The effect of varying training on neural network weights and visualizations

Emmett Fountain, Joe Rasmus
Williamston High School, Williamston, Michigan

SUMMARY

Neural networks are used throughout modern society to solve many problems commonly thought of as impossible for computers. As their use becomes more widespread, an issue of measurement arises. In order to create metrics with which to measure the ability of neural networks, new techniques must be constantly developed. The purpose of this research is to determine whether varying training produces measurable and consistent patterns in the visualizations and weights of Convolutional Neural Networks. In order to carry out this investigation, a convolutional neural network was designed and run with varying levels of training to see if consistent, accurate, and precise changes or patterns could be observed. To determine if such a change or pattern existed, the weights and filters were analyzed through visualizations, both qualitatively and quantitatively. Several patterns were discovered in the visualizations, but they were inconsistent across layers and the quantitative models were only consistent in specific circumstances. This indicated that while training introduced and strengthened patterns in the weights and visualizations, the patterns observed may not be consistent between all neural networks.

INTRODUCTION

Neural network technology, and machine learning more generally, is present throughout much of our modern world, from self-driving cars to text recognition (1). Machine intelligence is becoming a crucial part of our society, most notably in healthcare, education, the automotive industry, and general safety (2). Neural networks, machine learning implementations which use systems of weights and biases to make decisions, are often part of these intelligent systems. Due to the integration of machine learning in highly important fields of society, deficiencies in our understanding of its function can often be devastating. If an automated vehicle fails to notice and recognize an obstacle, it could result in death, similarly if an automated doctor in the not-so-distant future were to misdiagnose a patient it could be equally fatal. Understanding the effect of training on components of a neural network deepens our insight into how the weights affect predictions of the neural networks. This deeper insight could open up the possibility for new techniques of measuring competency of a network other than test accuracy, which is often heavily situational and can overrepresent functionality due to overfitting (that is, when a neural network works only on the specific training set or test set). As the field of machine learning grows, it becomes more important to understand the effects of training on neural networks and their weights - numbers that influence the output of a neural network that are crucial to their operation.

This investigation focuses on neural networks. At its simplest, a neural network works very similarly to a mathematical function, with weights modifying input values multiplicatively and biases modifying them additively. One way of "learning" new mappings from input to output is supervised learning where the network is fed both an input output pair which is known to be correct. For example, you could provide an image of a cat as input and the classification "cat" as the output. With this input and output the network calculates loss, a measure of how inaccurate the network's prediction was. The network then uses an algorithm, to modify the weights and biases to better map the input to the correct output. This kind of basic neural network is often called a fully connected neural network.

Convolutional Neural Networks (CNNs) are neural networks which contain at least one, but usually more, convolutional layers. Layers at their most basic are steps in a neural network which data must pass through, usually being modified in some way before being output to the next layer or as an output. Convolutional layers in particular, are layers which apply convolutions to an image. A convolution is an image analysis technique in which a filter - an ordered set of weights - is iterated over the input, modifying it to create a new output. Due to convolutions being the core of this type of neural network, these networks are often used to analyze and make predictions based off of images.

A simple CNN was created using Python and then trained with Cifar-10 (3), an image database used for machine learning. The filters used in the convolutions were then analyzed using techniques similar to those outlined during the 1989 Neural Information Processing Systems conference (4). This analysis generated qualitative and quantitative data used to compare the varying levels of training. My research aims to contribute to an understanding of the effect of training on weights by determining whether consistent and measurable patterns appear in visualizations of weights at varying degrees of training. To accomplish this goal, visualizations of weights were analyzed to look for patterns at varying levels

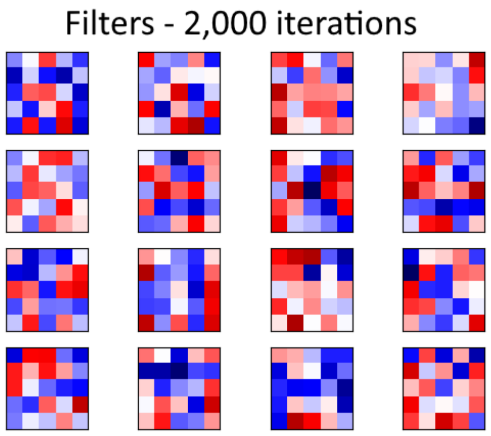

Figure 1. Weights of filters with little training are randomly distributed. All 16 filters in layer 1, after 2,000 iterations. Each panel is one of the filters of layer 1, the colors represent the value of the weights with blue being low and red being high. The exact values of the weights are not important in this situation because all we need to see are the patterns in the intensity and sign. Note the prevalence of intense weights throughout the filters, mixed seemingly randomly.

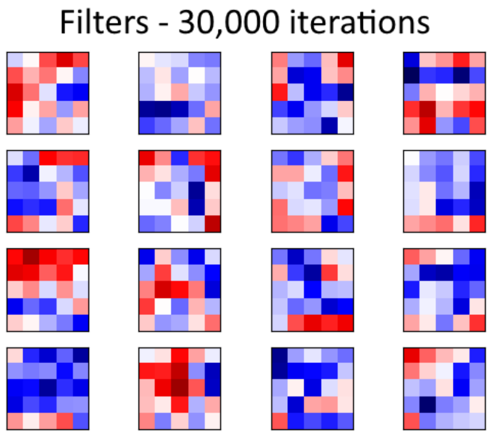

Figure 2. All 16 filters in layer 1, after 30,000 iterations. Weights of filters with more training form clusters Each panel is one of the filters of layer 1, the colors represent the value of the weights with blue being low and red being high. The exact values of the weights are not important in this situation because all we need to see are the patterns in the intensity and sign. There are far fewer intense weights, which appear mostly in clusters with a similar sign.

of training. In the analysis, a pattern was found and termed "softness", it was tracked both as a quantitative variable and as a qualitative variable. We predicted that quantifiable patterns would be detected, but they would most likely not be consistent.

RESULTS

The first qualitative trend noticed was a change in the distribution of strong weights among the filters. When there was little training, the weights were distributed with a random spread. As training increased, clusters of weights began to form. The distribution of weights across the filters changed from a random spread with little training to a spread with clusters of strong weights when there were higher training levels (**Figures 1 & 2**). Two other visualizations were created to analyze the weights. Both visualizations contained a number of panels, each with an image which has had a unique filter applied to it. The visualizations differed in the base images that were used; one used an image of a bird taken from the Cifar-10 dataset, while the other used an image of black and white static noise. A prominent pattern in these visualizations was the increasing prevalence of "soft" images when filters are better trained, seen in both the images generated from static and the bird images in both the first and second layer of both of these image sets (**Figures 3 & 4**). Images which are soft have pixels where close location corresponds with close

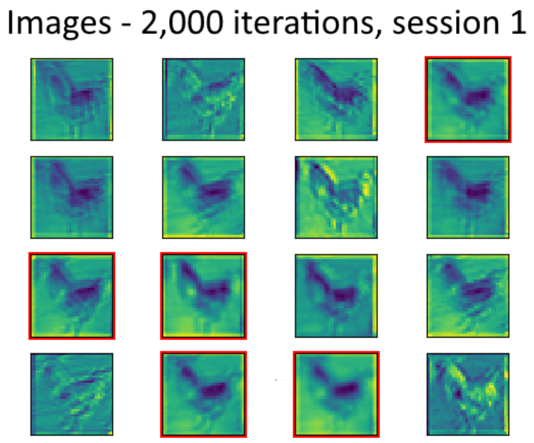

Figure 3. Images generated by applying all 16 filters in layer 1, after 2,000 iterations, to an image in the "bird" class of Cifar-10. Applying filters with little training to sample images generates images with little softness. Each panel was generated using a different filter from layer 1. Very few filters produced images that are soft (soft images highlighted in red).

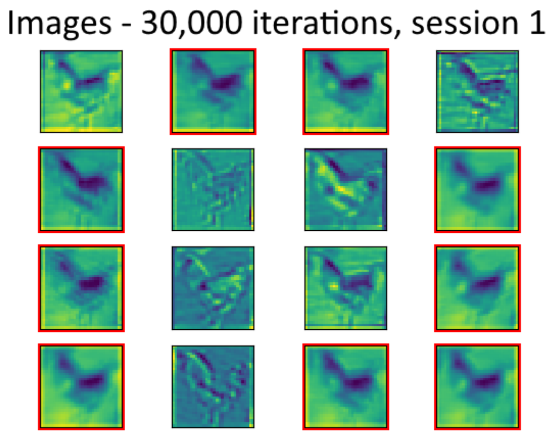

Figure 4. Images generated by applying all 16 filters in layer 1, after 30,000 iterations, to an image in the "bird" class of Cifar-10. Applying filters with more training to sample images generates images with greater softness. Each panel was generated using a different filter from layer 1. Far more filters produced images that are soft compared to the filters that were trained to 30,000 iterations (soft images highlighted in red).

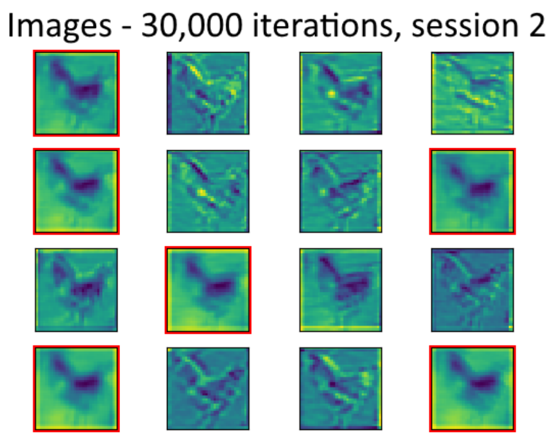

Figure 5. Images generated by applying all 16 filters in layer 1, after 30,000 iterations, to an image in the "bird" class of Cifar-10. Each panel was generated using a different filter from layer 1. The amount of softness is highly variable even with the same amount of training. While there are still more soft images then in Figure 3 with 2,000 iterations, there are noticeably less than in Figure 4, which had the same amount of training and was just a different session.

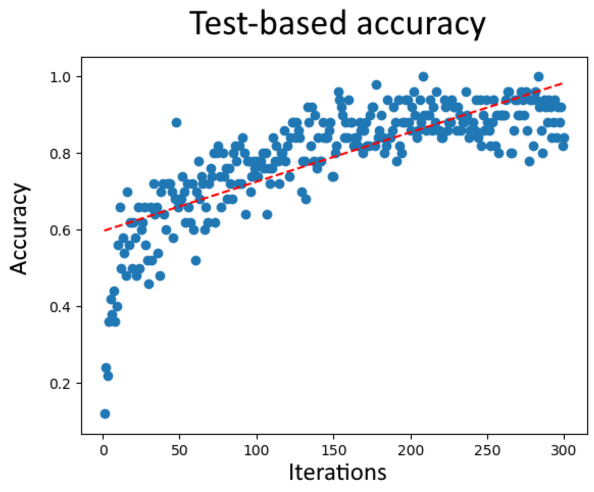

Figure 6. Test-based accuracy during training, reported at intervals of 10 iterations. Test based accuracy suggests a non-linear model. This figure includes the least squares regression line in red as a reference.

values, visually this means there are very few edges and little pixelation. When trained to 2,000 iterations, few of the 16 filters in layer 1 generated images which appeared soft. However, when trained to 30,000 iterations far more of the images were soft. This trend also occurred to some extent with the images generated by applying filters to static noise and when using filters from layer two (data not shown). However, the specific number of soft images varied per session even when the number of iterations remains constant. In two separate sessions filters were trained with 30,000 iterations but the filters of one session produced far more soft images than the other despite the same amount of training (**Figures 4 & 5**).

A graph was generated that showed the accuracy during training (**Figure 6**). This helped show the training trend for the model which gives an idea of what kind of numerical models might be plausible while analyzing the quantitative data. We observed that the rate of increase in accuracy decreased at higher training levels suggesting that a pattern would probably not be linear.

Almost all of the data had very low R and R-squared (R^2) values, meaning that not many of the linear regression equations fit the data very well. The notable exceptions were power functions from the second layer (**Figure 7**). The data generated from the bird image had an R2 value of about 0.9220 and the data generated from the static noise image had an R2 value of about 0.8996. Other than these two, all of the R2 values were below 0.8. Power functions for layer two

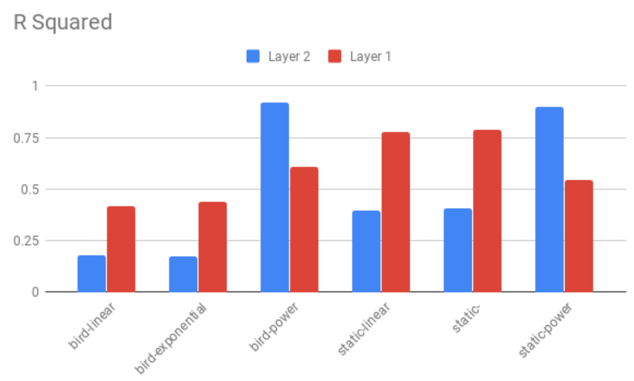

Figure 7. Bar graph showing the values of R2 for a line comparing softness to number of iterations. The graph includes the R2 values for all combinations of variables, that being layer, model, and image. In layer 2, the power model has a very high R2 value for both images. A high R2 value for the power model in layer 2 suggests that this model fits the data well.

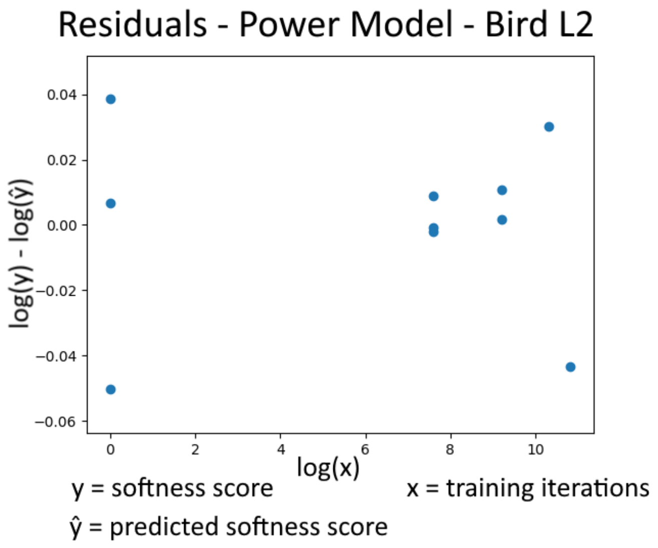

Figure 8. Residual graph showing the residuals for a power model of softness scores. Softness scores were calculated from the bird images generated from layer 2 as described in the methods section. There is not a strong curvature, which indicates that the power model would remain somewhat consistent if the model were extended to higher and lower batch sizes.

of the bird and static noise data, also had the lowest *p*-values for the two-tailed linear regression t-test with 1.05x10-5 and 2.90x10-5 respectively. However, these two were not the only regression equations that were significant at α = 0.05. All of the first layer data had significant *p*-values, but very low R2 values; however, from layer two, only the power function and the exponential model generated from static noise were significant at α = 0.05.

The residual graphs provided additional information as to whether the regression equations held up over time, an example of which can be found below (**Figure 8**). All of the graphs had some degree of curve, but several demonstrated more random scatter than others. The most random graphs were from layer two of the static noise straightened as a power function, both layer one and two of the bird image unstraightened, and layer one of the bird image straightened as a power function. The rest showed much more severe curving, which suggested that the pattern falls off as the data is extended.

DISCUSSION

We observed some consistent patterns in the weights and visualizations which were very localized or inconsistent. There is some evidence to suggest that consistent patterns are present, but evidence also suggests that these patterns may not be consistent across neural network models. The existence of consistent patterns in the weights was supported by qualitative evidence. We observed patterns in the visualizations of the weights through the clustering of strong weights, as well as the increasing number of filters that produce soft images (**Figures 1 & 2 and 3 & 4**). The existence of patterns was further supported by the quantitative evidence, which demonstrates some consistency and rigor to these trends. In layer 2 of the data generated from static noise, there exists an accurate power function that shows a strong relationship between training and softness. A high R2 value suggested that there was a strong correlation between the scores generated by the image and the number of iterations (**Figure 7**). The low *p*-value tells us that there was a significant linear correlation in the data when straightened as a power function. Finally, the random scatter of the residual graph showed that the power function is relatively consistent even as the pattern is extended. (**Figure 8**)

However, evidence suggests that the patterns might not be present in other neural network models or even when testing separate sessions. For example, a power function seemed to be the best fit for the data in layer 2, having higher R2 values and lower *p*-values than any other model for both images. The issue is that this same model was not consistent for layer 1, which is simply a different level of abstraction and doesn't have any fundamental differences from layer 2. Layer 1 has different models for each image which presents the problem that the patterns would be inconsistent as layers vary tremendously across models. The patterns would most likely be inconsistent with different test images and when using different neural networks. Along with this quantitative inconsistency, the qualitative trends were inconsistent across sessions (**Figures 4 & 5**). Therefore, more advanced quantitative research may prove unfruitful.

Overall, there were consistent patterns in Neural Network weights and visualizations within a specific model, but those patterns may not extend much beyond that specific model. In order for the findings of this research to have an impact on applied machine learning, more research would be needed to widen the scope of these findings.

While we conducted this research to the best of our ability, we could take several steps to improve the validity of this research or to further it. One such step would be to use computers that can better handle machine learning. It would have been highly informative to see how the data extended beyond this point, and it might have prompted some new insights. Other steps that could have been taken to generate a more representative sample and outline stronger trends include increasing the sample size, running more sessions, and applying the weights to more images. If future research was to be conducted it would be important to apply more advanced image analysis techniques and statistical analyses so that trends can be better quantified and new trends might be discovered.

METHODS

In order to test for patterns, a Convolutional Neural Network (CNN) was used; as it provides an acceptable level of complexity but is also simple enough that its weights can be easily accessed and analyzed. The CNN was built in Python with the TensorFlow library (5); the CNN had two convolutional layers and two fully connected layers. The batch size is variable, but due to hardware limitations a batch size of 50 was employed to generate the data. Cifar-10 (4) was used to generate the data, as it has a relatively large sample of images, but a manageable number of 10 classes. A function saved the convolutional filters as NumPy array files once the set number of training loop iterations had been completed.

A separate Python program was designed to view the filters in a variety of fashions. This program would display the filters as grids of colors ranging between blue and red depending on the value of the weight for each grid square. The program could also apply the filters to a sample image, in order to see how the filters affect the images they are applied to. The filters and images generated by the filters were grouped by which session and convolutional layer they were from. There were 5 different levels of training: 1, 2,000, 10,000, 30,000, and 50,000 iterations. The sessions with only one iteration were treated as baselines since they were essentially random as they had no time to train. Then, small, moderately small, moderately large, and large numbers of iterations were chosen so that the differences in visualizations at noticeably different training levels could be compared. Due to time and computing restraints, more sessions with fewer iterations were run than those with many iterations. Three sessions with 1 and 2,000 iterations were run, two sessions of 10,000 and 30,000 iterations were run, and only one session of 50,000 iterations was run.

In order to analyze the change in the filters, three kinds of images were qualitatively analyzed for each filter. This qualitative analysis was meant mostly as a heuristic to find possible patterns and support the quantitative analysis. The first of these image types was the red and blue color interpretation of the filters. The next image type was generated when the filters were applied to a 32 x 32 image with randomized black and white pixels. The original black and white image was generated as a 32 x 32 matrix with random ones and zeros, which was then transferred to an image with ones representing white pixels and zeros representing black pixels. The final image type was generated by applying the filter to a sample image from Cifar-10. For this sample image, a simple bird image was chosen to further reveal patterns not identified by applying the black and white image. The images of the same number of iterations and that shared a layer were analyzed together. The images were originally analyzed qualitatively and note was taken of any recurring patterns or features in a layer. The number of "soft" images for each group was also recorded. To quantify image trends, a python program was developed to objectively measure "softness" of an image.

The program first normalizes the images so that the value of each pixel is a floating point between 0 and 1. Once the images are normalized, a 3 x 3 filter is repeatedly applied to each image. This filter computes the standard deviation in that 3 x 3 area and returns it to the location in a matrix where the sample 3 x 3 area was taken. The standard deviations of each image were then averaged for each image, representing a score for the image which should theoretically be lower for softer images. After this score was computed, the scores for each layer were summed to give a total score for the layer. These scores were paired with their respective number of training iterations allowing the scores of each layer to be analyzed as a function of training iterations. Using these scores and their corresponding training iterations, statistics were generated in order to determine whether there was a relationship between training iterations and the softness score. A python program was used to generate a least squares regression line for the relationship between scores and iterations, R, the *p*-value for a two-tailed linear regression t-test with a null hypothesis of zero slope, and graphs of the residuals (**Figure 8**). As the accuracy graphs were clearly curved, the data was straightened to conform to exponential and power function models. Straightening, also called re-expressing, is a process which linearizes non-linear data, which serves many purposes. The same calculations were performed on both of the straightened data sets and the unstraightened data.

Linear regression equations can be used to describe the straightened data and as such different models for a set of data can be compared. Linear data allows for a linear regression t-test to be performed to statistically determine whether the data is linear and thus fits the model which it was straightened for. In general, straightening works by determining which model the data follows and applying the inverse of that to the data. This effectively "undoes" the original function leaving the data as a set of linear points. Curves can often be described by one of two models; exponential, and power models. As such, in order to determine which model described the data curve, the data was straightened as if it was one of these. Several tests were run on the resulting, now linear, data to see which model it fit best.

Received: May 29, 2019
Accepted: November 26, 2019
Published: December 4, 2019

REFERENCES

1. Bojarski, Mariusz, *et al*. "End to End Learning for Self-Driving Cars." *arXiv.org ArXiv:1604.07316 [Cs]*, Apr. 2016., arxiv.org/abs/1604.07316.
2. Amato, Filippo, *et al*. "Artificial Neural Networks in Medical Diagnosis." *Journal of Applied Biomedicine*, vol. 11, no. 2, ScienceDirect, Jan. 2013, pp. 47–58., doi:10.2478/v10136-012-0031-x.
3. Krizhevsky, Alex. "Learning Multiple Layers of Features from Tiny Images" cs.toronto.edu*,* April 8 2009, ch. 3, cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.
4. Wejchert, Jakub and Gerald Tesauro. "Neural Network Visualization." *Advances in Neural Information Processing Systems 2 (NIPS 1989),* Morgan-Kaufmann, 1990, pp. 9. papers.nips.cc/paper/286-neural-network-visualization.pdf.
5. Abadi, Martín, *et al*.. "TensorFlow: Large-scale machine learning on heterogeneous systems", tensorflow.org, Nov 9 2015, static.googleusercontent.com/media/research.google.com/en//pubs/archive/45166.pdf.