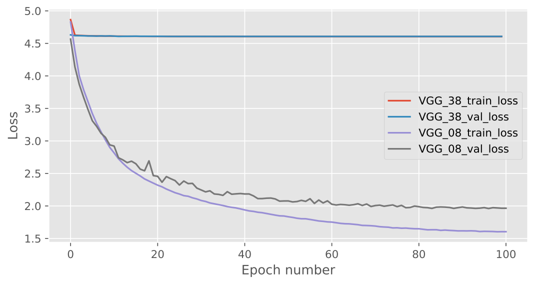
MLP Coursework 2

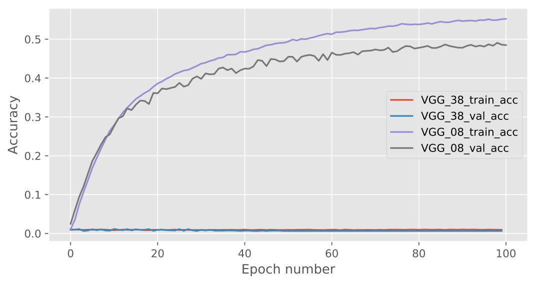
s1803764

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 feedforward 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 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

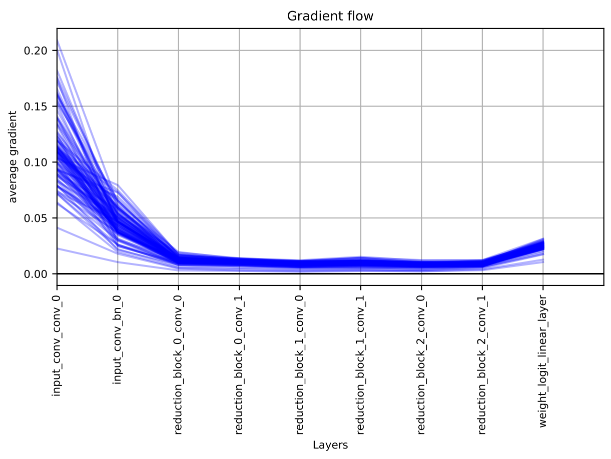


Figure 2. Gradient flow on VGG08

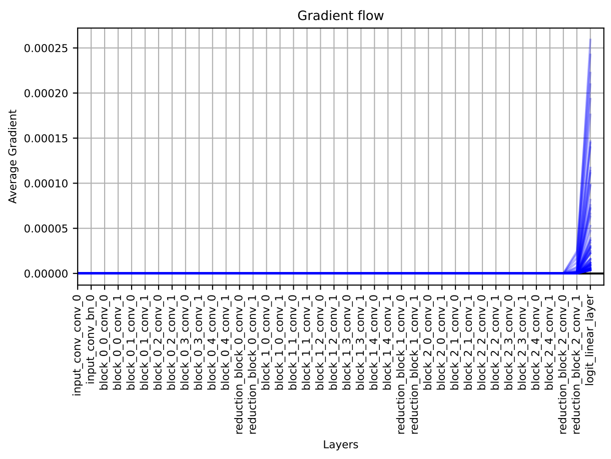


Figure 3. Gradient Flow for VGG38

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:

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

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. **[After analysing figures 1, 2, and 3 it is evident that the VGG38 model suffers from the vanishing gradient problem. This is particularly evident from figure 3 that shows the gradient flow graph for this model, where the majority of layers have consistent gradients of 0 (or very close to 0) with a spike in gradient magnitudes in the last 2 layers. This is extremely characteristic of this problem as the backpropagation algorithm shrinks the gradients exponentially more as it goes from the last to the first layer resulting in the lower layers (the first 36 layers in this case) having consistent exponentially small gradients (very close to 0), and the higher layers (the last 2 layers in this case) having highly variable gradients throughout all the epochs (Bohra, 2021). The significant changes in the magnitude of the higher layers is due to the fact that no meaningful transformations were performed on the input data (since the weights stay constant and do not converge towards an optimum for the lower layers), and thus although gradient descent keeps trying to find the optimal setting of weights for the output layer by trying lots of different settings it will never find anything useful as the input data that is applied against these weights has no meaningful information (due to the effectively "random" transformations performed on input data in the lower layers).**

This problem is especially evident when comparing the performance of the VGG38 model with the shallower VGG8 model. We would expect the VGG38 to perform better than VGG8 due to the very large increase in the number of layers, however, this is not the case. We can see from figure 1b that the train and validation accuracies for the VGG38 model stagnate around 0 throughout all the training epochs, unlike the VGG8 model where the train and validation accuracies continue to increase with the number of epochs in a logarithmic fashion. From these results it is evident that there is something wrong with the VGG38 network, however, if we had no prior knowledge about this problem how would we be able to identify it's root cause? Given we know that the VGG8 model works and that weights are what determines a model's performance we can compare the gradient flow (a measure of the changes in weights throughout the epochs) of this model with that of VGG38 in order to understand what the gradients for a working network should look like, and how the VGG38 model differs. After inspecting figures 2 and

3, we can see that the gradient flows for these models are extremely different, with the most significant differences being that: the VGG38 model has almost no variability in gradients for the input layers, the majority of gradients in the layers of the VGG38 model are 0 (or very close to 0), and the gradients in the output layers of the VGG38 model have very high variability. Given the magnitude of differences found from these gradient flow visualisations it is evident that the problem behind VGG38's poor performance lies in the way that the weights for the majority of layers in the VGG38 network remain constant throughout all the epochs and do not converge towards an optimum. Thus even if we had no prior knowledge of this problem it's diagnosis is rather straightforward when you use the right network performance and parameter setting visualisations] .

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

[Batch normalization is a technique for training very deep neural networks that standardizes the inputs to a layer for each mini-batch. This technique allows us to use much higher learning rates, be less careful about weight initialisation, and eliminate the need for smoothing techniques such as Dropout (as it offers some regularisation effect) (Brownlee, 2019).

Batch normalisation uses minibatch statistics in order to normalise the activations of each layer. It consists of adding an operation in the model just before or after the activation function of each hidden layer. This operation simply zero-centers and normalizes each input, then scales and shifts the result using two new parameter vectors per layer: one for scaling, the other for shifting. In other words, the operation lets the model learn the optimal scale and mean of each of the layer's inputs. To zero-center and normalize the inputs, the algorithm needs to estimate each input's mean and standard deviation. It does so by evaluating the mean and standard deviation of the input over the current mini-batch (hence the name "Batch Normalization") (Ioffe & Szegedy, 2015).

The Batch Normalizing Transform algorithm
Consider a mini-batch $\mathcal{B}$ of size m. Since the normalization is applied to the inputs of each activation independently, let us focus on a particular activation a^k and omit k for clarity. We have m inputs for this activation a in the mini-batch $\mathcal{B}$:

$$\mathcal{B} = \{a_1 ... a_m\} \ \textbf{where} \ a_i = w_i x$$

1. **Compute the mean and variance of $\mathcal{B}$**
   $\mu_{\mathcal{B}} = \frac{1}{M} \sum_{i=1}^{M} u_i^m$ **and** $\sigma_{\mathcal{B}}^2 = \frac{1}{M} \sum_{i=1}^{M} (a_i - \mu_{\mathcal{B}})^2$
2. **Normalise all the values in the mini-batch**

$\hat{a}_i = \frac{a_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}}$
where ϵ is a very small constant used for numerical stability

3. **Shift and scale all the normalised values in the mini-batch using learned parameters γ and β**
$z_i = \gamma_i \hat{a}_i + \beta_i = \textbf{batchNorm}(a_i)$

Once this algorithm has been completed for all the activations the γ and β parameters are then updated through gradient descent using an exponential moving average (to give more importance to the latest iterations).

So how are we going to evaluate this model now? Unlike the training phase, we may not have a full batch to feed into the model during the evaluation phase. To tackle this problem, we use the final means and standard deviations determined for each activation to normalize the testing data using the Batch Normalizing Transform algorithm (Huber, 2020)] .

4.2. Residual connections

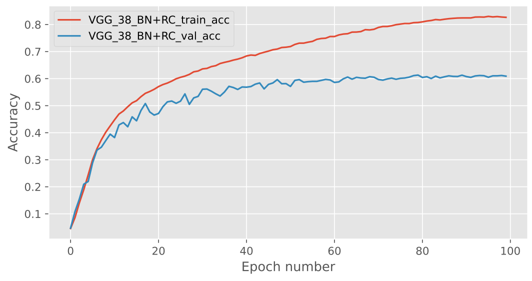
[Residual connections (or skip connections) are used to allow gradients to flow through a network directly without passing through non-linear activation functions. Non-linear activation functions, by nature of being non-linear, cause the gradients to vanish/explode for very deep neural networks. Residual connections in practice essentially form a "bus" which flows right the way through the network helping to prevent vanishing/exploding gradients by making these paths carry gradient throughout the entire network. Each block of network layers taps the values at a point along the bus, and then adds values onto the bus. This means that the blocks do affect the gradients, and conversely, affect the forward output values too (Perkins, 2018).

The principal idea behind residual connections is that is easier for deep neural networks to optimize residual mappings of the input rather than the original, unreferenced mapping of the input (He et al., 2016). These residual mappings can then be used to calculate the desired underlying mapping of the input:
Let $H(x)$ be the desired underlying mapping. We let the stacked non-linear layers in our network fit another mapping of $F(x) = H(x) - x$. Thus we can use this to recast the original mapping into $F(x) + x$ which is equivalent to $H(x)$.

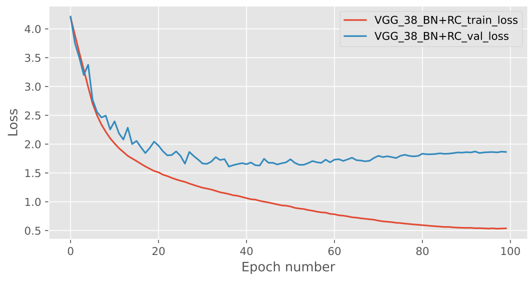
****Note since we are performing an addition operation across different layers we must ensure the dimensions of $F(x)$ and x match. This is typically ensured by just creating these residual connections in a single block of convolutional layers.***

Thus to implement this technique in training we simply create a connection from the activation output in layer n to the activation input in layer $n + 2$. This ultimately allows us to retain the activation output from layer n, skip the non-linear activation in layer $n + 1$, and add this output to the activation input in layer $n + 2$ (thus forming the residual mapping). This is then repeated all the way down the network.

Now, due to the fact these residual connections were used to alter layer inputs throughout the network we do not have to perform any changes on our model for evaluation! Due to the fact these residual connections were used in training they must be used in evaluation too to ensure the model treats inputs in the same way] .



(a) Accuracy by epoch



(b) Loss by epoch

Figure 4. Training and validation curves in terms of classification accuracy (a) and loss (b) on the CIFAR100 dataset for the best performing model: VGG38 BN+RC

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

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	1.47	57.57	1.98	47.52
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	0.57	82.02	1.79	61.8

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.

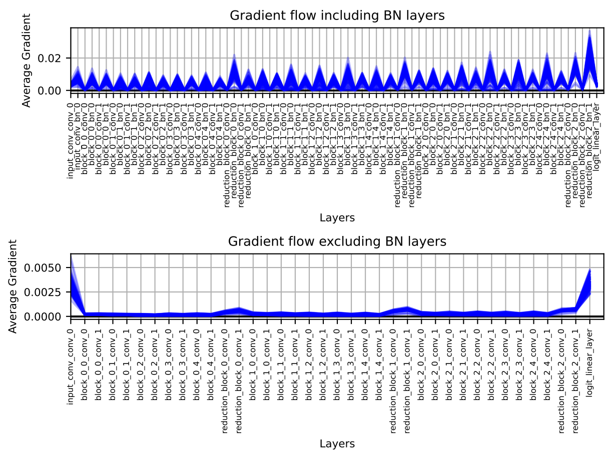


Figure 5. Gradient flow graphs for the best performing model: VGG38 BN+RC

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

[From the results in tables 1 and 2, it is evident that the combination of batch normalisation and residual connections was a great solution to the vanishing/exploding gradient problem given VGG38 BN+RC produced the best model with 61.8% validation accuracy.

This is further reinforced when comparing the gradient flow of the *working* VGG08 model in figure 2 with that of the VGG38 BN+RC model in figure 5. We can see from this comparison that the shapes for these gradient flow graphs are very similar with high gradient variability throughout all the epochs in all the layers (illustrated by the thick band of blue lines), and spikes in gradients at the input and output layers. Conversely, given our comparison of the gradient flow for the VGG08 and VGG38 models in section 2 this finding also indicates the large disparity between the gradient flow of the VGG38 model in figure 3 and the VGG38 BN+RC model in figure 5. The high similarity of gradient flows between the working models (VGG08 and VGG38 BN+RC) and the large disparities of gradient flow between the VGG38 models (VGG38 and VGG38 BN+RC) is a direct indication of how batch normalisation and residual connections can fix the vanishing/exploding gradients problem for a deep neural network.

In the future, if I were to run further experiments on our VGG08 and VGG38 models I would first like to experiment more with the configuration of batch normalisation and residual connections on our VGG38 model to get a better understanding of how well these techniques address the vanishing/exploding gradient problem in various settings. For batch normalisation I would like to experiment with different placements of batch normalisation layers w.r.t. the activation function for varying types of activation functions, and for residual connections I would like to experiment with the combinations of residual connections with varying skip sizes used in our network.

Now, using the optimal configuration for batch normalisation and residual connections found in the previous experiments to optimise our VGG38 BN+RC model even further we can still optimise the input parameters of our network. These input parameters include: the number of epochs used, the batch size, the number of filters, the number of blocks in a stage, the number of stages, and the random seed used for weight initialisation. To be able to produce the optimal solution we would need to be able to optimise these parameters in conjunction. Evidently this would be very computationally expensive to optimise given each parameter combination would need to be used to train the entire network. Thus to solve this I would make use of a technique called Gaus-

Model	LR	# Params	Val loss	Val acc	Test loss	Test acc
VGG38	1e-3	336 K	4.61	0.64	4.61	1
VGG38 BN+RC	1e-2	339 K	1.79	61.8	1.83	60.37

Table 2. Experiment evaluation results

sian Processes to minimise the amount of tests to be run. This technique observes the performances of previously observed parameter settings and uses Bayesian optimisation to choose the next parameter settings which maximises the probability of improving performance] .

7. Conclusion

[In conclusion, it is evident that when trying to train very deep neural networks both batch normalisation and residual connections serve as great techniques to address the vanishing/exploding gradient problem. This enables our deep models to outperform the shallower ones and thus negate the effect of the degradation problem. This is evident from the 59.37% test accuracy improvement the VGG38 BN+RC model gave from the original VGG38 model.

In the future, to further understand the vanishing/exploding gradient problem I would like to perform experiments using different known techniques for solving the vanishing/exploding gradient problem (in combination and alone), and test how different neural network architectures with 38 layers handle this vanishing/exploding gradient problem.

Firstly, experimenting with different known techniques for solving the vanishing/exploding gradient problem would be very useful as it would allow us to compare the effectiveness of varying techniques, and the relevance of them in different domains of neural network classification. These techniques include: using modified optimisation algorithms (such as RMSProp and gradient clipping), using modified hidden unit transfer functions (such as LSTM), using a weight filler (such as Xavier initialization), or adding "auxiliary classifiers" to middle layers.

Lastly, testing how different neural network architectures with 38 layers handle this vanishing/exploding gradient problem would be extremely useful as it would give us an insight into which architectures work best against this problem. This could prove extremely beneficial as once we have found the best performing architecture against this problem we could implement some of the known vanishing/exploding gradient problem solving techniques on this architecture. However, we must note that performing such comparisons can be rather unreliable due to the extremely high variability of setup in these different architectures, and thus in order to reliably compare these model architectures we would need to optimise each of them as far as possible. The main architectures we would test would be artificial neural networks (in particular residual networks), and recurrent neural networks] .

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.

Bohra, Yash, Jun 2021. URL https://www.analyticsvidhya.com/blog/2021/06/the-challenge-of-vanishing-exploding-gradients-in-deep-neural-networ

Brownlee, Jason. A gentle introduction to batch normalization for deep neural networks, Jan 2019. URL https://machinelearningmastery.com/batch-normalization-for-training-of-deep-neural-networks/.

Glorot, Xavier and Bengio, Yoshua. Understanding the difficulty 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.

Huber, Johann. Batch normalization in 3 levels of understanding - towards data science, Nov 2020. URL https://towardsdatascience.com/batch-normalization-in-3-levels-of-understanding-14c2da90a338#ad2e.

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.

Perkins, Hugh. What are "residual connections" in rnns?, 2018. URL https://stats.stackexchange.com/questions/321054/what-are-residual-connections-in-rnns.

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.