MLP Coursework 2: Investigating the Optimization of Convolutional Networks

s2043919

Abstract

In this report, we have two Convolutional neural networks(CNN). One is a broken CNN VGG_38 with 37 layers and another is a healthy CNN VGG_08 with 7 layers. To debug this problem, we compared the gradients flows of VGG_38 with VGG_08 and found that gradients of VGG_38 converge to 0 while VGG do not. Therefore, we concluded that training VGG_38 is more challenging because gradients vanish throughout backpropagation according to chain rule as network depth increases in VGG_38. This problem is called vanishing gradients. To solve this problem, we reviewed methods in three papers: (Ioffe & Szegedy, 2015), (He et al., 2015) and (Huang et al., 2016), which include batch normalization, residual network and dense convolutional network respectively. We chose batch normalization as the solution, which proves to be effective in solving this problem with validation and test accuracy of the baseline network reaching 50.24% and 49.42% respectively. We searched over learning rate 0.001, 0.0001, 0.005, 0.01 and batch size 0.005, 0.01, 0.045 to explore a potentially better solution. The baseline network has a learning rate of 0.001 and a batch size of 100. We found that baseline network is actually the optimal solution with stable training and testing accuracy. Moreover, we found that both too large or too small learning rate and batch size bring poor generalization in VGG_38.

1. Introduction

CNN has become one of the most powerful architectures in computer vision problems. When training CNN, we might expect that deeper CNN produces better results than shallow one as deeper model is more complex and might portray more features of images. However, it is simply not the case. We have two models in our problem setting. One is VGG_38 with 37 layer while another is VGG_08 with only 7 layers. From their loss and accuracy curve in the figure shown in the material, VGG_38 turns out to be a broken network which gets stuck and stops optimization after training several epochs. On the contrary, VGG_08 is a healthy model which can be optimized well. To debug this problem, we plotted the gradient flows of VGG_08 and VGG_38 and compared them. We found that gradients in VGG_38 converge to 0 as training more epochs. For VGG_08, the gradients do not have this problem. This problem is vanishing gradients, which typically exists in deep neural network. Vanishing gradients happens because gradients decrease exponentially to 0 throughout backpropagation based on chain rule in deep network. To solve the problem, we reviewed three papers (Ioffe & Szegedy, 2015), (He et al., 2015) and (Huang et al., 2016), all of which concerns solving the vanishing gradients problem. The proposed solutions include batch normalization, residual network and dense convolutional network. Batch normalization works by making distribution of inputs stable. For residual network, it works by adopting the strategy of 'skipping layer' and forcing the deep network to learn identity functions. For densely convolutional network, it works by connecting each layer to all the subsequent layers. We adopted batch normalization as our solution. Batch normalization was added before each activation function in convolutional layers. The result shows that validation and test accuracy of batch-normalized baseline network reaches 50.24% and 49.42% respectively without overfitting, which proves that batch normalization is successful in solving vanishing gradients. To find a potentially better solution, we searched over learning rate and batch size. Initially, baseline network has learning rate of 0.001 and batch size of 100. We experimented on other networks with learning rate being 0.0001, 0.005, 0.01, 0.045 and batch size being 50, 200. We found that baseline network performs best. Moreover, we got an empirical conclusion that too large or too small learning rate and batch size both achieved bad performance in this network. Finally, we tested the optimal model on multiple seeds(0,10,20) and all of them worked effectively with test accuracy reaching 49.42%, 48.16% and 47.89% respectively.

To conclude, batch normalization is particularly useful in solving vanishing gradients. After experiments, we improved the validation and test accuracy from 0 to 50.24% and 49.42% without overfitting.

2. Identifying training problems of a deep CNN

The loss of VGG_38 stops decreasing after several epochs of training while VGG_08 proves to be a healthy model with normal loss and accuracy curve. The problem in VGG_38 is vanishing gradients, which refers to a phenomenon that gradients become too small to optimize the model as depth of network increases. Gradients are derived

by backpropagation. By chain rule, the derivatives of the network are moving from layer by layer down to the initial one. Each layer multiplies the derivatives computed from the next layer to its own derivatives and passes the them forward. For shallow networks, this is not a problem because the layer structure is not complex enough. However, for deep networks with more layers involved, this problem gets amplified as the gradients might decrease exponentially to 0 if local error gradient of each layer is smaller than 1 during backpropagation, which leads to problem of vanishing gradients.

To visualize the problem quantitatively, we computed the absolute average gradient of each layer in each epoch in VGG_38 and VGG_08 and concatenated them to plot gradient flows, which are shown in figure 1. In VGG_38, the gradients of all layers converge to 0 as epoch number increases. In contrast, VGG_08 is totally different and does not have this problem.

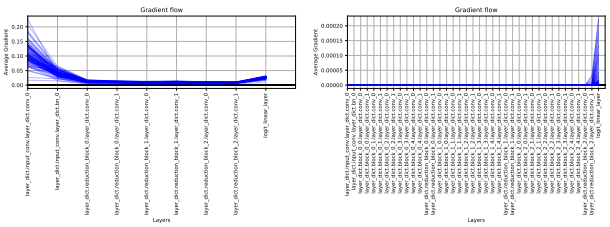


Figure 1. Gradient Flow in each layer for VGG_08 VGG_38 network

Gradient flows in VGG_38 are closely related to vanishing gradients problems. We can see that all gradients vanish in the last epoch, which is exactly the problem of vanishing gradients. According to gradient descent learning algorithm $weight = weight - learning\_rate \times gradient$, we can infer that if gradients converge to 0, weights remain almost unchanged, which makes the model unable to optimize. Therefore, the loss curve of VGG_38 converges after several epochs. As the weights in all layers stop optimization, the model has poor ability to represent features, which makes both training and validation accuracy close to zero.

3. Background Literature

We choose Ioffe & Szegedy (2015), He et al. (2015) and Huang et al. (2016) to review.

First paper is Ioffe & Szegedy (2015). This paper addresses the problem described by the author as internal covariate shift, which refers to a phenomenon that distribution of each layer's inputs changes as parameters in previous layers change when training deep neural network. This makes the optimization slow because it requires lower learning rate as well as accurate parameter initialization and also makes the model likely to get stuck in the saturated regime, which is amplified as network depth increase. Actually, vanishing gradients can be considered as a consequence of internal covariate shift, which is clarified in the paper. The paper proposes a solution which divides training set into mini-batches and performs batch normalization on each of them. Batch normalization works by making each dimension of inputs in each batch have mean of zero and variance of 1, which significantly reduces internal covariate shift by stabilizing distribution of inputs. Batch normalization is proved to tolerate higher learning rates and regularize the model without dropout. The paper applied batch normalization to an image classification model and achieved the same accuracy with faster training by changing training parameters. Moreover, using an ensemble of batch-normalized networks challenges benchmarks. In conclusion, batch normalization solves internal covariate shift by stabilizing distribution of inputs throughout training. Batch normalization is more tolerant to high learning rates and can regularize the model without using dropout. However, there are still some confusing part that could be clarified. For example, the author did not give a detailed explanation of why batch normalization could regularize model.

The second paper is He et al. (2015). This paper addresses the problem of degradation, which refers to a phenomenon that accuracy gets saturated and then degrades rapidly as network depth increases. To further elaborate this problem, considering a shallow model and a deep model, there is a solution in which layers can be copied from shallow model to the deep model and the additional layers use identity mappings. This indicates that the accuracy of deep model should be higher or at least equal to the shallow counterpart. However, the results of experiments turn out to be the opposite not because of overfitting. This paper introduces deep residual learning framework as the solution. Instead of sticking to fitting the original model with optimal parameters, deep residual learning adopts a strategy of 'skipping layers'. Suppose $H(x)$ to be the desired underlying mapping with a few stacked layers to be fit and x refers to the inputs of the first layer in these stacked layers. To optimize $H(x)$, an alternative way would be optimizing a residual function $F(x) = H(x) - x$, which is easier to optimize. Then the original function becomes $H(x) = F(x) + x$. Experiments were performed on ImageNet classification and CIFAR-10 set and both of them achieved good performance. We can infer that residual learning can address vanishing gradients because residual network can be considered as a way to force the model to learn identity function whose gradient is 1, which prevents vanishing gradients to a large extent. In conclusion, residual learning addresses degradation by skipping layers to make deep network easier to learn identity mappings. However, there is still something that the author did not clarify. For example, why optimizing residual mappings is easier than optimizing the original mappings. The author made this conclusion empirically. Moreover, the author did not go enough depth about how residual network solves vanishing gradients.

The third paper is (Huang et al., 2016). This paper addresses vanishing gradients by introducing dense convolutional network(DenseNet), which connects each layer to all the subsequent layers. In detail, each layer in DenseNet

takes feature maps from all the preceding layers and concatenates them together, then passes its output to all the subsequent layers following the feed-forward nature. Due to the architecture of DenseNet, it requires fewer layers to train a deep network. As a result, there are fewer paramters to train than traditional convolutional network because it is unnecessary to learn redundant feature maps. The key benefits of DenseNet include alleviating vanishing gradients problem, strengthening feature propagation, encouraging feature reuse and reducing the number of parameters. The reason why DenseNet works in solving vanishing gradients is that DenseNet connects each layer to its subsequent layers and enables the each layer to have direct access to gradients from loss function as well as the original inputs, which makes the gradients more connected with the overall performance of the network. The paper evaluated DenseNet on CIFAR-10, CIFAR-100, SVHN and ImageNet and all of them achieved comparable accuracy with fewer training parameters. In conclusion, DenseNet works by connecting each layer directly to all its subsequent layers. DenseNet can alleviate vanishing gradients, strengthen feature propagation, encourage feature reuse and reduce the number of parameters. However, one drawback of this paper is that it did not cover more depth and details in terms of how DenseNet solves vanishing gradients.

4. Solution Overview

We adopted batch normalization as the solution because vanishing gradients problem is essentially caused by internal covariate shift and batch normalization is focused on solving this problem. Although other two solutions in (He et al., 2015) and (Huang et al., 2016) might also work, the reason why we did not select them is either due to the difficulty of implementation or lack of dedication to vanishing gradients. Generally speaking, batch normalization solves vanishing gradients problem by stabilizing distribution of inputs. Technically speaking, Batch normalization works by making each dimension of inputs in each mini-batch have mean of zero and variance of 1. Suppose a layer with d-dimensional inputs $x = (x_1....x_d)$, dimension k can be normalized to be $\frac{x_k - \mu_\beta}{\sqrt{\sigma_\beta^2}}$, in which μ_β and σ_β refer to the mini-batch mean and mini-batch variance respectively. In order to preserve the model's representation ability, two learnable parameters γ_k and β_k are adopted to make the normalized value to be $\gamma_k \widehat{x_k} + \beta_k$, in which $\widehat{x_k}$ refers to the normalized value. Actually, setting $\gamma_k = \sqrt{\sigma_\beta^2}$ and $\beta_k = \mu_\beta$ can recover the initial functions if this is optimal. However, batch normalization is not needed during validation or testing because the output should only depend on the input when predicting. The solution is to compute the overall mean and variance composed with scale and shift throughout the mini-batches for each dimension.

The reasons that batch normalization works are as follows. Firstly, stable distribution makes the network less likely to get stuck in saturated regime because internal covariate shift (Ioffe & Szegedy (2015)) is caused by frequent shifts of inner parameters, which leads to vanishing gradients. In this aspect, training would be largely improved and training loss as well as validation loss will shift from getting sudden stuck to decreasing continuously until convergence.

Another reason is that batch normalization tolerates higher learning rates. In traditional neural networks, higher learning rates might lead to vanishing or exploding gradients and optimization getting stuck in a saturated regime. Actually, increase of learning rate can be considered as a scale-up of parameters. the paper Ioffe & Szegedy (2015) proves that batch normalization makes training more robust and resilient to changes of parameter scale. For a scalar α,

$$BN(Wu) = BN((\alpha W)u)$$

. We can also infer that

$$\frac{\partial BN((\alpha W)u)}{\partial u} = \frac{\partial BN(Wu)}{\partial u}, \frac{\partial BN((\alpha W)u)}{\partial (\alpha W)} = \frac{1}{\alpha}\frac{\partial BN(Wu)}{\partial W}$$

From the formulas above, we can conclude that scale-up of parameters does not affect back-propagation and larger weights actually make gradients smaller, which proves our assumption. Moreover, we could infer empirically that gradients flows would be more stable compared to traditional network because gradients of batch-normalized network are more resilient to learning rates.

Moreover, batch normalization also regularizes network without dropout because a training example is deeply connected with other examples in a mini-batch. Besides, the output of training network is not deterministic for a given example depending on other examples. Empirically, this could largely reduce overfitting.

The pseudo codes of batch normalization, training and inference of batch-normalized network are presented in algorithm 1, 2 and 3.

Algorithm 1 Batch Normalization

Input: Inputs x with m dimensions over a mini-batch: $\{x_1...m\}$;
Learnable parameters for scale and shift: γ, β
Output: y_i =BN(x_k)
$\mu_\beta \longleftarrow \frac{1}{m}\sum_{k=1}^{m} x_k$ ▷ mean of mini-batch
$\sigma_\beta^2 \longleftarrow \frac{1}{m}\sum_{k=1}^{m}(x_k - \mu_\beta)^2$. ▷ variance of mini-batch
$\widehat{x_k} \longleftarrow \frac{x_k - \mu_\beta}{\sqrt{(\sigma_\beta)^2 + \xi}}$ ▷ normalization
$\gamma_k \longleftarrow$BN(x_k) $= \gamma\widehat{x_k} + \beta$ ▷ scale and shift

Algorithm 2 Training Batch Normalization Network

Input: Network N to be trained.
Inputs of each activation$x_{(1...K)}$
Output: Trained batch-normalized network N_{tr}
$N_{tr} \longleftarrow N$.
for $k \leftarrow 1$ to K **do**
Add batch normalization $y_{(k)}$ on N_{tr} (Algorithm 1).
Replace $x_{(k)}$ with $y_{(k)}$.
end for

Algorithm 3 Inference with Batch-Normalized Network

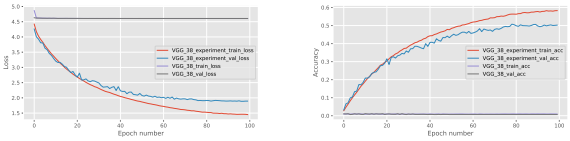
Input: Already trained Batch-normalized network N_{tr}.
Output: Batch-normalized inference network N_{inf}
$N_{tr} \longleftarrow N_{inf}$
for $k \leftarrow 1$ to K **do**
$x \longleftarrow$ inputs in kth activation.
$\mu_\beta \longleftarrow$ the means in kth activation.
$\sigma^2_\beta \longleftarrow$ the variances in kth activation.
$E[x] \longleftarrow E_\beta[\mu_\beta]$
$Var[x] \longleftarrow \frac{m}{m-1} E_\beta[\sigma^2_\beta]$
In N_{inf}, replace y to be $y = \frac{\gamma}{\sqrt{Var[x]+\xi}} x + (\beta - \frac{\gamma E[x]}{\sqrt{Var[x]+\xi}})$
end for

5. Experiments

CIFAR100 is a dataset consisting of 60000 32×32 coloured images with 100 classes, each of which has 600 images. In this experiment, the dataset is divided into a training set with 47500 images, a validation set with 2500 images and a testing set with 10000 images. The training set is grouped into 475 mini-batches and batch size is 100. The network that this experiment is based on is VGG_38. There are 1 EntryConvolutionalBlock plus 3 stages of blocks and 1 linear layer in VGG_38. An EntryConvolutionalBlock consists of 1 convolutional layer with batch normalizaiton. Among the 3 stages of blocks, each stage has 5 ConvolutionalProcessingBlocks and 1 ConvolutionalDimensionalityReductionBlock. A ConvolutionalProcessingBlock consists of 2 convolutional layers with leaky_relu as activation function. ConvolutionalDimensionalityReductionBlock includes 2 convolutional layers, with an average pooling layer in the middle and leaky_relu as activation function. To sum up, VGG_38 consists of 37 convolutional layers plus 1 linear layer. The size of kernel is 3×3 and number of filters is 32. Number of input channels is always 32 but the size of convolutions shift from 32 to 16, 8, 4 and finally to 1. The dimension of outputs is (100,32) as there are 100 classes and 32 channels. The total number of parameters is 338500. Some initial parameters including learning rate, batch size , seed and weight_decay_coefficient are 0.001, 100, 0, 0 respectively.

The goal of the experiment is to explore to what extent batch normalization solves vanishing gradients and tune hyper-parameters to find a relatively effective solution in this problem set. Firstly, we created two new blocks named as ConvolutionalProcessingWithBatchNormBlock and ConvolutionalDimensionalityReductionWithBatchNormBlock originated from ConvolutionalProcessingBlock and ConvolutionalDimensionalityReductionBlock respectively and added batch normalization before activation functions for the two blocks. Then replace the old blocks with the new blocks that we created.

Firstly, we would like to see whether batch normalization solves vanishing gradients. We run the baseline network without changing any parameters. The error and accuracy curves are presented in figure 2. The figure of gradients flows is presented in figure 3.



(a) Error curve (b) Accuracy curve

Figure 2. Loss and accuracy curve of VGG_38 before and after batch normalization

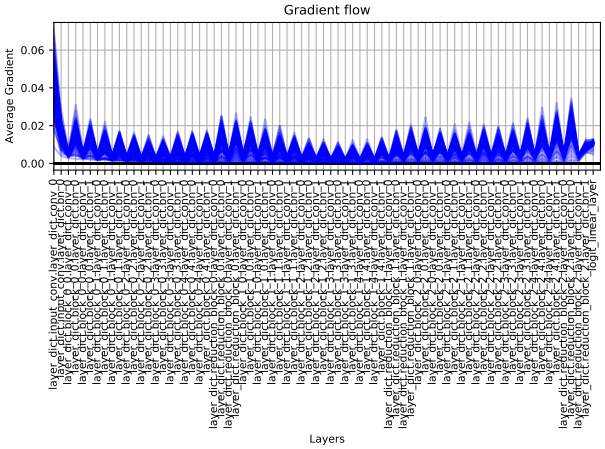


Figure 3. Gradients flows of VGG_38

Afterwards, we do a hyper-parameter search over learning rate. We evaluated on the following networks, all of which are trained on training set and evaluated on validation set.

BN-Baseline: VGG_38 batch-normalized network.

BN-x1/10: The initial learning rate is decreased by a factor of 10 to 0.0001.

BN-x5: The initial learning rate is increased by a factor of 5 to 0.05.

BN-x10: The initial learning rate is increased by a factor of 10 to 0.01.

BN-x45: The initial learning rate is increased by a factor of 45 to 0.45.

The training and validation accuracy is presented in figure 4. Table 1 and 2 show the loss and accuracy.

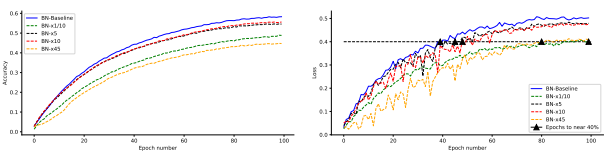


Figure 4. Training and validation accuracy of batch-normalized VGG_38 models with different learning rates, vs. epoch number.

Models	Training loss	Validation loss	Test loss
BN-Baseline	1.45	1.89	1.90
BN-x1/10	1.85	2.30	2.24
BN-x5	1.61	1.96	1.96
BN-x10	1.56	1.96	1.98
BN-x45	1.97	2.18	2.17

Table 1. Training, validation and test loss of batch-normalized VGG_38 models with different learning rates.

Models	Training accuracy	Validation accuracy/epochs to near 40%	Test accuracy
BN-Baseline	58.29%	50.24%/39	49.42%
BN-x1/10	48.83%	40.12%/45	41.53%
BN-x5	54.55%	47.92%/48	48.39%
BN-x10	55.62%	47.48%/80	48.05%
BN-x45	44.71%	40.92%/99	40.83%

Table 2. Training, validation and test accuracy of batch-normalized VGG_38 models with different learning rates.

Then we want to discover how batch size affects the performance. The networks are presented below.

BN-Baseline: VGG_38 batch-normalized network.

BN-x1/2: The initial batch size is decreased by a factor of 1/2 to 50.

BN-x2: The initial batch size is increased by a factor of 2 to 200.

The figure of training and validation accuracy is presented in figure 5. The table 3 and 4 show the loss and accuracy.

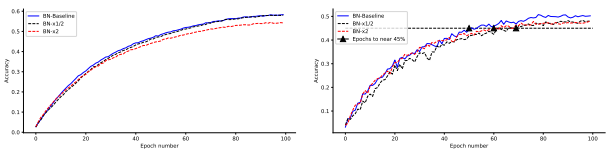


Figure 5. Training and validation accuracy of batch-normalized VGG_38 models with different batch sizes, vs. epoch number.

Afterwards, we adopted the baseline network and changed seeds to see the performance. The networks are presented below.

BN-Baseline: VGG_38 batch-normalized network.

BN-seed=10: VGG_38 batch-normalized network with seed being 10.

Models	Training loss	Validation loss	Test loss
BN-Baseline	1.45	1.89	1.90
BN-x1/2	1.63	1.94	1.93
BN-x2	1.45	1.97	1.96

Table 3. Training, validation and test loss of batch-normalized VGG_38 models with different learning rates.

Models	Training accuracy	Validation accuracy/epochs to near 45%	Test accuracy
BN-Baseline	58.29%	50.24%/50	49.42%
BN-x1/2	54.20%	48.08%/60	48.16%
BN-x2	58.09%	48.04%/69	47.89%

Table 4. Training, validation and test accuracy of batch-normalized VGG_38 models with different learning rates.

BN-seed=20: VGG_38 batch-normalized network with seed being 20.

The figure of training and validation accuracy is presented in 5. Table 5 and 6 show the loss and the accuracy.

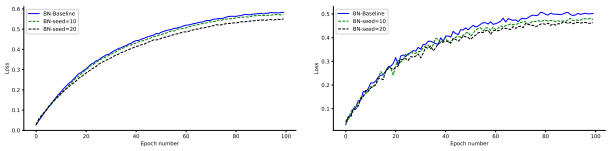


Figure 6. Training and validation accuracy of batch-normalized VGG_38 models with different seeds, vs. epoch number.

Models	Training loss	Validation loss	Test loss
BN-Baseline	1.45	1.89	1.90
BN-seed=10	1.49	1.94	1.96
BN-seed=20	1.60	1.98	1.98

Table 5. Training, validation and test loss of batch-normalized VGG_38 models with different learning rates.

6. Discussion

From experimental results shown in figure 2 and 3, We observe that vanishing gradients is solved completely as loss and accuracy return to normal without overfitting and

Models	Training accuracy	Validation accuracy	Test accuracy
BN-Baseline	58.29%	50.24%	49.42%
BN-seed=10	57.31%	47.60%	48.65%
BN-seed=20	55.07%	46.32%	47.39%

Table 6. Training, validation and test accuracy of batch-normalized VGG_38 models with different learning rates.

gradients flows fluctuate regularly, which proves that batch normalization is useful. Actually, the normalized baseline network has performed quite well. The motivation for the following experiments is to search over learning rate and batch size to find whether there exists a better solution than the baseline.

First, we search over learning rate. From figure 4 and table 1 and 2, we can see that Baseline network performs best with validation accuracy reaching 50.24% and test accuracy reaching 49.42%. BN-x1/10 and BN-x45 perform the worst with similar accuracy. BN-x5 and BN-x10 both perform well but no better than BN-Basline. Moreover, baseline network takes fewest epochs to reach a validation accuracy of 40%. The results seem to be inconsistent with the results in (Ioffe & Szegedy, 2015), which proves that increasing learning rate could achieve a training speedup without side effects. The reason is that in this problem set, we adopt the learning rate scheduler in pytorch called CosineAnnealingLR, which is a periodic learning rate scheduler based on Cosine function. The learning rate ranges from the learning rate that we adopted to a minimum value, which is 0.00002 by default. Hence the conclusion from (Ioffe & Szegedy, 2015) can not be applied to our problem. From our experimental results, We can gain an empirical knowledge that too small learning or too large learning rate make the model generalize badly. Ultimately, we would prefer the initial learning rate 0.001.

Then we search over batch size. From figure 5, we can observe that baseline network wins again. BN-x1/2 and BN-x2 both achieved similar accuracy, which is lower than BN-Baseline. The reason is that for smaller batch size, each mini-batch only consists of a small number of training examples to be trained using SGD(stochastic gradient descent), which adds too much noise through generalization and results in low accuracy. However, it does not mean larger batch size would produce better generalization because larger batch size require more computation, which makes SGD less computational efficient and brings more inaccuracy in gradient estimates. This is why too large batch size does not generalize better. Therefore, empirically there is a balance between large and small batch size. In this case, 100 for batch size is relatively the optimal solution.

From (Ioffe & Szegedy, 2015), we know that batch normalization also regularizes the model, which means no regularization is needed for batch-normalized networks. Therefore, we do not need to change weight decay coefficient, which is 0 by default. Actually, the paper suggests reducing L2 weight regularization. This is the reason why we do not search over weight decay coefficient.

Finally, from all the analysis above, we adopt BN-Baseline to be the final network. We test the model on multiple seeds. The results are presented in figure 6 and table 5 and 6.

7. Conclusions

To conclude, in this report, we explored the optimization problem in training deep CNN architectures due to vanishing gradients problems. We showed the benefits of the proposed solution to this problem by batch normalization. The experimental results show that batch normalization works effectively in solving this problem. From the gradients flows of batch-normalized network, which demonstrates its stability, it proves that vanishing gradients is essentially caused by small gradients multiplied together throughout backpropagation based on chain rule. By changing parameters to explore potentially better solutions, we searched over learning rate and batch size and performed experiments on different models for each case. We ended up finding that the optimal solution is the baseline network with learning rate being 0.001 and batch size to being 100. Afterwards, we performed several experiments on the optimal model with multiple seeds.

In future, we would like to gain further insights into the solution and problem by exploring how to search over more combination of hyper-parameters such as kernal size or stride to find a model that generalizes better based on the empirical knowledge that too large or too small learning rate and batch size both lead to poor generalization in this problem. Moreover, we could explore combination of residual network (He et al., 2015) or DenseNet (Huang et al., 2016) with batch normalization to make the model generalize better

References

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL http://arxiv.org/abs/1512.03385.

Huang, Gao, Liu, Zhuang, and Weinberger, Kilian Q. Densely connected convolutional networks. *CoRR*, abs/1608.06993, 2016. URL http://arxiv.org/abs/1608.06993.

Ioffe, Sergey and Szegedy, Christian. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015. URL http://arxiv.org/abs/1502.03167.