Drop-in: An Attempted Dropout Variation

Hannah Lerner
University of Massachusetts Amherst
College of Information and Computer Science
hmlerner@umass.edu

1. Abstract

When training a neural network, researchers often look for methods to make models more robust. One of the methods used is Dropout [3]. Dropout works by randomly removing features from a layer of the network during training and scaling the features accordingly. This is effectively acting as a regularizer for the model. In regularizing, Dropout allows for other complexities to be added to the model (more features for example). The idea behind this is to try and help models be able to identify the most important nodes as well as how to work out a good result even if some of those nodes are not present.

A problem that can occur during training however is when too many node end up dying. This leads to much slower training rates as well as decreased performance. In this paper I explore the idea of ”Drop-in” where in I revive dead features instead of removing living ones and then scale the rest of the values down to scale. In doing this, I allow potentially wrongly-removed features to have another chance in the training model as well as offer a place to highlight inconsistently behaving features and create a path for their removal.

2. Introduction

Currently, a major issue when training neural networks is that it can become very difficult to discover certain ”bad” nodes due to them not really appearing too much during training. This can lead to them having a more influential role in the neural network than they should, and ultimately reducing the overall performance of the model.

2.1. Drop-in

In this paper, I propose a possible fix for this issue. I offer a new function: ”Drop-in” as a method to try and identify these problematic nodes and create pathways to either focus more on them, or to ignore them. The idea is very similar to Dropout [3], but instead of randomly removing features and scaling the rest up, I will be randomly reviving dead features and scaling the rest down.

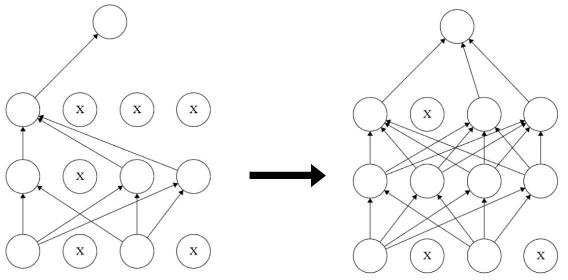

Figure 1. Above illustrates how Drop-in works in a more visual context.

The basic plan to go about testing this new Drop-in function will be to first go back to a previous homework assignment for this class [4] where I implemented Dropout by hand. Here, I are provided with pretty much all the framework needed to create a new layer ”Drop-in” that follows the same design principles as the implemented Dropout function, but without needing to create an entire infrastructure from scratch. It also offers a multi-layer fully-connected network that will offer a good base model to test the functions with. This experiment will be run on fully-connected networks only as the effects of Drop-in will have a more direct influence on the network as a whole.

Using this previously created structure allows gives a fairly straightforward testing platform for the forward and backward pass of the layer as it was already created to test very similar features for the Dropout implementation I were required to do previously.

2.2. Implementation

The implementation process will be fairly simple as I will be testing this Drop-in layer using the FullyConnectedNet class from the homework two assignment [4] as a general architecture. I will be creating a total of 12 different neural networks for this experiment. The three testing categories go as follows: a network with no Dropout and no Drop-in, a network with Dropout, but no Drop-in, a network with Drop-in, but no Dropout, and a network with both Drop-in as well as Dropout. Each of these three model types will be tested with 2 layers, 3 layers, and 5 layers to try and determine where each one's strength is. Given that Dropout is a regularization technique, and Drop-in is more or less the opposite (will increase model complexity), The expectation going into this experiment is that Dropout will perform better on more complex models where it prevents overfitting, and the Drop-in models will perform better on the less complex models where it prevents underfitting. The input for each of these models will be preprocessed image data and the output will be the classification predictions.

2.3. Evaluation

The reported values in the tables are test set accuracies based on a softmax loss function, but the line graphs depict the training and validation data. This is so as to show the history of the accuracies of both training and validation at each epoch.

3. Related Work

While in the preliminary stages of this assignment, I were looking into various ways to apply neural networks to different tasks. A dermatologist-level classifier paper [2] really caught the eye and I used that as a starting point for working out much of our understanding of how work done in this project can generalize to many different types of tasks.
The basis for this entire project stems from the original Dropout paper [3]. In this paper, the layer function Dropout is discussed in great detail and sets up the framework for the changes that I propose in this paper. As a summary, the actual Dropout function works by randomly removing features from the layer so as to improve the robustness of the overall model. A helpful diagram from the original paper [3] illustrates how the Dropout functionally works.

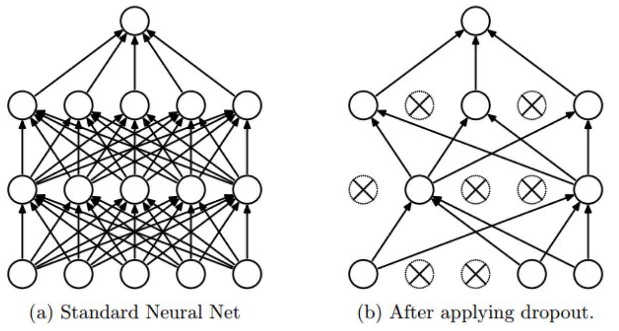

Figure 2. Above illustrates how Dropout works in a more visual context.

A major contributing resource to this project was the base code from the second homework assignment for this class [4]. This source provided me with many framework and testing resources for this project that allowed me to streamline many otherwise lengthy processes.

4. Technical Approach

Originally, the neural networks were to be trained on the ICIS dataset [1]. This dataset is made up of 30000 images of dermascopic photo data. For testing, I wrote a script to organize the data into folders based on classification labels. As a start, I wanted to run 5000 images through a basic pre-trained model to get an idea of what a baseline will look like. Unfortunately, during testing, I discovered that the data is extremely lopsided (only about 2% of the images are malignant) which makes for a less than ideal dataset for processing. The model produced a very high accuracy on the data, but with data that is 98% benign it can be really easy to produce high accuracy data without producing a good model. In response to this issue with the data, I decided to shift gears and move to the CIFAR-10 dataset.

4.1. Base Model Settings

This data was split 50000 training images (49000 train, 1000 validation), and 1000 test images. The input images can fall into 1 of 10 different classes. All of the models were run using the following settings:

hyperparameter	setting
loss	softmax
optimizer	Adam
learning rate	0.001
number of epochs	10
batch size	100

4.2. Drop-in Calculation

In this section, I will describe the implementation of the Drop-in algorithm. The basic idea behind Drop-in follows along the same logic as Dropout in that I are manipulating specific features in the network to have more (in the case of Drop-in), or less (in the case of Dropout).

To go about performing Drop-in on a layer, I follow a very similar method to how Dropout is implemented. The basic idea is to find all of the lost features in the layer, then randomly revive a percentage of them to some pre-defined uniform random value. In order to balance out the layer, I will need to scale the input layer by:

$$\frac{N_x}{N_x + p * N_0 * R}$$

where N_x is the number of living features in the layer, p is the fraction of dead features to revive, N_0 is the number of dead features in the layer, and R is a pre-defined uniform random value. This layer is put into a simple feedforward network and the model is then run on the dataset.

4.3. Error Checking

I was able to check over the Drop-in implementation calculation fairly accurately due to tools left for the class to complete homework assignment 2. Specifically, there was a script provided [4] that was able to show the difference mean value of the input image and the modified image after running the forward pass of Drop-in. This allowed for some very reliable error checking to ensure the forward pass was balancing the layer properly. The gradient checking tool made ensuring the backward pass was correct trivial as well.

4.4. Architecture

The actual architecture of the network can be shown as follows:

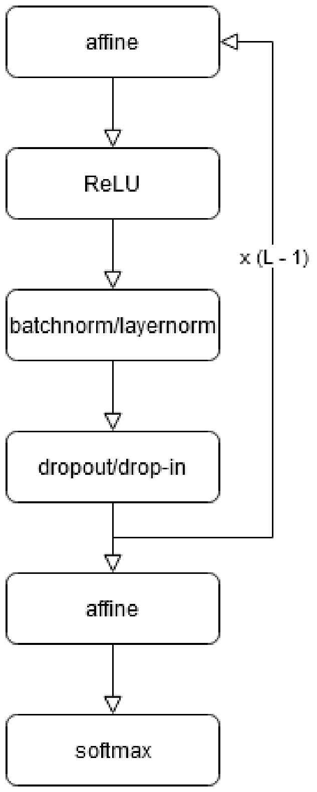

Figure 3. Above is a diagram of the network architecture used for this experiment. L represents the number of layers in the network.

This basic architecture allows for us to restrict the number of changing variables in the experiment to 4 primary factors: The number of layers the network will have, whether or not to use batchnorm or layernorm, whether or not to use Dropout or Drop-in, and what the p value for Dropout or Drop-in will be. In order to further restrain variable parameters, I decided that this experiment will not use batchnorm or layernorm. While this will most likely decrease the performance of the overall model, I felt that it could possibly provide a place for Drop-in to perform better than Dropout as there is a higher chance of nodes incorrectly being ignored or given attention.

5. Experiment

This experiment was designed to see how Drop-in performs in comparison to Dropout over both different values of p as well as with a varying number of layers of the neural network.

5.1. Setup

This experiment can be broken down into two primary parts, a model visualization test on a sample set of data to illustrate the difference in training/validation history of Dropout and Drop-in at different p values. To do this part of the experiment, I picked a subset of the CIFAR-10 dataset with 1000 training images. From there, I fine-tuned the model to work well with the baseline dataset making sure that it was not overfitting or underfitting. After that, I ran the models through using the same hyperparemeters as the baseline, but testing with Dropout or Drop-in at different p values. This way I were able to see if there is any clear distinction between how Dropout performs during training versus how Drop-in performs.

The second part of this experiment is to run through each neural network type described in section 4.3. I decided to test the performance of Drop-in, Dropout, a baseline model with neither Drop-in or Dropout, and a combined model with both Drop-in and Dropout over 2, 3, and 5-layer neural networks. The primary reason behind this was to see how each model performed when the network had varying levels of complexity to them the idea being if Drop-in performs well with less complex models, but worse with more complex models, then it is likely not fulfilling its role as a regularizer and is unlikely to be helpful addition to the neural network.

Each of these 12 networks were set up to train with values of p ranging sequentially from 0.1 to 1 and then taking the model with the highest validation accuracy and using that to predict the test images. The lower scoring p value models are discarded during this process.

5.2. Results and Evaluation

Before I go through the results of the 12 network test, It is important to first show the training progression of the Drop-in model in comparison to Dropout. I did not use the full dataset for this as it takes a significant amount of time to train and is not necessary to visualize how the network is training. For this first experiment, I fine-tuned a simple 3-layer neural net to perform well on the data without Dropout or Drop-in. Once tuned, I ran through each value of p with this model to see how Dropout and Drop-in would perform in comparison to each other at different values. Results of this experiment can be seen below:

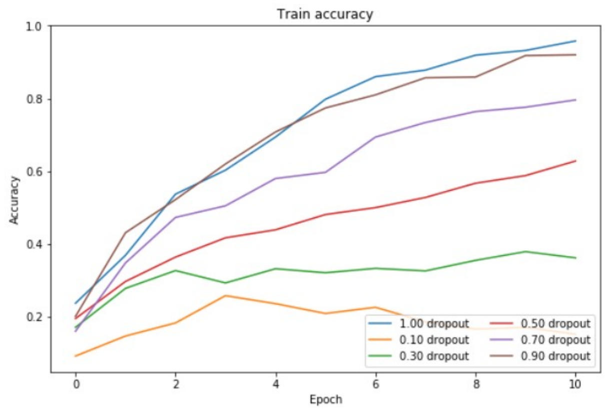

Figure 4. Above is the training history of a simple 3-layer neural net with Dropout

With Dropout, this model was able to produce a fairly even spectrum of training accuracies over values of p. This is expected as Dropout is a known regularizer and decreasing the value of p effectively removes more and more pathways for the network to flow which results in it becoming less and less complex. The hope for Drop-in is to see somewhat similar of a spectrum showing that it can act as an effective regularizer.

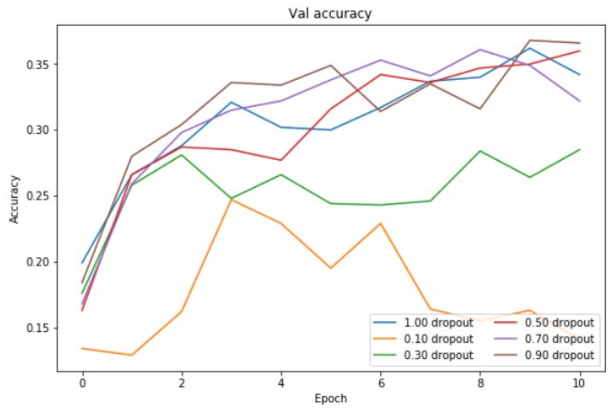

Figure 5. Above is the validation history of a simple 3-layer neural net with Dropout

Here again, the model produces very consistent results. the networks that were the least complex underfit, but a couple of the Dropout models with higher training accuracies had a better validation accuracy than the baseline. This is a very good sign for Dropout as this behavior demonstrates that it can act as an effective regularizer. Ideally this is similar to what the Drop-in accuracy history will look like.

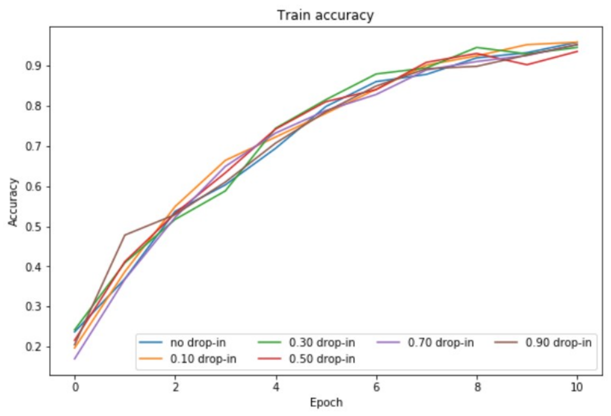

Figure 6. Above is the training history of a simple 3-layer neural net with Drop-in

Unfortunately, the results from the Drop-in training indicate no significant regularization is taking place. All of the p value models for Drop-in yield about the same high final training accuracy. While not absolutely conclusive, this graph strongly suggests that Drop-in will not work as a regularizer. It still does have a chance to perform well in the second test, but this graph here is an indicator that the initial premise for using Drop-in does not apply during training.

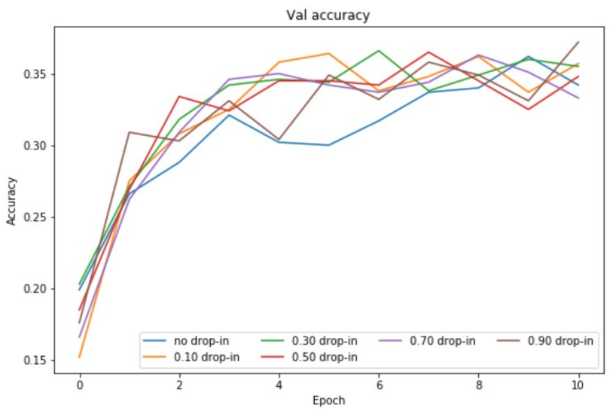

Figure 7. Above is the validation history of a simple 3-layer neural net with Drop-in

Here we can see all of the accuracies are in more or less the same spot. Such results again suggest that regularization is not taking place and that Drop-in is not likely to perform well at the task of identifying problematic features. While this part of the experiment indicates a grim future for the rest, there is still hope that Drop-in can improve network performance when put into the different neural networks at varying levels of complexity.

Below are the results from the full run through of each of the 12 networks running on the sequence of p values from 0.1 to 0.9.

Method	2 layers	3 layers	5 layers
Baseline (no Drop-in, no Dropout)	0.389	0.475	0.494
no Dropout, Drop-in	**0.352**	**0.444**	**0.501**
Dropout, no Drop-in	0.382	0.519	0.503
Dropout, Drop-in	**0.383**	**0.462**	**0.478**

These results indicate that while Drop-in improves performance in the shallower networks, its performance begins to lag in the deeper network. This comes as no surprise as the results from the first experiment (specifically Figure 6) heavily indicated that something like this might happen. Given that Drop-in does not appear to work as a regularizer, it cannot adjust models well enough to prevent overfitting. One potential reason why Drop-in failed was that by reviving previously dead features, Drop-in is offering more pathways for the network to use which can help it fit to the training set better. When it has limited pathways to work with (as in the case of Dropout), it is forced to find more general solutions, but when it is given the option of any pathway, it will just continue to train on the good ones during training, leaving the network less robust than it would otherwise be.

One interesting result however was the performance of Dropout immediately followed by Drop-in. This was the only model sequence that consistently performed worse than all of the others (except Dropout in a very shallow network, but that can be expected). This poor performance is most likely due to taking away learned features with regularization, and then immediately putting random features back in their place. While it is adding noise to the layer, it is also likely making it difficult for the network to properly learn its own features.

5.3. Classified Image data

While it is useful to see what the training, validation, and test accuracies are, it is also very important to look on how the models perform on the images themselves. Below are three examples of correctly classified and missclassified test images. The top row holds the ground truth labels, while the bottom row holds the model predictions.

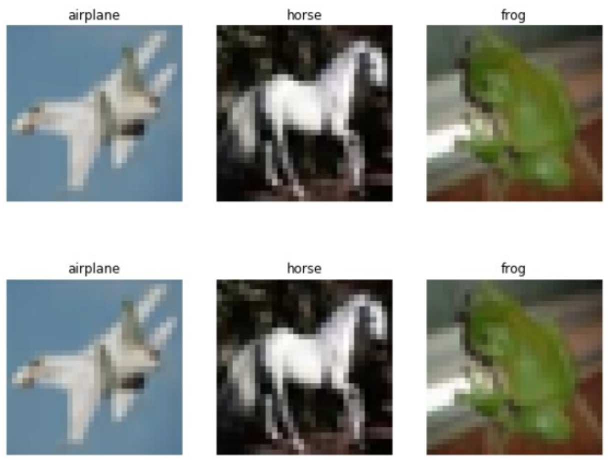

Figure 8. Above are a couple examples of images that were classified correctly.

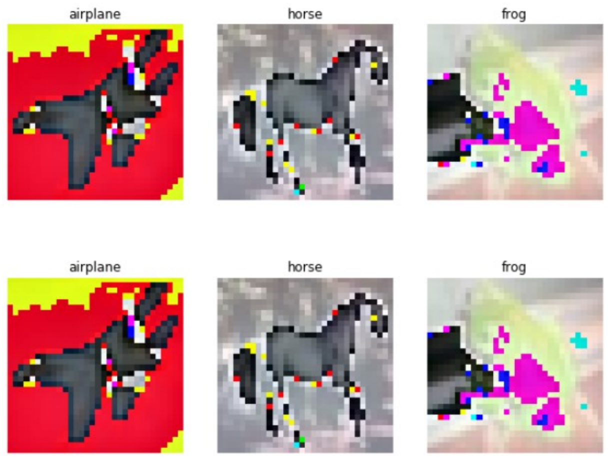

Figure 9. Above are a couple examples of normalized images that were classified correctly.

Looking at these examples, each image seems pretty clear that it is within the class the model predicted. There seem to be pretty cut and dry lines around the objects of focus in each image and each of those lines give a pretty clear outline of what the image is without even looking into coloring. This becomes obvious when looking at the normalized image that is fed into the model for prediction. Here we can see that the normalized images are able to make pretty distinctive boundary lines between different parts of the blob which makes for an overall clearer view of the image at large.

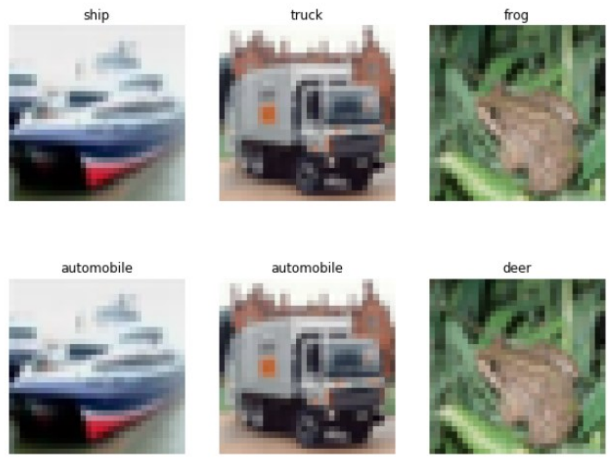

Figure 10. Above are a couple examples of images that were classified incorrectly.

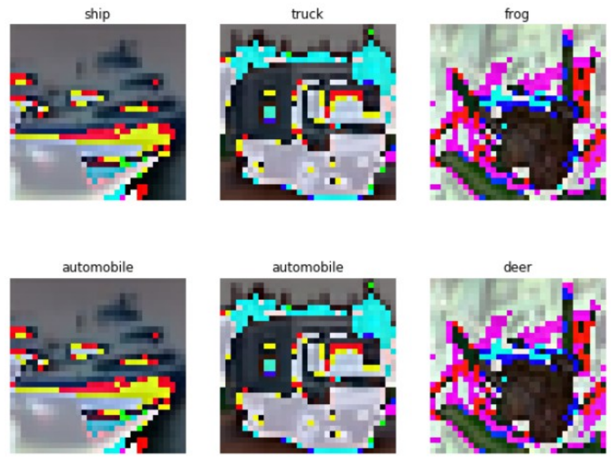

Figure 11. Above are a couple examples of normalized images that were classified incorrectly.

Looking at the missclassified images shown above brings up an interesting point with the overall model accuracy. Both the ship and the truck were missclassified to automobile. As far as predicting between the 3 goes, it is expected that more errors will occur here. Specifically, looking at the truck, it becomes clear when looking at the normalized image that the scale of the object in the image is lost. This makes it really difficult to differentiate between a truck and another automobile. The most likely reason the frog was missclassified in this image is how the blades of grass merge with the blob of the frog to look almost like antlers in the normalized image.

6. Conclusion

The goal of this project was to test a new regularization method "Drop-in" and see how it performed in relation to Dropout [3]. This experiment was able to rely heavily on previously submitted code from homework 2 of this class to provide a helpful testing infrastructure and ensure that the Drop-in calculations are sound. Unfortunately, Drop-in did not perform to a satisfactory level in the experiment. The biggest indicator that it was not performing well was when it was run to test the training accuracy over time (Figure 6) and it showed no indication of reducing the training accuracy as p changes. The original suspicion that Drop-in was not performing well was later cemented by the larger-scale test on the entire dataset which yielded no significant increase in performance (specifically in the higher complexity models that were run). Overall, I believe the idea behind Drop-in had merit, but unfortunately did not pan out during testing.

7. Future Work

While Drop-in did not appear to offer any significant improvements to the network, there are a couple of changes implementation side of this experiment that could change the results.

The first and primary change that could be made in the future would be to replace the value that replaces the dead feature. At the moment, this implementation replaces the zero value with a predefined random variable between zero and one. I implemented it this way for this experiment because I thought it would be more likely to force the network to pay more attention to it. Another possible solution for Drop-in would be to replace the dead features with a representative value from the rest of the layer such as a mean value.

References

[1] Veronica Rotemberg et. al. "A Patient-Centric Dataset of Images and Metadata for Identifying Melanomas Using Clinical Context". In: (). URL: https://doi.org/10.34970/2020-ds01. (accessed: 11.19.2020).

[2] Andre Esteva. *Dermatologist-level classification of skin cancer with deep neural networks*. URL: https://doi.org/10.1038/nature21056. (accessed: 11.19.2020).

[3] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *J. Mach. Learn. Res.* 15.1 (Jan. 2014), pp. 1929–1958. ISSN: 1532-4435.

[4] CS 682 Staff. *COMPSCI 682 Neural Networks: A Modern Introduction Homework Assignment 2*. URL: https://compsci682-fa20.github.io/assignments/assignments2020/assignment2/. (accessed: 11.19.2020).