Activation Dropout, Inverse Dropout and Correlated Dropout

Ignacio M. Gavier
University of Massachusetts at Amherst
igavier@umass.edu

Abstract

Deep neural networks that prevent overfitting are desiderable models not only for classification tasks, but for every prediction problem in machine learning. Different approaches to avoid overfitting have been proposed, including regularization penalty and dropout techniques. Standard Dropout is the oldest and most common dropout technique used for preventing networks to overfit the training data, where randomly selected neurone from the hidden layers are deactivated with probability 50%. The Standard Dropout technique was proved to outperform the previous multilayer perceptrons models. In the following years, different dropout techniques were proposed to improve the Standard Dropout. In this work, we present three new techniques: Activation Dropout (DropAct), Inverse Dropout (DropInv) and Correlated Dropout (DropCorr). These models are inspired in the Standard Dropout, but with different approaches on the deactivation probabilities distributions of the hidden neurone. When evaluating the proposed techniques on different architectures to classify MNIST and CIFAR-10 datasets, we found that they perform as well as the Standard Dropout, but they are slower, especially DropCorr.

1. Introduction

Deep Neural Networks models have been applied to different fields, like computer vision, speech recognition, natural language processing, etc. Their structure allows us to easily increase their capacity by adding more parameters into the network. However, big networks tend to overfit the training data, leading to a poor performance during testing. Many techniques have been developed to confront overfitting, such as adding an ℓ_2 penalty on the network weights.

Dropout is an effective regularization technique, introduced in 2012 by Srivastava et al. [6], that prevents neural networks to overfit the training data, leading to a significantly better performance during test time. In the paper, the authors proposed the Standard Dropout, which turns off randomly selected neurone with 50% of deactivation probability during training time. This technique significantly improved the performance of previous network used not only in computer vision, but also in general prediction tasks.

Originally, the dropout technique was used to avoid overfitting, but it was also extended to a variety of tasks, like compressing neural networks and Monte Carlo dropout, which measures the uncertainty of predictions in neural networks [4]. As a consequence, this technique has become an area of significant interest within the field of machine learning, leading to related works that outperformed the Standard Dropout.

During the subsequent years after Standard Dropout, several dropout techniques were developed for neural networks, including DropConnect [7], Gaussian Dropout [8], Standout [1], Variational Dropout [3], etc. All these techniques were inspired in the original Standard Dropout and all consist of modifying the activation values of the hidden neurone.

⋆ ⋆ ⋆

In this work, we review some of the most used dropout techniques in Section 2, such as the Standard Dropout (2), and also DropConnect, Gaussian Dropout, Variational Dropout and Standout (2). Also, we describe our proposed techniques, DropAct, DropInv and DropCorr and their technical details in Section 3. In Section 4 we show the performance of our techniques compared with the other dropout techniques, using the MNIST and CIFAR-10 datasets with different model architectures. We analyze the training time of each technique and the achieved performance prediction for each dataset.

2. Background

Neural networks are composed by multiple layers, and depending on the task they are performing, their architecture can vary greatly. In general, for image classification tasks, the most common models are the convolutional networks. These models, after the first convolutional layers, have the sequence of fully connected layers to decode the features extracted in the convolutional layers, and calculate the scores for each label. Each of these fully connected lay-

ers consists of an affine layer, which is a linear multiplication of an input vector by a weight matrix plus a bias, followed by an non-linear activation layer. The output of a given fully connected layer can be written as:

$$\mathbf{y} = f(\mathbf{W}\mathbf{x}),$$

where $\mathbf{W}$ is the weight matrix and $f(\cdot)$ is the activation function. Note that here we are assuming that the bias has been absorbed into the product.

Activation Layer

One of the most common activation functions $f(\cdot)$ used for neural networks in computer vision, is the ReLU [5]. The ReLU function can be written as $f(x) = \max(0, x)$ and it is known to work much faster than the logistic function.

Standard Dropout

Standard Dropout layers randomly select neurone to be deactivated during training time with probability p. We can express the output of a given Affine-ReLU-StandardDropout layer as:

$$\mathbf{y} = f(\mathbf{W}\mathbf{x}) \circ \mathbf{m}, \;\; m_i \sim (1-p)\text{-Bernoulli},$$

where $\circ$ is the Hadamard product. Here, $\mathbf{m}$ is a vector of zeros and ones that behaves like a mask vector of the outputs after the activation layer. During test time, the layer does not deactivate any neuron, and the output needs to be rescaled by a factor of $(1-p)$ in order to it has the same first moment than the output during training time:

$$\mathbf{y} = (1-p)f(\mathbf{W}\mathbf{x}).$$

Another version of the Standard Dropout is the one that avoids multiplying by $(1-p)$ during test time. Therefore, the fully connected layer during training is written as:

$$\mathbf{y} = f(\mathbf{W}\mathbf{x}) \circ \frac{\mathbf{m}}{1-p}, \;\; m_i \sim (1-p)\text{-Bernoulli},$$

and the output during test time is just

$$\mathbf{y} = f(\mathbf{W}\mathbf{x}).$$

This variation receives the name of inverted Standard Dropout, while the previous one is called vanilla Standard Dropout. In practice, if Standard Dropout is used, the inverted version is implemented since faster testing time is preferable.

Other Dropout Techniques

Following with the Standar Dropout ideas, different dropout techniques were proposed. One of the most common variations is the one that instead of deactivating neurone, it randomly deactivates weights in the weight matrix. This technique is called DropConnect [7] and the output is represented as:

$$\mathbf{y} = f((\mathbf{W} \circ \mathbf{M})\mathbf{x}), \;\; \text{where } m_{ij} \sim (1-p)\text{-Bernoulli}.$$

In this kind of dropout, $\mathbf{M}$ is a matrix of ones and zeros that behaves like a matrix mask for the weight matrix and they have the same dimensions. For test time, the authors took advantage of the approximately Gaussian distribution of the product $\mathbf{W}\mathbf{x}$: a sample was taken from this Gaussian and passed to the neuron activation function.

Gaussian Dropout [8] layers are designed to train faster than Standard Dropout by taking advantage of the fact that the output values follow a Gaussian distribution. Therefore, instead of incorporating a Bernoulli multiplicative noise to the matrix weight, they proposed a multiplicative Gaussian noise with the same mean and variance:

$$\mathbf{y} = f(\mathbf{W}\mathbf{x}) \circ \mathbf{m}, \;\; \text{where } m_i \sim \mathcal{N}(0, \alpha),$$

where $\alpha = \frac{p}{1-p}$ is the standard deviation, which is equal to the standard deviation of the Bernoulli divided by $(1-p)$, so their means can also match.

Variational Dropout [3] is a dropout technique inspired in Gaussian Dropout that uses variational inference to estimate the posterior of the Gaussian noise. They introduce a Kullback-Leibler divergence as a regularizer for the standard deviation of the noise, letting α to be adaptive.

Standout [1] layers are similar to Standard Dropout, but they have an adaptive value for the probability that a neuron is deactivated. We can express the output of a fully connected layer using Standout as:

$$\mathbf{y} = f(\mathbf{W}\mathbf{x}) \circ \mathbf{m}, \;\; m_i \sim g((\mathbf{\Pi}\mathbf{x})_i)\text{-Bernoulli},$$

where $\mathbf{\Pi}$ is a matrix of identical size than $\mathbf{W}$, and $g(\cdot)$ is an activation function satisfying $g : \mathbb{R} \to [0, 1]$. The values of $\mathbf{\Pi}$ are learned at the same time as $\mathbf{W}$ is learned. The authors found that setting $\mathbf{\Pi} = \alpha\mathbf{W} + \beta$ is a good approach for saving space and computation time, where α and β are parameters to be learned during training. Also, they propose different activation functions $g(\cdot)$ for the mask, such as the logistic function or the inverted logistic function. During test time, the output has to be rescaled accordingly:

$$\mathbf{y} = f(\mathbf{W}\mathbf{x}) \circ g(\mathbf{\Pi}\mathbf{x}).$$

The described techniques are some of the most common dropout layers used in fully connected layers. The Standard Dropout is the fastest and simplest dropout, because it samples Bernoulli experiments for the number of output dimensions in that layer. DropConnect, for example, samples more Bernoulli experiments since it uses one for each weight. Therefore, it needs the size of the input times the size of the output samples. Gaussian Dropout is almost as

fast as Standard Dropout because sampling two independent values from a Gaussian distribution is almost as efficient as sampling two independent Bernoulli values. Variational Dropout is slower than Gaussian Dropout since it needs to minimize de Kullback-Leibler divergence.

In the following sections, we will describe different types of dropout algorithms and how they perform compared with the Standard Dropout and the other dropout methods described above. We will analyze the validation and test accuracy they achieve for standard architectures working on the MNIST and CIFAR-10 datasets. Also, we will measure the time efficiency of the proposed algorithms.

3. Approach

The proposed dropout algorithms for this work are DropAct, DropInv and DropCorr. The three techniques inherits the basic properties of the Standard Dropout, but different ways to select the deactivated neurone. The main variation with respect to the previous techniques is that deactivated elements are not independent from each other. They are sampled altogether from a multivariate distribution probability. We briefly describe each technique and then show the details:

- **DropAct:** The underlying idea for this dropout technique is to deactivate neurone that have higher activation values.
- **DropInv:** This technique is similar to DropAct, but instead of probably deactivate neurone with higher activation values, it deactivates neurone with values close to zero.
- **DropCorr:** In this algorithm, the dropout layer deactives set of neurone that correlatively activate at the same time.

For all the dropout techniques mentioned in Section 2, there is an underlying independence for deactivating elements (neurone or weights). That is, for every element, the probability of being deactivated does not depend on the fact that a different element is deactivated. In our approach, we tweak this idea by making dependent deactivations. For the three techniques, given a deactivation probability p, we select $k = \lfloor pN \rfloor$ neurone to be deactivated, being N the number of neurone in that particular layer. The k neurone are selected from a multivariate probability distribution as we will mention in the following sections.

In order to express the new dropout algorithms, we use the notation from the previous section.

3.1. DropAct

The output of an Affine-ReLU-DropAct layer can be expressed as:

$$\mathbf{y} = f(\mathbf{W}\mathbf{x}) \circ \mathbf{m}.$$

Here, we sample m_i to be zero from a multinomial distribution without replacement with probabilities $p_i \sim x_i$. One of the advantages this technique has is that avoids setting to zero a neuron that has been previously set to zero by the activation function. The algorithm for creating the vector mask $\mathbf{m}$ is described below. We represent the input to the dropout layer as $\mathbf{x}$.

Algorithm 1: DropAct

$\mathcal{I} = \{1, \ldots, N\}$;
$m_i = 1$;
$p_i \sim x_i$;
Rescale probabilities accordingly;
for *iter in* $[1, \ldots, k]$ **do**
  Sample j from $\mathcal{I}$ with corresponding probabilities;
  $\mathcal{I} \leftarrow \mathcal{I} - j$;
  $m_j = 0$;
  Rescale probabilities accordingly;
end

3.2. DropInv

Similar to DropAct, but we sample m_i to be zero from a multinomial distribution without replacement with probabilities $p_i \sim 1/x_i$. Since the activation values after a ReLU layer are probably to be zero, we do not take into account the values that are zero. An overlaying intuition of this kind of dropout is that it enhances activations with higher values, while avoids taking into account the activations with lower values. This intuition can be interpreted as the network paying attention to the most important neurone in the network.

The algorithm for creating the vector mask $\mathbf{m}$ is similar to DropAct, and is described below. We represent the input to the dropout layer as $\mathbf{x}$.

Algorithm 2: DropInv

$\mathcal{I} = \{1, \ldots, N\}$;
$m_i = 1$;
$p_i \sim 1/x_i$ if $x_i > 0$, else $p_i = 0$;
Rescale probabilities accordingly;
for *iter in* $[1, \ldots, k]$ **do**
  Sample j from $\mathcal{I}$ with corresponding probabilities;
  $\mathcal{I} \leftarrow \mathcal{I} - j$;
  $m_j = 0$;
  Rescale probabilities accordingly;
end

3.3. DropCorr

This dropout technique is similar to the previous dropouts, but we sample m_i not using the multinomial distribution, but from a multivariate Gaussian. We want to deactivate neurone being correlated at the same time. Therefore, in order to estimate the covariance matrix, we use maximum likelihood estimation:

$$\mathbf{S} = \sum_i \left(f(\mathbf{Wx}) - \overline{f(\mathbf{Wx})}\right)\left(f(\mathbf{Wx}) - \overline{f(\mathbf{Wx})}\right)^T,$$

where $\overline{f(\mathbf{Wx})})$ is the mean value of $f(\mathbf{Wx})_i$.

The algorithm for creating the vector mask $\mathbf{m}$ is slightly different from DropAct and DropInv, and is described below. We represent the input to the dropout layer as $\mathbf{x}$.

Algorithm 3: DropCorr

$\mathcal{I} = \{1, \ldots, N\}$;
$m_i = 1$;
$\mathbf{x}_c = \mathbf{x} - \overline{\mathbf{x}}$;
$\mathbf{S} = \sum_n \mathbf{x}_c \mathbf{x}_c^T$;
Sample vector s from $\mathcal{N}(0, \mathbf{S})$;
for *iter in* $[1, \ldots, k]$ **do**
  Take j from $\mathcal{I}$ that maximizes $\mathbf{s}_j$;
  $\mathcal{I} \leftarrow \mathcal{I} - j$;
  $m_j = 0$;
end

4. Experiments

In this section, we describe the experiments we run to test the dropout layers compared with other dropout algorithms. We compare our techniques with Standard Dropout, Gaussian Dropout and Variational Dropout. We first analyze their performance on MNIST and CIFAR-10 datasets, wich are widely used for classification tasks in computer vision, and then compare the methods considering the time they take to run during training time.

Implementation details

The models we use are convolutional networks with two fully connected hidden layers. Each hidden layer has a ReLU activation function and is concatenated with a dropout layer.

The loss function that we use is the Cross Entropy Loss. We use Adam optimizer for updating parameters, with minibatch SGD and a batch size of 100. The learning rate was set to 5×10^{-4} and weight decay was set to 10^{-3}.

The main library used for this work is Pytorch.

Each dropout algorithm was tested within the same network architecture, using the same hyperparameters and the same seed initializer.

MNIST

MNIST dataset contains handwritten digits in 28×28 grayscale images. The training set contains $60,000$ samples with labels, while the testing set contains $10,000$ samples with labels. We use a fixed validation set of 5% for checking the performance of the network during training.

The architecture used to solve de MNIST classification problem is:

$$\begin{aligned}
&\text{In}_{(1\times28\times28)} \rightarrow \text{Conv}_{(16|3)} \rightarrow \text{ReLU} \rightarrow \\
&\text{Conv}_{(32|3)} \rightarrow \text{ReLU} \rightarrow \text{Pool}_{(2)} \rightarrow \\
&\text{FC}_{(6,272|1,000)} \rightarrow \text{ReLU} \rightarrow \textbf{Dropout}_{(0.2)} \rightarrow \\
&\text{FC}_{(1,000|1,000)} \rightarrow \text{ReLU} \rightarrow \textbf{Dropout}_{(0.2)} \rightarrow \text{Out}_{(1,000|10)}
\end{aligned}$$

Here, the first subindex of the Conv layers indicates the number of filters, and the second index indicates the size of the filters. For all the dropout layers, we set $p = 0.2$. For the Dropout layers, we tested the performance of our algorithms DropAct, DropInv and DropCorr compared to Standard Dropout, Gaussian Dropout and Variational Dropout, since they are the most commonly used dropout layers.

In the Figure 4 we plot values for validation accuracy using the mentioned dropout layers. We use the blue color for our dropout techniques, while the red color is for existing dropouts. We can see that the Standard Dropout layer makes the network achieve the highest validation accuracy, but Gaussian, DropCorr and DropInv achieve an accuracy almost as high as the Standard. In addition, we can see that DropAct is the one that has the biggest difficulty to increase the validation accuracy. This is produced by the fact that DropAct deactivates neurone having higher values. Consequently, since the *most important* values during the forwarding pass are turned off, the network is likely to not learn at the same pace as the other techniques.

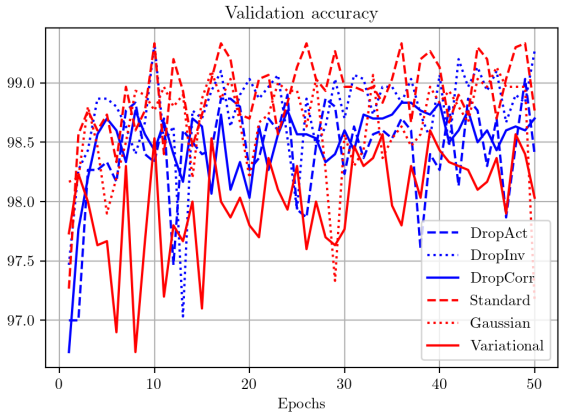

Figure 1. Plot of validation accuracy values for different dropout techniques, using the same network architecture in MNIST.

However, when we compare the performance of each ar-

chitecture on the test set in Table 1, we see a slightly better performance in the DropInv layer. Together with DropCorr, they are the networks that generalize the more, since they achieve the best performances. However, it is not a significant improvement compared with the Standar Dropout technique. On the other hand, DropAct also has a good performance over test set, despite the fact that it could not achieve a high validation accuracy. In other words, this technique also makes the network perform with good generalization.

Dropout Technique	% Test Accuracy
DropAct	98.43
DropInv	99.10
DropCorr	98.60
Standard	98.49
Gaussian	96.18
Variational	97.88

Table 1. Values of accuracies over MNIST test data for different dropout layers in the same architecture.

Each one of the implemented dropout layers has different training times. In particular, the Standar Dropout has the cheapest computational cost since it only creates a vector $\mathbf{m}$ of Bernoulli samples. Gaussian and Variational Dropouts are computationally more expensive than Standard, since they sample a vector $\mathbf{m}$ of independent normal distributed variables. DropAct and DropInv, on the other hand, sample the mask vector using a multinomial distribution, which is even computationally more expensive than the previous techniques. Finally, DropCorr has the most complex operations, since it needs to estimate the correlation matrix in each iteration.

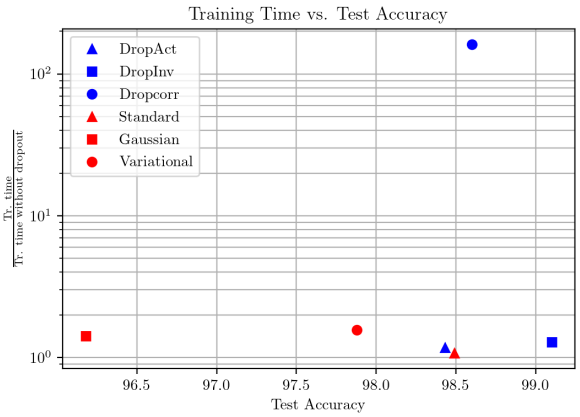

Figure 2. Plot of accuracy over MNIST test set versus training time compared to the architecture without dropout, for different dropout techniques, using the same architecture.

The computational cost of each dropout technique in Figure 4 is represented in the y-axis, while the x-axis represent the test accuracy. We can note that the training time for DropCorr is extremely larger than the rest of the techniques because of the reasons mentioned above. The Standard Dropout is much more efficient than the rest of the techniques since it runs almost in the same time as the architecture without dropouts and performs almost as well as the rest of the techniques. In addition, we observe that DropAct and DropInv trained slightly faster than Gaussian and Variational.

CIFAR-10

CIFAR-10 dataset contains 32×32 colored images of 10 different classes, such as dogs, airplanes, ships, cars, etc. The training set contains $50,000$ samples with labels, while the testing set contains $10,000$ samples with labels. Again, we use a validation set of 5% for checking the performance of the network during training.

For this classification problem, we use a deeper convolutional architecture concatenated with a fully connected network. We use BatchNormalization layers [2] to avoid the activation values collapse to zero:

$$
\begin{aligned}
&\text{In}_{(3\times32\times32)} \rightarrow \text{Conv}_{(32|3)} \rightarrow \text{ReLU} \rightarrow \text{BN} \rightarrow \\
&\text{Conv}_{(32|3)} \rightarrow \text{ReLU} \rightarrow \text{BN} \rightarrow \text{Pool}_{(2)} \rightarrow \\
&\textbf{Dropout}_{(0.2)} \rightarrow \text{Conv}_{(64|3)} \rightarrow \text{ReLU} \rightarrow \text{BN} \rightarrow \\
&\text{Conv}_{(64|3)} \rightarrow \text{ReLU} \rightarrow \text{BN} \rightarrow \text{Pool}_{(2)} \rightarrow \\
&\textbf{Dropout}_{(0.3)} \rightarrow \text{Conv}_{(128|3)} \rightarrow \text{ReLU} \rightarrow \text{BN} \rightarrow \\
&\text{Conv}_{(128|3)} \rightarrow \text{ReLU} \rightarrow \text{BN} \rightarrow \text{Pool}_{(2)} \rightarrow \\
&\textbf{Dropout}_{(0.4)} \rightarrow \text{FC}_{(2048|256)} \rightarrow \text{ReLU} \rightarrow \\
&\textbf{Dropout}_{(0.5)} \rightarrow \text{FC}_{(256|256)} \rightarrow \text{ReLU} \rightarrow \\
&\textbf{Dropout}_{(0.5)} \rightarrow \text{Out}_{(256|10)}
\end{aligned}
$$

We run experiments on CIFAR-10 similar to the experiments over MNIST dataset, checking performance on the test set and analyzing curves for validation accuracy during training. For this network, we use DropAct and DropInv, and compare them wiht Standard Dropout and Gaussian Dropout. DropCorr was not implemented in this architecture since calculating the correlation matrix in each iteration during training for each dropout layer would result in a very large period of time, as we noted in the previous section. Also, Variational Dropout was also not implemented since it was designed to be included only in fully connected layers, and we would not be able to make a fair comparison with the other dropouts because of that difference in the architectures.

In Figure 4 we show the validation accuracy for the mentioned dropout techniques. We can observe similar tendencies in the curves as the ones seen in the validation accuracy for MNIST dataset. The Standard Dropout is the technique that achieves the highest validation accuracy at the beginning, but DropInv and Gaussian reach similar values of validation accuracy a few epochs after. Moreover, Gaussian

outperforms the rest of the dropouts at the end of the training. DropAct, as seen in the MNIST dataset, is the one that has the biggest difficulty to increase the validation accuracy, but after 100 epochs it achieves a validation accuracy similar to the other techniques.

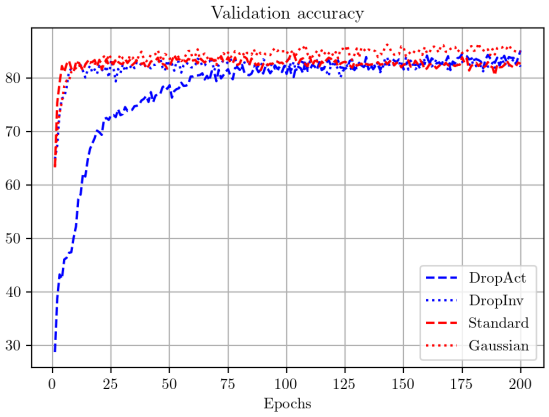

Figure 3. Plot of validation accuracy values for different dropout techniques, using the same network architecture in CIFAR-10.

Also, test accuracies are presented in Table 2. Unlike the test accuracies for the MNIST dataset, the best dropout techniques over CIFAR-10 using the described architecture are the Gaussian Dropout and DropAct. However, Standard Dropout and DropAct perform very similar, with an accuracy of less than three percentual points below the best.

Dropout Technique	% Test Accuracy
DropAct	83.16
DropInv	81.57
Standard	81.40
Gaussian	84.15

Table 2. Values of accuracies over CIFAR-10 test data for different dropout layers in the same architecture.

5. Conclusion

We presented three new dropout techniques based on three different ideas, deactivating neurone with high values (DropAct), deactivating neurone with low values (DropInv) and deactivating correlated neurone at the same time (DropCorr). The three of them resulted in similar performance than the Standard Dropout technique when they are used to train networks to classify MNIST and CIFAR-10 dataset.

We saw that DropAct has difficulties to achieve a high validation accuracy since it deactivates highly active neurone. A possible way we could solve this problem is to make the layer vary the deactivation probability alternating from p to 0 in different epochs. This approach could potentially make the validation accuracy increase more in the epochs where there is no deactivation, and step away from overfitting in the epochs where there are deactivations.

On the other hand, DropInv acts like a sparsifier since it deactivates neurone with low activation values. We saw that DropInv performs almost as well as the Standard Dropout, although it is slightly slower than the Standard.

Finally, DropCorr is the technique with largest training time, since it computes an estimation of the covariance matrix in a given layer. This calculation makes the algorithm extremely slow compared with the other techniques, taking three orders of magnitude more than them. However, it performed almost as well as the DropInv or Standard techniques when tested on MNIST dataset. A potential solution for making this technique work faster is to not estimate the covariance matrix in each iteration, but do it every N epochs. N could even be variable, starting with a small number and increasing its value with time during training. The intuition behind this idea is that the covariance matrix should vary as much as the weights of the network vary. Therefore, when the network starts training, the weights are strongly updated, making necessary to estimate the covariance matrix with more frequency, but when finishing training, the weights are subtly updated.

Nevertheless, we found that Standard Dropout is still the fastest technique, and also achieves one of the best performances in both architectures we trained. DropInv, DropAct and DropCorr performs similar to Standard Dropout but they take more running time during training, making them less efficient than Standard.

References

[1] J. Ba and B. Frey. Adaptive dropout for training deep neural networks. In *Advances in neural information processing systems*, pages 3084–3092, 2013.

[2] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[3] D. P. Kingma, T. Salimans, and M. Welling. Variational dropout and the local reparameterization trick. In *Advances in neural information processing systems*, pages 2575–2583, 2015.

[4] A. Labach, H. Salehinejad, and S. Valaee. Survey of dropout methods for deep neural networks. *arXiv preprint arXiv:1904.13310*, 2019.

[5] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010.

[6] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[7] L. Wan, M. Zeiler, S. Zhang, Y. Le Cun, and R. Fergus. Regularization of neural networks using dropconnect. In *International conference on machine learning*, pages 1058–1066, 2013.

[8] S. Wang and C. Manning. Fast dropout training. In *international conference on machine learning*, pages 118–126, 2013.