DropAlan: An Input-Weighted Approach to Dropout Regularization

Alan Casallas*
MIT
(Dated: December 13, 2018)

Dropout and RELU-based activation functions are ubiquitous in the machine learning industry today. However, there is a discrepancy between the network that Dropout trains and the resulting network that is used for inference. Additionally, RELU neurons can suffer from the 'dead RELU' problem, where a neuron that outputs 0 during training can no longer train. To address these issues, I introduce DropAlan, a technique where no activation function is used. Instead, the dropout rate for each neuron is a function of the input $w^T x$ so that in expectation, the output is similar to a RELU unit. Experimental results on the CIFAR-10 dataset were interesting, showing a version of DropAlan to score slightly lower than traditional Dropout but converge much faster. Results on the MNIST dataset were very favorable towards DropAlan, showing it matched the performance of Dropout but converged faster.

I. INTRODUCTION

Dropout is a commonly used form of regularization for neural networks. The technique consists of randomly setting the output of selected neurons to 0 with some probability p during each training iteration. When the trained net is used for inference, all neurons are used. Neuron outputs are scaled by $\frac{1}{1-p}$ to roughly set the expectation of an output neuron to the same value it would have without Dropout. [1]

In this paper I will introduce DropAlan, a regularization technique that involves dropping a neuron during training with a probability dependent on the input to the neuron. In other words, the dropout probability for each training iteration is a function of the weighted sum of the input vector, or $p = f(w^T x)$.

In my implementation of DropAlan, the probability of dropping a neuron is 0 if the input is greater than 0, and the probability increases as the input became more negative. This approach adheres to the philosophy espoused by the recent use of RELU-based activation functions, which deterministically output 0 or a small negative value if the input is negative, and a positive output for positive inputs. In the following sections, I will present the reasons for my interest in developing DropAlan, experimental results on the CIFAR-10 and MNIST datasets, and a brief theoretical analysis of its probabilistic properties.

II. DESCRIPTION

The DropAlan technique works by randomly dropping a neuron (setting its output to 0) during each training iteration according to a probability that is a function of the input to the neuron. Two separate dropout probability functions were used in this research, which I call

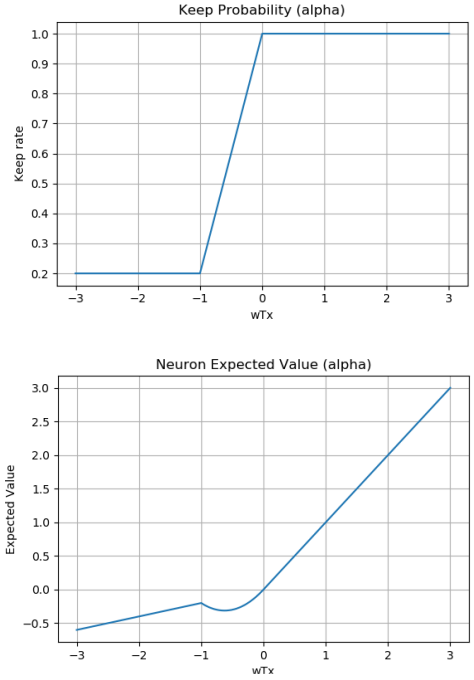

FIG. 1. α Probability Function and Expectation

probability distributions α and β:

$$p_\alpha(w^T x) = \begin{cases} .8, & w^T x < -1, \\ -.8 * w^T x, & -1 < w^T x < 0, \\ 0, & w^T x > 0. \end{cases}$$

$$p_\beta(w^T x) = \begin{cases} 1 + \frac{.2}{w^T x}, & w^T x < -1, \\ .8, & -1 < w^T x < 0, \\ 0, & w^T x > 0. \end{cases}$$

Both of these distributions attempt to emulate the philosophy of RELU activation functions in different ways. As Figure 1 shows, the 'keep' probability of distribution α exhibits a curve reminiscent to RELU-based functions,

* alancas@mit.edu

where positive input values are kept and negative input values are discarded. However, as seen in the second plot of the figure, the expectation of the neuron output with distribution α is dissimilar to a RELU activation function, most jarringly in the curve exhibited in the interval $-1 < w^T x \leq 0$.

Probability distribution β is designed such that the expectation of the neuron will be similar to a RELU-based activation funcion, as can be observed in Figure 2. The disadvantage of this distribution is that the probability requires tensor division and is thus more computationally expensive.

One of the motivations behind DropAlan was the dead RELU problem. It is well known [2] that when training a network with RELU activation functions, some neurons fall into a state where they output 0 and never leave that state. This is understandable, since the gradient of the RELU function is 0 for $w^T x < 0$. Other activation functions have been developed to deal with this problem [3], one of the most successful of which is the ELU activation function. Nevertheless, even the ELU activation function can have a small gradient for very negative values of $w^T x$.

In the DropAlan approach, no activation function is used; the output of a neuron is simply the dot product $w^T x$. However, negative values of $w^T x$ will be dropped with greater probability. Thus, although in expectation the output of neurons with negative inputs will be small, these neurons will be given a greater chance to 'escape' from their low-output state during gradient descent iterations when they are not dropped.

Prior to my experiments, I performed a literature review on variants of Dropout. DropConnect, described in [4], is a version of Dropout where individual weights, rather than neurons, are dropped randomly. Data-dependent Dropout is explored in [5]; specifically, the paper describes a method of keeping a running count of the dataset's mean and variance and changing the dropout rate according to those statistics. However, I could not find a version of Dropout similar to DropAlan, where the Dropout rate for each neuron is a function of $w^T x$, activation functions are not used, and the Dropout probability function is set to mimic an activation function in expectation.

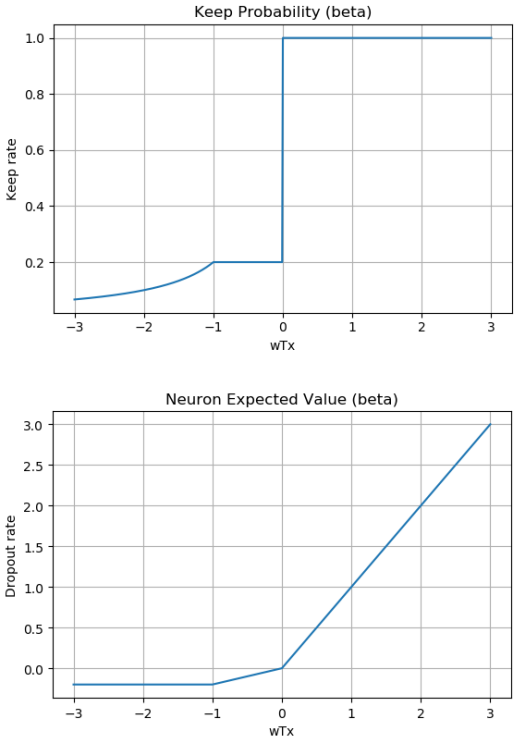

FIG. 2. β Probability Function and Expectation

III. EXPERIMENTAL RESULTS

To test the performance of DropAlan, I decided to train several neural networks on the CIFAR-10 and MNIST datasets. I planned to train four networks: 1) a basic Convolutional Neural Network composed of three convolutional layeres, followed by a 1000-unit dense layer, and a 10-unit dense output layer, 2) the same CNN, but with a dropout layer after the 1000-unit dense layer, 3) the same CNN, but with a DropAlan layer using probability distribution α as the 1000-unit dense layer, and 4) the same CNN, but with a β DropAlan layer. I emphasize that where DropAlan layers were used, no activation function was used for the corresponding dense layer. All input was subtracted by the training set mean and divided by 255.0, so that the input contained both negative and positive values. The network was trained using a Gradient Descent Optimizer with a .01 learning rate on a Sparse Softmax Cross Entropy loss function.

The purpose of this experiment was not to outperform existing state-of-the-art results on CIFAR-10, such as the results achieved by RESNET, but instead to compare how the DropAlan technique compares to traditional Dropout and what adjustments may be needed to improve performance.

Full source code and experiment results can be found in my GitHub repository: `https://github.com/acasallas/6860-final-project`

A. DropAlan Implementation

I used the Tensorflow framework to perform the experiment. Although the Tensorflow Dropout layer is not suited to a network that requires setting the dropout rate based on the value of the neuron input, I was able to re-write the Tensorflow dropout code to suit my needs. Most importantly, I implemented functionality to apply a different dropout probability to each unit. Below is the critical code, which replaces the tf.nn.dropout() method

for the purpose of an DropAlan layer. It uses a tensor of random variables of the same dimension as the input batch and effectively applies a different dropout probability to each neuron.

```
def mydropout(x, keep_prob):
    #keep_prob is a tensor of
    #probabilities between 0.0 and 1.0
    rand_tensor += tf.random.uniform(
        keep_prob.shape)
    binary_tensor = tf.floor(rand_tensor)
    ret = tf.multiply(x,binary_tensor)
    return ret
```

To calculate the drop probability (equivalent to 1-$keep_prob$) during each iteration, I enabled eager execution in Tensorflow and overrode the tf.keras.Layer.call() method. The code used to calculate a matrix of drop probabilities for both distribution α and distribution β is found below:

```
def call(self, inputs):
    #prob alpha
    rate_matrix1 =
        tf.clip_by_value(
            inputs*(-.8)
            ,0,.8)

    #prob beta
    rate_matrix2=
        tf.clip_by_value(
        1.0+tf.divide(.2,inputs),
        .8,.999)*
        tf.sign(tf.nn.relu(-1*inputs))
```

I took care to ensure my implementation was correct. I confirmed that setting all drop probabilities to 1.0 caused the network to exhibit 0.10 accuracy (and a cross entropy loss of 2.30), meaning it was randomly guessing answers. I also printed tensors while developing my code to ensure probabilities and neuron outputs were being correctly calculated.

B. Results

The table below shows the results of training the networks on the CIFAR-10 dataset. The accuracy column gives the highest score that the network achieved on a validation set, and the convergence column is the training iteration step in which the score was achieved.

Net	Accuracy	Convergence
CNN	0.724	113,000
CNN w/ Dropout	0.736	110,000
CNN w/ DropAlan, Prob α	0.699	28,000
CNN w/ DropAlan, Prob β	0.704	35,000
CNN w/ DropAlan, Prob β (scaled)	0.723	42,000

TABLE I. CIFAR-10 dataset results

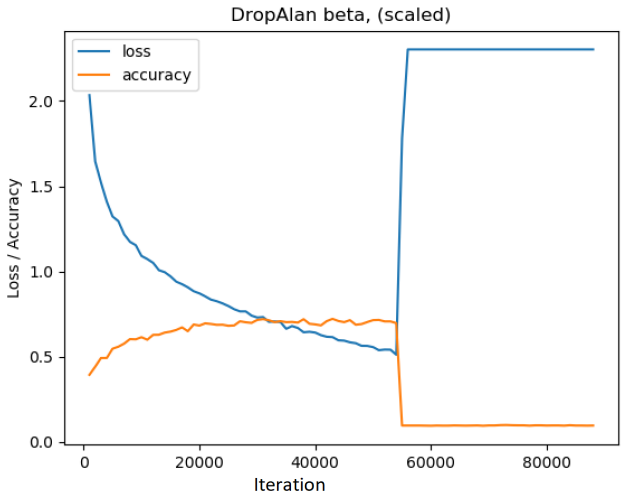

FIG. 3. DropAlan loss and accuracy

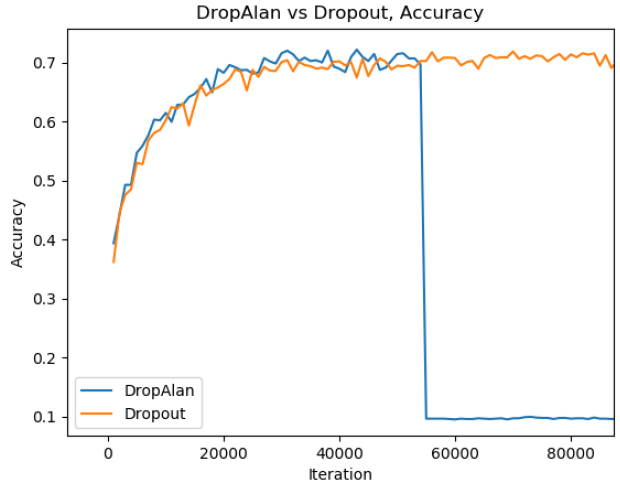

FIG. 4. Dropout vs. DropAlan Accuracy

As the table shows, the first two DropAlan networks did not score as well as traditional CNNs, although they converged to their peak score sooner. However, the last entry in the table 'CNN w/ DropAlan, Prob β (scaled)' achieved a score comparable to a CNN without Dropout, but, more significantly, converged to its maximum score much sooner than traditional CNNs did.

This last network was an accidental experiment, as I forgot to remove the snippet of code from the Tensorflow base that scaled the output of a dropout neuron by $\frac{1}{1-p}$. Thus, this network used DropAlan, but with the scaling technique of traditional Dropout. Although this network produced a better result than the other two DropAlan networks, it crashed after about 55,000 iterations. Figure 3 shows a graph of the loss and accuracy of this network. At a certain point, the network lost its ability to inference, as the loss increased to 2.30 and the accuracy

decreased to 0.10, both indicative of a complete lack of inference for a dataset with 10 labels.

I cannot yet say with certainty why the training crashed. One possibility is that there was division by zero in my code, since the probability function contains an interval for which the drop rate is $1 + \frac{.2}{w^T x}$. Another possibility is that the drop rate neared 1.0 for too many neurons, similar to the dead RELU problem of traditional networks. This is possible with the β distribution because the drop probability tends to 1.0 as $w^T x$ tends to negative infinity. It is worth noting that I tried running the network again, with the same result: a crash at around 50,000 iterations.

However, despite crashing, this DropAlan network converged faster than the traditional Dropout CNN. While DropAlan reached its peak accuracy of 0.723 at around iteration 42,000, the Dropout CNN only had a peak accuracy of .704 by that iteration. Figure 4 plots the validation accuracies of the CNN w/ Dropout vs. the DropAlan network. It is evident that before the DropAlan network crashed, it had converged to its peak performance faster.

The table below shows the results of training the network on the MNIST dataset:

Net	Accuracy	Convergence
CNN	0.9888	84,000
CNN w/ Dropout	0.9906	115,000
CNN w/ DropAlan, Prob α	0.9843	96,000
CNN w/ DropAlan, Prob β	0.9819	97,000
CNN w/ DropAlan, Prob β (scaled)	0.9906	102,000

TABLE II. MNIST dataset results

In this case, the last DropAlan network beat the performance of the first CNN and exactly matched the performance of the traditional Dropout network. Furthermore, it once again converged to its peak performance faster than the Dropout network. The DropAlan network did not crash during training as it did on the CIFAR-10 training set.

IV. THEORETICAL ANALYSIS

One of the motivations behind my development of DropAlan was the possibility of developing a variant of Dropout that was easier to analyze theoretically. In fact, generalization bounds and other theoretical analysis of traditional Dropout suffer from a significant problem: the trained network is different than the final network used for inference. To be more precise, we can represent the output of a Dropout layer during training symbolically, as shown below, where θ is a Bernoulli random variable tensor, $*$ represents element-wise multiplication, and σ represents some non-linear activation function. Thus, the output $\hat{y}$ is a random variable even for a fixed input x:

$$\hat{y} = \sigma(w^T x * \theta)$$

The creators of Dropout intended for the trained network to be deterministic, and ideally recommended using the expectation of the trained network for inference. The expectation of a single layer is below, and the expectation of an entire neural net is complicated by the fact that we are taking expectations of a composition of functions.

$$\mathbb{E}[\sigma(w^T x * \theta)]$$

Due to the non-linearity of the network activation functions σ_i, the expectation of the entire network can be difficult to calculate theoretically. For a network with N Dropout neurons, calculating the expectation numerically would involve attempting all 2^N possible combinations of random variables θ, which is infeasible. Thus, the creators of dropout recommend using an approximated network for inference, which involves approximating the expectation of a layer as follows:

$$\mathbb{E}[\sigma(w^T x * \theta)] \cong \sigma(w^T x * \mathbb{E}[\theta])$$

In the case of traditional dropout, the above expression evaluates to:

$$\mathbb{E}[\sigma(w^T x * \theta)] \cong \sigma(w^T x * (1 - p))$$

This approximation has empirically been shown to produce good results in practice, as the widespread proliferation of Dropout can attest to. Theoretically, however, the approximation means that the scaling factor $\frac{1}{1-p}$ that a dropout neuron is multiplied by is not precisely the correct factor to scale by in order to preserve the expectation of the output of a neuron. More significantly, the approximation complicates the analysis of generalization bounds of neural networks trained with dropout. Most generalization bound techniques depend on the assumption that the same network that was trained will be used for inference, as opposed to an approximation to that network.

The work done in [6] investigates this discrepancy, and defines the concept of an Expectation-Linear Layer. The formula for an Expectation-Linear layer, adapted to the notation I have defined, is as follows:

$$||\mathbb{E}[\sigma(w^T x * \theta)] - \sigma(w^T x * \mathbb{E}[\theta])||_2 = 0$$

The paper states that networks with non-linear activation functions, such as RELU and the sigmoid function, do not have Expectation-Linear Layers, but can exhibit an approximately Expectation-Linear behavior.

However, I claim that DropAlan layers are Expectation-Linear for a fixed input x. This is because they do not use activation functions, rather the output of a neuron is simply the dot product $w^T x$. Thus, the σ function in the neural net representation above is just an identity function, and the expression reduces to:

$$||\mathbb{E}[(w^T x * \theta)] - (w^T x * \mathbb{E}[\theta])||_2 = 0$$

For a fixed x, we can see that the expression for Expectation-Linearity becomes:

$$||[(w^T x * \mathbb{E}[\theta]) - (w^T x * \mathbb{E}[\theta])||_2 = 0$$

$$0 = 0$$

Futhermore, the above use of linearity of expectation will extend to compositions of DropAlan layers, since no non-linear activation functions are used.

Thus, the DropAlan architecture allows us to understand what the output of a layer will be in expectation, and we do not have to rely on approximations as in the case of traditional Dropout. We can thus have a better understanding of the effect that any scaling factor we apply will have on the neuron output.

V. CONCLUSION

The DropAlan technique with probability distribution β was able to achieve similar or better accuracy results as compared to Dropout, but converged quicker. There is definetly room for further development, as training crashed on the CIFAR-10 dataset. However, the technique shows promise, especially in its ability to more quickly converge to its final solution.

Future research on the technique could focus on adjusting the hyperparameters of the β distribution to improve on the results shown in this paper. Furthermore, the experiments in this paper can be re-run with more granularity; I only recorded accuracy and loss information after every 1000 iterations. Additionally, exploring the effect of using RELU-based activation functions or Batch Normalization along with DropAlan is another avenue of future research.

More theoretical work can be done on this approach to. Although my brief theoretical treatment showed that the expectation of a network using DropAlan, as opposed to traditional Dropout, can be easier to understand, this was contingent on taking the expectation with regards to a fixed input x. Future work can explore how the expectation of a DropAlan layer compares to the expectation of a layer with Relu and Dropout and also exploring whether better generalization bounds can be developed for DropAlan than the ones we have for Dropout.

It is also worth analyzing why scaling the output of a DropAlan neuron improved accuracy, and whether such scaling is inherently necessary or simply indicative of a neuron output expectation that works better for inference. Once refined, the DropAlan technique may one day lead to neural networks that converge quicker and are more accurate than current state-of-the-art systems.

[1] Nitish Srivastava, Geoffrey Hinton,Alex Krizhevsky,Ilya Sutskever,Ruslan Salakhutdinov *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*. Journal of Machine Learning Research, 2014.

[2] He Hu *Symmetric Rectified Linear Units for Fully Connected Deep Models*. Lecture Notes in Computer Science, vol 11062. Springer, Cham, 2018.

[3] Bing Xu, Naiyan Wang, Tianqi Chen, Mu Li *Empirical Evaluation of Rectified Activations in Convolution Network* . arXiv preprint arXiv:1505.00853., 2015.

[4] Li Wan, Matthew Zeiler, Sixin Zhang, Yann LeCun, Rob Fergus *Regularization of Neural Networks using DropConnect*. Proceedings of the 30 th International Conference on Machine Learning, 2013.

[5] Zhe Li, Boqing Gong, Tianbao Yang *Improved Dropout for Shallow and Deep Learning*. 30th Conference on Neural Information Processing Systems, 2016.

[6] Xuezhe Ma, Yingkai Gao, Zhiting Hu, Yaoliang Yu, Yuntian Deng, Eduard Hovy *Dropout With Expectation-Linear Regularization*. The Seventh International Conference on Machine Learning, 2017.