DROPAUT: AUTOMATIC DROPOUT APPROACHES TO LEARN AND ADAPT DROP RATES

Anonymous authors
Paper under double-blind review

ABSTRACT

Over time, it has been shown that Dropout is one of the best techniques to fight overfitting and at the same time improve the overall performance of deep learning models. When training with Dropout, a randomly selected subset of activations are set to zero within each layer based on a hyper-parameter called drop rate. Finding a suitable drop rate can be very expensive, especially nowadays where modern neural networks contain a large number of parameters. We introduce *DropAut*, a completely data-driven extension of Dropout which enables the model to learn and adapt the drop rate based on the task and data it is dealing with. However, both Dropout and *DropAut* exploit the same drop rate for all the units of a layer, but this could be a sub-optimal solution since not all of them have the same importance. Based on this, we also propose two *DropAut* extensions called *UnitsDropAut* and *BottleneckDropAut* which additionally allow the model to learn and use different and specific drop rates for each unit of a layer. We first derived a bound on the generalization performance of Dropout, *DropAut*, *UnitsDropAut* and *BottleneckDropAut* and then we evaluated the proposed approaches using different kinds of neural models on a range of datasets, showing good improvements over Dropout in all the experiments conducted. The code is available at https://github.com/<anonymous>.

1 INTRODUCTION

In recent years, deep learning has led to considerable advances in almost all the tasks of Natural Language Processing (NLP), Computer Vision (CV) and Speech fields. These improvements can be attributed to the use of increasingly powerful neural models such as Recurrent Neural Networks (RNN) Hochreiter & Schmidhuber (1997); Cho et al. (2014), Convolutional Neural Networks (CNN) LeCun et al. (1999) and, recently, Transformers Vaswani et al. (2017). Modern architectures contain millions or billions of parameters which are needed to provide the necessary representational power to the models, but the use of these huge networks could lead to a greater probability of overfitting.

In 2012, Hinton et al. proposed a new form of regularization called Dropout Hinton et al. (2012), specifically designed to fight overfitting. With Dropout, the activations of hidden units for each training case are stochastically set to zero at training time. More specifically, each element of a layer's output is kept with probability p (keep rate), otherwise is set to zero with probability q = 1 - p (drop rate), where p (or q[1]) is a hyper-parameter. However, although in Srivastava et al. (2014) the authors have provided guidelines for setting q, it can be difficult to understand which is the correct probability to use Park & Kwak (2017). In their paper, the authors recommended using a q of 0.5 for hidden layers but, over time, architectures have evolved, increasingly preferring lower drop rates. Nowadays, the best way to find a good value for q is to try different configurations using a validation set, an approach that could be very expensive and could still lead to a sub-optimal solution.

In this paper, we propose *DropAut*, a completely data-driven extension of Dropout which allows the model to automatically learn and adapt the drop rate during the training based on the task and data it is dealing with. Basically, a *DropAut* layer works like a Dropout one but tries to learn q as a function of the inputs instead of considering it as a hyper-parameter. Furthermore, Dropout uses a constant probability for omitting units, but not all units of a layer have the same importance, some may be

[1] In modern deep learning frameworks, the hyper-parameter is actually q.

more receptive to certain features and consequently may be more important than others. Dropout will ignore this confidence and drop these units out stochastically. Based on this, we also propose two *DropAut* extensions called *UnitsDropAut* and *BottleneckDropAut* which additionally allow the model to learn and use different and specific drop rates for each unit of the layer.

By using standard datasets such as MNIST Lecun et al. (1998), CIFAR-10/100 Krizhevsky & Hinton (2009) and IMDb Movie Reviews Maas et al. (2011), we show that all three proposed approaches improve the performance of each tested networks, namely feedforwards, CNNs and Transformers, surpassing the classic Dropout in almost all the experiments conducted.

The paper is organized as follows: Section 2 describes relevant previous work; Section 3 formally describes the *Automatic Dropout* approaches; Section 4 reports experiments and results done to evaluate the proposed solutions; finally conclusions are reported in Section 5.

2 Related Work

Overfitting is still one of the biggest problems to deal with when trying to train very large networks. Over time, a wide range of techniques for regularizing the training have been developed. Dropout was introduced in Hinton et al. (2012) as a powerful form of regularization for feedforward networks and is implemented by setting hidden activations to zero with some fixed probability during the training. On the contrary, all activations are kept during the evaluation phase except that the output is scaled according to the dropout probability. This technique works well as a robust type of bagging Breiman (1996), which discourages the co-adaptation of neighboring units within the network.

DropConnect Wan et al. (2013) is an extension of Dropout also for regularizing feedforward networks. Instead of setting to zero the activations, it sets a randomly picked subset of weights within the network to zero with a fixed probability. Standout Ba & Frey (2013) is another extension of Dropout where the dropout probability for each hidden unit is computed using a binary belief network that shares parameters with the deep network. Our *UnitsDropAut* is similar in using a network to predict different dropout probability for each hidden unit but it has its own weights which are trained jointly with the deep network. Furthermore, our method does not require the scale (α) and bias (β) hyper-parameters expected by Standout. In addition, we have proven that our proposal does not work well only with feedforward networks but also with CNN and Transformer architectures.

Compared to the original work on Dropout, Srivastava et al. (2014) provided more exhaustive experimental results, also showing that applying Dropout to CNNs aided generalization. Despite this, Dropout is not often used with Convolutional layers as it appears to be less powerful than when used with Fully Connected layers Tompson et al. (2015). This can be attributed to the fact that neighbouring pixels in images share much of the same information. If any of them are dropped out then the information will likely still be passed on from the neighbouring pixels that are still active.

In an attempt to increase the effectiveness of Dropout in Convolutional layers, several variations have been proposed. Spatial Dropout Tompson et al. (2015) randomly discards entire feature maps rather than individual pixels, effectively bypassing the issue of neighbouring pixels passing similar information. Probabilistic weighted pooling Wu & Gu (2015) drops with some probability activations in each pooling region. Max-drop Park & Kwak (2017) drops the maximal activation across feature maps or channels with some probability. In the same work, the authors propose also Stochastic Dropout, another variation of Dropout whose dropout probability varies for each iteration based on a probability distribution. Similarly, our approaches change the dropout probabilities at each iteration but it is the network that decides how to (and if to) change them rather than doing it stochastically.

3 Methods

Like the classic Dropout, all the proposed approaches can be implemented as a layer of a neural network that takes the output vector of the previous layer as input and turns it into a new vector with the same dimensions but with some units disabled. This layer is called Automatic Dropout (AD) Layer and can be placed either after the input layer or after a hidden one.

During the training stage, the AD Layer predicts a vector of drop probabilities using an internal neural network called AD Network. Each item of this vector parameterizes an independent Bernoulli

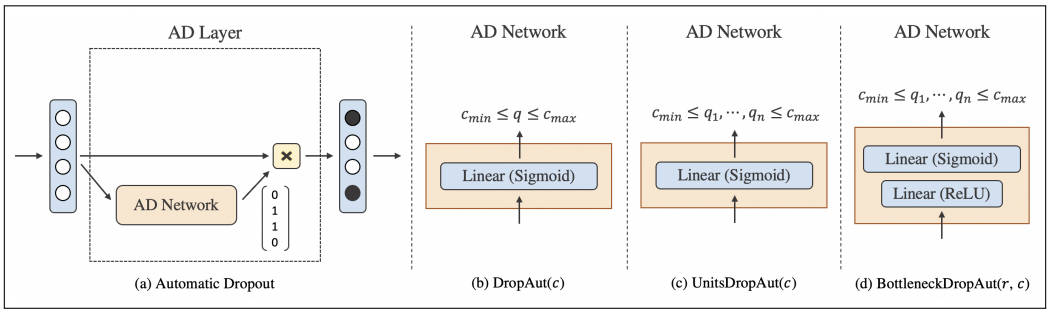

Figure 1: (a) High level overview of the Automatic Dropout (AD) layer. (b) AD Network of *DropAut* composed by one Linear layer with a single unit; q in this case is a scalar. (c) AD Network of *UnitsDropAut* composed by a single Linear layer but with n units; $\boldsymbol{q}$ here is a n-dimensional vector. (d) AD Network of *BottleneckDropAut* composed by a dimensionality-reduction Linear layer (bottleneck) and a dimensionality-increasing Linear layer; $\boldsymbol{q}$ again is a n-dimensional vector.

distribution and this allows for the sampling of a binary mask, i.e. a new vector containing only zeros and ones. This mask is then multiplied element-wise with the input vector to get the final result returned by the layer. This is illustrated in Figure 1(a). The process just described is the same for all the proposed approaches with the exception of the AD Network which is built differently and specifically by the method considered. Figure 1(b), Figure 1(c) and Figure 1(d) show the three different types of AD Network implemented respectively by *DropAut*, *UnitsDropAut* and *BottleneckDropAut*.

In addition, each proposal is equipped with an optional hyper-parameter called constraint range c which allows the narrowing of the probability values that the AD Network can predict within a specified range. See Appendix A for further details about this useful hyper-parameter.

When applying Dropout, one needs to compensate for the fact that at training time a portion of the units were deactivated. To do so, it is necessary to scale the activations of a layer after omitting some units Hinton et al. (2012); Srivastava et al. (2014). Accordingly, there exist two common strategies: scale up the retained activations of the considered layer by multiplying them by $\frac{1}{p}$ at training time[2] or scale down the activations by multiplying them by the keep rate p at test time. We have decided to implement the scale of activations at test time. We will motivate this choice in Subsection 4.1.1.

After this brief informal introduction, the approaches are formally defined below starting from the classic Dropout. For simplicity, let's consider a simple feedforward network although the following applies to any type of network and any dimensionality of the inputs.

3.1 DROPOUT

Consider a neural network with L hidden layers and let $l \in \{1, \ldots, L\}$ index them. Let $\boldsymbol{x} \in \mathbb{R}^m$ be the vector of inputs into layer l and $\boldsymbol{y} \in \mathbb{R}^n$ the vector of outputs from layer l, where m is the input features and n is the number of units of l. $\boldsymbol{W} \in \mathbb{R}^{m \times n}$ and $\boldsymbol{b} \in \mathbb{R}^n$ are the weights and biases at layer l. The standard feedforward operation can be described as (for any l):

$$\boldsymbol{y} = f\left(\boldsymbol{W}^T \boldsymbol{x} + \boldsymbol{b}\right) \tag{1}$$

where f is an activation function.

Applying Dropout to layer l, the feedforward operation becomes (for any hidden unit i):

$$\widetilde{\boldsymbol{y}} = f\left(\boldsymbol{W}^T \boldsymbol{x} + \boldsymbol{b}\right) \tag{2}$$

$$m_i \sim Bernoulli(q) \tag{3}$$

$$\boldsymbol{y} = \widetilde{\boldsymbol{y}} * \boldsymbol{m} \tag{4}$$

where $*$ denotes an element-wise product, q is the drop rate hyper-parameter and $\boldsymbol{m} \in \mathbb{R}^n$ is a vector of independent Bernoulli random variables each of which has probability q of being 0.

[2]This method is also known as Inverted Dropout and is generally the preferred way to implement Dropout in modern deep learning frameworks.

At test time, activations are scaled down multiplying them by the keep rate p, as explained above:

$$\boldsymbol{y} = p * f\left(\boldsymbol{W}^T\boldsymbol{x} + \boldsymbol{b}\right) \tag{5}$$

With Inverted Dropout instead, the equation 4 must be modified as follows:

$$\boldsymbol{y} = \frac{\widetilde{\boldsymbol{y}} * \boldsymbol{m}}{p} \tag{6}$$

and nothing is done at test time.

3.2 DROPAUT

DropAut works exactly like Dropout with the only difference that q is no longer a hyper-parameter but a scalar value that is predicted by a neural network called AD Network. For *DropAut*, the AD Network is composed only of one Linear layer with a single unit, as shown in Figure 1(b). Consequently, q in the equation 3 is now computed as:

$$q = \sigma\left(\boldsymbol{W}_{AD}^T\widetilde{\boldsymbol{y}} + \boldsymbol{b}_{AD}\right) \tag{7}$$

where $\boldsymbol{W}_{AD} \in \mathbb{R}^{n \times n'}$ and $\boldsymbol{b}_{AD} \in \mathbb{R}^{n'}$ are the weights and biases of the AD Linear layer, $n' = 1$ and σ is the sigmoid activation function. If there were multiple dimensions, q would be obtained by computing the mean of elements across all dimensions.

If the hyper-parameter c is specified, then a function $g : \mathbb{R} \to [c_0, c_1]$ is applied such that if $q < c_0$, then it is set to c_0 while if $q > c_1$, then it is set to c_1:

$$q = \min(\max(q, c_0), c_1) \tag{8}$$

At test time, q is computed by the AD Network using its learned weights and the activations are scaled down as follows:

$$\widetilde{\boldsymbol{y}} = f\left(\boldsymbol{W}^T\boldsymbol{x} + \boldsymbol{b}\right) \tag{9}$$

$$\boldsymbol{y} = (1 - \sigma\left(\boldsymbol{W}_{AD}^T\widetilde{\boldsymbol{y}} + \boldsymbol{b}_{AD}\right)) * \widetilde{\boldsymbol{y}} \tag{10}$$

3.3 UNITS DROPAUT

Unlike *DropAut*, *UnitsDropAut*'s AD Network does not predict q as a scalar value but as an n-dimensional vector $\boldsymbol{q} \in \mathbb{R}^n$. The AD Network is composed again of a single Linear layer but with n units. Therefore, the main difference with *DropAut* is that a different and specific q for each hidden unit will be generated, as shown in Figure 1(c). Equations 7, 8 and 3 now become:

$$\boldsymbol{q} = \sigma\left(\boldsymbol{W}_{AD}^T\widetilde{\boldsymbol{y}} + \boldsymbol{b}_{AD}\right) \tag{11}$$

$$q_i = \min(\max(q_i, c_0), c_1) \tag{12}$$

$$m_i \sim Bernoulli(q_i) \tag{13}$$

where $\boldsymbol{W}_{AD} \in \mathbb{R}^{n \times n}$ and $\boldsymbol{b}_{AD} \in \mathbb{R}^n$ are the weights and biases of the AD Linear layer.

3.4 BOTTLENECK DROPAUT

BottleneckDropAut is an extension of *UnitsDropAut* with an additional layer called bottleneck layer. Like the *UnitsDropAut*, the AD Network of the *BottleneckDropAut* will generate a specific drop rate for each hidden unit. Basically, the AD Network of the *BottleneckDropAut* is composed of two Linear layers placed one after the other, as shown in Figure 1(d). The first, the bottleneck layer, acts as a dimensionality-reduction layer while the second is a dimensionality-increasing layer which restores the input dimensionality. The choice of how much the bottleneck layer should reduce the dimensionality depends on the hyper-parameter reduction ratio r.

A bottleneck layer could be useful as it reduces the number of parameters of the network and improves its generalization skills. Indeed, by converting the input vector to a low-dimensional representation (i.e., an embedding), the bottleneck step forces the AD Network to learn a more general representation of the feature combinations, allowing to get a more robust network at the end.

With the addition of the second Linear layer, Equation 11 becomes:

$$\boldsymbol{v} = \delta\left(\boldsymbol{W}_{AD_1}^T \widetilde{\boldsymbol{y}} + \boldsymbol{b}_{AD_1}\right) \tag{14}$$

$$\boldsymbol{q} = \sigma\left(\boldsymbol{W}_{AD_2}^T \boldsymbol{v} + \boldsymbol{b}_{AD_2}\right) \tag{15}$$

where $\boldsymbol{v} \in \mathbb{R}^{\frac{n}{r}}$, $\boldsymbol{W}_{AD_1} \in \mathbb{R}^{n \times \frac{n}{r}}$ and $\boldsymbol{b}_{AD_1} \in \mathbb{R}^{\frac{n}{r}}$ are the otputs, weights and biases of the bottleneck layer, $\boldsymbol{W}_{AD_2} \in \mathbb{R}^{\frac{n}{r} \times n}$ and $\boldsymbol{b}_{AD_2} \in \mathbb{R}^n$ are the weights and biases of the output layer and δ is the ReLU Nair & Hinton (2010) activation function.

The idea of the *BottleneckDropAut* is similar to that of a Squeeze and Excitation (SE) layer of the SENet Hu et al. (2018) but without the Squeeze part and without the Global Average Pooling layer in the Excitation stage. Moreover, our *BottleneckDropAut* outputs a vector of independent Bernoulli random variables that is used to drop out some units of the layer rather than a vector of real numbers useful to recalibrate the features of the layer based on the correlations between them.

4 EXPERIMENTS

We conducted several experiments across a range of tasks, datasets and model architectures in order to evaluate our *Automatic Dropout* approaches. We used the TensorFlow framework Abadi et al. (2015) for implementing the architectures and ran the first two experiments on a MacBook Apple M1 Pro 14-Core-GPU workspace with 16 Gigabyte of RAM and the others on a Linux machine with a single Tesla P100 GPU with 16 Gigabyte of RAM. In all experiments, our goal was only to compare our approaches with Dropout rather than matching or exceeding the state of the art.

4.1 MNIST

MNIST Lecun et al. (1998) is one of the most popular computer vision datasets consisting of 28×28 black and white images of handwritten digits, from 0 to 9. The dataset consists of $60,000$ examples for the training set, $10,000$ for the test set and 10 classes.

On this dataset, we tried two different architectures, a feedforward and a CNN. In both cases, the only preprocessing applied is to scale the pixel values to the $[0, 1]$ range before inputting to our models. No data augmentation was used. In addition to Dropout and the proposed approaches, the Baseline was also analyzed, i.e. the base network without Dropout or *Automatic Dropout*.

4.1.1 FEEDFORWARD MODEL

For our first experiment on this dataset, we examined the same simple feedforward model of Srivastava et al. (2014) and Wan et al. (2013) composed by two Linear hidden layers with 800 units each and ReLU as activation functions. The first hidden layer takes the image pixels as input, while the second's output is fed into a 10-class softmax classification layer. Dropout or *Automatic Dropout*, when used, is placed after both the Linear hidden layers.

With this experiment, we also tried to verify whether the *Automatic Dropout* approaches work better by scaling activations during the training or test phase. The hypothesis is that, since each method produces drop rates through a neural network, it might make sense to scale the activations during the test phase in order to exploit the weights learned during the training phase. To understand if the above is correct, we examined a version with the scale during the training and a version with the scale during the test for each *Automatic Dropout* approach.

We first trained each model for 50 epochs in order to study their behavior during the training. For this training stage, which we called stuff training, we used Adam Kingma & Ba (2014) as optimizer, a batch size of 128 and the 10% of the training set as validation set; the drop rate of Dropout was set to 0.5 as in Srivastava et al. (2014) and Wan et al. (2013) and the learning rate was fixed at 0.001. For *BottleneckDropAut*, r was set to 16 as in Hu et al. (2018). Figure 2 shows the training trend of the different models during the stuff training.

The Baseline overfits almost immediately. The model overfits also with Dropout but later than the Baseline and has a much better training trend in general. The same is true for the three test scale whose trend is very similar to that of Dropout. On the other hand, the trends of the three train scales

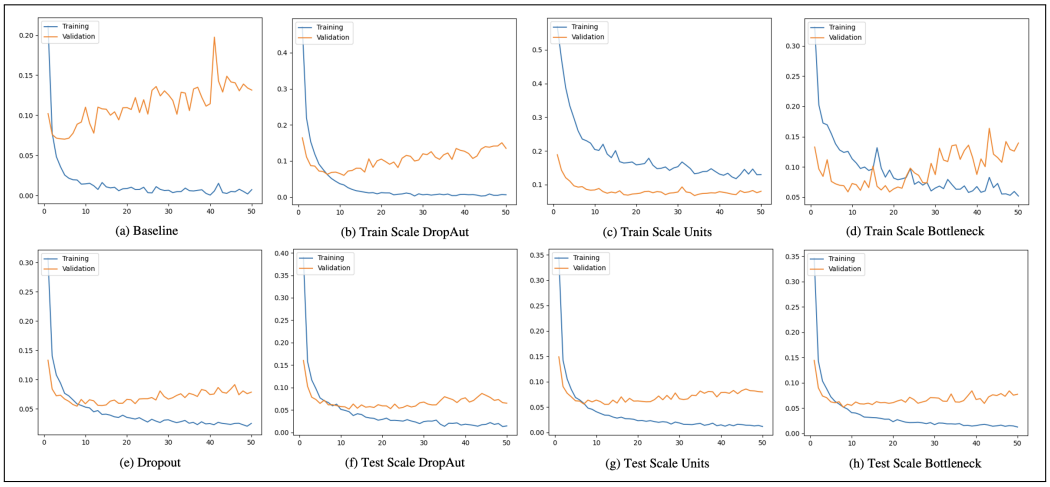

Figure 2: Training and validation loss of each model examined during the stuff training. Here, x and y axes represent respectively the number of training epochs and the values of the loss function.

are more particular, with the *Train Scale DropAut* which is the only one that is good enough; in fact, the *Train Scale UnitsDropAut* even underfits, with much lower validation loss than the training one while the *Train Scale BottleneckDropAut* has a very jagged trend, with continuous ups and downs.

In addition to the trend of the training, we also analyzed how the approaches change and adapt their drop rates during the stuff training and the sparsity property of the *Automatic Dropout*. See Appendices B.1 and B.2 for further details.

After the stuff training, we performed two more training stages on each model: first, we trained with the early stopping[3] technique to find out the epoch e in which the model performed best in terms of validation loss; second, we trained again using the whole training set for e epochs. The resulting model was then evaluated on the test set. Table 1 summarizes the results obtained on the test set.

Table 1: MNIST test accuracy with feedforward models.

	Train Time	**Best Epoch**	**Test Accuracy**	**Test Time**	**Parameters**
Baseline	6 ms/step	7	0.9801	5 ms/step	1,276,810
Dropout	7 ms/step	14	0.9834	5 ms/step	1,276,810
Train scale DropAut	8 ms/step	9	0.9821	5 ms/step	1,278,412
Test scale DropAut	7 ms/step	13	**0.9838**	5 ms/step	1,278,412
Train scale UnitsDropAut	11 ms/step	27	0.9803	5 ms/step	2,558,410
Test scale UnitsDropAut	9 ms/step	9	**0.9840**	6 ms/step	2,558,410
Train scale BottleneckDropAut	11 ms/step	10	0.9740	5 ms/step	1,438,510
Test scale BottleneckDropAut	9 ms/step	9	**0.9842**	6 ms/step	1,438,510

Basically, the three test scales achieve a better accuracy than Baseline and Dropout in a lower number of epochs. Much worse are the train scales, with the *Train Scale BottleneckDropAut* which is even worse than the Baseline. However, the *Automatic Dropout* approaches are a bit slower in terms of step execution speed due to the increased computation required by the training of the AD Networks.

Given these results, we decided to scale activations at test time in the remainder of the experiments.

4.1.2 CNN MODEL

For our second experiment on this dataset, we examined a more complex network in order to study how the proposed approaches perform in terms of test accuracy compared to the Baseline and Dropout. The model chosen is called SimpleNet Hasanpour et al. (2016) and is a fairly deep

[3]Stop the training as soon as performance on a validation set starts to get worse.

CNN architecture consisting of 13 layer blocks, each of which is composed of a Convolutional layer, a Batch Normalization Ioffe & Szegedy (2015) layer and a ReLU activation function. In addition, there are also Max Pooling layers after the first 4 blocks, after another 3 blocks, after 2 more blocks and finally after still 3 blocks. The architecture is ended by a classifier composed by a Global Max Pooling layer and a Linear layer with 10 units and softmax as activation function. Dropout or *Automatic Dropout*, when used, is placed after all the Max Pooling layers.

We re-implemented the PyTorch Paszke et al. (2019) version of SimpleNet[4] in TensorFlow and trained the models for 400 epochs using Adam as optimizer and a batch size of 120. The learning rate starts from 0.001 and exponentially decays during training with a decay rate of 0.98. For Dropout, we show the results using both 0.1 and 0.2[5] as drop rate values and for *BottleneckDropout*, r was set to 16. This configuration was found using a validation set consisting of the 10% of the training set. Table 2 summarizes the details of the different models on the MNIST test set.

Table 2: MNIST test accuracy with CNN models.

	Train Time	**Test Accuracy**	**Test Time**	**Parameters**
Baseline	136 ms/step	0.9966	37 ms/step	5,492,682
Dropout_01	139 ms/step	0.9967	37 ms/step	5,492,682
Dropout_02	139 ms/step	0.9966	37 ms/step	5,492,682
DropAut	143 ms/step	**0.9970**	40 ms/step	5,493,839
UnitsDropAut	148 ms/step	**0.9975**	43 ms/step	5,772,362
BottleneckDropAut	151 ms/step	**0.9975**	47 ms/step	5,528,722

Dropout does not improve the performance of the model, actually following the assertion of the Ioffe & Szegedy (2015)'s authors that when Batch Normalization is used, it is not useful to use Dropout too. On the contrary, all the three proposed approaches seem to work better than Dropout together with Batch Normalization, effectively increasing the accuracy of the models at the cost, however, of some training and test speed. In particular, *UnitsDropAut* and *BottleneckDropAut* show good improvements over both the Baseline and the classic Dropout.

4.2 CIFAR

The CIFAR-10 and CIFAR-100 Krizhevsky & Hinton (2009) datasets consist of $60,000$ colour images of size 32×32 pixels. CIFAR-10 has 10 classes while CIFAR-100 contains 100 classes. Each dataset is split into a training set with $50,000$ images and a test set with $10,000$ images.

On these datasets, we studied how the proposed approaches perform in terms of test accuracy compared to the Dropout using a Transformer architecture known as Vision Transformer (ViT) Dosovitskiy et al. (2021). However, in their paper the authors mention that ViT is very data-hungry and pre-training it on a large-sized dataset is essential to achieve state of the art performances on medium-sized datasets like CIFAR. Since we were not interested in the state of the art, we implemented a version called SL-ViT[6] Lee et al. (2021) that allows the training even on smaller datasets.

Before inputting the images to our models, both datasets were normalized using per-channel mean and standard deviation. Data augmentation was applied with the following schema: resizing of the images from 32×32 to 72×72; flipping images horizontally; introducing 2% of rotation variation; introducing 20% of vertical and horizontal zoom.

4.2.1 VISION TRANSFORMER: CIFAR-10

For our third experiment on the CIFAR-10 dataset, the Baseline coincides with the network with Dropout, as the basic model already has Dropout with a drop rate of 0.1. In particular, following Dosovitskiy et al. (2021), Dropout or *Automatic Dropout* is placed after every Linear layer except for the qkv-projections and directly after adding positional to patch embeddings.

[4]Available on github at https://github.com/Coderx7/SimpleNet_Pytorch.

[5]0.2 is actually the drop rate value used by the authors.

[6]Available on github at https://github.com/aanna0701/SPT_LSA_ViT.

According to Lee et al. (2021), the number of layers (i.e., Transformer blocks) was set to 9, the hidden size to 192, the number of attention heads to 12 and the hidden size of the MLP to 384. Moreover, the patch size of the patch embedding layer was set to 8 and we added learnable positional encodings to the patch embeddings to retain positional information. Note that the resize of the images allowed us to exploit a higher sequence length (i.e., number of patches) of 81 rather than 16.

We trained the models for 200 epochs using AdamW Loshchilov & Hutter (2019) as optimizer with a base learning rate of 0.003, warmup over the first 10 epochs and cosine learning rate decay. We used a batch size of 128, a label smoothing Szegedy et al. (2016) of 0.1 and a weight decay of 0.0001. For *BottleneckDropAut*, r was set to 16. Moreover, unlike the previous experiments, we decided to also specify a value for the constraint range c hyper-parameter, see Equation 8. In particular, we trained and tested the models with *DropAut*, *UnitsDropAut* and *BottleneckDropAut* either by not using c or by setting c to $[0.0, 0.2]$ or to $[0.0, 0.1]$. The hypothesis here is that some steps that a Transformer performs on a vision dataset, such as the attention one, are crucial to understand the relationships between the input tokens. Consequently, having too high drop rates may mean that there are higher probabilities to drop essential information.

Table 3: CIFAR-10/100 test accuracy with SL-ViT models.

	Train Time		**Test Accuracy**		**Test Time**		**Parameters**	
	CIFAR-10	**CIFAR-100**	**CIFAR-10**	**CIFAR-100**	**CIFAR-10**	**CIFAR-100**	**CIFAR-10**	**CIFAR-100**
Dropout	560 ms/step	570 ms/step	0.8639	0.6028	189 ms/step	190 ms/step	17,532,691	17,550,061
DropAut	563 ms/step	575 ms/step	0.8170	0.5187	204 ms/step	205 ms/step	17,538,824	17,556,194
DropAut_02	564 ms/step	577 ms/step	0.8590	0.6019	205 ms/step	207 ms/step	17,538,824	17,556,194
DropAut_01	564 ms/step	577 ms/step	**0.8645**	**0.6070**	205 ms/step	207 ms/step	17,538,824	17,556,194
UnitsDropAut	577 ms/step	587 ms/step	0.8174	0.5541	225 ms/step	227 ms/step	19,293,589	19,310,959
UnitsDropAut_02	579 ms/step	589 ms/step	0.8593	0.5974	226 ms/step	229 ms/step	19,293,589	19,310,959
UnitsDropAut_01	579 ms/step	589 ms/step	**0.8665**	**0.6085**	226 ms/step	229 ms/step	19,293,589	19,310,959
BottleneckDropAut	583 ms/step	595 ms/step	0.8241	0.5597	230 ms/step	232 ms/step	17,758,435	17,775,805
BottleneckDropAut_02	584 ms/step	597 ms/step	0.8571	0.5996	231 ms/step	234 ms/step	17,758,435	17,775,805
BottleneckDropAut_01	584 ms/step	597 ms/step	**0.8660**	**0.6150**	231 ms/step	234 ms/step	17,758,435	17,775,805

Looking at Table 3, the hypothesis seems to be correct: for a Transformer on a vision dataset with a fairly small sequence length, the use of lower drop rates is crucial in order for the model to learn well in a limited number of epochs. In fact, *DropAut*, *UnitsDropAut* and *BottleneckDropAut* without c have lower performance than Dropout; specifying a c with a tight upper limit such as 0.1 instead not only improves the performance but also makes them superior to Dropout. This means that the model benefits from using drop rates even lower than 0.1, or potentially not using Dropout at all.

4.2.2 Vision Transformer: CIFAR-100

To verify whether the CIFAR-10 hypothesis applies to another vision dataset, we examined exactly the same models and training configurations also on the CIFAR-100 dataset during our fourth experiment. Table 3 shows the accuracy of the models on the CIFAR-100 test set. Obviously, the accuracies are generally lower than CIFAR-10 because CIFAR-100 is a more complex dataset, which requires much more fine-grained recognition as some classes are very visually similar. Nevertheless, the behavior of the models follows that already observed for CIFAR-10, with the three approaches with the upper limit set at 0.1 which carry to the best accuracy levels again.

4.3 IMDb Movie Reviews

IMDb Movie Reviews is a binary sentiment analysis dataset consisting of $50,000$ reviews from the Internet Movie Database (IMDb) labeled as positive or negative. The dataset has an even number of positive and negative reviews and is split into a training and test set with $25,000$ reviews each.

With our fifth experiment, we examined a Transformer Encoder architecture similar to that of BERT Devlin et al. (2019). Specifically, models were trained from scratch on the dataset without pre-training using single sequences as inputs (rather than pairs of sequences packed together) and not using segment embeddings. Following Vaswani et al. (2017), the number of layers (i.e., Transformer blocks) was set to 6 and the number of self-attention heads to 8. We used a hidden size of 256 and a hidden size of the MLP of 512. Moreover, according to Vaswani et al. (2017) and Devlin et al.

(2019), Dropout or *Automatic Dropout* was applied to the output of the attention, to the output of the MLP and to the sums of the embeddings and the positional encodings. We used learnable positional encodings to retain positional information. Before inputting the sequences to our models, each text was preprocessed by removing the HTML tags and any multiple spaces and converting it to lower case. After that, as in Devlin et al. (2019), the sentences were tokenized using the WordPiece Wu et al. (2016) model with a $30,000$ token vocabulary and their length was set to a maximum of 512.

We trained the models for 50 epochs using Adam as optimizer with a base learning rate of 0.0001, warmup over the first 5 epochs and linear learning rate decay. We used a batch size of 64, a drop rate of 0.1 for Dropout and the same c ranges of the previous experiment. For *BottleneckDropAut*, r was set to 16. Table 4 shows the results obtained on the IMDb test set.

Table 4: IMDb test accuracy with Transformer Encoder models.

	Train Time	**Test Accuracy**	**Test Time**	**Parameters**
Dropout	2 s/step	0.8648	484 ms/step	22,016,257
DropAut	2 s/step	**0.8801**	492 ms/step	22,019,598
DropAut_02	2 s/step	0.8746	496 ms/step	22,019,598
DropAut_01	2 s/step	0.8714	496 ms/step	22,019,598
UnitsDropAut	2 s/step	**0.8820**	503 ms/step	22,871,553
UnitsDropAut_02	2 s/step	0.8746	508 ms/step	22,871,553
UnitsDropAut_01	2 s/step	0.8713	508 ms/step	22,871,553
BottleneckDropAut	2 s/step	**0.8790**	513 ms/step	22,126,289
BottleneckDropAut_02	2 s/step	0.8704	515 ms/step	22,126,289
BottleneckDropAut_01	2 s/step	0.8676	515 ms/step	22,126,289

In this case, all the approaches overcame Dropout but in particular those without specifying c, which in previous experiments were unsuccessful, proved to be the best with a gap of about 2% compared to Dropout. This different behavior despite the similar architecture may be due to the type of dataset (vision for CIFAR versus textual for IMDb) or to the different sequence lengths (81 for CIFAR against 512 for IMDb). However, we leave a more careful analysis of this topic to future work.

5 Conclusion and Future Work

In this work, we presented three Dropout extensions able to automatically detect how to regularize the activities of the layers of a neural model. We proposed *DropAut* which predicts the drop rate q using an internal neural network called AD Network, *UnitsDropAut* whose AD Network predicts different drop rates for each unit of the layer and *BottleneckDropAut* which is similar to *UnitsDropAut* but with a slightly different AD Network architecture.

We have proven that our methods have similar behaviors to the classic Dropout using different neural models on a range of datasets, showing good training trends and improvements in performance. Moreover, we have seen that *DropAut*, *UnitsDropAut* and *BottleneckDropAut* work well also together with other widely used techniques such as data augmentation and Batch Normalization. The other side of the coin is that they are slower though this is compensated by the fact that the expensive phase of finding a good value for the hyper-parameter q is no longer necessary. In addition, we have shown that the use of specific drop rates for the different units of a layer brings performance benefits.

Although *UnitsDropAut* turned out to be the best approach from the experiments, it is the heaviest in terms of parameters. In general, if model size is not an issue, *UnitsDropAut* might be the ideal solution; if instead memory is an important point, *BottleneckDropAut* could be better. On the other hand, if both speed and memory are crucial, *DropAut* is the solution to be preferred.

While these initial results are encouraging, many challenges remain. One is to further investigate the different behavior found on datasets of different types with a Transformer architecture examining other textual and vision datasets. Another topic might be to investigate deeper architectures for the AD Network to see if it can lead to further improvement in performance. Finally, an extension of this work could be the deactivation of entire layers of the network rather than some of its units. This could help bring even more generalization and robustness to the model.

REFERENCES

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

Jimmy Ba and Brendan Frey. Adaptive dropout for training deep neural networks. In C.J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger (eds.), *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013. URL `https://proceedings.neurips.cc/paper/2013/file/7b5b23f4aadf9513306bcd59afb6e4c9-Paper.pdf`.

Leo Breiman. Bagging Predictors. *Machine Learning*, 24:123–140, 1996. doi: 10.1007/BF00058655.

Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder–decoder approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pp. 103–111, Doha, Qatar, oct 2014. Association for Computational Linguistics. doi: 10.3115/v1/W14-4012. URL `https://aclanthology.org/W14-4012`.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL `https://aclanthology.org/N19-1423`.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=YicbFdNTTy`.

Seyyed Hossein Hasanpour, Mohammad Rouhani, Mohsen Fayyaz, and Mohammad Sabokrou. Lets keep it simple, using simple architectures to outperform deeper and more complex architectures, 2016. URL `https://arxiv.org/abs/1608.06037`.

Mohamed Hebiri and Johannes Lederer. Layer sparsity in neural networks, 2020. URL `https://arxiv.org/abs/2006.15604`.

Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012. URL `http://arxiv.org/abs/1207.0580`.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997. doi: 10.1162/neco.1997.9.8.1735.

Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7132–7141, 2018. doi: 10.1109/CVPR.2018.00745.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis Bach and David Blei (eds.), *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 448–456, Lille, France, 07–09 Jul 2015. PMLR. URL `https://proceedings.mlr.press/v37/ioffe15.html`.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical Report 0, University of Toronto, Toronto, Ontario, 2009.

Yann Lecun, Leon Bottou, Y. Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791.

Yann LeCun, Patrick Haffner, Lëon Bottou, and Yoshua Bengio. Object recognition with gradient-based learning. In *Shape, Contour and Grouping in Computer Vision*, pp. 319–345. Springer Verlag, 1999. ISBN 3540667229. doi: 10.1007/3-540-46805-6_19.

Seung Hoon Lee, Seunghyun Lee, and Byung Cheol Song. Vision transformer for small-size datasets, 2021. URL https://arxiv.org/abs/2112.13492.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pp. 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P11-1015.

Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pp. 807–814, Madison, WI, USA, 2010. Omnipress. ISBN 9781605589077.

Sungheon Park and Nojun Kwak. Analysis on the dropout effect in convolutional neural networks. In Shang-Hong Lai, Vincent Lepetit, Ko Nishino, and Yoichi Sato (eds.), *Computer Vision – ACCV 2016*, pp. 189–204, Cham, 2017. Springer International Publishing. ISBN 978-3-319-54184-6.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2818–2826, 2016. doi: 10.1109/CVPR.2016.308.

Jonathan Tompson, Ross Goroshin, Arjun Jain, Yann LeCun, and Christoph Bregler. Efficient object localization using convolutional networks. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 648–656, 2015. doi: 10.1109/CVPR.2015.7298664.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In Sanjoy Dasgupta and David McAllester (eds.), *Proceedings of the 30th International Conference on Machine Learning*, Proceedings of Machine Learning Research, pp. 1058–1066, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR. URL https://proceedings.mlr.press/v28/wan13.html.

Haibing Wu and Xiaodong Gu. Towards dropout training for convolutional neural networks. *Neural networks : the official journal of the International Neural Network Society*, 71:1–10, 07 2015. doi: 10.1016/j.neunet.2015.07.007.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation, 2016. URL https://arxiv.org/abs/1609.08144.

A THE CONSTRAINT RANGE HYPER-PARAMETER

Each proposed approach gives you the possibility to narrow the probability values that the AD Network can predict within a specified range through the hyper-parameter constraint range c. By default, its value coincides with the probability range $[0.0, 1.0]$ (i.e., the output of the sigmoid function) but in some cases it may be useful to specify a different range in which the network can search.

For example, in Srivastava et al. (2014) the dropout probability is much lower (0.1 or 0.2) if Dropout is applied to the input layer rather than to a hidden one. In this case, it may be useful to set an upper limit that the predicted probabilities cannot exceed by specifying a c of $[0.0, 0.2]$ or even $[0.0, 0.1]$.

Probably, during the training, the AD Network will be able by itself to understand that the values to be predicted must be lower in such cases, but specifying the range in advance allows the faster finding of good probabilities, without looking into spaces that may not be interesting. In addition, it also limits the risk of the AD Network being trapped in non-useful places and therefore never finding an optimal solution.

It is also important to note that specifying a very small c like $[0.0, 0.1]$ is not equivalent to using the classic Dropout with a drop rate $q = 0.1$. This is because the output of the AD Network is only constrained in a range whose maximum value is 0.1 but it does not necessarily mean that it must always predict 0.1 as a drop rate: it could in fact also predict a lower value or even decide not to use the Dropout at all ($q = 0.0$).

B ADDITIONAL ANALYSES

This section shows other details of the analysis done on the *Automatic Dropout* approaches during the stuff training on the MNIST dataset with the simple feedforward model of Subsection 4.1.1.

B.1 TREND OF DROP RATES

One of the great differences of *DropAut*, *UnitsDropAut* and *BottleneckDropAut* compared to the classic Dropout is that the drop rates are predicted by an internal neural network called AD Network. Consequently, the deep network can choose or change them during the course of a training according to its needs and based on the data it is dealing with.

Since DropAut generates a single drop rate which is the same for all units, it was possible to study the trend of the predicted drop rate during the training epochs through the simple trend charts shown in Figure 3.

It is interesting to see how behaviors change according to the chosen activation scale method. For the *Train Scale DropAut*, both rates decrease almost exponentially during the epochs until reach practically 0 from the 20th epoch. In practice, from epoch 20, the network becomes the Baseline

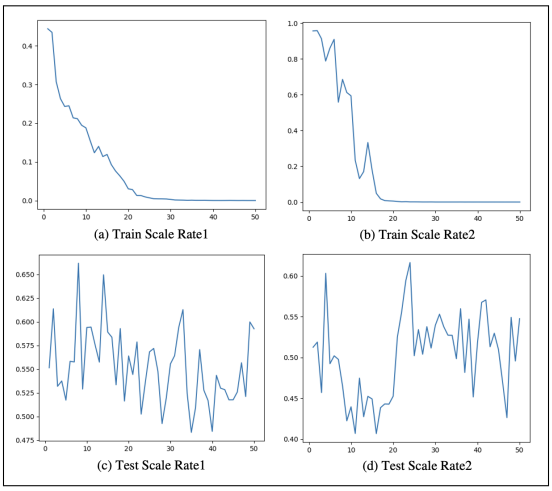

Figure 3: Drop rate trend of (a) the *Train Scale DropAut* layer placed after the first hidden layer; (b) the *Train Scale DropAut* layer placed after the second hidden layer; (c) the *Test Scale DropAut* layer placed after the first hidden layer and (d) the *Test Scale DropAut* layer placed after the second hidden layer, during the stuff training. For each chart, x and y axes represent respectively the number of training epochs and the values of the drop rate.

(with the addition of the activations scale). On the other hand, the charts of the *Test Scale DropAut* are more interesting, where drop rates vary over the epochs in a range between $[0.48, 0.68]$ for the first layer and $[0.40, 0.65]$ for the second layer. However, towards the end of the training, the predicted drop rate is generally higher than 0.5 for both layers, showing that Srivastava et al. (2014); Wan et al. (2013)'s choice of 0.5 as the hyper-parameter value for Dropout could be close to the correct solution but presumably sub-optimal.

UnitsDropAut and *BottleneckDropAut* instead predict a different drop rate value for each unit of the layer, so it was necessary to draw heatmaps to study the trend of their predicted drop rates. Figure 4 shows the heatmaps for *UnitsDropAut*, where the darker the color the lower the drop rate and vice versa.

Unlike the *Train Scale DropAut* where the drop rate becomes practically zero after a while, the opposite happens with the *Train Scale UnitsDropAut*. Especially for the second hidden layer, the drop rates are quite high during the whole training. The second *Train Scale UnitsDropAut* layer practically does not let almost anything go forward. This explains the underfitting shown in Figure 2. Also in this case, the trends of the test scales are more stable. It is interesting to see that some drop rates are very high for some units and this is because the network has probably understood that those units are not very important. Conversely, much lower drop rates are assigned by the network to the units that it considers most important.

Finally, Figure 5 shows the heatmaps for the *BottleneckDropAut*. The charts are similar to those of the *UnitsDropAut* with the only difference of the behaviour of the *Train Scale BottleneckDropAut* for the second hidden layer which is more similar to that of the *Train Scale DropAut* layer. In fact, here we have again very low drop rates, tending to zero during the training.

B.2 SPARSITY

Another analysis we conducted during the stuff training is the study of the sparsity of the activations after applying the *DropAut*, *UnitsDropAut* or *BottleneckDropAut* layer. Sparsity is a good and desirable property for a neural network because it can save computational resources, facilitate interpretations, and prevent overfitting Hebiri & Lederer (2020). It is verified from Srivastava et al. (2014) that in the fully connected layer, the activations are sparser when Dropout is used. Moreover, Park & Kwak (2017) confirmed that this statement also holds for CNNs in both lower and higher layers.

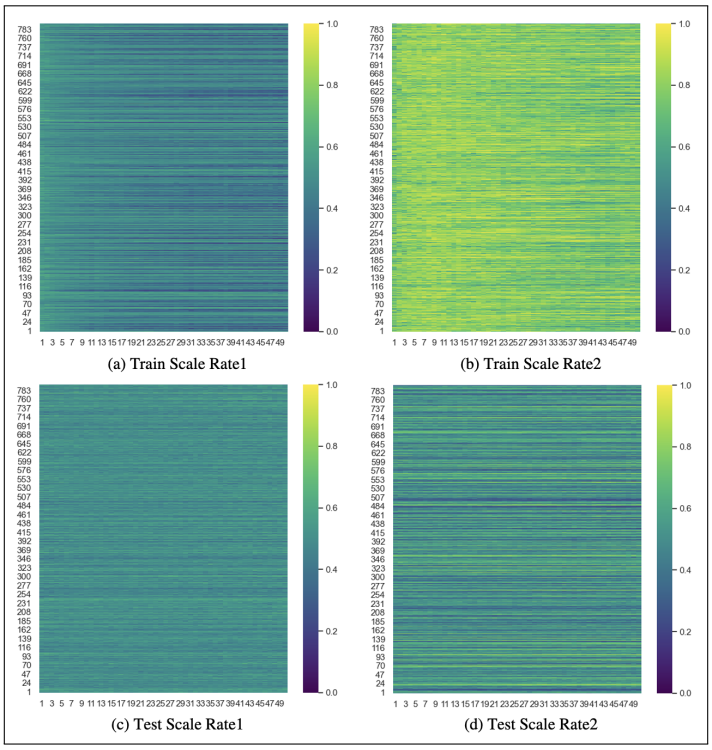

Figure 4: Drop rates trend of (a) the *Train Scale UnitsDropAut* layer placed after the first hidden layer; (b) the *Train Scale UnitsDropAut* layer placed after the second hidden layer; (c) the *Test Scale UnitsDropAut* layer placed after the first hidden layer and (d) the *Test Scale UnitsDropAut* layer placed after the second hidden layer, during the stuff training. For each chart, x and y axes represent respectively the number of training epochs and the number of units.

To verify if the proposed approaches also lead to the sparsity of the activations, we calculated the mean activations of the units in both the hidden layers of the feedforward model on a random mini-batch taken from the test set, starting from the classic Dropout shown in Figure 6.

In a good sparse model, there should only be a few highly activated units for any data case. Moreover, the average activation of any unit across data cases should be low. This is actually true for both the hidden layers. Indeed, the histogram of mean activations of the first hidden layer shows a sharp peak at zero and the histogram of the second hidden layer shows that most units have a small mean activation of about 0.3. Definitely, very few units have high activations.

In Figure 7, the histograms relating to the train scale versions of *DropAut*, *UnitsDropAut* and *BottleneckDropAut* are reported. From the histograms it is clear that all the proposed approaches create much more sparse activations than the classic Dropout, especially the *UnitsDropAut* which, as one might have expected, definitively leads to a much more sparsity than the others. This is largely due to the fact that, especially regarding the second hidden layer, the predicted drop rates of the *Train Scale UnitsDropAut* are very high (Figure 4). The sparsity of the test scale versions, shown in Figure 8, are much more similar to that of the Dropout, with also in this case the *UnitsDropAut* having the greatest sparsity.

Ultimately, we can conclude by saying that, like the classic Dropout, the proposed approaches also lead to sparse activations of a layer after being applied.

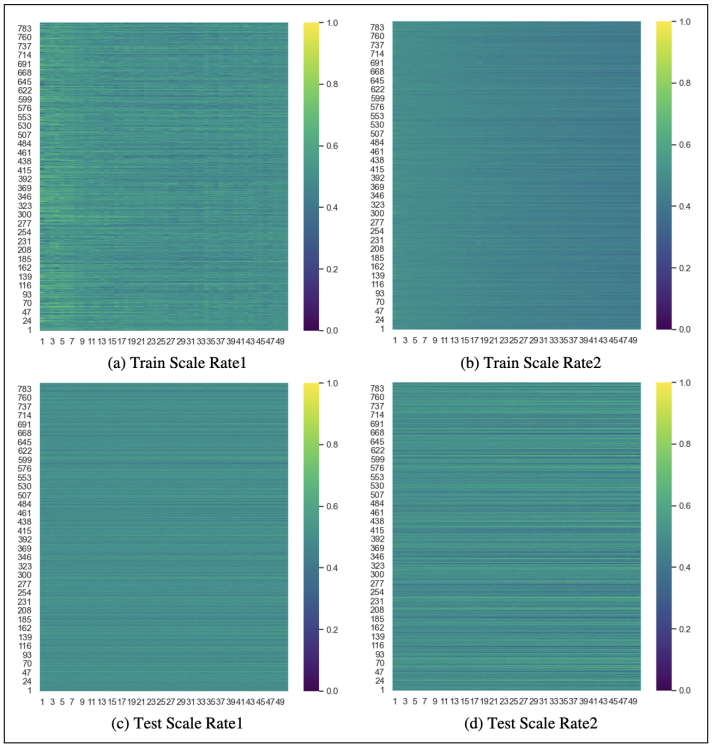

Figure 5: Drop rates trend of (a) the *Train Scale BottleneckDropAut* layer placed after the first hidden layer; (b) the *Train Scale BottleneckDropAut* layer placed after the second hidden layer; (c) the *Test Scale BottleneckDropAut* layer placed after the first hidden layer and (d) the *Test Scale BottleneckDropAut* layer placed after the second hidden layer, during the stuff training. For each chart, x and y axes represent respectively the number of training epochs and the number of units.

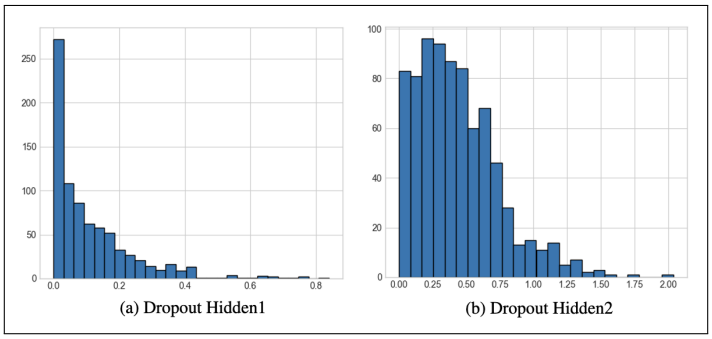

Figure 6: Histogram of mean activations of (a) the first hidden layer after applying Dropout and (b) the second hidden layer after applying Dropout. For each histogram, x and y axes represent respectively the values of mean activations and the number of units.

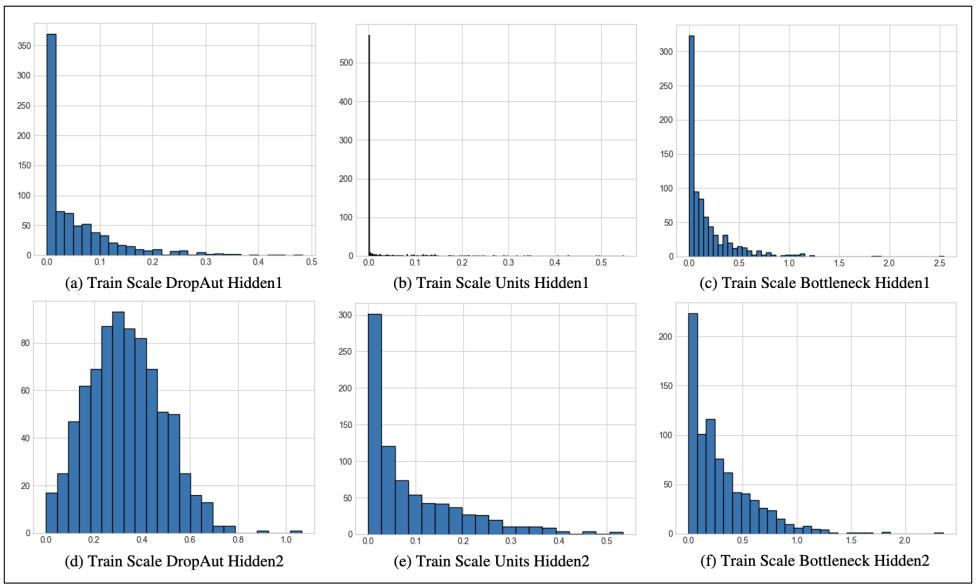

Figure 7: Histogram of mean activations of (a) the first hidden layer after applying *Train Scale DropAut*; (b) the first hidden layer after applying *Train Scale UnitsDropAut*; (c) the first hidden layer after applying *Train Scale BottleneckDropAut*; (d) the second hidden layer after applying *Train Scale DropAut*; (e) the second hidden layer after applying *Train Scale UnitsDropAut* and (f) the second hidden layer after applying *Train Scale BottleneckDropAut*.

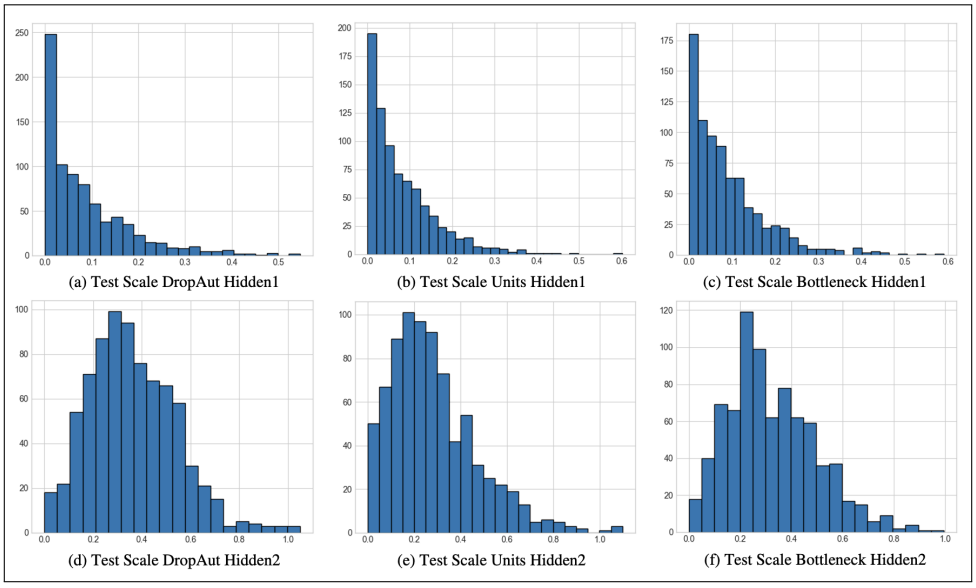

Figure 8: Histogram of mean activations of (a) the first hidden layer after applying *Test Scale DropAut*; (b) the first hidden layer after applying *Test Scale UnitsDropAut*; (c) the first hidden layer after applying *Test Scale BottleneckDropAut*; (d) the second hidden layer after applying *Test Scale DropAut*; (e) the second hidden layer after applying *Test Scale UnitsDropAut* and (f) the second hidden layer after applying *Test Scale BottleneckDropAut*.