.
- [46] S. Semeniuta, A. Severyn, and E. Barth. Recurrent dropout without memory loss, 2016.
- [47] R. Sennrich, B. Haddow, and A. Birch. Improving neural machine translation models with monolingual data, 2016.
- [48] C. Shorten and T. M. Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1), July 2019.
- [49] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.
- [50] S. S. Talathi and A. Vartak. Improving performance of recurrent neural network with relu nonlinearity, 2016.
- [51] J. Tompson, R. Goroshin, A. Jain, Y. LeCun, and C. Bregler. Efficient object localization using convolutional networks, 2015.
- [52] T. Tran, T. Pham, G. Carneiro, L. Palmer, and I. Reid. A bayesian data augmentation approach for learning deep models. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30, pages 2797–2806. Curran Associates, Inc., 2017.
- [53] D. Ulyanov, A. Vedaldi, and V. Lempitsky. Instance normalization: The missing ingredient for fast stylization, 2017.
- [54] L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, and R. Fergus. Regularization of neural networks using dropconnect. In S. Dasgupta and D. McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1058–1066, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.
- [55] Y. Wang, X. Pan, S. Song, H. Zhang, C. Wu, and G. Huang. Implicit semantic data augmentation for deep networks, 2020.
- [56] C. Wei, S. Kakade, and T. Ma. The implicit and explicit regularization effects of dropout, 2020.
- [57] J. Wei and K. Zou. Eda: Easy data augmentation techniques for boosting performance on text classification tasks, 2019.
- [58] H. Wu and X. Gu. Towards dropout training for convolutional neural networks. *Neural Networks*, 71:110, Nov 2015.
- [59] S. Wu, H. R. Zhang, G. Valiant, and C. R. On the generalization effects of linear transformations in data augmentation, 2020.
- [60] Y. Wu and K. He. Group normalization, 2018.
- [61] Q. Xie, Z. Dai, E. Hovy, M.-T. Luong, and Q. V. Le. Unsupervised data augmentation for consistency training, 2020.
- [62] S. Yun, D. Han, S. J. Oh, S. Chun, J. Choe, and Y. Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features, 2019.
- [63] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz. mixup: Beyond empirical risk minimization, 2018.
- [64] H. Zhang, Y. N. Dauphin, and T. Ma. Fixup initialization: Residual learning without normalization, 2019.
- [65] X. Zhang, Q. Wang, J. Zhang, and Z. Zhong. Adversarial autoaugment, 2019.
- [66] Z. Zhong, L. Zheng, G. Kang, S. Li, and Y. Yang. Random erasing data augmentation, 2017.
- [67] B. Zoph, E. D. Cubuk, G. Ghiasi, T.-Y. Lin, J. Shlens, and Q. V. Le. Learning data augmentation strategies for object detection, 2019.
- [68] H. Zou and T. Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society, Series B*, 67:301–320, 2005.

A. Figures

A.1. Naive Initialization [13]

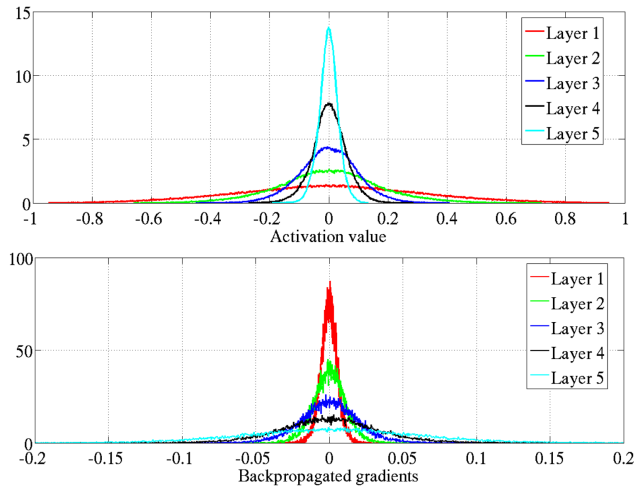


Figure 2. Normalized histogram of activation values (top) and back-propagated gradients (bottom) for a five-layer neural network with hyperbolic tangent activation and naive weight initialization via uniform distribution. Note how both values quickly saturate, the activations during the forward pass, as well as the gradients during the backward pass.

A.2. Xavier Initialization [13]

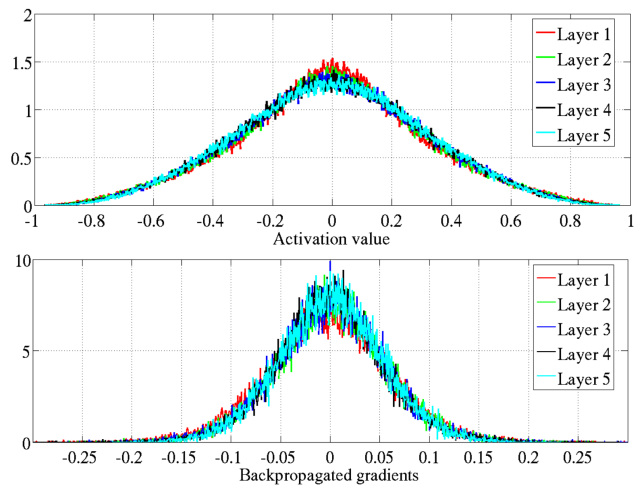


Figure 3. Normalized histograms of activations values (top) and back-propagated gradients (bottom) for a five-layer neural network with hyperbolic tangent activation and Xavier Initialization. Note, how both values remain stable during the forward pass as well as during the backward pass.

A.2.1 He Initialization Performance [18]

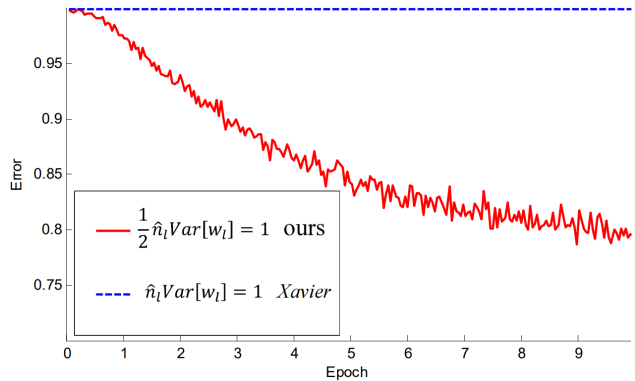


Figure 4. Validation error of He Initialization vs Xavier Initialization for a 30-layer neural network with ReLU activation functions. Note, how He Initialization is able to learn while Xavier Initialization completely stalls.

A.3. Layer Normalization Performance [2]

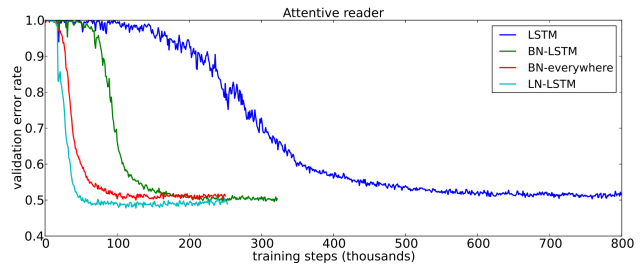


Figure 5. Validation error during training of a LSTM model. Layer Normalization achieves significantly faster optimization and better accuracy at convergence than Batch Normalization.

A.4. Group Normalization Performance [60]

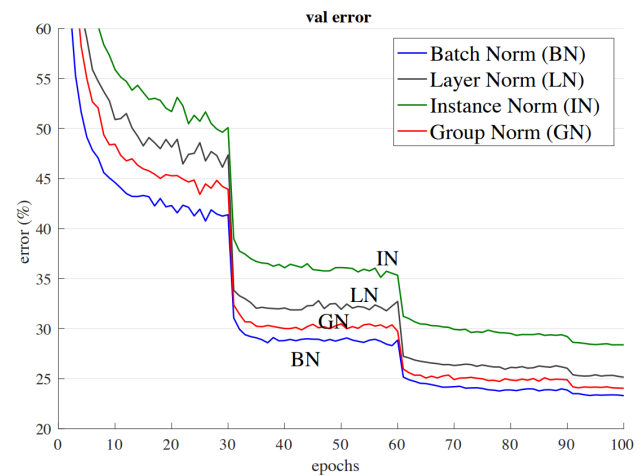


Figure 6. Validation error for different normalization methods during training a ResNet-50 model on ImageNet. For CNNs, Group Normalization achieves similar performance like Batch Normalization, while outperforming Layer and Instance Normalization.

A.5. Group Normalization Scalability [60]

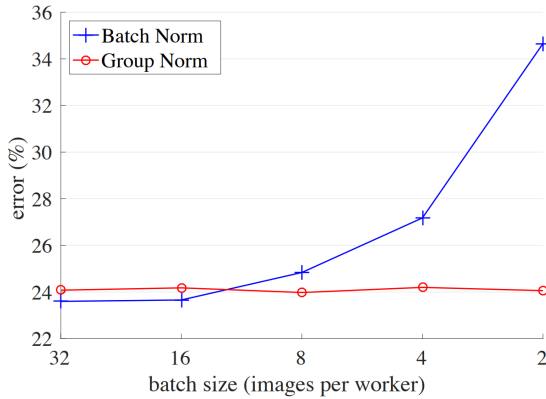


Figure 7. ImageNet validation error of a ResNet-50 model based on varying batch sizes. For large batch sizes Group Normalization works similarly well as Batch Normalization, but it scales significantly better for smaller batch sizes.

A.6. MC-DropConnect Uncertainty Estimation [36]

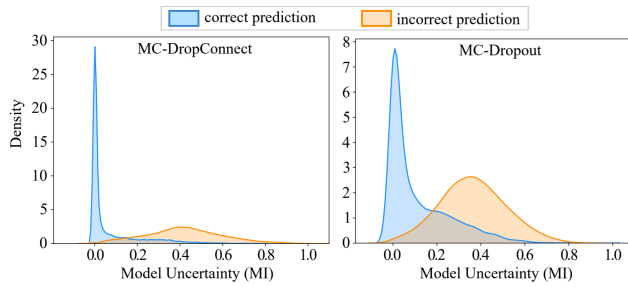


Figure 8. Distribution of the model uncertainty values for MC-DropConnect vs. MC-Dropout on CIFAR-10 test samples. Note, that MC-DropConnect produces significantly higher model uncertainty values for incorrect predictions.

A.7. Recurrent Dropout Approach [46]

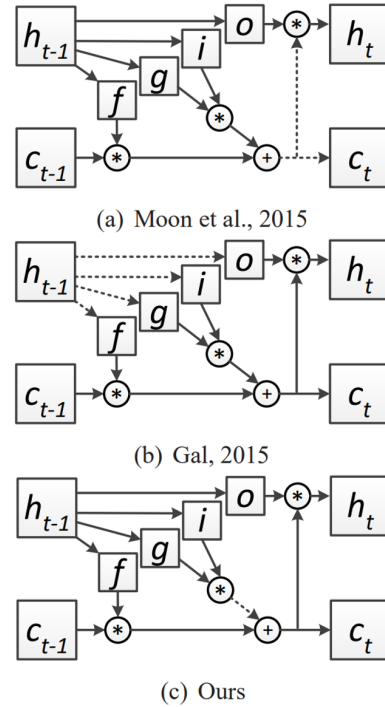


Figure 9. Three approaches of Dropout in recurrent connections of LSTM networks: (a) RNN-Drop, (b) Variational RNN Dropout and (c) Recurrent Dropout. Dashed arrows refer to dropped connections. Input connections are omitted for clarity.

A.8. SamplePairing Performance [22]

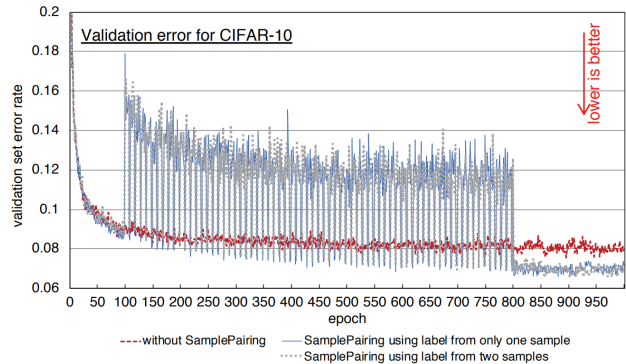


Figure 10. CIFAR-10 validation error of a simple CNN model. One can see, that performing SamplePairing with using label from only one sample performs similarly well as using a simple averaged label from two samples (basically MixUp).

B. Tables

B.1. Curriculum Dropout Performance [38]

Dataset	Architecture	Configuration (n or $n\bar{\theta}$ fixed)	Classes	Unregularized network	Dropout [13, 25]	Anti-Curriculum	Curriculum Dropout (percent boost [27] over Dropout [13, 25])
MNIST [19]	MLP	n	10	98.67	+0.38	+0.04	+0.36 (-5.3%)
	CNN-1	n	10	99.25	+0.15	-0.05	+0.18 (20.0%)
Double MNIST	CNN-2	n	55	92.48	+1.42	+0.73	+2.35 (65.5%)
	CNN-2	$n\bar{\theta}$	55	92.48	+0.87	+0.53	+1.11 (27.6%)
SVHN [21]	CNN-2	n	10	84.63	+2.35	+1.17	+2.65 (12.8%)
	CNN-2	$n\bar{\theta}$	10	84.63	+1.59	+1.51	+2.06 (29.6%)
CIFAR-10 [16]	CNN-1	n	10	73.06	+0.22	-0.68	+0.62 (182%)
CIFAR-100 [16]	CNN-1	n	100	39.70	+1.01	+0.01	+1.66 (64.4%)
Caltech-101 [9]	CNN-2	n	101	28.56	+4.21	+1.57	+4.72 (12.1%)
Caltech-256 [10]	CNN-2	n	256	14.39	+2.36	-0.22	+3.23 (36.9%)

Table 1. Comparison of Curriculum Dropout versus standard Dropout and Anti-Curriculum for image classification problems on different datasets. Note, that Curriculum Dropout frequently yields better generalization than standard Dropout.

B.2. CutMix Performance [62]


Image	ResNet-50	Mixup [48]	Cutout [3]	CutMix
				
Label	Dog 1.0	Dog 0.5 Cat 0.5	Dog 1.0	Dog 0.6 Cat 0.4
ImageNet	76.3	77.4	77.1	78.6
Cls (%)	(+0.0)	(+1.1)	(+0.8)	(+2.3)
ImageNet	46.3	45.8	46.7	47.3
Loc (%)	(+0.0)	(-0.5)	(+0.4)	(+1.0)
Pascal VOC	75.6	73.9	75.1	76.7
Det (mAP)	(+0.0)	(-1.7)	(-0.5)	(+1.1)

Table 2. Performance of MixUp, Cutout, and CutMix on ImageNet classification, ImageNet localization, and Pascal VOC 07 detection tasks compared to a ResNet-50 baseline. CutMix significantly outperforms MixUp and Cutout.

B.3. AutoAugment Performance [7]

Dataset	Model	Baseline	Cutout [12]	AutoAugment
CIFAR-10	Wide-ResNet-28-10 [67]	3.9	3.1	2.6±0.1
	Shake-Shake (26 2x32d) [17]	3.6	3.0	2.5±0.1
	Shake-Shake (26 2x96d) [17]	2.9	2.6	2.0±0.1
	Shake-Shake (26 2x112d) [17]	2.8	2.6	1.9±0.1
	AmoebaNet-B (6.128) [48]	3.0	2.1	1.8±0.1
	PyramidNet+ShakeDrop [65]	2.7	2.3	1.5 ± 0.1
Reduced CIFAR-10	Wide-ResNet-28-10 [67]	18.8	16.5	14.1±0.3
	Shake-Shake (26 2x96d) [17]	17.1	13.4	10.0 ± 0.2
CIFAR-100	Wide-ResNet-28-10 [67]	18.8	18.4	17.1±0.3
	Shake-Shake (26 2x96d) [17]	17.1	16.0	14.3±0.2
	PyramidNet+ShakeDrop [65]	14.0	12.2	10.7 ± 0.2
SVHN	Wide-ResNet-28-10 [67]	1.5	1.3	1.1
	Shake-Shake (26 2x96d) [17]	1.4	1.2	1.0
Reduced SVHN	Wide-ResNet-28-10 [67]	13.2	32.5	8.2
	Shake-Shake (26 2x96d) [17]	12.3	24.2	5.9

Table 3. Test set error rates of AutoAugment vs. Cutout and Baseline with various models and image datasets. AutoAugment consistently outperforms Cutout and achieves state-of-the-art results.

B.4. Adversarial AutoAugment Performance [65]

Model	Baseline	Cutout	AutoAugment	PBA	Our Method
Wide-ResNet-28-10	3.87	3.08	2.68	2.58	1.90±0.15
Shake-Shake (26 2x32d)	3.55	3.02	2.47	2.54	2.36±0.10
Shake-Shake (26 2x96d)	2.86	2.56	1.99	2.03	1.85±0.12
Shake-Shake (26 2x112d)	2.82	2.57	1.89	2.03	1.78±0.05
PyramidNet+ShakeDrop	2.67	2.31	1.48	1.46	1.36±0.06

Table 4. Top-1 test error of various models on CIFAR-10 dataset. Adversarial AutoAugment significantly outperforms Cutout, AutoAugment as well as Population Based Augmentation (PBA).

B.5. RandAugment Performance [8]

	baseline	PBA	Fast AA	AA	RA
CIFAR-10					
Wide-ResNet-28-2	94.9	-	-	95.9	95.8
Wide-ResNet-28-10	96.1	97.4	97.3	97.4	97.3
Shake-Shake	97.1	98.0	98.0	98.0	98.0
PyramidNet	97.3	98.5	98.3	98.5	98.5
CIFAR-100					
Wide-ResNet-28-2	75.4	-	-	78.5	78.3
Wide-ResNet-28-10	81.2	83.3	82.7	82.9	83.3
SVHN (core set)					
Wide-ResNet-28-2	96.7	-	-	98.0	98.3
Wide-ResNet-28-10	96.9	-	-	98.1	98.3
SVHN					
Wide-ResNet-28-2	98.2	-	-	98.7	98.7
Wide-ResNet-28-10	98.5	98.9	98.8	98.9	99.0

Table 5. Test accuracy of various models on different datasets. Note, that RandAugment (RA) performs similarly well or even outperforms its alternatives, namely AutoAugment (AA), Fast AutoAugment (Fast AA) and Population Based Augmentation (PBA).

B.6. Uncertainty-based Sampling Performance [59]

Dataset	Model	Baseline	AA	Fast AA	PBA	RA	Adv. AA	Ours
CIFAR-10	Wide-ResNet-28-10	96.13	97.32	97.30	97.42	97.30	98.10	97.89%(±0.03%)
	Shake-Shake (26 2x96d)	97.14	98.01	98.00	97.97	98.00	98.15	98.27%(±0.05%)
	PyramidNet+ShakeDrop	97.33	98.52	98.30	98.54	98.50	98.64	98.66%(±0.02%)
CIFAR-100	Wide-ResNet-28-10	81.20	82.91	82.70	83.27	83.30	84.51	84.54%(±0.09%)
SVHN	Wide-ResNet-28-10	96.9	98.1	-	-	98.3	-	98.3%(±0.03%)
ImageNet	ResNet-50	76.31	77.63	77.60	-	77.60	79.40	79.14%

Table 6. Test accuracy of different image classification models on various dataset. Uncertainty-based Sampling achieves similar performance like Adversarial AutoAugment (Adv. AA), thus outperforms AutoAugment (AA), Fast AutoAugment (Fast AA), Population Based Augmentation (PBA), and RandAugment (RA) in particular.