Final Report

Anurag Dutt, Wei Liu, Ruizhe Gao

**Abstract**: Overfitting is a serious problem when using deep neural networks with a large number of parameters. Large networks also take more time to train and test, making it less practical in the real world. To address this problem, Dropout is widely employed. By randomly removing units and its connections from the network, the technique significantly reduces co-adapting effects during training. In our report, we mainly investigate this technique using three datasets, MNIST, CIFAR10, and CIFAR100, and compare the result with related papers. The result suggests that Dropout leads to an increase of accuracy making predictions. Batch Normalization is another powerful technique in deep learning. However, combining Dropout and Batch Normalization together often result in a worse performance. We explore different strategies dealing with this problem. At the end, we further apply these strategies to GANs.

1. Paper Replication
   1. Results on MNIST

The MNIST dataset consists of 2828 pixel handwritten digit images, which are classified into 10 digit classes. We employ the same architecture in [1]. All networks use 0.8 as Dropout rate for input layers, and 0.5 for hidden layers respectively. The batch size we use is 200, and the number of training epochs is 1000. Table 1 compares the performance with [1]. The error rates we obtain are slightly larger than those in the paper. We think this may be due to the different batch size and number of epochs used in our report as opposed to [1]. However, the result still shows the effectiveness of Dropout. Without Dropout, the best error rate is 1.42% (ReLUs units, 3 layers). Adding Dropout and max-norm constraint reduces the error rate to 1.24%. Increasing the size of the network can further improve the performance. A network with 2 layers and 4096 units gets down to 1.1% error. We also try replacing ReLU with Maxout activation, and the error rate is a little bit higher (1.15%).

Table 1 MNIST

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Method | Unit Type | Architecture | Error% | Error% in the paper |
| Standard NN | Logistic | 2 layers, 800 units | 1.58 | 1.6 |
| Dropout NN | Logistic | 3 layers, 1024 units | 1.25 | 1.35 |
| Dropout NN | ReLU | 3 layers, 1024 units | 1.42 | 1.25 |
| Dropout NN + max-norm constraint | ReLU | 3 layers, 2048 units | 1.24 | 1.06 |
| Dropout NN + max-norm constraint | ReLU | 3 layers, 4096 units | 1.17 | 1.04 |
| Dropout NN + max-norm constraint | ReLU | 2 layers, 4096 units | 1.1 | 1.01 |
| Dropout NN + max-norm constraint | Maxout | 2 layers, (5\*240) units | 1.15 | 0.94 |

* 1. Results on CIFAR-10

The CIFAR-10 dataset consists of 50,000 training images and 10,000 test images. Each sample is 32 pixels by 32 pixels in the RGB color model drawn from 10 categories. The dataset comes in 6 batches, we choose 5 of them as training batches and the rest one as a test batch. Each of batch has around 1000 images per class.

We use three convolutional layers followed by max-pooling layers. Furthermore, we add two fully connected hidden layers followed three convolutional layers and all unit use rectified linear activation function. We apply Dropout with the probability of retaining the unit being p=0.5, 0.8 and 0.9 for different layers.

Table 2 shows the error rate obtained by different methods. The baseline is 25.7% using only convolutional nets and max pooling layers. Adding Dropout in fully-connected layers lead to 5.76% increase of accuracy. The error rate is further reduced to 18.54% by adding Dropout to all layers. Besides, we also use WRN and Densenet to test Dropout. The result further confirm that dropout can reduce overfitting in most cases.

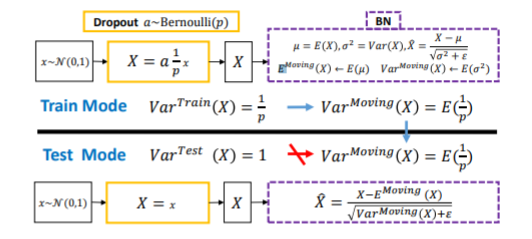
Table 2 CIFAR-10

|  |  |  |
| --- | --- | --- |
| Method | Error% | Error% in the paper |
| Conv Net + max pooling | 25.70 | 14.98 |
| Conv Net + max pooling + Dropout in fully-connected layers | 19.94 | 14.32 |
| Conv Net + maxpooling + Dropout in all layers | 18.54 | 12.61 |
| DenseNet | 10.18 | - |
| DenseNet + Dropout (p=0.5) | 10.04 | - |
| WRN | 8.73 | - |
| WRN + Dropout (p=0.5) | 8.36 | - |

1. Combine Dropout and Batch Normalization

Since the birth of Dropout, it has been proved to be significantly effective in preventing large networks from overfitting. As introduced in [5], Batch Normalization is another powerful technique introduced to speed up the architectures and improve the performance as regularizers. However, combining the above two techniques together often leads to a worse result.

[2] first answers the question in theoretical and statistical aspects. They discover that there exists mismatch of variance when the state changes from train to test, which could account for the disharmony between Dropout and Batch Normalization. As illustrated in Figure 1, Dropout would scale the variance by its retain ratio, yet Batch Normalization still maintains its variance. They name this scheme as “Variance Shift”.



* 1. Two strategies to avoid Variance Shift

Inspired by [2], we make an extension by applying two strategies to address the problem of “Variance Shift”.

* + 1. Apply Dropout after all Batch Normalization layers (Strategy 1)

There would exists the Variance Shift in an architecture only if we have a Dropout layer before a Batch Normalization layer. Thus, one way to avoid the problem is to simply place dropout layers after all Batch Normalization layers. In the following sections, when using Strategy 1, we would only employ Dropout in fully-connected layers.

* + 1. Modify the form of Dropout (Strategy 2)

Instead of completely removing the Variance Shift effect from our networks, we could also reduce the impact by modifying the form of Dropout. First, we would generate random variables from a uniform distribution that lies in , i.e., generate . Then, we would change each hidden activation to . By doing this, we add a layer that functions like Dropout but has a variance shift ratio, defined as the ratio of test variance to train variance, much closer to 1.0 in comparison with the traditional Dropout layer. We conduct experiments on MNIST, CIFAR-10 and CIFAR-100 using different values (0.1, 0.2, 0.3) for .

* 1. Experimental results
     1. MNIST

From Table 3, we can see that when combining Dropout and Batch Normalization together, the architecture performs worse on MNIST (Error% 0.89). To avoid the scaling on feature-map before every Batch Normalization layer, we only add Dropout on the fully connected layers after all Batch Normalization layers. Such an operation could bring 0.13% improvements on MNIST as opposed to only using Batch Normalization. We could also modify the formula of Dropout by adding a uniform distributed random variable . This leads to 0.13~0.2% increase of accuracy.

Table 3 MNIST (Dropout + Batch Normalization)

|  |  |
| --- | --- |
| Method (CNN) | Error% |
| Batch Normalization | 0.85 |
| Batch Normalization + Dropout(all,p=0.9,fc,p=0.5) | 0.89 |
| Dropout(all,p=0.9;fc,p=0.5) | 0.75 |
| Batch Normalization + Dropout(fc,p=0.5) + strategy1 | 0.72 |
| Batch Normalization + Dropout(fc,p=0.5) + strategy2(beta=0.1) | 0.73 |
| Batch Normalization + Dropout(fc,p=0.5) + strategy2(beta=0.2) | 0.64 |
| Batch Normalization + Dropout(fc,p=0.5) + strategy2(beta=0.3) | 0.72 |

* + 1. CIFAR-10 and CIFAR-100

Besides CIFAR-10, we also use CIFAR-100 to test the two strategies. The CIFAR-100 dataset consists of 3232 color images from 100 categories. The dataset contains train set and test set. We use the same architecture in 1.2.

First, we implement two strategies to CIFAR-10 dataset. As reported in table 4, when we add both Batch normalization and Dropout to all layers without strategies, we got error rate around 21.64%. Then, we implement the first strategy which applies Dropout after all BN layers, and we found that the error rate goes down to 16.42%. Furthermore, we change Dropout into a more variance-stable form based on strategy 2. The new form of Dropout achieves 4.64~5.18% increase of accuracy. In conclusion, improvements can be observed by using both strategies on Cifar10 set compared with the original results. Also, if we remove batch normalization and only apply strategy2 to Dropout, the error rate is around 19~20% which is close to the results from original methods without strategies.

Table 4 CIFAR-10 (Dropout + Batch Normalization)

|  |  |
| --- | --- |
| Method (CNN) | Error% |
| No Dropout and Batch Normalization | 25.70 |
| Dropout (fc, p=0.5) | 19.94 |
| Batch Normalization | 19.79 |
| Dropout (fc,p=0.5) + strategy2 (beta=0.1) | 19.70 |
| Dropout (fc,p=0.5) + strategy2 (beta=0.2) | 20.40 |
| Dropout (fc,p=0.5) + strategy2 (beta=0.3) | 19.76 |
| Batch Normalization + Dropout (all, p=0.8) | 21.64 |
| Batch Normalization + Dropout (fc, p=0.5) + strategy1 | 16.42 |
| Batch Normalization + Dropout (all, p=0.5) + strategy2 (beta=0.1) | 17.00 |
| Batch Normalization + Dropout (all, p=0.5) + strategy2 (beta=0.2) | 16.79 |
| Batch Normalization + Dropout (all, p=0.5) + strategy2 (beta=0.3) | 16.46 |

Then we implement two strategies to CIFAR-100 dataset. From table 5, we can see that if we only add Dropout or batch normalization separately to the neural network, the accuracy rate is around 28~29%. Also, if we only add strategy2 with Dropout, no significant improvements are reported. Furthermore, when we apply Dropout and batch normalization together without strategies, the accuracy drops to around 20%. Thus, from the results above, strategies have led to a significant increase in accuracy. We use the first strategy which applies Dropout after all BN layers, and the accuracy goes up to 36.54% with Dropout equals 0.5 and 33.48% with Dropout equals 0.8. And for the second strategy, we add the uniform noise to each convolutional layer and the new structure achieves 3~6% increase of accuracy with different parameter values.

Table 5 CIFAR-100 (Dropout + Batch Normalization)

|  |  |
| --- | --- |
| Method (CNN) | Accuracy% |
| No Dropout and Batch Normalization | 20.12 |
| Dropout (fc, p=0.5) | 28.17 |
| Batch Normalization | 29.77 |
| Dropout (fc,p=0.5) + strategy2 (beta=0.1) | 30.23 |
| Dropout (fc,p=0.5) + strategy2 (beta=0.2) | 28.11 |
| Dropout (fc,p=0.5) + strategy2 (beta=0.3) | 25.47 |
| Batch Normalization + Dropout (all, p=0.5) | 20.38 |
| Batch Normalization + Dropout (fc, p=0.5) + strategy1 | 36.54 |
| Batch Normalization + Dropout (fc, p=0.8) + strategy1 | 33.48 |
| Batch Normalization + Dropout (all, p=0.5) + strategy2 (beta=0.1) | 33.46 |
| Batch Normalization + Dropout (all, p=0.5) + strategy2 (beta=0.2) | 36.04 |
| Batch Normalization + Dropout (all, p=0.5) + strategy2 (beta=0.3) | 33.16 |

1. Apply Dropout and Batch Normalization to GANs

Appendix

Reference

[1] Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting." The Journal of Machine Learning Research 15.1 (2014): 1929-1958.

[2] Li, Xiang, et al. "Understanding the disharmony between dropout and batch normalization by variance shift." arXiv preprint arXiv:1801.05134 (2018).

[3] Cogswell, Michael, et al. "Reducing overfitting in deep networks by decorrelating representations." arXiv preprint arXiv:1511.06068 (2015).

[4] Wan, Li, et al. "Regularization of neural networks using dropconnect." International Conference on Machine Learning. 2013.

[5] Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." arXiv preprint arXiv:1502.03167 (2015).

[6] Zheng, Zhedong, Liang Zheng, and Yi Yang. "Unlabeled samples generated by gan improve the person re-identification baseline in vitro." arXiv preprint arXiv:1701.07717 3 (2017).

[7] Mordido, Gonçalo, Haojin Yang, and Christoph Meinel. "Dropout-GAN: Learning from a Dynamic Ensemble of Discriminators." arXiv preprint arXiv:1807.11346 (2018).