DropoutVs BatchNormalization

INTRODUCTION

Overfitting and long training time are two fundamental challenges in multilayered neural network learning and deep learning in particular. Dropout and batch normalization are two well-recognized approaches to tackle these challenges. While both approaches share overlapping design principles, numerous research results have shown that they have unique strengths to improve deep learning. Many tools simplify these two approaches as a simple function call, allowing flexible stacking to form deep learning architectures. Although their usage guidelines are available, unfortunately no well-defined set of rules or comprehensive studies to investigate them concerning data input, network configurations, learning efficiency, and accuracy. It is not clear when users should consider using dropout and/or batch normalization, and how they should be combined (or used alternatively) to achieve optimized deep learning outcomes. In this paper we conduct an empirical study to investigate the effect of dropout and batch normalization on training deep learning models. We use multilayered dense neural networks and convolutional neural networks (CNN) as the deep learning models, and mix dropout and batch normalization to design different architectures and subsequently observe their performance in terms of training and test CPU time, number of parameters in the model (as a proxy for model size), and classification accuracy. The interplay between network structures, dropout, and batch normalization, allow us to conclude when and how dropout and batch normalization should be considered in deep learning. The empirical study quantified the increase in training time when dropout and batch normalization are used, as well as the increase in prediction time (important for constrained

environments, such as smartphones and low-powered IoT devices). It showed that a non-adaptive optimizer (e.g. SGD) can outperform adaptive optimizers, but only at the cost of a significant amount of training times to perform hyperparameter tuning, while an adaptive optimizer (e.g. RMSProp) performs well without much tuning. Finally, it showed that dropout and batch normalization should be used in CNNs only with caution and experimentation (when in doubt and short on time to experiment, use only batch normalization).

Overfitting and dropout

Overfitting is a common challenge in training machine learning models. In general, overfitting happens when the model performs well on the training data, but performs poorly on test data, i.e. the model has low training error and high test error. *Regularization* is a set of techniques used to reduce overfitting [5]. Some of these techniques are model-specific, such as preprunning and postprunning for decision trees, some techniques act on the gradient descent optimization algorithms, and others act on the input data, artificially creating new training data

Other techniques go beyond acting on only one model. One such technique is *model ensembling*, combining the output of several models, each trained differently in some respect, to generate one final answer. It has dominated recent machine learning competitions. Although model ensembling performs well, it requires a much larger training time by definition (compared to training only one model). Each model in the ensemble has to be trained either from scratch or derived from a base model in significant ways.

When tackling overfitting for deep learning, *dropout* proposed to randomly change the network architecture, to minimize the risks that the learned weight values are highly customized to the underlying training data, and therefore cannot be generalized well to test data. In essence, dropout simulates model ensembling without creating multiple networks. It has been widely adopted since its publication, in part because it does not require fundamental changes to the network architecture, other than adding the dropout layers.

Despite its simplicity, dropout still requires tuning of hyperparameters to work

well in different applications. The original paper mentions the need to change the learning rate, weight decay, momentum, max-norm, number of units in a layer, among others. Getting dropout to work well for a given network architecture and input data requires experimentation with these hyperparameters. Adding dropout to a network increases the convergence time. Then, after adding dropout, we need to train models with different combinations of hyperparameters that affect its behavior, further increasing training time.

Another equally important, if not more significant, complicating factor is that existing dropout evaluations were tested with the standard stochastic descent gradient (SGD) optimizer. Most networks today use adaptive optimizers, e.g. RMSProp, commonly used in Keras examples. Some of the recommendations in the dropout paper, for example, learning rates and weight decay values, do not necessarily apply when an adaptive optimizer is used.

This observation naturally leads to techniques focusing on improving the model training efficiency by helping the models converge faster. One such technique commonly used for deep learning is *batch normalization*.

Training efficiency and batch normalization

Before *batch normalization* was introduced, the time to train a network to converge depended significantly on careful initialization of hyperparameters (e.g. initial weight values) and on the use of small learning rates, which lengthened the training time. The learning process was further complicated by the dependency of one layer on its preceding layers. Small changes in one layer could be amplified when flowing through the other network layers. Batch normalization significantly reduces training time by normalizing the input of each layer in the network, not only the input layer. This approach allows the use of higher learning rates, which in turn reduces the number of training steps the network needs to converge. Similar to dropout, using batch normalization is simple: add batch normalization layers in the network. Because of this simplicity, using batch normalization would be a natural candidate to be used to speed up training of different combinations of hyperparameters needed to optimize the use of dropout layers (it would not speed up each epoch during training, but would help converge faster).

Dropout and batch normalization

Because deep learning architectures often have a large number of weight values for tuning, whereas the training process only has a limited number of samples, overfitting becomes a significant challenge. On the other hand, existing approaches to avoid overfitting, such as decision tree pruning or constrained optimization, are either too specific or too expensive, therefore cannot be applied to general deep learning frameworks. Finding simple and effective approaches to avoid overfitting for deep learning is a practical challenge.

To prevent neural networks from overfitting, Srivastava et al. [28] introduced dropout and described how specific hyperparameters (learning rate, momentum, max-norm, etc.) affect its behavior and provided detailed recommendations on how to train networks using dropout. Most of the recommendations are given in ranges of values for each hyperparameter. For example, increase learning rate by 10 to 100 times, use momentum between 0.95 and 0.99, apply max-norm with values from 3 to 4, etc. The dropout rate itself is recommended as a range between 0.5 and 0.8 (for hidden layers). Mixing this number of hyperparameters and their ranges result in a large matrix of combinations to try during training.

To accelerate the deep network training, Ioffe and Szegedy [7] introduced batch normalization. It shows that batch normalization enables higher learning rates by reducing internal covariate shift, but does not prescribe a value or a range to be used. It also recommends reducing L2 weight regularization and to accelerate learning rate decay. Finally, it recommends removing dropout altogether and counting on the regularization effect provided by batch normalization. This claim has been studied in more recent articles (some of them are referenced below). These additional investigations resulted in recommendations to use dropout together with batch normalization in some scenarios.

Noticing the success of the dropout and batch normalization for deep learning,

reconciles dropout and batch normalization for some applications and shows that combining them reduces the error rate in those applications. Its specific recommendation is to apply dropout after batch normalization, with a small dropout rate.

In order to better understand regularization in batch normalization, shows that using dropout after batch normalization layers is beneficial if the batch size is large (256 samples or more) and a small (0.125) dropout rate is used.

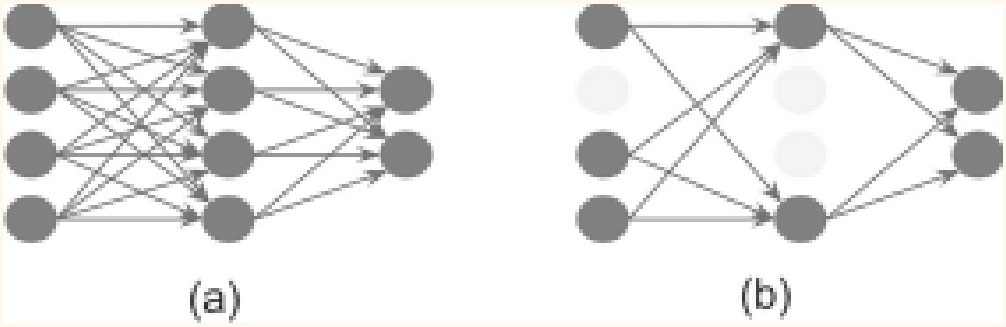

Fig. 1 A conceptual view of dropout. Instead of using a fixed network structure, dropout randomly removes units from a fully connected network (left) to create a sub-model (right)

In summary, while there is empirical research of dropout and batch normalization, with some practical recommendations for settings for each of them, our research work reported in this paper differs from existing works in two main aspects:

1. Our work not only investigates the effect of the dropout and batch normalization layers, but also studies how they behave with respect to different optimizers: the SGD optimizer, commonly used in publications, and an adaptive optimizer (RMSProp), commonly used in commercial applications.
2. Our research investigates the efficiency and effectiveness of the networks using dropout and batch normalization combined training. The networks are measured

in terms of the time to train the networks, the time a trained network takes to predict values, and the memory consumption of the network. These factors are important for real-life applications, where the best possible accuracy is not the only deciding factor to adopt a deep neural network architecture.