Visualizing Internal Neural Network Dynamics Through Network Analysis Of Co-activation Graph Of Neural Network.

Onat Ozer

Neural networks have shown themselves to be one of the most powerful tools in machine learning, becoming an essential building block as a part in any machine learning application. However, a clear issue with neural networks is the 'black box' dilemma, often, predictions made by the neural network don't appear to have a clear rationale behind them, and often involve learning some alien correlations between data points that may or may not truly reflect the underlying truth. The goal of this project is to help alleviate this problem by using network analysis to illuminate the trends in what the neural network is learning during training.

I. Introduction and Background

At it's most simple level, a neural network is a set of layers where each layer receives some input vector $\mathbf{x_i} \in \mathbb{R}^n$, which are then multiplied by that layer's weights: $\mathbf{W}_i \in \mathbb{R}^{m \times n}$, plus some bias term: $\mathbf{b}_i \in \mathbb{R}^m$. The resulting vector then has some non-linear activation function $\sigma(\cdot)$ applied to it, resulting in the following equations being applied at every layer:

$$\mathbf{z}_i = \mathbf{W}_i \mathbf{x_i} + \mathbf{b}_i \mathbf{a}_i = \sigma(\mathbf{z}_i) \tag{1}$$

This inherit structure lends itself to being represented as a graph. We can take any fully-connected layer and build a graph off of it, where the nodes are each of the n dimensions of our input vector $\mathbf{x_i} \in \mathbb{R}^n$, and the edges are the weights $\mathbf{W}_i \in \mathbb{R}^{m \times n}$ we've computed throughout training. As an example, Using python networkx we can visualize the following fully-connected network:

$$\begin{aligned}
&\mathbf{x} \in \mathbb{R}^2 \\
&\mathbf{z}_1 = \mathbf{W}_1 \mathbf{x} + \mathbf{b}_1, \quad \mathbf{W}_1 \in \mathbb{R}^{4\times 2}, \quad \mathbf{b}_1 \in \mathbb{R}^4 \\
&\mathbf{a}_1 = \text{ReLU}(\mathbf{z}_1) \\
&\mathbf{z}_2 = \mathbf{W}_2 \mathbf{a}_1 + \mathbf{b}_2, \quad \mathbf{W}_2 \in \mathbb{R}^{3\times 4}, \quad \mathbf{b}_2 \in \mathbb{R}^3 \\
&\mathbf{a}_2 = \text{ReLU}(\mathbf{z}_2) \\
&\mathbf{z}_3 = \mathbf{W}_3 \mathbf{a}_2 + \mathbf{b}_3, \quad \mathbf{W}_3 \in \mathbb{R}^{1\times 3}, \quad \mathbf{b}_3 \in \mathbb{R} \\
&\hat{y} = \mathbf{z}_3
\end{aligned}$$

as:

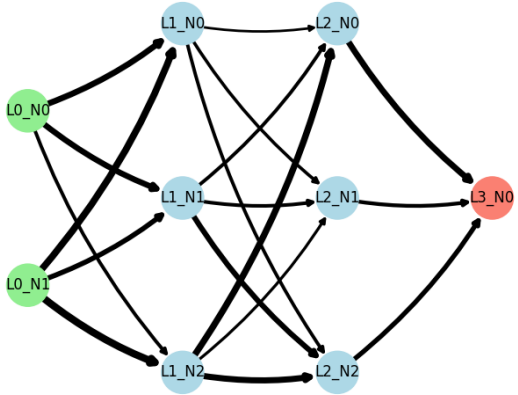


Fig. 1 Networkx visualization of the simple FCN.

But we can go even further in how we represent a neural network as a graph. When we visualize a neural network, we don't simply just want to see it architecture, ideally we want to see how the connections between neurons are forming throughout the process of learning that the network undergoes. In service of this goal, a helpful tool that we can use is the Co-Activation graph of a neural network. Formally, co-activation values are extracted over a set of inputs S. A co-activation value W_{ij}^{kl} between a neuron i in layer k and a neuron j in layer l is defined as the correlation (can be any well-defined correlation metric) of the activation values A of the two neurons over the set of inputs S

$$W_{ij}^{kl} = \text{Corr}(A(i, k, S), A(j, l, S)) \tag{2}$$

We can now replace the weights in the graph that we've previously made with the co-activation values, giving us a co-activation graph. The benefit we now have is that we can now see what inputs activate which sections of the network more strongly, and we can see the extent to which certain neurons are more correlated with each other. Previous papers have used these methods in analyzing relatively niche model architectures, such as reinforcement learning policies or auto-encoders. For this study, I'm hoping to analyze the simplest, most foundational building block of modern machine learning architectures so that these results will be more generally applicable to a wider audience. As part of this analysis, we require three more things, a community detection algorithm, a correlation metric, and a network centrality algorithm. For this paper, I used the following:

A. PageRank

The PageRank centrality assigns an importance score to each node by considering both its direct connections and the importance of its neighbors [1]. Let N be the total number of nodes in G, and let $\mathbf{PR} = [PR(n_1), PR(n_2), \ldots, PR(n_N)]^\top$ be the PageRank vector, initialized with $PR(n_i) = 1$ for all i. The PageRank is computed iteratively using [2]:

$$PR(n_i) = \frac{1-d}{N} + d \sum_{n_j \in \mathcal{N}(n_i)} \frac{A_{ji} \cdot PR(n_j)}{D(n_j)}, \tag{3}$$

where:

- $d \in (0, 1)$ is the damping factor (I used $d = 0.85$),
- $\mathcal{N}(n_i)$ is the set of neighbors of node n_i,
- $A_{ji} = W_{ji}$ is the weight of the edge between nodes n_j and n_i,
- $D(n_j)$ is the weighted degree of node n_j, defined as:

$$D(n_j) = \sum_{n_k \in \mathcal{N}(n_j)} A_{jk}. \tag{4}$$

The iterative process continues until convergence, i.e., when $\|\mathbf{PR}^{(t)} - \mathbf{PR}^{(t-1)}\| < \epsilon$ for a predefined $\epsilon > 0$. All Page Rank calculations were performed using networkx [3]

B. Pearson Correlation

The **Pearson correlation coefficient** measures the strength and direction of the linear relationship between two variables X and Y. Its value ranges between -1 and $+1$. With -1 meaning that the two variables are perfectly negatively correlated, and $+1$ meaning that they're perfectly positively correlated.

Mathematical Definition

Given two variables X and Y with n paired observations, the Pearson correlation coefficient r_{XY} is defined as:

$$r_{XY} = \frac{\sum_{i=1}^{n} (X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^{n} (X_i - \bar{X})^2} \sqrt{\sum_{i=1}^{n} (Y_i - \bar{Y})^2}}$$

where:

- X_i, Y_i are individual data points,
- $\bar{X}, \bar{Y}$ are the sample means of X and Y,
- n is the number of paired observations.

C. Greedy modularity communities

The Greedy Modularity Communities algorithm is a method for detecting communities in a graph by greedily optimizing the modularity measure. [4]

Modularity Definition

Given an undirected graph $G = (V, E)$ with n nodes and m edges, the modularity Q of a partition of the graph into communities is defined as:

$$Q = \frac{1}{2m} \sum_{i,j} \left[A_{ij} - \frac{k_i k_j}{2m} \right] \delta(c_i, c_j)$$

Where:

- A_{ij} is the adjacency matrix: 1 if there is an edge between nodes i and j, and 0 otherwise.
- k_i is the degree of node i.
- m is the total number of edges in the graph.
- c_i is the community assignment of node i.
- $\delta(c_i, c_j)$ is the Kronecker delta function: 1 if $c_i = c_j$, 0 otherwise.

A higher value of Q indicates a stronger community structure.

Algorithm Overview

The algorithm proceeds as follows:

1) Initialize each node in its own community.
2) Iteratively merge the pair of communities that results in the largest increase (or smallest decrease) in modularity.
3) Stop when no further increase in modularity is possible.

Modularity Gain Calculation

At each step, the algorithm computes the change in modularity ΔQ for merging two communities C_i and C_j. The modularity gain is given by:

$$\Delta Q = \left[\frac{\sum_{\text{in}} + 2e_{ij}}{2m} - \left(\frac{\sum_{\text{tot}} + k_j}{2m} \right)^2 \right] - \left[\frac{\sum_{\text{in}}}{2m} - \left(\frac{\sum_{\text{tot}}}{2m} \right)^2 + \frac{k_j}{2m} - \left(\frac{k_j}{2m} \right)^2 \right]$$

Where:

- e_{ij} is the number of edges between communities C_i and C_j.
- $\sum_{\text{in}}$ is the total number of internal edges within the combined community.
- $\sum_{\text{tot}}$ is the sum of the degrees of the nodes in the combined community.

II. Methods

For this paper, the network I will primarily be analyzing is the famous LeNet5 [5], trained to predict lables in the MNIST dataset [6].

For reference, LeNet5 has the following architecture:

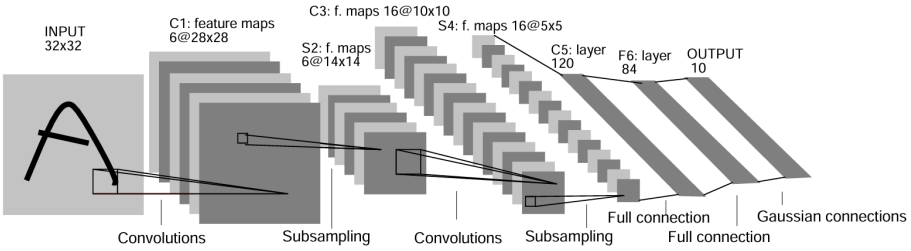
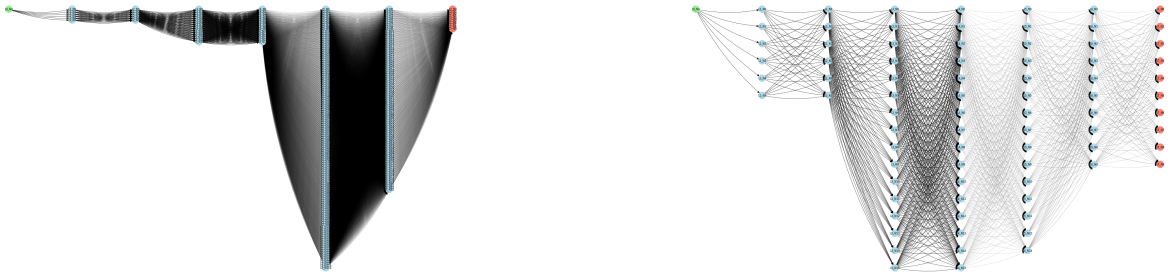


Fig. 2 LeNet5 Neural Network Architecture Structure.

If we try to visualize the full architecture using networkx, we find the resulting visual is often very cluttered and difficult to interpret. To ease this problem, I reduced the amount of nodes in each of the fully connected layers in the model by a factor of 8. Below is a side by side comparison of the 2, showing why such a step is helpful.



(a) LeNet5 not reduced at all **(b) LeNet5, with a reduction factor of** 8

Fig. 3 Comparison of the networkx visualization of LeNet5 on the full network vs on the reduced version of the network

The first part of this experiment involves the comparison of our model during three different stages of training. The first being on initialization, with random weights, the second being after one full epoch of training, and the final being the model after five full epochs of training (the point where the model starts to converge). The MNIST dataset is then split into 11 different groups, one for each digit in the dataset plus every digit combined. The co-activation values are then computed for each of these 11 groups. For this part of the experiment, the dataset will be split into a training and test set, where the co-activations will be computed using the test set. The model will be trained using MSE loss.

The second part of this experiment involves Training the model under 2 separate conditions. The first condition where the model has to predict the labels as defined by the MNIST dataset, and the second condition where the model only has to predict whether the given digit is even or odd. We then compare the co-activation graphs on the MNIST dataset with only the even numbers and with only the odd numbers.

Model Analysis and Parameter Choice

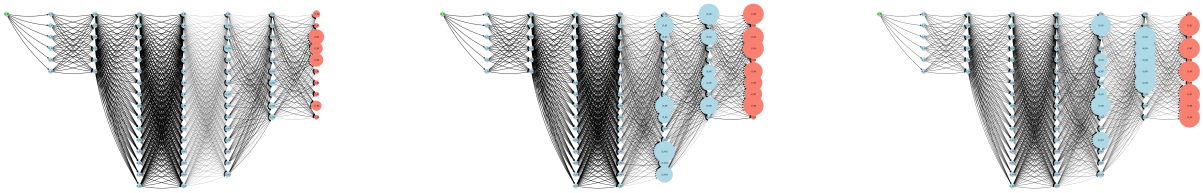
Throughout the course of this paper, I tried a variety of different community detection methods (info map, louvain community detection, greedy modularity community) and correlation methods (spearman, pearson), ultimately setting on a pearson correlation with greedy modularity community detection because those gave me the best results. To visualize the network itself, I wrote a custom spacing function that placed each node a set vertical can horizontal distance apart. Empirically, I found that a horizontal spacing of 4 and vertical spacing of 8 worked best for me. For the node sizes, I set them to be 500 on the draw graph function in networkx, and for the Page Rank visualizations I set the min node size to be 350, and the max node size to be 5,000. The exact model code and plots can all be found at[7].

III. Results

Model learns which nodes to use more throughout training

When a deep neural network is first initialized, its weights are essentially set to random, meaning that the correlation between the activations of nodes will be roughly the same across the entire network. What we expect then is that the page rank score of any one node in the network to be relatively small. However as training advances, we would expect the network build associations within the data and learn how different features correlate with each other, resulting in higher page rank values.

In fact this is exactly what we find happens. In figure 4, we have the co-activation graphs of LeNet5 computed from the entire test dataset. What's then graphed are those nodes multiplied by their respective Page Rank values.



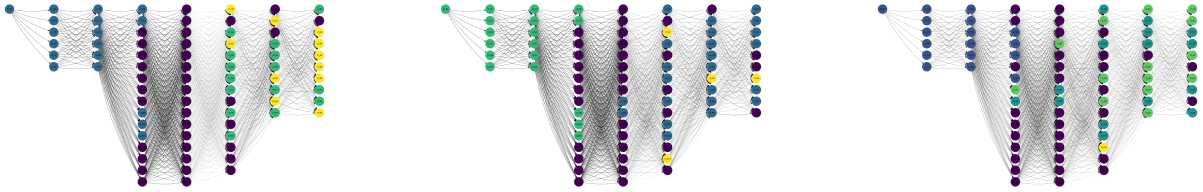
(a) No Training Done (b) One Epoch of Training (c) Fully trained Network

Fig. 4 Visualization of Page Rank Computed from Entire Test Set

Node community groups become more diverse throughout training

Following the spirit of the previous section, we now observe the community dynamics of our neural network throughout training, as measured by the greedy modularity algorithm. Since network activation correlations are homogeneous upon network initialization, we expect to find simple groups at the start of training, and more complicated groups as training proceeds.

And as we can see from figure 5, this is in fact exactly what takes place. When no training has took place, we see 4 simple groups, each clustered towards their region of the network, but as training proceeds, the communities become more and more inter leaven, showing us that training in a neural network involves learning a more elaborate interleaving of neuron activations.

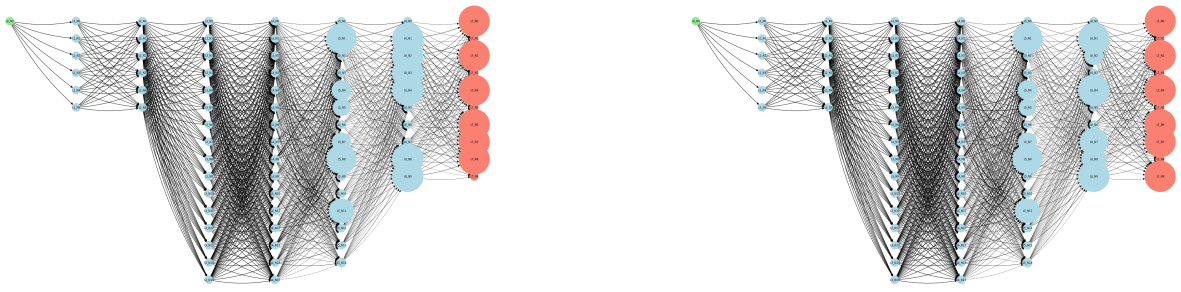


(a) No Training Done (b) One Epoch of Training (c) Fully trained Network

Fig. 5 Visualization of Greedy Modularity Communities Computed from Entire Test Set

Network uses similar nodes to identify inputs with similar features

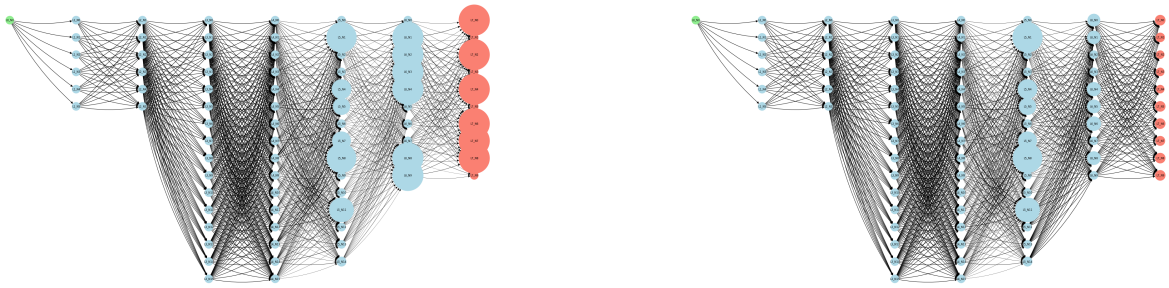
Next, we move on from visualizing the co activations of the entire test dataset to just the test dataset with certain labels. Ideally, once a network becomes fully trained, it learns how to identify general features from the input using different neurons/ different connections of neurons. To test if this hypothesis actually holds, we see in figure 6 the Page Rank visualization of the co-activation graph computed from only inputs of the image 6 and only from inputs of the image 9. Since 6 and 9 are about as similar as any 2 digits can be, we would expect these graphs to look very similar to each other.



(a) Activation Graph On Only 6 (b) Activation Graph On Only 9

Fig. 6 Comparison of Activation Graph Computed on Only Images of 6 vs Only on Images of 9

And this is exactly what we find. To further verify our suspicion, we can compare the co-activation fo graph 6 with the co-activation of a digit that shares almost no visual features with 6, 7.



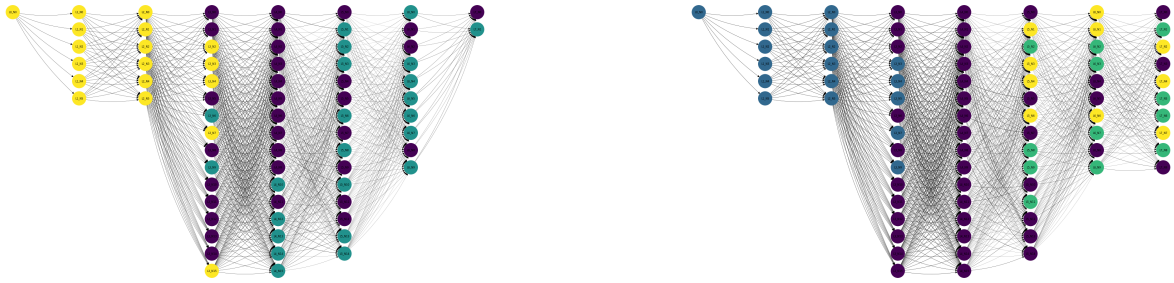
(a) Activation Graph On Only 6 **(b) Activation Graph On Only 7**

Fig. 7 Comparison of Activation Graph Computed on Only Images of 6 vs Only on Images of 7

Changing the target during training impacts the complexity of the underlying trends the network learns

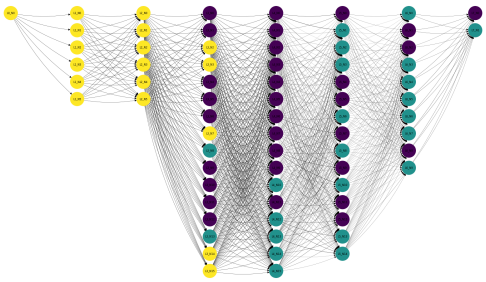
For this final result, we will focus on the second experiment I outlined in the methods section. Here I want to make a slight modification to the LeNet5 model we've been viewing so far, instead of training it to predict what value each digit has, I instead just train it to predict whether a given digit in question is even or odd. I then want to compare the co-activation graphs generated through only even digits and through only odd digits.

In theory, there's nothing preventing our modification of LeNet5 from learning exactly the same activation structure as the normal LeNet5 model. Both models achieve near identical accuracy when evaluated on the test dataset. And after all, if you can successfully predict the exact number an image of a digit represents, you can predict whether the image of the number is even or odd. However, in practice, we might suspect that since the network isn't being forced to learn the intricate details separating the individual digits, it isn't 'pushed' to learn the activation patterns that the other model would. To see which hypothesis actually holds, we look to figures 8 and 9.

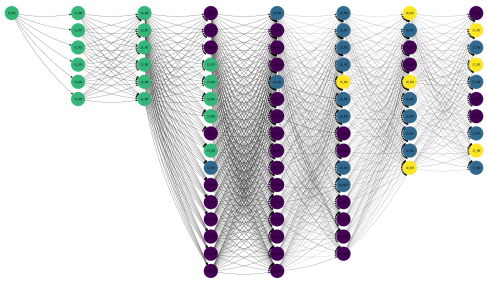


(a) Activation Graph of Even/Odd Network for Even Numbers **(b) Activation Graph of Regular Network for Even Numbers**

Fig. 8 Visualization of Communities for LeNet with 2 Labels vs With 9



(a) Activation Graph of Even/Odd Network for Odd Numbers



(b) Activation Graph of Regular Network for Odd Numbers

Fig. 9 Visualization of Communities for LeNet with 2 Labels vs With 9

And as we can see, the ladder hypothesis is confirmed, with the network prediction labels 0-9 clearly having a more complicated co-activation community clustering. This tells us that even if the architecture and hyper-parameters of two networks are exactly the same, simply having a different loss function can result in a completely different model after training.

IV. Discussion

Despite the relative simplicity of the tools we used, we were able to learn a great deal about the underlying functioning of neural networks throughout the course of these experiments. The fact that we were able to get consistent results and get results that aligned with our intuitions and understanding how how neural networks operate shows us that these methods do expose some underlying truth in how deep neural networks operate. Although I only looked at LeNet5 trained on the MNIST digit classification task, I feel that the trends I observed should hold for different network architectures performing different non-classification tasks.

A big limitation of this analysis is that I only really looked at the co-activation values for linear layers, neglecting the question of how I would handle more complicated network architecture components such as convolutional or residual layers. The absence of a procedure for computing the co-activation of convolutional layers is especially present in this paper, since LeNet5 is partially a convolutional neural network that feeds into linear layers. Another limitation is that a lot of the visualization parameters I used so that I could make the network graphs viewable were fine-tuned specifically to LeNet5, ideally there would be some method to generate a viewable networkx visual that generalizes to any network architecture.

References

[1] Page, L., "The PageRank citation ranking: Bringing order to the web," Technical report, Stanford InfoLab, 1999.

[2] Selani, D., and Tiddi, I., "Knowledge extraction from auto-encoders on anomaly detection tasks using co-activation graphs," *Proceedings of the 10th International Conference on Knowledge Capture (K-CAP)*, ACM, 2021, pp. 65–71.

[3] Hagberg, A., Swart, P., Chult, D. S., and contributors, "NetworkX: Network Analysis in Python," `https://networkx.org/`, 2025. Accessed: 2025-04-28.

[4] Clauset, A., Newman, M. E. J., and Moore, C., "Finding community structure in very large networks," *Physical Review E*, Vol. 70, No. 6, 2004, p. 066111. https://doi.org/10.1103/PhysRevE.70.066111.

[5] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P., "Gradient-Based Learning Applied to Document Recognition," Tech. rep., Stanford Vision Lab, 1998. http://vision.stanford.edu/cs598_spring07/papers/Lecun98.pdf.

[6] Kazemi, H., "MNIST Dataset," https://www.kaggle.com/datasets/hojjatk/mnist-dataset, 2020. Accessed: 2025-04-28.

[7] Ozer, O., "CMPLXSYS_530_final_paper," https://github.com/onatozer/CMPLXSYS_530_final_paper, 2025. GitHub repository, accessed: 2025-04-28.