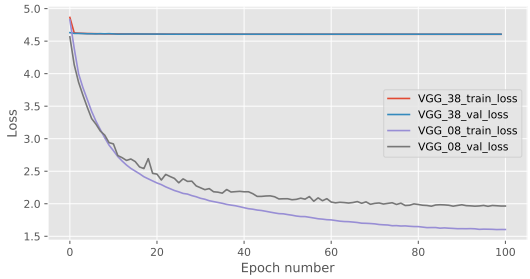
MLP CW: Batch Normalisation and Residual Connections for Vanishing/Exploding Gradients

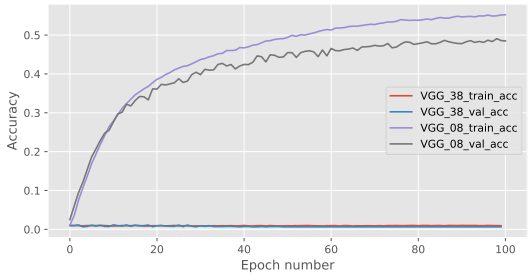
Mohamad Harah

Abstract

Deep neural networks have become the state-of-the-art in many standard computer vision problems thanks to more powerful neural networks and large labeled datasets. While very deep networks allow for better deciphering of the complex patterns in the data, training these models successfully is a challenging task due to problematic gradient flow through the layers, known as vanishing/exploding gradient problem (VGP and EGP respectively). In this report, we first analyze this problem in VGG models with 8 and 38 hidden layers on the CIFAR100 image dataset, by monitoring the gradient flow during training. We explore known solutions to this problem including batch normalization or residual connections, and explain their theory and implementation details. Our experiments show that batch normalization and residual connections effectively address the aforementioned problem and hence enable a deeper model to outperform shallower ones in the same experimental setup.



(a) Loss per epoch



(b) Accuracy per epoch

Figure 1. Training curves for VGG08 and VGG38

1. Introduction

Despite the remarkable progress of deep neural networks in image classification problems (Simonyan & Zisserman, 2014; He et al., 2016), training very deep networks is a challenging procedure. One of the major problems is the VGP, a phenomenon where gradients from the loss function shrink to zero as they backpropagate to earlier layers, hence preventing the network from updating its weights effectively. This phenomenon is prevalent and has been extensively studied in various deep network including feed-forward networks (Glorot & Bengio, 2010), RNNs (Bengio et al., 1993), and CNNs (He et al., 2016). Multiple solutions have been proposed to mitigate this problem by using weight initialization strategies (Glorot & Bengio, 2010), activation functions (Glorot & Bengio, 2010), input normalization (Bishop et al., 1995), batch normalization (Ioffe & Szegedy, 2015), and shortcut connections (He et al., 2016; Huang et al., 2017).

This report focuses on diagnosing the VGP occurred in the VGG38 model and addressing it by implementing two standard solutions. In particular, we first study the "broken" network in terms of its gradient flow, norm of gradients with respect to model weights for each layer and contrast it to ones in the healthy VGG08 to pinpoint the problem. Next, we review two standard solutions for this problem, batch normalization (BN) (Ioffe & Szegedy, 2015) and residual connections (RC) (He et al., 2016) in detail and discuss how they can address the gradient problem. We first incorporate batch normalization (denoted as VGG38+BN), residual connections (denoted as VGG38+RC), and their combination (denoted as VGG38+BN+RC) to the given VGG38 architecture. We train the resulting three configurations, and VGG08 and VGG38 models on CIFAR-100 dataset and present the results. The results show that though separate use of BN and RC does tackle the vanishing/exploding gradient problem, therefore enabling the training of the VGG38 model, the best results are obtained by combining both BN and RC.

2. Identifying training problems of a deep CNN

[ ]

Concretely, training deep neural typically involves three steps, forward pass, backward pass (or backpropagation

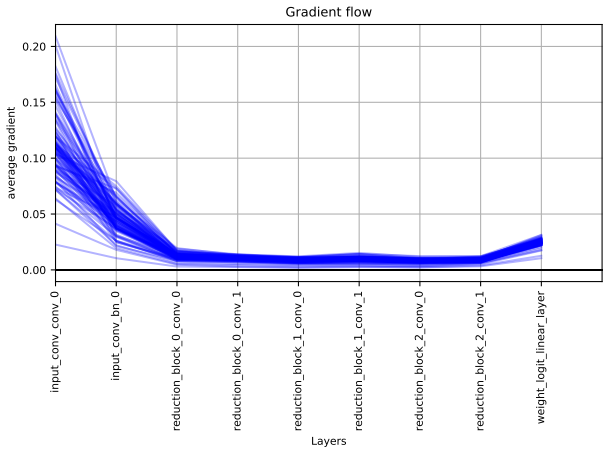


Figure 2. Gradient flow on VGG08

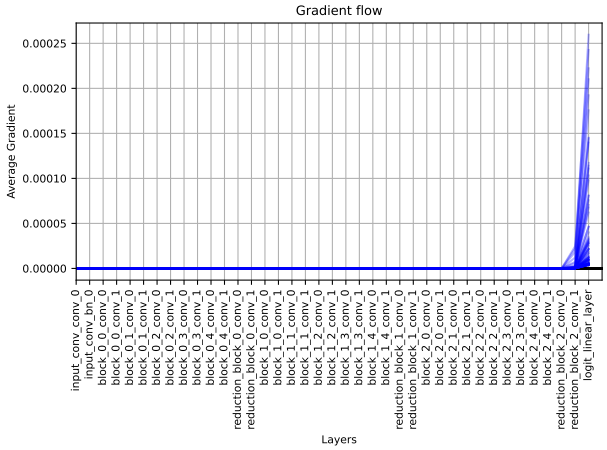


Figure 3. Gradient Flow on VGG38

algorithm (Rumelhart et al., 1986)) and weight update. The first step involves passing the input x^0 to the network and producing the network prediction and also the error value. In detail, each layer takes in the output of the previous layer and applies a non-linear transformation:

$$\boldsymbol{x}^{(l)} = f^{(l)}(\boldsymbol{x}^{(l-1)}; W^{(l)}) \tag{1}$$

where (l) denotes the l-th layer in L layer deep network, $f^{(l)}(\cdot, W^{(l)})$ is a non-linear transformation for layer l, and $W^{(l)}$) are the weights of layer l. For instance, $f^{(l)}$ is typically a convolution operation followed by an activation function in convolutional neural networks. The second step involves the backpropagation algorithm, where we calculate the gradient of an error function E (e.g. cross-entropy) for each layer's weight as follows:

$$\frac{\partial E}{\partial W^{(l)}} = \frac{\partial E}{\partial \boldsymbol{x}^{(L)}} \frac{\partial \boldsymbol{x}^{(L)}}{\partial \boldsymbol{x}^{(L-1)}} \cdots \frac{\partial \boldsymbol{x}^{(l+1)}}{\partial \boldsymbol{x}^{(l)}} \frac{\partial \boldsymbol{x}^{(l)}}{\partial W^{(l)}}. \tag{2}$$

This step includes consecutive tensor multiplications between multiple partial derivative terms. The final step involves updating model weights by using the computed $\frac{\partial E}{\partial W^{(l)}}$ with an update rule. The exact update rule depends on the optimizer.

A notorious problem for training deep neural networks is the vanishing/exploding gradient problem (Bengio et al., 1993) that typically occurs in the backpropagation step when some of partial gradient terms in Eq. 2 includes values larger or smaller than 1. In this case, due to the multiple consecutive multiplications, the gradients w.r.t. weights can get exponentially very small (close to 0) or very large (close to infinity) and prevent effective learning of network weights.

Figures 2 and 3 depict the gradient flows through VGG architectures (Simonyan & Zisserman, 2014) with 8 and 38 layers respectively, trained and evaluated for a total of 100 epochs on the CIFAR100 dataset. **[Looking at figure 1. we see that VGG 38 Accuracy is always around zero for all epochs and it's loss is around 4.6 for all epochs. This performance tells us that VGG 38 is somehow "broken". to investigate further, we try to plot Gradient flow for VGG 8 and VGG 38 in figures 2 and 3 respectively. by observing the two figures we notice that VGG 38 suffers from Vanishing Gradient Problem because the gradient of most of the layers vanish to zero (gets very close to zero) and it gets a very small insignificant values only in the last couple of layers. this will prevent the weight from changing its value. and Hence, it will cause the model learning to slow down a lot and in the worst case, which is this case, it makes the model stop learning completely.]**

.

3. Background Literature

In this section we will highlight some of the most influential papers that have been central to overcoming the VGP in deep CNNs.

Batch Normalization (Ioffe & Szegedy, 2015) BN seeks to solve the problem of internal covariate shift (ICS), when distribution of each layer's inputs changes during training, as the parameters of the previous layers change. The authors argue that without batch normalization, the distribution of each layer's inputs can vary significantly due to the stochastic nature of randomly sampling mini-batches from your training set. Layers in the network hence must continuously adapt to these high variance distributions which hinders the rate of convergence gradient-based optimizers. This optimization problem is exacerbated further with network depth due to the updating of parameters at layer l being dependent on the previous $l-1$ layers.

It is hence beneficial to embed the normalization of training data into the network architecture after work from LeCun *et al.* showed that training converges faster with this addition (LeCun et al., 2012). Through standardizing the inputs to each layer, we take a step towards achieving the fixed distributions of inputs that remove the ill effects of ICS.

Ioffe and Szegedy demonstrate the effectiveness of their technique through training an ensemble of BN networks which achieve an accuracy on the ImageNet classification task exceeding that of humans in 14 times fewer training steps than the state-of-the-art of the time. It should be noted, however, that the exact reason for BN's effectiveness is still not completely understood and it is an open research question (Santurkar et al., 2018).

Residual networks (ResNet) (He et al., 2016) One interpretation of how the VGP arises is that stacking non-linear layers between the input and output of networks makes the connection between these variables increasingly complex. This results in the gradients becoming increasingly scrambled as they are propagated back through the network and the desired mapping between input and output being lost. He *et al.* observed this on a deep 56-layer neural network counter-intuitively achieving a higher training error than a shallower 20- layer network despite higher theoretical power. Residual networks, colloquially known as ResNets, aim to alleviate this through the incorporation of skip connections that bypass the linear transformations into the network architecture. The authors argue that this new mapping is significantly easier to optimize since if an identity mapping were optimal, the network could comfortably learn to push the residual to zero rather than attempting to fit an identity mapping via a stack of nonlinear layers. They bolster their argument by successfully training ResNets with depths exceeding 1000 layers on the CIFAR10 dataset. Prior to their work, training even a 100-layer was accepted as a great challenge within the deep learning community. The addition of skip connections solves the VGP through enabling information to flow more freely throughout the network architecture without the addition of neither extra parameters, nor computational complexity.

4. Solution overview

4.1. Batch normalization

[As originally mentioned (Ioffe & Szegedy, 2015) Batch normalisation for mini-batch B can be done by making each scalar feature x_i in B have zero mean and unit variance. the mini batch mean μ_B can be denoted by:

$$\mu_B \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \quad (1)$$

where m is the size of the mini batch. while the variance is giving by:

$$\sigma_B^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i-\mu_B)^2 \quad (2)$$

each x_i can be normalised to $\hat{x}_i$ by subtracting the mean and dividing by the variance:

$$\hat{x}_i \leftarrow \frac{x_i-\mu_B}{\sqrt{\sigma_B^2+\epsilon}} \quad (3)$$

where ϵ is a small noise that can be added to prevent division by zero when the variance is zero. after that x_i can be scaled and shifted by γ and β to maintain the expressive power of the network, where γ and β are learnable parameters:

$$y_i \leftarrow \gamma\hat{x}_i + \beta \quad (4)$$

where y_i is the result from batch normalising x_i.

By batch normalizing the input throughout the network. normalised input will lie within a limited range of numbers and it cannot reach the far edges of the activation function. this will prevent activation functions from saturating and then the derivative value won't get too small or too large. thus the model learning speed will improve

To make normalisation follow a convolutional property we jointly normalize all the activations in a mini-batch so for a mini-batch of size m and feature maps of size p x q, we a mini-batch of size $m = |B| = m.pq$. Now, everything we mentioned so far applies in training time. but in test time, things are little different. Instead of directly calculating the mean and variance μ_B, σ_B^2 from equations (1) and (2), we estimate them. Typically it's done using exponentially weighted average for μ_B and σ_B^2 across mini-batches from training time. then those estimated values can be plugged in equations (3) and (4) to normalise test data.] .

4.2. Residual connections

[As shown in the original Deep Residual Learning (He et al., 2016) the example residual block consists of the following weights layer and activation functions:

$$x \rightarrow weight_1 \rightarrow ReLU \rightarrow weight_2 \rightarrow ReLU$$

by assuming $H(x)$ as the ideal predicted output that matches the ground truth and let $H(x) = F(x)+x$. where x is the input and $F(x)$ can be obtained from processing x with two weight layers as follows:

$$x \rightarrow weight_1 \rightarrow ReLU \rightarrow weight_2$$

and $F(x)$ can be obtained from:

$$F(x) + x \rightarrow ReLU$$

since $H(x) = F(x) + x$, for $H(x)$ to be ideal, $F(x)$ needs to be ideal as well. this means that the two weights layers should be capable of producing the $F(x)$ for an ideal $H(x)$. There are two kinds of residual connections: The first can be used when x and $F(x)$ have same dimensions, in this case the identity shortcuts x can be directly used.

$$y = F(x, W_i) + x$$

The second type is used when x $F(x)$ have different dimensions. In that case, a shortcut can still perform identity mapping with extra zero paddings with increased dimensions or a projection shortcut can be used to match the dimension using:

$$y = F(x, W_i) + W_s x$$

Residual connections can help solving the vanishing gradient problem because of their ability to add blocks inputs directly to the end of the block without passing it through activation functions that may potentially squash or help vanishing the gradient which will result in higher blocks derivatives overall. Residual connec-

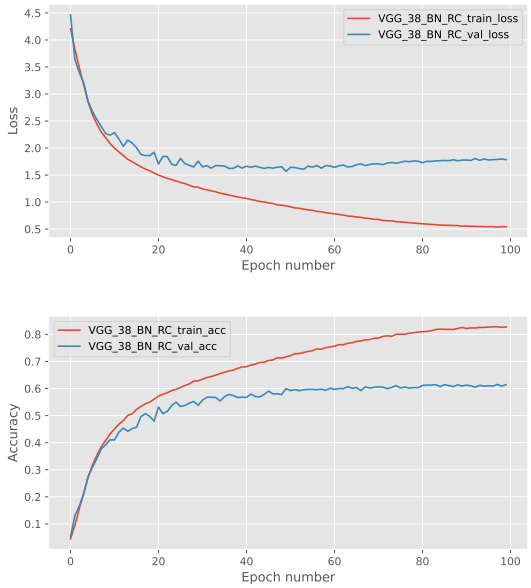


Figure 4. Training curves for VGG38 with BN & RC

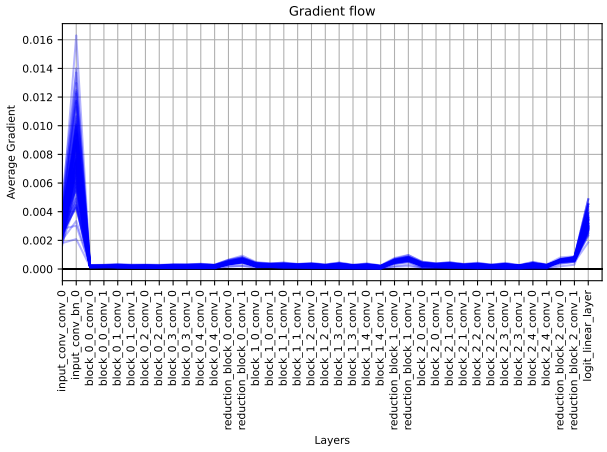


Figure 5. Gradient Flow on VGG38 with BN & RC

tions behave the same in train time and test time] .

5. Experiment Setup

[ ]

[ ]

[ ]

We conduct our experiment on the CIFAR-100 dataset (Krizhevsky et al., 2009), which consists of 60,000 32x32 colour images from 100 different classes. The number of samples per class is balanced, and the samples are split into training, validation, and test set while maintaining balanced class proportions. In total, there are 47,500; 2,500; and 10,000 instances in the training, validation, and test set, respectively. Moreover, we apply data augmentation strategies (cropping, horizontal flipping) to improve the generalization of the model.

With the goal of understanding whether BN or skip connections help fighting vanishing gradients, we first test these methods independently, before combining them in an attempt to fully exploit the depth of the VGG38 model.

All experiments are conducted using the Adam optimizer with the default learning rate (1e-3) – unless otherwise specified, cosine annealing and a batch size of 100 for 100 epochs. Additionally, training images are augmented with random cropping and horizontal flipping. Note that we do not use data augmentation at test time. These hyperparameters along with the augmentation strategy are used to produce the results shown in Figure 1.

When used, BN is applied after each convolutional layer, before the Leaky ReLU non-linearity. Similarly, the skip connections are applied from before the convolution layer to before the final activation function of the block as per Figure 2 of (He et al., 2016)

6. Results and Discussion

[Here we discuss experiments on two VGG model architectures, VGG08 the consists of 7 convulotional layers and one fully connected layer while VGG38 consist of 37 convulotional layers and one fully connected layer. two types of blocks were used to build layers. Convolutional Processing Block and Convolutional Dimensionality Reduction Block. Both of the blocks consist of two covolutional layers and a RelU activation function after each layer with the later containing an additional average pooling layer after the first RelU function and before the second convulotional layer. We run our first experiment on VGG08 with learning rate of 1e-3 on Adam optimiser, batch size of 100 and 100 epochs with no regularisation we get a model with 60k parameters and a training accuracy of 51.59% and validation accuracy of 46.86%. we run our second experiment on the deeper model VGG38 with the same number of epochs, batch size and learning rate, hoping that it will have the capacity learn further. we get a 336k parameters model with training accuracy of 0.1% and validation accuracy of 0.1%. we investigate further to see the problem behind these results and as we saw in figure 3 we discover that the model suffers from vanishing gradient problem (VGP). we try to run 5 experiments on VGG38 to solve its VGP. the first experiment consists of inserting batch normalisation layers (BN) right before each RelU activation function to normalise weights values hoping that it prevents gradient from taking too small values i.e vanishing. we run our experiment and we get a model with 339k parameters, around 3k more parameters than the base model. our VGG38 with BN scores a training accuracy of 56.15% and a validation accuracy of 48.04% which is already an improvement over VGG08. we try another solution to solve VGP on VGG38 which is creating residual connections (RC), hoping that by skipping some activation functions it

Model	LR	# Params	Train loss	Train acc	Val loss	Val acc
VGG08	1e-3	60 K	1.74	51.59	1.95	46.84
VGG38	1e-3	336 K	4.61	00.01	4.61	00.01
VGG38 BN	1e-3	339 k	1.55	56.15	1.95	48.04
VGG38 RC	1e-3	336 K	1.33	61.52	1.84	52.32
VGG38 BN + RC	1e-3	339 K	1.26	62.99	1.73	53.76
VGG38 BN	1e-2	339 K	1.70	52.28	1.99	46.72
VGG38 BN + RC	1e-2	339 K	0.54	82.72	1.78	61.44

Table 1. Experiment results (number of model parameters, Training and Validation loss and accuracy) for different combinations of VGG08, VGG38, Batch Normalisation (BN), and Residual Connections (RC), LR is learning rate.

will keep the gradient value high enough to not vanish. our VGG38 with RC results 61.52% training accuracy and 53.76% validation accuracy which is better than using BN alone. we conduct a third experiment that combines VGG38 with BN and RC since both solutions improve the performance of VGG38 Independently. our VGG38 model with BN and RC gets a slightly better results than VGG with RC, with a training accuracy of 62.99% and validation accuracy of 53.76%. we try to train VGG38 with BN but this time with learning rate of 1e-2 since according to (Ioffe & Szegedy, 2015) with BN, back propagation through a layer is unaffected by the scale of its parameters, thus higher learning rates can be used. and the resulted scores were 52.28% and 46.72% which are worse than the ones with learning rate of 1e-3, but they comparable to them and that's very decent for a higher learning rate. Additionally, according to (He et al., 2016) RC allows deep models to use faster learning rates. so we put this into test by training VGG38 with BN and RC with learning rate of 1e-2. the resulted scores were the best among all experiments. the model achieved 82.72% training accuracy and 61.44% validation accuracy. However, by looking at the training/validation loss and accuracy plots in figure 4, we see that the validation loss and accuracy starts to diverge after around 50 epochs. meaning that the generalisation gap is getting bigger. hence, the model is overfitting.

Results: From our experiments we notice that both of BN and RC work for solving VGP and to accelerate model learning. but each of them does that differently. RC had more effect on solving the VGP problem than BN since it directly contributes towards reducing the number of activations the gradient has to pass through, thus reducing it's chance to vanish. while in BN it reduces the chances of VGP indirectly by reducing internal covariant shift, which is the change in disribusion of network activations, by limiting the distrubution within fixed values or boundries.

Further Experiments that could be made: Since our best performing model is overfitting, it's worth looking into some regularisation techniques. like applying weight penalty with weight decay 1e-4, 1e-5 or 1e-6 and see how it performs. we can also consider using dropout layers. even though (Ioffe & Szegedy, 2015) had better validation performance after removing dropout layers, it might be worth looking into. Additionally, We can use BN and RC and a combination of the two with VGG08 and see how it performs with them. we could also use RC only on VGG38 with learning rate of 1e-2 since we saw that the combination of both BN and RC got better results with higher learning rate while BN alone did not. we can also experiment with even a higher learning rate of 1e-1 and see if it accelerate or improve model's performance over all.] .

7. Conclusion

[We can conclude that our solutions of adding batch normalisation and residual connections to VGG model allowed us to train a deep version of it, the VGG38, which without the proposed solutions, suffered from the vanishing gradient problem. Additionally, our solutions didn't only allow us to train a deep architecture, but they also helped regularising and improving the performance of that architecture. interesting questions that we can may for further experiments are, does our solution only regularise deep architectures that may suffer from vanishing gradient problem? or will it improve performance for shallow architectures like VGG8? how far can we scale the depth of our model with BN and RC Implemented into it? and will deeper model always produce better results if they're regularised properly? (Tan & Le, 2019) suggests that this may not necessary be the case. sometimes very deep architectures can be very inefficient compared to the accuracy improvement they provide. and they can reach a level where no further significant improvement can be achieved by scaling further.] .

References

Bengio, Yoshua, Frasconi, Paolo, and Simard, Patrice. The problem of learning long-term dependencies in recurrent networks. In *IEEE international conference on neural networks*, pp. 1183–1188. IEEE, 1993.

Bishop, Christopher M et al. *Neural networks for pattern recognition*. Oxford university press, 1995.

Glorot, Xavier and Bengio, Yoshua. Understanding the dif-

ficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256. JMLR Workshop and Conference Proceedings, 2010.

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

Huang, Gao, Liu, Zhuang, Van Der Maaten, Laurens, and Weinberger, Kilian Q. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708, 2017.

Ioffe, Sergey and Szegedy, Christian. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pp. 448–456. PMLR, 2015.

Krizhevsky, Alex, Hinton, Geoffrey, et al. Learning multiple layers of features from tiny images. 2009.

LeCun, Yann A, Bottou, Léon, Orr, Genevieve B, and Müller, Klaus-Robert. Efficient backprop. In *Neural networks: Tricks of the trade*, pp. 9–48. Springer, 2012.

Rumelhart, David E, Hinton, Geoffrey E, and Williams, Ronald J. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

Santurkar, Shibani, Tsipras, Dimitris, Ilyas, Andrew, and Mądry, Aleksander. How does batch normalization help optimization? In *Proceedings of the 32nd international conference on neural information processing systems*, pp. 2488–2498, 2018.

Simonyan, Karen and Zisserman, Andrew. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Tan, Mingxing and Le, Quoc. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pp. 6105–6114. PMLR, 2019.