Analyzing the Effect of Batch Normalization and Dropout Layers on CNN Performance

Dymytriy Zyunkin
5/11/2022
Rutgers ECE

Abstract
Dropout and batch normalization layers are vitally important to train neural network efficiently by removing matrix multiplication operations that are minimally important to the final outcome and improve the network performance by preventing overfitting. This report analyzes the performance of a LeNet5 model trained on MNIST with the MNIST dataset and compares the effects of Batch Normalization and Dropout layers on the test accuracy of the model, as well as how these layers impact the training time.

Introduction
Improving the speed, stability, and performance of deep learning algorithms is of paramount importance to delivering a trustworthy application. Both batch normalization and dropout layers accomplish this goal, although they take on vastly different approaches. Batch normalization allows the model to mitigate an effect called the covariate shift - the change in distribution of the inputs. During training, each layer's inputs are normalized by subtracting the mean and dividing by the standard deviation of the current batch, not the overall dataset. Batch normalization adds additional computational overheads to each epoch, but may result in fewer epochs achieving the desired test accuracy. BN may also help accommodate higher learning rates and enables easier weight initialization since the weights converge to the desired values quicker. Additionally, it enabled better scaling for activation functions. A visual representation of the batch normalization method can be found below:

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Figure 1 - A mathematical description of Batch Normalization steps

The dropout layer is introduced to ensure minimum overfitting of the model - that is to ensure that the model is not trained specifically to perform well on training data, but fails on test data. It accomplishes this goal by approximating some layer outputs randomly, meaning that they are not influencing the output of that layer. This increases the number of iterations required to converge, but the overall number of iterations may be decreased. An illustrative example of how the Dropout layer effects the forward pass through a neural network can be seen below:

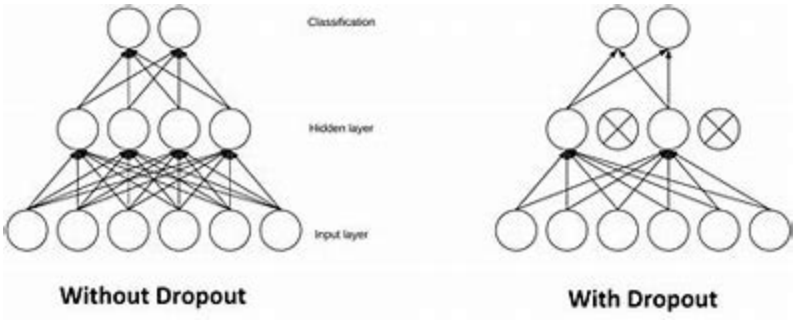


Figure 2 - Showing the engaged neurons with dropout layers

Related Work

Batch normalization was introduced in the 2015 paper “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift” by Sergey Ioffe’s and Christian Szegedy. The general background for normalizing data is to train a multivariate network on data that has massively different scales. Further, this idea was extended to normalizing inter-layer connections to normalize inputs for the next subsequent layer. This allows for easier integration into many activation functions, as gradient saturation problems are alleviated by holding all inputs within a certain range.

Dropout layers ensure that neurons in each layer are randomly activated and applied gradient descent to help the model converge to an acceptable solution faster. Dropout layer increases the number of iterations required to converge, but the overall number of iterations may be decreased. Dropout layers were first introduced in the 2012 paper “Improving neural networks by preventing co-adaptation of feature detectors” by Geoffrey Hinton, et al. Further developments in dropout layers have been made by Srivastava, et al. in their 2014 journal paper “Dropout: A Simple Way to Prevent Neural Networks from Overfitting.” Both studies found that introducing dropout layers to a network enhances its results for classification tasks.

Data Description

The data that was used as a benchmark for evaluation of the model was taken from the MNIST dataset of 50,000 training images and 10,000 test images of handwritten digits. The images are all of 32 by 32 pixels in resolution and are normalized to [0,255] in grayscale. To ease the data preprocessing and exfiltration, it was preprocessed using the native PyTorch methods that do not require explicit downloading from the LeCun database. Additionally, a data loader native to PyTorch was used to expedite all batching procedures and remove the necessity for manual shuffling.

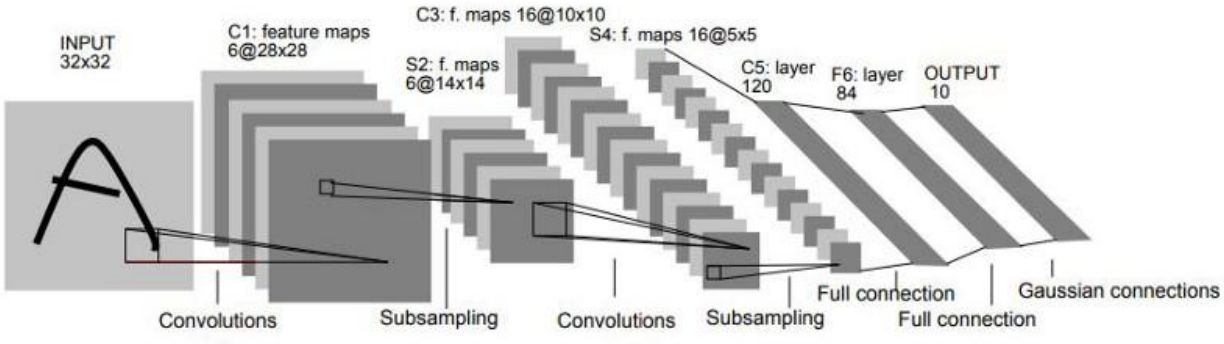
Method Description

To ensure consistency of the results, a certain baseline must be established to compare all results against. Furthermore, several hyperparameters must be frozen to ensure consistency in

training and validation. The number of epochs was frozen at 5 since this was a sensible trade-off between training time and satisfactory results. The batch size was fixed at 128 as it is a sensibly large batch size for gradient descent, but it is not overbearing given the CPU capabilities. The learning rate was fixed at 0.05 with no momentum added, as per the class suggestions. The results from the training are shown in Table 1 and 2 below.

Model Description

As per the assignment instructions, all experiments were conducted on a LeNet5 model. Below is a sample model architecture published in [LeCun et al., 1998]:



LeNet5 accepts a 32 by 32 grayscale input which is passed through a first convolutional layer with 6 feature extractors. Thus, the input to the network is a 32 by 32 by 6. A kernel of the size 5 by 5 reduces the feature map to a 28 by 28, which is then subsampled by a 2 by 2 maxpool layer, leaving behind a 14 by 14 feature map. The convolution operation repeats, but now with 16 and then subsequently 10 filters which then get flattened and fed forward to a fully connected layer. The next three layers reduce the dimensionality from 120 to 84 and finally 10.The number of dimensions in the output layer corresponds to the number of classes present in the dataset.

Experimental Procedure and Results

The original LeNet5 model is trained four separate times with no BN or Dropout, BN, Dropout, and both BN and Dropout. Further, the hyperparameter tuning for the Dropout layer is toggled between 0.1 and 0.5 to see the effect that an increased dropout ratio would have on the training speed as well as the test accuracy. Further, the batch normalization is added in the convolutional layer using the nn.BatchNorm2d() command and the dropout layer is placed in the fully connected layer. Below are the two tables condensing the results:

	P = 0.1	P = 0.2	P = 0.5	P = 0.8
Baseline	84.1743s	84.1743s	84.1743s	84.1743s

BN	94.8565s	94.8565s	94.8565s	94.8565s
Dropout	74.8126s	86.6575s	74.8981s	71.5549s
BN + Dropout	87.1156s	85.2019s	85.2713s	84.7738s

Table 1 - Comparing results of training time with batch normalization and dropout

	P = 0.1	P = 0.2	P = 0.5	P = 0.8
Baseline	0.9832	0.9832	0.9832	0.9832
BN	0.9883	0.9883	0.9883	0.9883
Dropout	0.9864	0.9876	0.9863	0.9614
BN + Dropout	0.9886	0.9902	0.9903	0.9821

Table 2 - Comparing results of test accuracy with batch normalization and dropout

Please note that there is no hyperparameter tuning for the batch normalization layers, so it is held static throughout the second line of the table.

Conclusion

Adding a batch normalization, dropout, or a combination of the two layers proves an effective measure to decrease the training time of a model and improve performance on the test set. Adding a batch normalization layer on its own improves the performance marginally, at the expense of a significant increase in training time (over 10% increase). Adding a sole dropout layer adds little to the training accuracy, but significantly reduces the training time. The higher the proportion of neurons dropped during the pass, the quicker was the training with a tradeoff for minor reductions in accuracy. Adding both batch normalization and dropout in training initially raises the total training time with low dropout rates, but a higher dropout rate can compensate for the additional computational overheads. Both BN and dropout do not significantly improve the performance of the model.

References

Key Deep Learning Architectures: LeNet-5 | by Max Pechyonkin | Medium
main.dvi (ucsd.edu)

Batch normalization | Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37 (acm.org)

How ReLU and Dropout Layers Work in CNNs | Baeldung on Computer Science

1207.0580.pdf (arxiv.org)

1207.0580.pdf (arxiv.org)

Slides and presentation from Introduction to Deep Learning