CS 168 Final Project:
A Survey on Regularization Techniques for Machine Learning Models

Rita Tlemcani
ritabt@stanford.edu

Cole Sohn
csohn@stanford.edu

Edmond Dilworth
edjd@stanford.edu

Abstract

Regularization techniques allow machine learning models to generalize better to unseen data through creating bias toward preferred models in a hypothesis class. Interested in Lectures 5 and 6, our group chose to research regularization techniques beyond those covered in lecture, specifically for limiting over-fitting in machine learning models. We write about Dropout, Data Augmentation, and Batch Normalization, explaining the techniques alongside some of their interesting applications and variations.

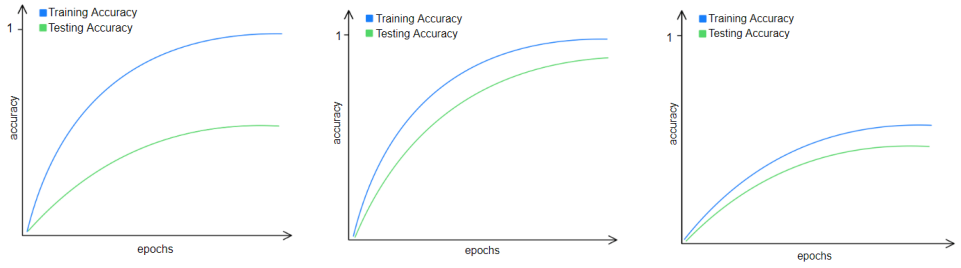
1. Introduction:

When training a machine learning model, the goal is that the model is able to correctly classify, with relatively high accuracy, data it has not seen during training. This concept is generalization and regularization is a tool used when training machine learning models to make sure that the model is generalizable.

Regularization makes a model more generalizable by preventing over-fitting. Over-fitting happens when a model learns the noise of the data too well. In this situation, the training accuracy of a classification model could be as high as 100% while the inference accuracy, during testing, could be as low as randomly choosing a label e.g. 50%. Figure 1a is an example representation of the training and testing accuracies when the trained model is overfitting. We can observe in Figure 1a that there is a large gap between the two final accuracies and that the training accuracy reaches almost 100%. When over-fitting happens, the model has high variance and cannot generalize to new data.

On the other hand, under-fitting happens when the model has high bias. Figure 1c is an example representation of a model underfitting. We can observe in Figure 1c that both accuracies are low. This usually happens when the model is not complex or expressive enough to capture enough information about the task at hand.

Ideally, when training a supervised learning model, the training and testing accuracy curves should look similar to the graph in Figure 1b. We can observe in Figure 1b that both accuracies are reaching high values at the end of around 90%. Most importantly, the gap between the training and testing isn't large meaning that the model is generalizable.

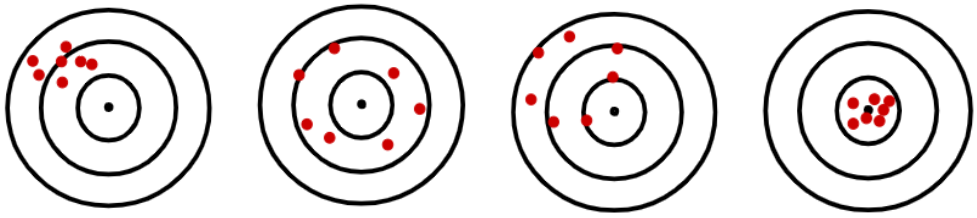


(a) Overfitting Model (b) Right Balance (c) Underfitting Model

Figure 1: Plots showing the training and testing accuracy per epoch for different types of models

When introducing regularization to a model, the bias-variance trade-off should be carefully considered. A model with low

complexity will train faster but the resulting trained model will have high bias while a more complex model could perfectly capture the training data but perform poorly on testing data with high variance. Figure 2 is a representation of the bias-variance concept. In the figure, the target value is at the center. Ideally, the trained model should have low bias and low variance and so the predicted values will be close to the target value as in Figure 2d.



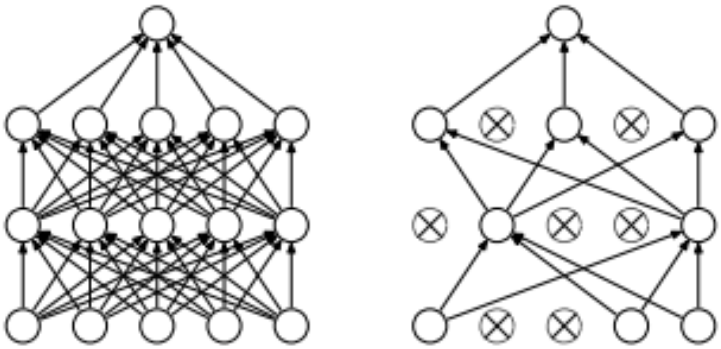
(a) High Bias Low Variance | (b) Low Bias High Variance | (c) High Bias High Variance | (d) Low Bias Low Variance

Figure 2: Darts Representation of the Bias and Variance Trade-off

The regularization techniques mentioned in this paper address the issue of over-fitting. We surveyed the current research on regularization techniques as well as the most commonly used practices and summarized our findings.

2. Dropout:

Dropout is a regularization technique for limiting over-fitting in neural networks [19]. It has been shown to be effective in a range of applications, like computer vision and speech recognition. At a high level, the technique "drops" units and their connections from the network during a training pass with some probability p. The effect is to train on a thinner network during that training, as seen in Figure 3.



(a) Standard Neural Net (b) After applying dropout.

Figure 3: Visual representation of a thinned network during Dropout [19]

.

When applying the network after training, at inference time, the units are not dropped and the outgoing weights of the node are scaled by $(1 - p)$ to ensure that the expected output from training matches the actual output of the testing model. Srivastava et al.[19] show that using dropout improves the performance of a network trained on the MNIST[14] dataset by reducing the test error from 1.6% on a standard neural network to 1.25% on a neural network with dropout layers effectively regularizing the training.

2.1. Theory

The effectiveness behind Dropout as a regularization technique can be conceptualized in a few ways. First, Dropout may limit complex co-adaptations in a neural network. During training examples, units in neural networks can update in ways that aim to fix mistakes in other units, creating complex co-adaptations and relationships that may not generalize well to other data [19]. On a simple level, by dropping out units randomly, Dropout may prevent these co-adaptations by preventing other units from relying on them. Second, since different thinned networks are examined during training, Dropout can be seen as approximately combining many different neural network architectures relatively efficiently. The training on different, thinned architectures may thus force the network to generalize better across model-choice. Third, Dropout can also be seen as simply adding noise to the neural network on account of the probabilistic dropping of units. Like other stochastic regularization techniques, this noise can prevent over-fitting to training data.

2.2. Variations

There are many possible modifications of Dropout, often with different ways of determining the probability of dropping units. As one simple example, the original dropout paper finds that input nodes tend to have a higher optimal probability of retention (a lower drop-out probability) [19]. This idea extends to an intuitive variation called "Adaptive Dropout," which allows different hidden units to be assigned different dropout probabilities [3]. Specifically, Ba et al. demonstrate that a binary belief network can be overlayed on the network to guide which units to drop [3].

Another variation called "DropConnect" randomly sets weights within the network to 0 rather than randomly removing units all-together. [23] DropConnect can be seen as a generalization of Dropout, where Dropout specifically zeros out weights to simulate removing a unit and its connections. DropConnect noticeably introduces sparsity of weights, by making previously-fully-connected layers randomly more sparse.[23] The visual distinction between DropConnect and Dropout is in Figure 4:

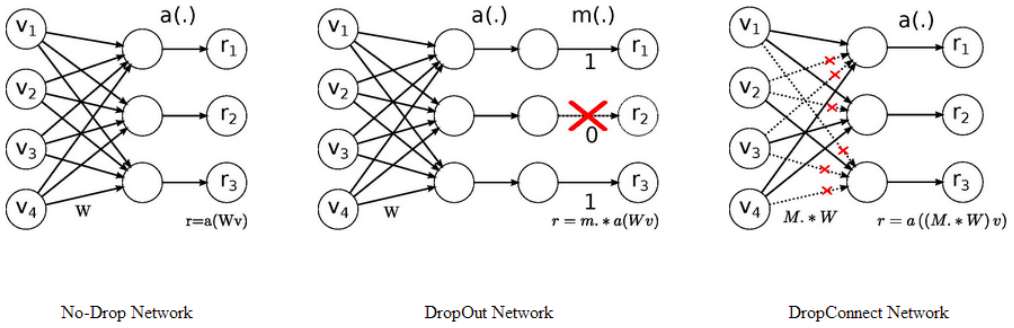


Figure 4: Visual representation of DropConnect compared to DropOut, where Dropout can be seen as randomly removing a unit (ie. not activating it), while DropConnect can be seen as randomly removing weights in the fully connected layer [1]

.

As a whole, Dropout provides a useful technique for regularizing neural networks through introducing noise, reducing co-dependencies, and combining different architectures. Dropout has inspired additional randomization techniques too, as will be discussed in the context of cutout in the section 3.

3. Data Augmentation:

Data augmentation is the process of transforming and manipulating images in the dataset to create new data. This strategy is specific to models trained on images such as convolutional neural networks (CNN). Modifying the images makes the trained model more robust to changes in the input images. These changes can be classified in different categories. In this survey we investigate two categories of data augmentation strategies, namely the image manipulation category, which is modifying the raw RGB values of the image, and the machine learning approach, which uses trained models to augment the data. By training on new additional data, the model is generalizable to data beyond the original dataset, making this approach a regularization technique.

3.1. Randomly Replacing a Portion of the Image:

Figure 5: Visualization of Random Erasing on an image[25]

This data augmentation method relies on randomly replacing a rectangle in the image with random values. This technique was simultaneously introduced by DeVries et al.[5] as "Cutout" and Zhong et al.[25] as "Random Erasing" in 2017. Both papers mention Dropout[19] as a main inspiration for their work. Instead of cancelling out random neurons in a neural network, this method cancels some pixels right at the input layer by erasing a part of the image.

Zhong et al.[25] discuss different approaches for picking the fill pixels in the erased rectangle. They investigate using random values, mean pixel value across the dataset, all 0, and all 255. For each one of the approaches, they train a ResNet18[8] network on the CIFAR-10[13] dataset. They report that using random values performs best with a test error of $4.31 \pm 0.07\%$ compared to the baseline test error, using no data augmentation, of $5.17 \pm 0.18\%$. Devries et al.[5] investigated combining cutout with other data augmentation techniques such as randomly mirroring images and randomly cropping images. They report that using a combination of Cutout, mirroring, and cropping for data augmentation on CIFAR-10 and training a ResNet18 achieves a test error rate of $4.72 \pm 0.21\%$ compared to the baseline test error of $10.63 \pm 0.26\%$.

Both papers discussed have showed that Cutout/Random Erasing works well quantitatively. Qualitatively, the main advantage of this technique is that it is simple to implement. The user only needs to define the cutout rectangle size range. This strategy also makes the trained model more robust to occlusion. Cutout or Random Erasing forces the network to learn from the whole image instead on focusing on one part. However, a disadvantage of this method is that for training on datasets such as MNIST[14], with pictures of handwritten digits, covering a random region of a number may change the label of the image. For example, covering the upper or lower half of an image of the 8 digit would change the label from 8 to 0. This limitation is dependent on the inference-time desired behavior, type of images in the dataset, and labels. This must be considered before using Cutout for data augmentation.

3.2. Adversarial Training:

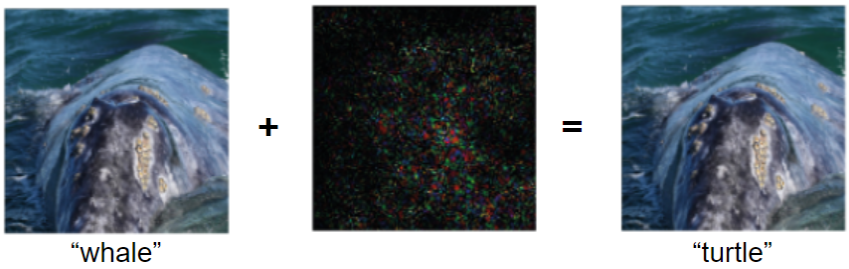


Figure 6: Generation of an adversarial example using Deepfool[15]. The original image is correctly classified as "whale" but with added perturbation generated from Deepfool the image is classified as "turtle"

Although some state-of-the-art models can achieve human-level performance on image classification tasks, small perturbations in the input images can mislead the model. The images created from these perturbations are called adversarial. Adversarial training is the best approach to make a model robust against adversarial examples[4]. Models can be trained to generate adversarial images as shown by Moosavi-Dezfooli et al.[15] with the DeepFool network. The DeepFool network efficiently computes small perturbations that will create an adversarial example. Figure 6 shows an example from the DeepFool network.

Adversarial examples can be used as training data as a data augmentation technique. This technique allows the training to tackle potential weak spots of the trained model. This also makes model safer and more reliable by determining weak decision boundaries[18].

A simple adversarial training strategy is to use a pretrained model such as DeepFool to generate adversarial examples for a label and use them for training. This makes the model more generalizable since very small added perturbations as in Figure 6 that are imperceptible to the human eye can mislead a state of the art neural network. This method can, however, be time consuming. Moosave-Dezfooli et al.[15] show that generating an adversarial example with DeepFool can take 10ms to 1100ms on a Mid-2015 MacBook Pro without CUDA depending on the dataset.

Roth et al.[16] propose an adversarial training method with less overhead. They introduced structured gradient regularization using the covariance structure of adversarial noise. They propose a data-dependent computable approximation for an intractable expectation. This method is also not dependent on the parameters of the classifier reducing the need for parametrization. They show that using this method on the MNIST[14] dataset improves the accuracy of adversarial data from 44.8% for a 9-layer CNN to 90% for the same network but with adversarial training using structured gradient regularization.

3.3. Style Transfer:

Figure 7: Example outputs of the style transfer CNN developed by Gatys et al.[7]. The first image to the left is a photograph and the two examples are the style transfer results of using The Starry Night by Vincent van Gogh and Der Schrei by Edvard Munch as reference style.

Neural style transfer (NST) is one of the most popular applications of artificial intelligence. It was first introduced by Gatys et al.[7]. NST uses two networks: a feature extractor and a transfer network. The key insight is that trained CNNs are complex feature extractors by default. Gatys et al.[7] use carefully picked layers of a pretrained model for content and style. The output image should have the same content as the input image but the style of the output image should be the same as the style of the style image used. In Figure 7, the content image is the photograph of the houses and the first style image is the Starry Night painting by Van Gogh.

Figure 8: Randomly generated style variations by Style Augmentation[12]. The first image from the left is from the Office dataset[17]. NST is applied to the image with random styles to generate the following images.

NST can be used as a data augmentation strategy. Jackson et al. introduced Style Augmentation[12] to randomize texture, contrast, and color in input data while preserving the shape and semantic content. They randomly sample a style embedding from a multivariate normal distribution instead of using a style image. Figure 8 shows an example output of Style Augmentation. The original cup, from the Office dataset[17], is the first image from the left. The styles used in the NST applied have been generated randomly. This approach makes the trained CNN robust against changes in color and shading intensities. Jackson et al. show that Style Augmentation confers accuracy gains of 6% to 17% compared to not using a data augmentation technique. They also compred Style Augmentation to color jitter (random perturbations in hue, contrast, saturation and brightness) and found an increase in accuracy of at least 4%.

NST also has regularization applications in the field of robotics in the sim2real transfer area. When training a reinforcement learning (RL) agent in simulation, transferring the learning to a real world scenario can often be challenging due to differences in textures and colors. Using NST or Style Augmentation on the images used for training a RL agent can make the agent more robust to these changes. This is a variation of the domain randomization approach introduced by Tobin et al.[20] where camera positions, lighting, and textures are randomized.

4. Batch Normalization:

Batch normalization was first introduced by Ioffe et al.[11] and it is part of a subset of regularization techniques that modify the training data to prevent over-fitting, by either applying it as a pre-processing step, or in-between layers in a deep neural network. When training a deep neural network with multiple layers, as the output of a previous layer changes, the input used to train the following layer also changes. Drastic changes in the distribution of input data to a layer can make it difficult for a layer to train. This problem is known as *the internal covariate shift*. Batch normalization standardizes the input passed to a layer to address the internal covariate shift. This has a regularizing effect as it can increase generalizability of models by adding noise to input data making models more robust to variation. In this case, the noise used to modify the training data is determined by the standard of deviation and mean of the minibatch, whose samples are often chosen randomly. Batch normalization also makes the training robust against non-optimized weight initializations.

To perform the standard batch normalization, mean and variance are calculated for a mini-batch.

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x_i \tag{1}$$

$$\sigma^2 = \frac{1}{m}\sum_{i-1}^{m}(x_i - \mu)^2 \tag{2}$$

Where m is the size of a mini-batch and x_i is the ith datapoint of the minibatch.

Next, 3 is used to normalize each dimension of the input data.

4.1. Decorrelated Batch Normalization:

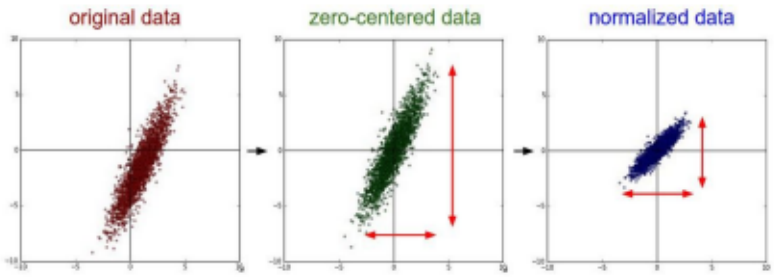


Figure 9: Visualization of how batch normalization changes the data by Fei-Fei Li[6]

The batch normalization method introduced by Ioffe et al.[11] used equation 3 to normalize the data

$$\hat{x}_i = \gamma\frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta \tag{3}$$

In equation 3, the mean and variance are used for the normalization performing regular standardization. This cannot correct for correlations in the input features as can be seen in Figure 9 from Fei-Fei Li's CS231n slides[6].

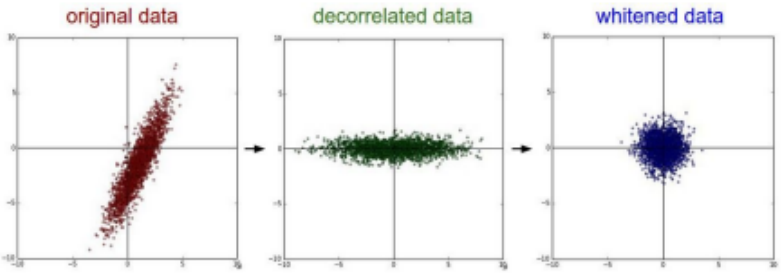


Figure 10: Visualization of how decorrelated batch normalization changes the data by Fei-Fei Li[6], Notice how the whitened data is centered around the origin.

To address this limitation, Huang et al.[10] introduced decorrelated batch normalization in which they whiten the activations of each layer within a mini-bacth using ZCA whitening. They introduce Equation 4 to perform the mini-batch whitening

$$\hat{x}_i = \Sigma^{-\frac{1}{2}}(x_i - \mu) \tag{4}$$

Where Σ is the mini-batch covariance matrix given by

$$\Sigma = \frac{1}{m}\sum_{j=1}^{m}(x_j - \mu)(x_j - \mu)^T \tag{5}$$

They addressed a previous limitation of using whitening in batch normalization by introducing an efficient backpropagation method. They proposed performing the backpropagation step of equation 4 by using an eigen decomposition to find the derivative of the inverse square root matrix. Using the full covariance matrix Σ corrects for correlations[6] effectively whitening the data as can be observed by the centered data in Figure 10. Huang et al. show that a ResNet[8] network trained on CIFAR-10[13] converges faster with decorrelated batch normalization compared to using regular batch normalization. Decorrelated batch normalization also reduces the test error by 0.98% on a ResNet20 trained on CIFAR-10. Huang et al. show that decorrelated batch normalization outperforms regular batch normalization in terms of convergence and regularization abilities.

4.2. Batch Normalization Variations:

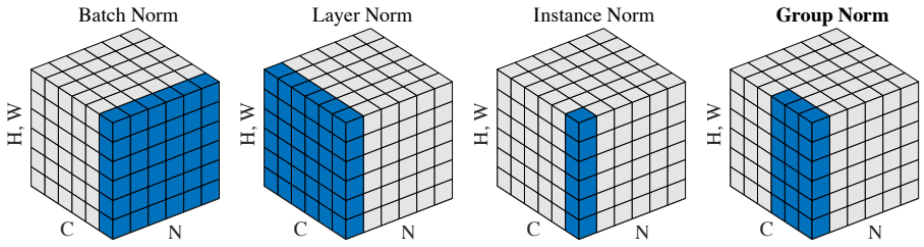


Figure 11: Visualization of different normalization approaches by Wu et al.[24].

Batch Normalization techniques have been extended in different ways to be viable for different use cases. Limitations of base batch normalization include the fact that it the computation and result are dependent on batch sizes. This can be seen by refering to variable m in 1 and 2. Therefore it is computationally different to train and test using different batch sizes.

4.2.1 Layer Norm:

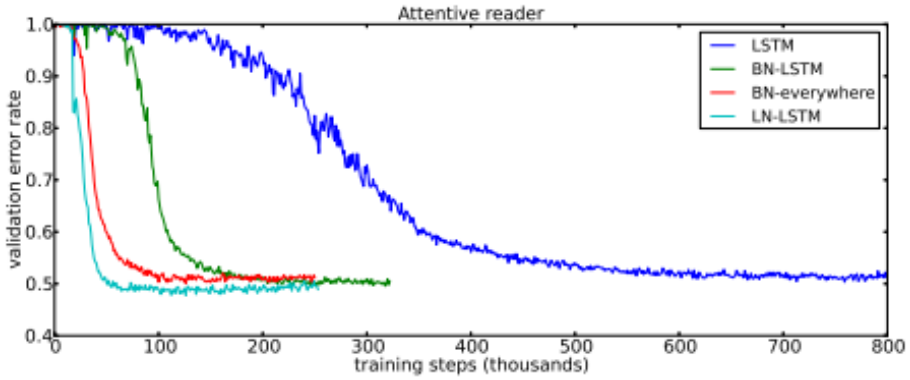


Figure 12: Validation error by epoch when training an LSTM[9] by Ba et al[2].

A limitation of batch normalization is that different computations are needed for test and training and that it does not generalize to recurrent neural networks. Ba et. al. [2] addressed these limitations and proposed layer norm. In layer normalization, the mean and variance terms are computed using the features of a single training input layer. This can be seen in Figure 11, where normalization is applied with the features of a single input instance. Ba et al. show that layer normalization leads to faster convergence in an LSTM[9] compared to regular batch normalization as can be seen in Figure 12.

4.2.2 Instance Norm:

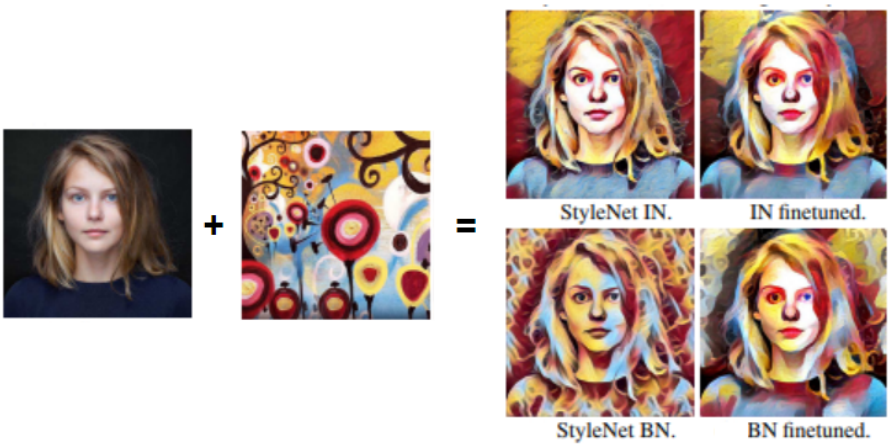


Figure 13: Visualization of how different batch normalization methods impact the output of StyleNet by Ulyanov et al.[22]

Instance Normalization was first proposed by Ulyanov et al. [21], showed how it was beneficial to normalize using mean and covariance from a single instance. They showed that using instance normalization can improve the stylized images generated from neural style transfer methods introduced in Section 3.3 compared to using regular batch normalization. Ulyanov et al.[22] used instance normalization to improve on the work of Gatys et al.[7] and showed the results in Figure 13. Using instance normalization improves the performance of the deep network generators which reduces the gap in stylization quality as can be observed in the difference between the bottom middle image and top middle image in Figure 13.

4.2.3 Group Norm:

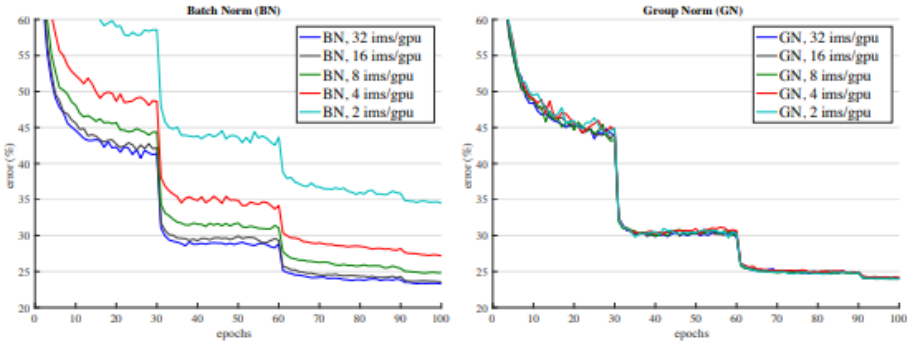


Figure 14: Visualization of how different batch sizes affect the validation errors of a ResNet50[8] that uses batch normalization and group normalization by Wu et al.[24]

Group normalization, proposed by Wu et al.[24], is an extension to batch normalizaiton through normalizing mean and variance by input channel groups with sizes independent of batch size. It addresses the limitation of batch norm being dependent on bath size. Wu et al. show that when decreasing batch sizes, batch normalization is prone to error while group normalization is not. Figure 14 shows that changing the batch size can affect the performance of a ResNet50[8] while using group normalization makes the training more robust to batch size as can be observed from the overlaying plot lines as well as a low final test error. Using group normalization can effectively reduce the need for hyperparameter tuning.

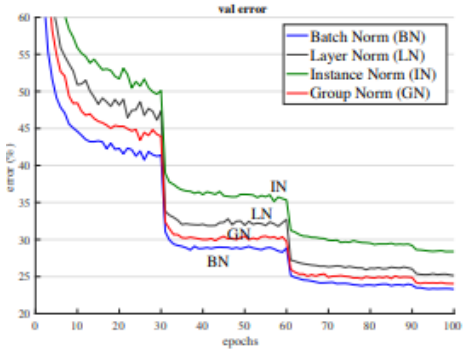


Figure 15: Comparison of validation error curves using different normalizatino techniques by Wu et al[24]

Wu et al. also show that group normalization outperforms layer norm and instance norm in Figure 15 that shows testing error per epoch after searching the batch size hyperparameter space for the best performing one. The batch size search can produce better results compared to group normalization as is shown in Figure 15.

5. Conclusion

In this survey, we provide an overview of regularization as a way to address over-fitting when training neural networks and reviewed common applicable regularization techniques such as dropout, data augmentation, and batch normalization.

As shown in this survey, regularization is an actively evolving field. There is currently no *free lunch* requlariza tion technique. Different techniques have different advantages for different model types and use cases. We have also seen in sections 3 and 4 that regularization can be achieved as in tandem with other goals such as training data-set size and model convergence speed.

While the regularization field of study is large and complex, this survey shows that picking a regularization technique is heavily tied to the desired behavior of the trained model. We have investigated two main areas of regularization strategies: modifying the dataset through data augmentation, and modifying the architecture of the model through Dropout. Both

approaches can be used simultaneously in methods such as batch normalization or combining strategies such as dropout with data augmentation.

References

[1] Regularization of neural networks using dropconnect. Technical report, NYU Center for Data Science. 3

[2] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. arXiv, 2016. 8

[3] L. J. Ba and B. Frey. Adaptive dropout for training deep neural networks. 2013. 3

[4] T. Bai, J. Luo, J. Zhao, B. Wen, and Q. Wang. Recent advances in adversarial training for adversarial robustness. In Z.-H. Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 4312–4321. International Joint Conferences on Artificial Intelligence Organization, 8 2021. Survey Track. 4

[5] T. DeVries and G. W. Taylor. Improved regularization of convolutional neural networks with cutout. arXiv, 2017. 4

[6] L. Fei-Fei, J. Johnson, and S. Yeung. Lecture 7: Training neural networks,part 2. http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture07.pdf, April 2018. 6, 7

[7] L. A. Gatys, A. S. Ecker, and M. Bethge. A neural algorithm of artistic style. In *Journal of Vision*, 2015. 5, 8

[8] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 4, 7, 9

[9] S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 11 1997. 8

[10] L. Huang, D. Yang, B. Lang, and J. Deng. Decorrelated batch normalization. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 7

[11] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In F. Bach and D. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France, 07–09 Jul 2015. PMLR. 6

[12] P. T. Jackson, A. Atapour-Abarghouei, S. Bonner, T. Breckon, and B. Obara. Style augmentation: Data augmentation via style randomization. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 5

[13] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. Technical report, University of Toronto, Toronto, Ontario, 2009. 4, 7

[14] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, volume 86, pages 2278–2324, 1998. 2, 4, 5

[15] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 4, 5

[16] K. Roth, A. Lucchi, S. Nowozin, and T. Hofmann. Adversarially robust training through structured gradient regularization. 2018. 5

[17] K. Saenko, B. Kulis, M. Fritz, and T. Darrell. Adapting visual category models to new domains. In K. Daniilidis, P. Maragos, and N. Paragios, editors, *Computer Vision – ECCV 2010*, pages 213–226, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. 5

[18] C. Shorten and T. M. Khoshgoftaar. A survey on image data augmentation for deep learning. In *Journal of Big Data*, 2019. 5

[19] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. In *Journal of Machine Learning Research*, volume 15, pages 1929–1958, 2014. 2, 3, 4

[20] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017. 6

[21] D. Ulyanov, A. Vedaldi, and V. Lempitsky. Instance normalization: The missing ingredient for fast stylization. arXiv, 2016. 8

[22] D. Ulyanov, A. Vedaldi, and V. Lempitsky. Improved texture networks: Maximizing quality and diversity in feed-forward stylization and texture synthesis. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 8

[23] L. Wan, M. Zeiler, S. Zhang, Y. LeCun, and R. Fergus. Regularization of neural networks using dropconnect. 2013. 3

[24] Y. Wu and K. He. Group normalization. arXiv, 2018. 7, 9

[25] Z. Zhong, L. Zheng, G. Kang, S. Li, and Y. Yang. Random erasing data augmentation. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2017. 4