Explainability, Fairness, and Security for Machine Learning in Sensitive Environments

Tristan Brigham
Yale University: CPSC 471
New Haven, CT 03301
tristan.brigham@yale.edu

1 Introduction

1.1 Problem Definition

Machine learning models are *not* called black boxes without reason. In most modern frameworks (especially gradient-based methods), it is difficult if not impossible to get a verifiably true understanding of the logic behind a neural network's output. Essentially every explainability method suffers from at least one (if not multiple) of:

1. It is extremely time consuming to implement (e.g. VQA (W. Kim, Son, and I. Kim 2021))
2. It is for simple models (e.g. decision trees, regressions)
3. Inference accuracy is substantially hurt (e.g. Cognitive Models (Blazek and Lin 2021))
4. The result is uncertain (e.g. LIME (Ribeiro, Singh, and Guestrin 2016), SHAP (Lundberg and Lee 2017))

If any of the above is the case for a given model, then it severely constrains where that model can be deployed. However, with ML continually being integrated into more systems with increasingly critical functions, it is becoming increasingly important to find methods that avoid the pitfalls above. Therein lies a key problem in machine learning I seek to address: *is it possible to have explanation methods that avoid every pitfall above?*

There are established best practices for developing models (Clarke 2024, Bachratý 2020, Ray 2024). I seek to provide the first comprehensive method for deploying machine learning in sensitive environments while avoiding the issues above by integrating new explainability and fairness methods into said well-defined best practices.

I define a sensitive environment as one where there exists the potential for unfair bias with an outsized impact. Given the imprecise nature of this definition, examples of such environments such as an insurance agency making decisions on whether to insure someone's property, a judicial system carrying out criminal justice, and a hospital making a diagnosis decision for a patient are included to help the reader. In the following sections, I will outline my contributions to any effective deployment of machine learning in sensitive environments. My approach utilizes novel methods to enforce explainability and fairness and draws on established cybersecurity principles to guarantee trust.

1.2 Relevance to Trustworthy Aspects

The ability to explain decisions in neural networks improves **every** one of the trustworthy aspects (robustness, explainability, privacy, and fairness) discussed in CPSC471 this semester. However, my methods most specifically target explainability, fairness, and adversarial robustness through concept-focused training, influence re-weighting, and weight freezing, respectively:

1. **Explainability:** The idea behind the concept-focused training is to push nodes to represent concepts and patterns present in the training data. The goal is to create a machine learning model where one can analyze the activations of concept nodes to understand what patterns the model has found in the data and is making its decisions on. I introduce an approach to more explainable machine learning models. The approach is similar to the CAV framework discussed in class in finding ways to represent data, but has advantages in its ability to find the most important concepts present in data automatically instead of requiring human intervention to define concepts and model integration (Druc et al. 2022).

2. **Fairness:** Using a combination of the broad explainability provided by the concept-focused training and other granular explainability methods such as Layer-wise Relevance Propagation (Binder et al. 2016), LIME (Ribeiro, Singh, and Guestrin 2016), or Grad-CAM (Selvaraju et al. 2019), the end user can gain insight into the logic behind models' decisions. Using the influence re-weighting approach I propose in this paper, the end user can decrease the influence that any concept or input has on the output. This method can be used to limit unfair or harmful decision factors and thereby increase the fairness of the trained model in substantially less time than full re-training.

3. **Adversarial Robustness:** In the final part of this framework I ensure that the model maintains user trust using known hashing algorithms such as SHA-256 through weight freezing (Bergstra and Middelburg 2017). Hashing the critical attributes of a model creates a fingerprint for the model that is extremely difficult to re-create. This directly relates to the trustworthy and robust aspects of ML we talked about in class by making secret permutations of the model impossible and upholding trust once the model has been hashed.

2 Related Works

Several existing methods seek to explain machine learning. Some of the most prevalent and related are noted below:

1. OpenAI used GPT-4 to analyze the neuron activations of GPT-2. The goal was to create an automated process for understanding the logic hidden in LLM neurons given how large the models can be. The authors found that GPT-4 could make basic inferences about the neuron activations (Bills et al. 2023).

2. Saliency maps along with their derivative methods have showed promise in explaining what parts of an input a model focuses on and uses to generate its output. I use saliency maps to calculate the individual neuron influence in this project (Simonyan, Vedaldi, and Zisserman 2014).

3. SHAP is one of the leading approaches for model-agnostic explanations of machine learning. It uses Shapley values to estimate the contribution that each feature provides to the model, considering both the feature itself and its combinations with other features (Ribeiro, Singh, and Guestrin 2016). However, this method runs in an exponential runtime and relies on models providing probabilistic and continuous outputs to score the confidence.

4. LIME creates a locally explainable (typically linear) surrogate model with a weighting function to adjust the influence that points have on the surrogate model to estimate what data points and features inform the output (Ribeiro, Singh, and Guestrin 2016).

5. Researchers have created neural networks where the sub-components are decision networks to decide whether a concept is present or not in the input. The outputs of that network are agglomerated and passed to another network that generates the final network output (Blazek and Lin 2021). Using the sub-decision networks guarantees that the model makes decisions based off of the concept model outputs which can improve interpretability with extreme added overhead.

Less exploration of intrinsically explainable machine learning methods has been done, and minimal research has focused on the topic since 2019 (Card, Zhang, and Smith 2019, Alvarez-Melis and Jaakkola 2018).

3 Proposed Approaches

Using my approach as an addition to best practices, machine learning models can begin to be deployed in sensitive environments with verifiable logic and results. My approach involves 3 aspects, the first two of which are novel:

3.1 Concept-Focused Training

Using a custom-defined loss function, I manipulate the gradients during training such that the parameters force the activations of neurons to represent patterns or ideas that the model will decide on. The inspiration for this process comes from Druc et al. 2022. In my method, instead of training surrogate models to differentiate concepts, the concepts are simply integrated into the node activations. The idea is that for any parameter configuration that is not at the absolute local minimum for data (which most models are not), there exists a manifold of model parameters that produce the same loss value. I aim to train the model such that it finds the optimal parameter configuration on that manifold for explainability.

Before training the model, I run PCA on the input data to get embeddings with minimized noise for the data points. By doing so, we can exploit the fact that PCA finds a low-dimensional representation while maintaining the most important features and variance for distinguishing classes (Kwak and Choi 2003). The embeddings can be found in Figures **??**, **??**, and **??**. An example number of data points with the minimum distance to each centroid is found in Figure 10b.

After running PCA, I run a clustering algorithm to generate C centroids for the data where C is a user-defined hyperparameter. The best results come when C is roughly $\frac{\sqrt{n}}{5}$ for the dimensionality of the data n. These centroids will essentially represent distinct concepts or ideas present in the data. A point's distance to these centroids judges how likely the point is to include the centroids' concepts.

For each of the training data points we compute the Mahalanobis Distance between the training point and each of the C centroids found above in the PCA representations of the data (Mclachlan 1999). Using the Mahalanobis distance accounts for any skew or covariance remaining within sub-clusters of the data.

Then, train the model using the original training data (not the PCA-decomposed version) where the same loss as normal is used. However, instead of applying the normal gradient to each parameter at each update step, transform the gradient g_i into g_i' for a single parameter i in the model as follows before clipping the gradient to mitigate infinite values:

$$g_i' = \epsilon \cdot \frac{s}{C} \sum_{j=1}^{C} \Bigg(((1 - \sigma(g_i, w_i)) \cdot (1 - m_{j,\text{norm}}) + \sigma(g_i, w_i) \cdot m_{j,\text{norm}}) \cdot \sigma(\|\Delta_o\|) \\ + (\sigma(g_i, w_i) \cdot (1 - m_{j,\text{norm}}) + (1 - \sigma(g_i, w_i)) \cdot m_{j,\text{norm}}) \cdot (1 - \sigma(\|\Delta_o\|)) \Bigg) + g_i$$

where m_{max} is the maximum Mahalanobis distance in the PCA space between any two points across all of the input data. $m_{j,\text{norm}} = \frac{m_j}{m_{\text{max}}}$ is the normalized mahalanobis distance from the original training input data to the j^{th} centroid computed above since m_j is the raw Mahalanobis distance.

$\sigma(g_j, w_j) = \sigma(\text{sign}(g_j) \cdot \text{sign}(w_j))$ represents a 1 if the sign of each of the values g_j and w_j is the same and 0 otherwise. g_j represents the j^{th} entry in the gradient and w_j represents the j^{th} weight in the weight matrix. It is used to update the gradient in the manner such that the outputs at the target layer are similar for inputs with a small Mahalanobis distance, and are disparate for different inputs.

o_j is the output of the layer that g' is updating when the j^{th} centroid was used as input, and o_c is the output of the layer given the current training point as input. Hence, $\Delta_o = o_j - o_c$. ϵ is a hyperparameter with a small value. s is a parameter that controls the strength of the gradient with respect to how deep we want the concept nodes to be $s = e^{-\left(\frac{l_c}{l_t} - p_e\right)^2}$ where l_c represents the index of the layer that the gradient is being applied to, l_t is the total number of layers there are, p_e is a value between 0 and 1 that controls how deep the concept nodes should be (0 means the inputs will represent the concepts, or the features are the concepts themselves, and 1 means the model outputs will represent the learned concepts). It is maximized and equal to 1 when $\frac{l_c}{l_t}$ is closest to p_e, and a

value of p_e such that the second-to-last or third-to-last layers contain the concept nodes is best as the direct impacts of certain concepts being present or not in the input data can be understood best there.

The idea behind this is to make the output of a given layer be close to a weighted average of the outputs where o_j is upweighted inversely with the mahalanobis distance from the current input to centroid j in the PCA space, and to make the outputs as different as possible when the distance between a centroid and the point is large.

We also limit the number of gradients that we perturb so that we only change the gradients that have the strongest negative or positive activations (typically the top 30% has shown good results).

Finally, to push the neurons to represent concepts, I apply weight-decay to the intermediate layers. Given the difference constraint imposed above with the weight-decay, the neurons start to represent a low-dimensional form of the required concepts to generate the correct outputs.

The output is a deep layer which activates different neurons for the presence of different ideas in the input data and contains enough information to generate correct outputs. In essence, these are the model-defined most important concepts in the data.

3.2 Influence Re-Weighting

Once a user finds what the model believes is the most important set of concepts in the input data through concept-focused training, the user can limit the influence of features deemed harmful or unfair through influence re-weighting without re-training the model from scratch. Influence of concepts are measured using layer-wise relevance propagation and saliency maps. We explore novel techniques such as stochastic fine-tuning, inversion fine-tuning, and amplitude influence-reduction as candidates for removing feature influence.

Stochastic fine-tuning trains the machine learning model using the original training data but replacing the harmful features with stochastic variables. The model should learn that these stochastic features should be ignored, and their influence is minimized. I have attempted to generalize this to concepts defined by combinations of input features without luck, so it only works for input features currently.

Inversion fine-tuning is the same process but instead of replacing the harmful features with noise, the features are inverted about their dataset means in the input data. I have attempted to generalize this to concepts defined by combinations of input features without luck.

$$x[i] = \text{mean}(x[:, i]) - 2 \cdot (x[i] - \text{mean}(x[:, i]))$$

In amplitude influence-reduction, we simply try to reconstruct the concept that we are attempting to remove using the other activations in the preceding layer, and adjust the other features' weights to offset this loss. I explored several algorithms to construct the optimal weighting, but the best outcome comes from running a regression using the other features in the training data as input values and adjusting the weights in the subsequent layer to offset the change in the harmful feature's weights. The weights that are multiplied with the harmful feature are set to zero, effectively turning off this feature. The error increases with this new regime, so some fine-tuning is required at the cost of re-introducing the concept slightly. Essentially, to remove the influence of node i' in a layer, we adjust the weights such that:

$$w'_j = \begin{cases} 0 & \text{if } j = i' \\ w_j \cdot \left(1 + \frac{\|w_{i'}\|}{\sum_{k=1}^{n} w_k}\right) & \text{otherwise} \end{cases}$$

If we are trying to remove a concept, we must limit our fine-tuning or else the concept will simply be re-learned in another neuron or set of neurons. Else, with concepts as input features one can fine-tune and enforce that the harmful feature's weights are zero without issue.

3.3 Weight Freezing

Finally, once a model is certified to possess certain properties by some trusted body, the deployed model must keep those properties. The stakeholders actually deploying the model likely do not want to re-run the tests locally to limit computation costs, comparing the weights of the local model against a remote version can be expensive given communication overheads, and it is risky to send

weights over communication networks incase malicious actors intercept the weights and potentially run attacks to infer data or model attributes.

Stakeholders can run a deterministic hashing algorithm over the important attributes (e.g. weights, activations, layer sizes, etc) of a model and compare this against the trusted body's hash of the model to see if the model still possesses the attributes that the trusted body said it does in a single communication step without communicating the weights or sensitive data. Given that hashing is not a novel concept, I refer the user to Martín-Fernández and Caballero-Gil 2022 for more information and use Geeks for Geeks 2023 as a resource for my implementation.

4 Experiment Section

4.1 Datasets

The first dataset I use is a life insurance decisions dataset from Prudential. The dataset was released in 2016 as part of a Kaggle competition. It provides anonymized insurance decisions with numerical representations of applicant attributes like income, height, and weight (Montoya et al. 2015).

There are 59,381 entries across 128 columns with a mean height and weight across the dataset of 0.707 and 0.293 respectively. Medical keywords are sparse (as evidenced by the low mean values) and the vast majority of the data is normalized to a distribution between 0 and 1.

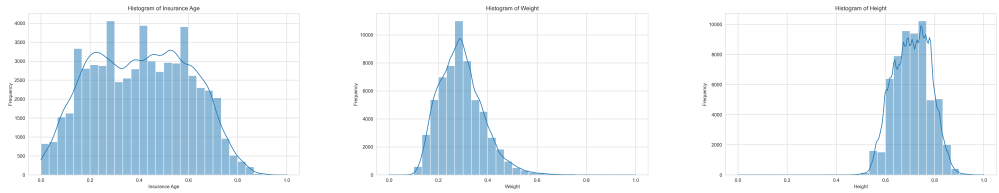
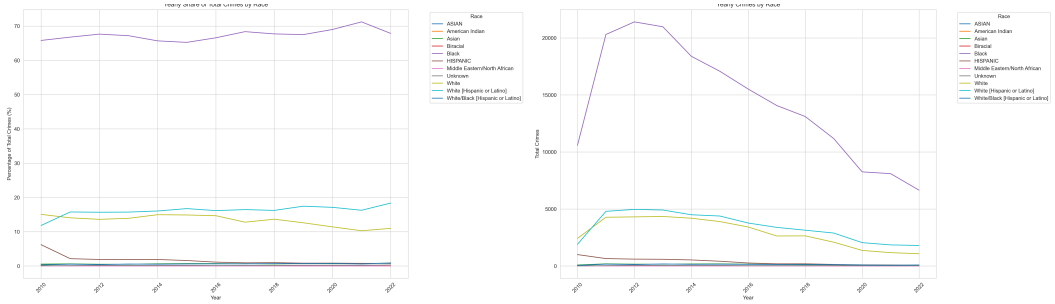


Figure 1: Histograms of Age, Weight, and Height

Because the dataset is anonymized, we cannot be certain what the response values included in the dataset mean. But, an educated guess given the relative prevalence of different values might be Declined, Postponed, Rated, Standard, Preferred, Elite, Smoker, and Accepted for values 0-7.

The Cook County Criminal Justice Dataset comes from the Chicago Open Data City Initiative (City of Chicago 2024, Cook County, Illinois Data Catalog 2024). This dataset will serve as a validation dataset given its larger size and easier interpretability due to lesser privacy constraints.

Analyzing the data, I find that young men are dramatically over-represented in the data compared to their share of the population. Additionally, as confirmed by numerous studies, people of minority backgrounds are systemically more likely to experience the criminal justice system in Chicago (M. K. Chen et al. 2022, Minor 2023, American Civil Liberties Union of Illinois 2015).



(a) Yearly Share of Total Crimes by Race

(b) Yearly Crimes by Race

Figure 2: Racial distribution of crime

4.2 Benchmarking

Training a basic neural network without concept-focused training yields medial results with uninterruptible intermediate layers. As a test, I run LIME on the base neural network which has a mediocre result on the life insurance dataset, and a poor result on the criminal justice dataset. The computed relative importance values are extreme and incorrect. LIME seems to worsen with the complexity of the dataset.

Given concept-focused training is a method to create intrinsically explainable machine learning, I fit an XGBoost model to the data as a baseline. XGBoost is a tree-based model that has been shown to be one of the most accurate explainable models (T. Chen and Guestrin 2016). We look to the validation accuracy and Cohen's Kappa values of the models (McHugh 2012).

The concept-focused neural network uses the gradient boosting equation explained above to generate concept-activated neurons in the second-to-last layer of the model. The Single Layer Concept-focused Neural Network only applies the gradient boosting to the second-to-last layer (none of the layers before it). All networks have a structure the same as the structure in Figure 10a.

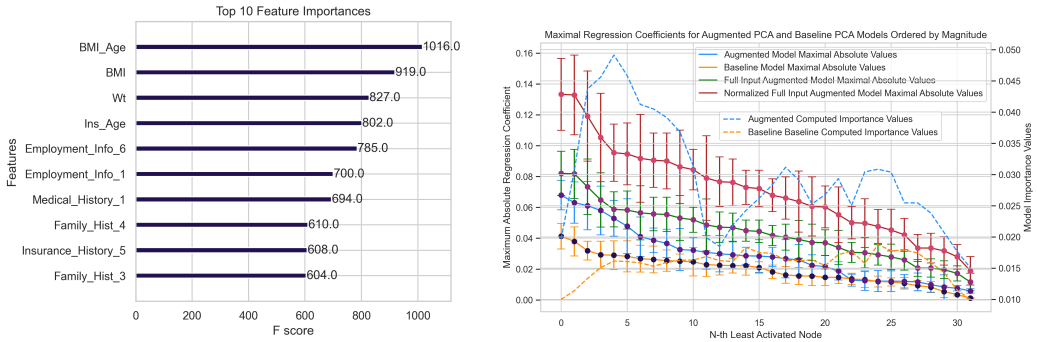
Table 1: Average Model Performance

Model	Cohen's Kappa	Validation Accuracy (%)
Vanilla XGBoost	0.6510	28.45
Unchanged Neural Network	0.4326	44.14
Concept-focused Neural Network	0.4127	43.91
Single Layer Concept-focused Neural Network	-0.01647	10.37

I find that the introduction of concept-focused training does not degrade the neural network performance when the gradient boosting is scaled up from the start of the model to the target layer. However, when the gradient boosting is only applied to a single layer the model collapses. This likely indicates that the concepts must be built up over the course of the neural network layers – they cannot be constructed immediately from an arbitrary layer's output.

The higher Cohen's Kappa value but lower accuracy in XGBoost comes from the tree model being better at not over-predicting a single class.

Now, looking to the relative explainability of the models, I find that the concept-focused neural network has better explainability than the XGBoost model.

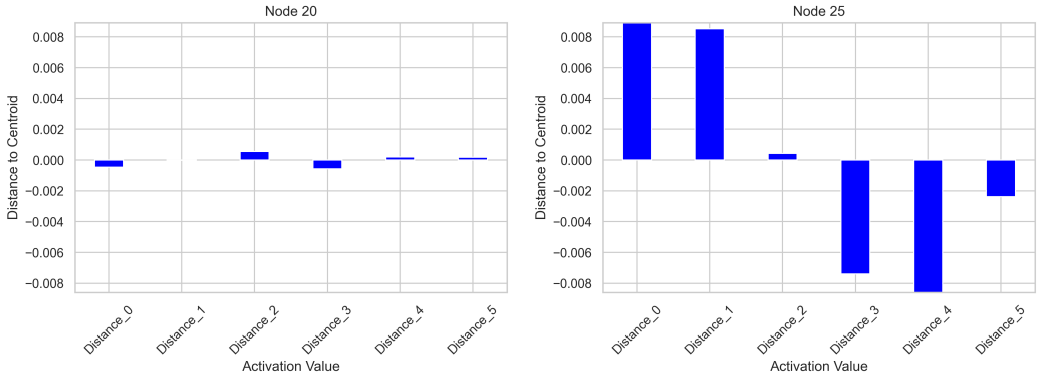


(a) XGBoost's Self-Reported Top Features (b) Concept Activation of Second-to-Last Layer Neurons

Figure 3: Explainability of Investigated Models

The XGBoost model is only able to provide high-accuracy feature importance scores for the input features. Although tree-based models can give somewhat inaccurate concept-based importances, the definitions of the concepts are rather arbitrary and it is difficult to define concepts that the model is actually using with any degree of confidence.

On the other hand, in the concept-focused model, I use established methods such as SHAP and GradCAM to get average individual feature importance in the model in only slightly more time than



(a) Concept neuron with weak concept utilization (b) Concept neuron with strong concept utilization

Figure 4: Concept Activation of Neurons

XGBoost's explanation framework. The model yields concept-relevance explanations instantly given the concept-focused training. On an instance level, we can compute the importance of any neuron in the model with existing methods, and cross-check what concepts the important neurons represent. In Figure 3b, we plot the augmented model trained on the PCA's maximal coefficients (which has the largest spike on the left), the augmented model trained on the full input with PCA distance (the second highest on the left), the un-augmented model trained on PCA, and the un-augmented model trained on the full input. The augmented model has several neurons which represent combinations of subsets of the model-defined concepts with high fidelity (the left side of the graph), while the rest of the neurons possess residual information for generating accurate outputs.

We run a regression on each of the neurons to find out which are most correlated with weighted combinations of concepts. A subset of the neurons in augmented models use the PCA-defined concepts (nodes represented on the left of the graph). These represent the concepts that the model find most informative. The latter neuron regressions revert to normal values with relatively high output influence which signifies important residual information.

Because neurons' computed influence has strongly positive covariance with the concept alignment, we can infer that the final layer of the model is using the concepts to compute the final model output. The total effect of the concept on the output is the magnitude of the regression coefficient times the node importance. Additional explanation information is plotted in the supplementary portion of this report at Figures 5a, 5b, **??**, and **??**.

Figure 4a shows a neuron in the deeper layers that does not necessarily use the model-defined concepts to make its decision. It is also a neuron with very weak influence on the output after training because of the weight-decay. The existence of these neurons shows that the model is selective in the concepts it defines. It leaves as many neurons with low influence and concept influence as possible. On the other hand, Figure 4b shows a neuron that uses a concept which is a combination of concepts 0 and 1, and the opposite of 3 and 4. To give an example, if the model decides concept 0 is a combination of features to represent violence exposure during childhood, concept 1 is the crime's violence level, concept 3 represents the convicted's employment status, and concept 4 is the racial bias of the arresting county's police force, this neuron could indicate a concept like the individual's likelihood to recidivate.

We test how well each of the methods do in removing concept influence compared to baseline retraining. The average runtime of each method is found in Table 5. Refer to Table 4 for the full study.

As seen by comparing Table 2 and Table 3, each of the PCA components have relatively equal influence on the output of the model. This seems counterintuitive at first since earlier PC's should represent larger shares of the data information. It should get easier to mitigate the loss increase due to the decreasing variance. Removing any single concept increases the validation loss. Additionally, the model doesn't seem to be affected until the concept is at least 50% removed when validation loss

Table 2: Ablation Study of Removing Different Principal Components' Influence Impact on Val Loss

Reduction Method	PC 0 (%)	PC 2 (%)	PC 5 (%)	PC 7 (%)	PC 9 (%)
Amplitude Reduction	1.90	1.86	1.86	1.86	1.88
Stochastic Fine Tuning	1.23	1.23	1.25	1.41	1.29
Inversion Fine Tuning	0.05	0.01	0.09	0.19	0.05
Simple Retraining	0.99	0.98	0.96	1.25	1.16

Table 3: Average Influence Reduction per Function from Removing Principal Components

Reduction Method	PC 0 (%)	PC 2 (%)	PC 5 (%)	PC 7 (%)	PC 9 (%)
Amplitude Reduction	-100.00	-100.00	-100.00	-100.00	-100.00
Stochastic Fine Tuning	-65.93	-68.57	-73.66	-66.78	-65.39
Inversion Fine Tuning	-22.73	-23.48	-24.75	-22.88	-21.85
Simple Retraining	-100.00	-100.00	-100.00	-100.00	-100.00

shoots up. The best method is the amplitude influence-reduction method with similar runtime to stochastic fine-tuning but the performance guarantees of re-training the model.

5 Conclusion

5.1 Key Takeaways

I sought to introduce a new framework for observable and fair neural networks without compromising on model effectiveness to open the black box of neural networks. The concept-focused training approach detailed in this project allows for stakeholders to understand the concepts that the model believes are most important to inference while influence re-weighting operations remove such harmful concepts' impact on the output of the model to enforce fairness.

These approaches do not compromise the model's accuracy or inference speed and should be integrated to ensure models in sensitive environments are fair and explainable. Without such broad methods, stakeholders take on too much risk and cannot claim to be deciding fairly.

5.2 Limitations of Method

Despite the forward progress this study has made into explainability, it suffers from some drawbacks. First: sometimes the explainability method just doesn't work. I estimate that I am able to guide the network towards defining and utilizing the PCA concepts roughly 60% of the time. I believe that with further tuning and optimization of the gradient boosting, this success probability can be increased. But, the potential for failure of concept convergence can increase training time and make the method prohibitively expensive for large models with high associated costs.

Additionally, after trying 4 clustering methods to define the concepts, I found that the optimal clustering scheme is highly model-dependent. Further investigation should be done to develop a method for choosing such a scheme. K-Means seems to have the most reliably good results, but is often not the best.

Finally, the method is time-consuming. Without low-level programming and potential integration of gradient boosting into common machine learning frameworks, this method in its currently non-optimized form can increase convergence time by an order of magnitude. Given how non-optimized the code is, I suspect this can be brought down, but any gradient manipulation will increase the training time.

6 Reproduction of Findings

For reproduction of the findings, please refer to the Github Repository at B. 2024.

Cited References

[1] David Alvarez-Melis and Tommi S. Jaakkola. "Towards Robust Interpretability with Self-Explaining Neural Networks". In: *Proceedings of the 32nd Conference on Neural Information Processing Systems (NeurIPS)*. Montréal, Canada, 2018.

[2] American Civil Liberties Union of Illinois. *Stop and Frisk in Chicago*. Mar. 2015. URL: https://www.aclu-il.org/en/publications/stop-and-frisk-chicago.

[3] Tristan B. *CPSC471 Final Project: A Framework for Model Security, Fairness, and Explainability*. https://github.com/TristanB22/CPSC471_Final. Code for the final project in CPSC471 at Yale University. 2024.

[4] Viktor Bachratý. "Machine Learning With Highly Sensitive Data". In: *Jumio Engineering & Data Science* (Aug. 2020). Accessed: 2024-05-03. URL: https://medium.com/jumio-engineering-data-science/machine-learning-with-highly-sensitive-data-%3Carticle-id%3E.

[5] J. A. Bergstra and C. A. Middelburg. *Instruction sequence expressions for the secure hash algorithm SHA-256*. 2017. arXiv: 1308.0219 [cs.PL].

[6] Steven Bills et al. *Language models can explain neurons in language models*. https://openaipublic.blob.core.windows.net/neuron-explainer/paper/index.html. 2023.

[7] Alexander Binder et al. *Layer-wise Relevance Propagation for Neural Networks with Local Renormalization Layers*. 2016. arXiv: 1604.00825 [cs.CV].

[8] Paul J. Blazek and Milo M. Lin. "Explainable neural networks that simulate reasoning". In: *Nature Computational Science* 1 (2021), pp. 607–618. DOI: 10.1038/s43588-021-00132-w.

[9] Dallas Card, Michael Zhang, and Noah A. Smith. "Deep Weighted Averaging Classifiers". In: *FAT* '19: Conference on Fairness, Accountability, and Transparency*. ACM. Atlanta, GA, USA, 2019, p. 13. DOI: 10.1145/3287560.3287595.

[10] M. Keith Chen et al. *Smartphone Data Reveal Neighborhood-Level Racial Disparities in Police Presence*. 2022. arXiv: 2109.12491 [econ.GN].

[11] Tianqi Chen and Carlos Guestrin. "XGBoost: A Scalable Tree Boosting System". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. ACM, Aug. 2016. DOI: 10.1145/2939672.2939785. URL: http://dx.doi.org/10.1145/2939672.2939785.

[12] City of Chicago. *City of Chicago Data Portal*. 2024. URL: https://data.cityofchicago.org/.

[13] Harrison Clarke. *Mastering MLOps: Best Practices for Secure Machine Learning Systems*. Accessed: 2024-05-03. 2024. URL: https://www.harrisonclarke.com/blog/mastering-mlops-best-practices-for-secure-machine-learning-systems.

[14] Cook County, Illinois Data Catalog. *Sentencing*. 2024. URL: https://datacatalog.cookcountyil.gov/Courts/Sentencing/tg8v-tm6u/data.

[15] Stefan Druc et al. *Concept Activation Vectors for Generating User-Defined 3D Shapes*. 2022. arXiv: 2205.02102 [cs.CV].

[16] Geeks for Geeks. *How to encrypt and decrypt strings in Python*. https://www.geeksforgeeks.org/how-to-encrypt-and-decrypt-strings-in-python/. Accessed: 2024-05-03. 2023.

[17] Wonjae Kim, Bokyung Son, and Ildoo Kim. *ViLT: Vision-and-Language Transformer Without Convolution or Region Supervision*. 2021. arXiv: 2102.03334 [stat.ML].

[18] Nojun Kwak and Chong-Ho Choi. "Feature Extraction Based on ICA for Binary Classification Problems". In: *IEEE Trans. Knowl. Data Eng.* 15 (2003), pp. 1374–1388. DOI: 10.1109/TKDE.2003.1245279.

[19] Scott Lundberg and Su-In Lee. *A Unified Approach to Interpreting Model Predictions*. 2017. arXiv: 1705.07874 [cs.AI].

[20] F Martín-Fernández and P Caballero-Gil. *Analysis of the new standard hash function*. 2022. arXiv: 2209.11857 [cs.CR].

[21] Mary L McHugh. "Interrater reliability: the kappa statistic". In: *Biochem Med (Zagreb)* 22.3 (2012), pp. 276–282.

[22] G. Mclachlan. “Mahalanobis Distance”. In: *Resonance* 4 (June 1999), pp. 20–26. DOI: 10.1007/BF02834632.

[23] Jasmine Minor. *Traffic study finds CPD 6 times more likely to stop Black drivers*. ABC7 Chicago. Nov. 2023. URL: https://abc7chicago.com/chicago-police-traffic-stops-free2move-study-racial-profiling/14054056/.

[24] Anna Montoya et al. *Prudential Life Insurance Assessment*. https://www.kaggle.com/competitions/prudential-life-insurance-assessment. Kaggle competition. 2015.

[25] Josh Ray. *Generative AI Security: 11 Best Practices*. https://www.cloudticity.com/blog/generative-ai-security-11-best-practices. Accessed: 2024-05-03. Mar. 2024.

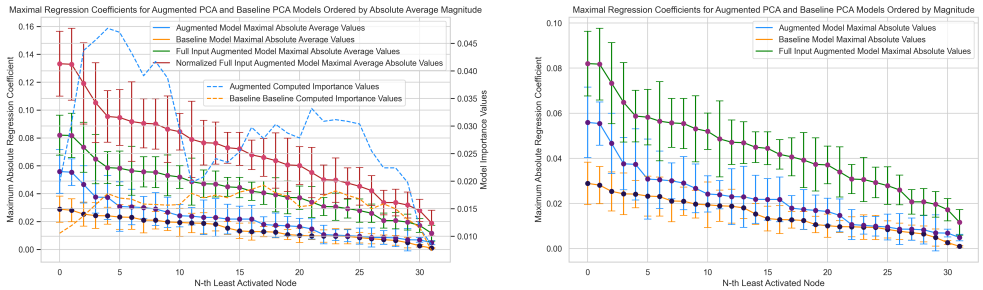
[26] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. *"Why Should I Trust You?": Explaining the Predictions of Any Classifier*. 2016. arXiv: 1602.04938 [cs.LG].

[27] Ramprasaath R. Selvaraju et al. “Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization”. In: *International Journal of Computer Vision* 128.2 (Oct. 2019), pp. 336–359. ISSN: 1573-1405. DOI: 10.1007/s11263-019-01228-7. URL: http://dx.doi.org/10.1007/s11263-019-01228-7.

[28] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. *Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps*. 2014. arXiv: 1312.6034 [cs.CV].

7 Supplementary Material

Below, the reader can find additional material that is helpful to the understanding of the project. Figures are included with brief explanations of their relevance to the project.



(a) Average Concept Importances with Normalized (b) Concept Importances without Normalized

Figure 5: Concept Importance Graphs

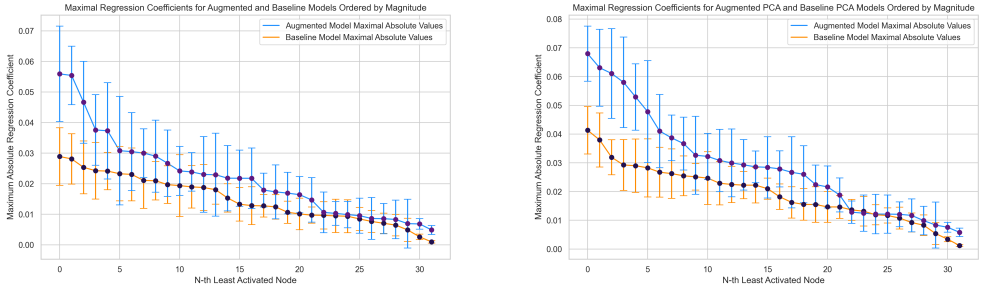


Figure 6: Concept Importance Graphs

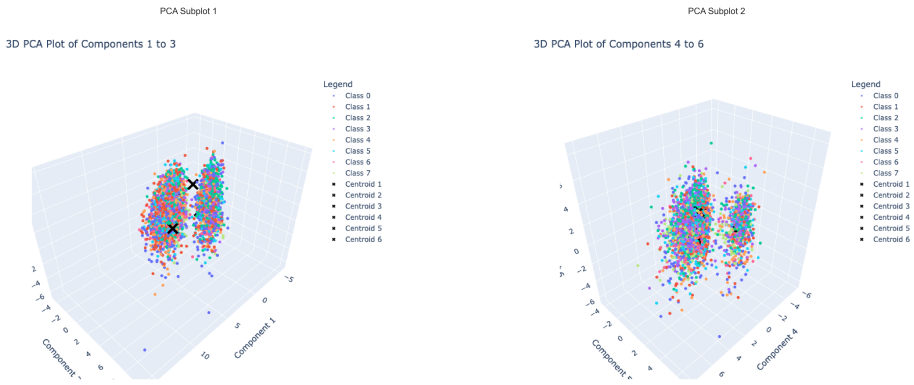


Figure 7: First and Second Sets of Principal Components with Centroids

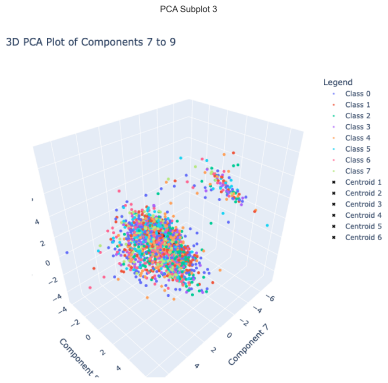
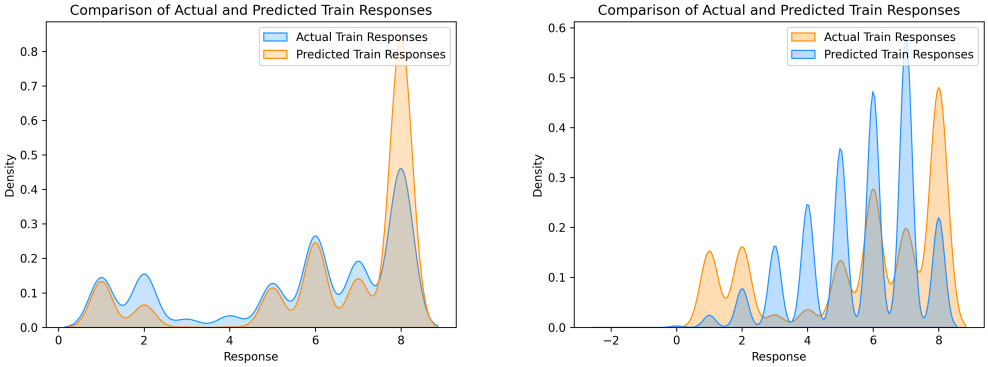
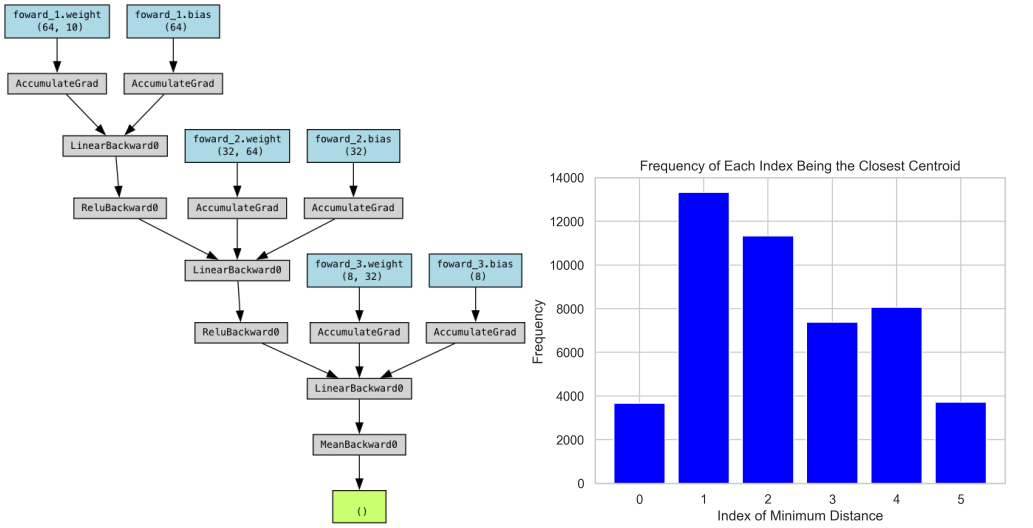


Figure 8: Third set of Principal Components with Centroids

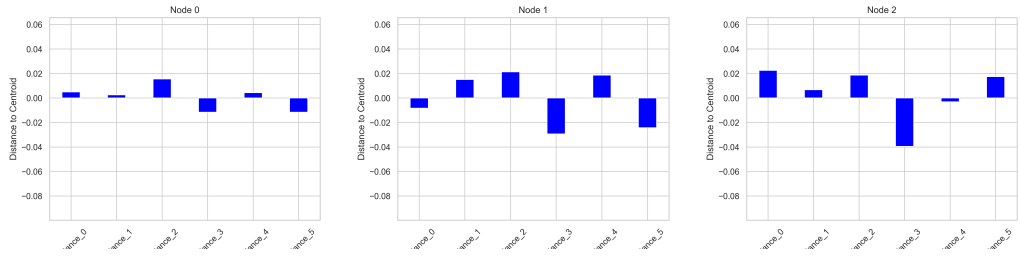


(a) Base Neural Network Train Accuracy (b) XGBoost Train Accuracy

Figure 9: Baseline Model Accuracy

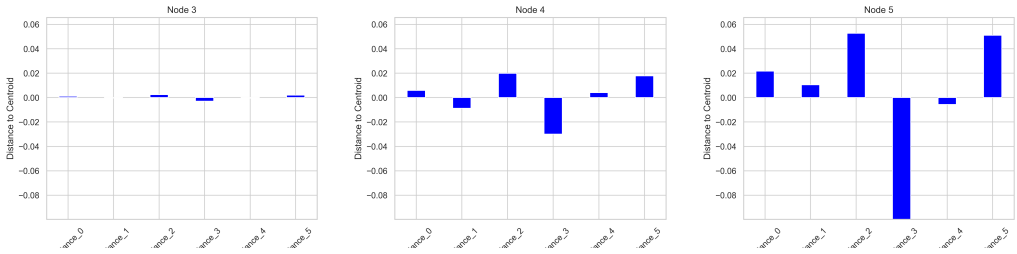


(a) Neural Network Structure (b) Minimum Distance from Centroids Histogram



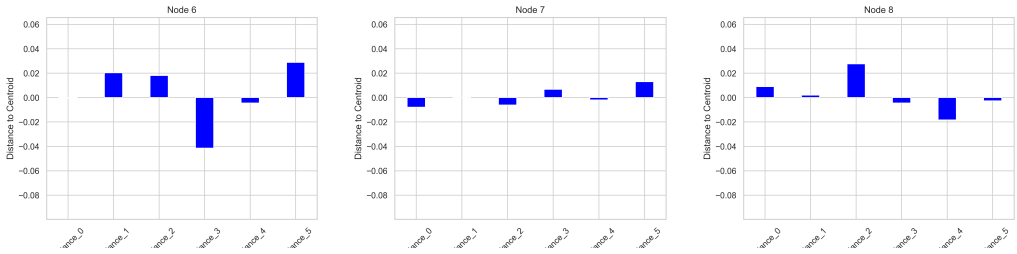
(a) (b) (c)

Figure 11: Regression Coefficients for Baseline Neurons 0, 1, and 2



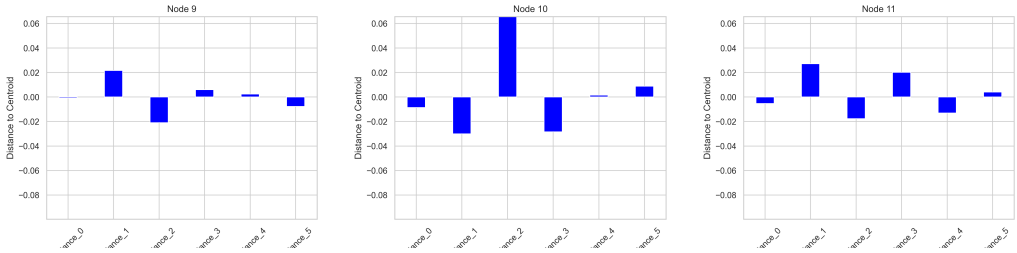
(a) (b) (c)

Figure 12: Regression Coefficients for Baseline Neurons 3, 4, and 5



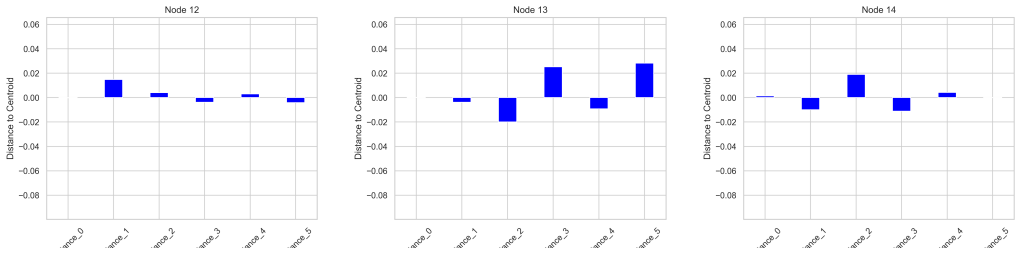
(a) (b) (c)

Figure 13: Regression Coefficients for Baseline Neurons 6, 7, and 8



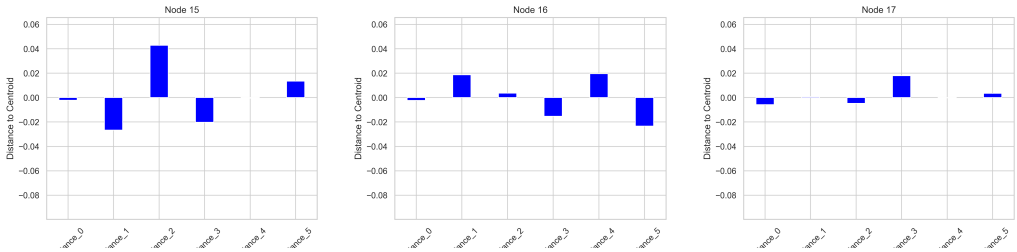
(a) (b) (c)

Figure 14: Regression Coefficients for Baseline Neurons 9, 10, and 11



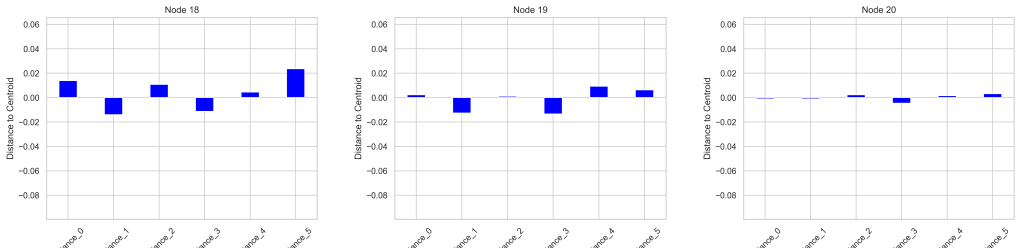
(a) (b) (c)

Figure 15: Regression Coefficients for Baseline Neurons 12, 13, and 14



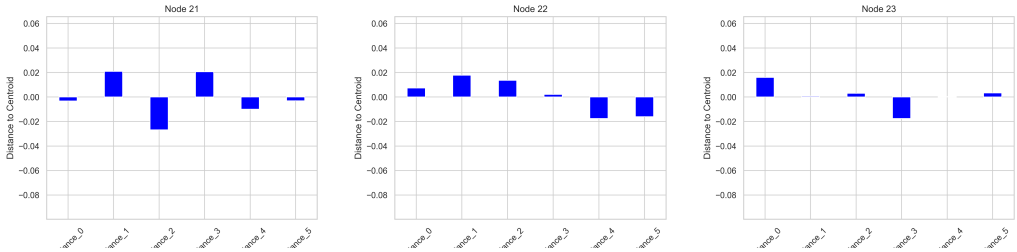
(a) (b) (c)

Figure 16: Regression Coefficients for Baseline Neurons 15, 16, and 17



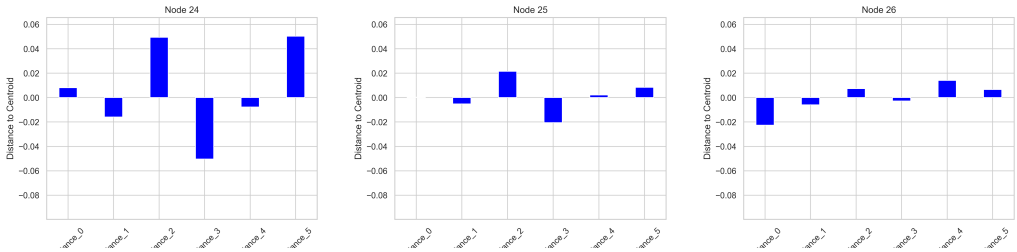
(a) (b) (c)

Figure 17: Regression Coefficients for Baseline Neurons 18, 19, and 20



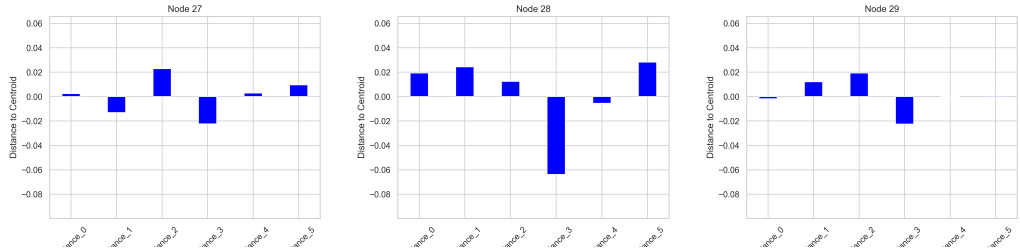
(a) (b) (c)

Figure 18: Regression Coefficients for Baseline Neurons 21, 22, and 23



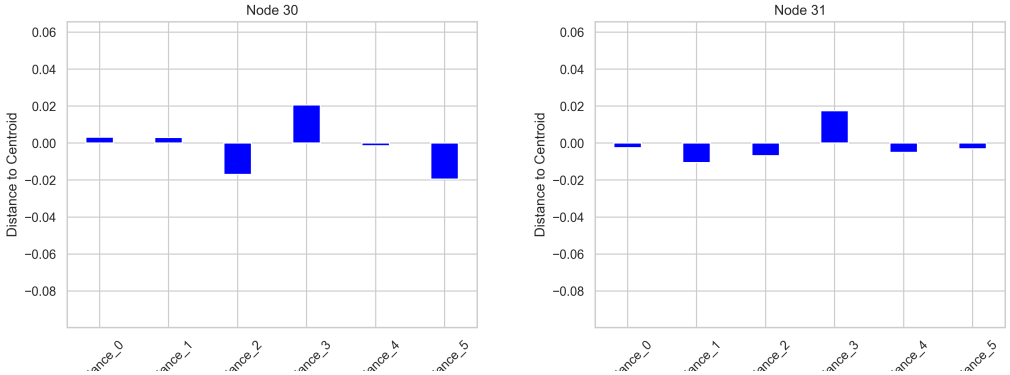
(a) (b) (c)

Figure 19: Regression Coefficients for Baseline Neurons 24, 25, and 26



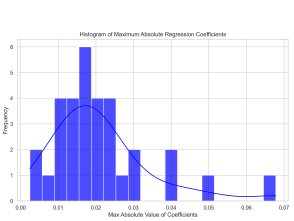
(a) (b) (c)

Figure 20: Regression Coefficients for Baseline Neurons 27, 28, and 29

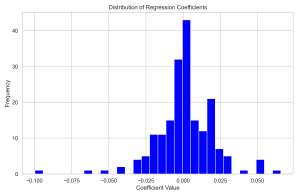


(a) (b)

Figure 21: Regression Coefficients for Baseline Neurons 30 and 31

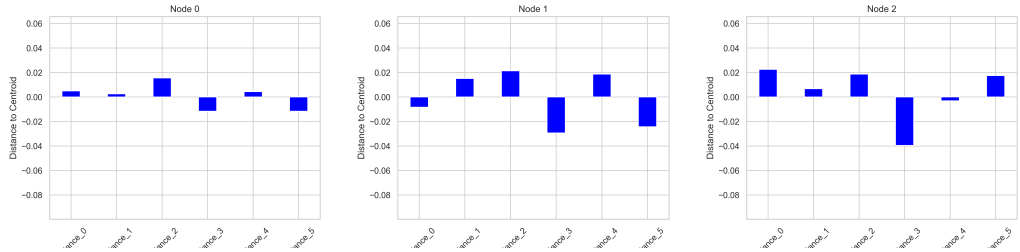


(a) Baseline Coefficient Histogram



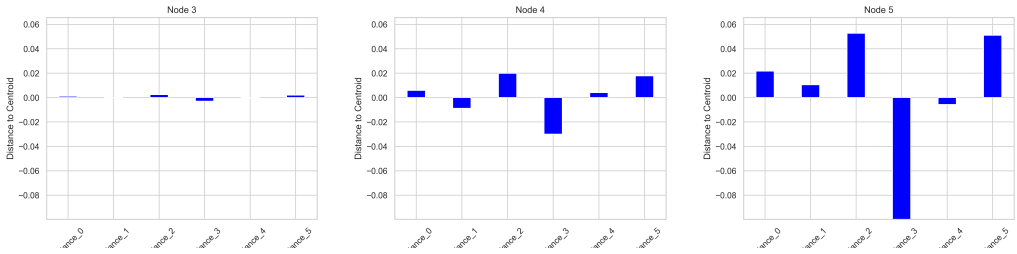
(b) Baseline Histogram of All Coefficients

Figure 22: Regression Coefficients for All Regression in Baseline



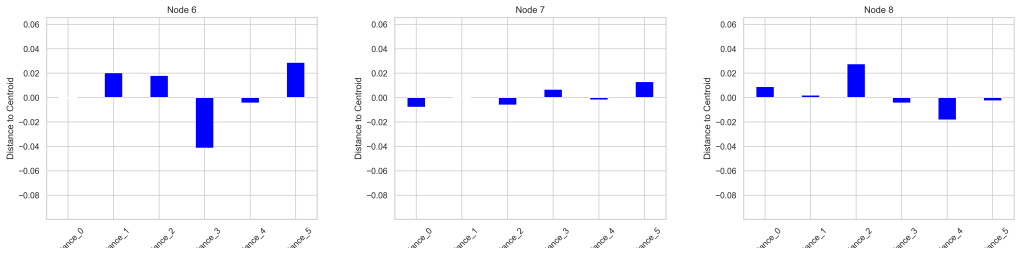
(a) (b) (c)

Figure 23: Regression Coefficients for Augmented Neurons 0, 1, and 2



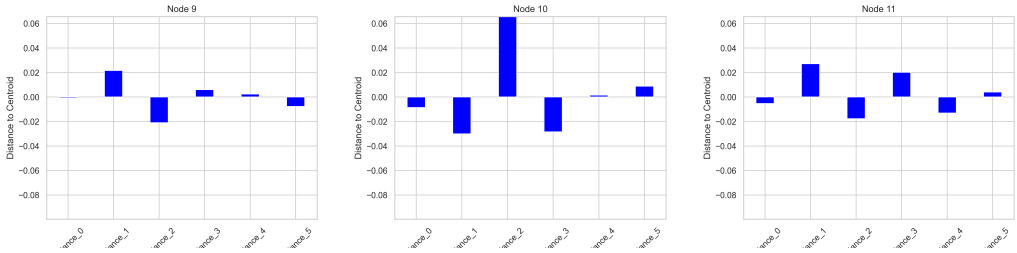
(a) (b) (c)

Figure 24: Regression Coefficients for Augmented Neurons 3, 4, and 5



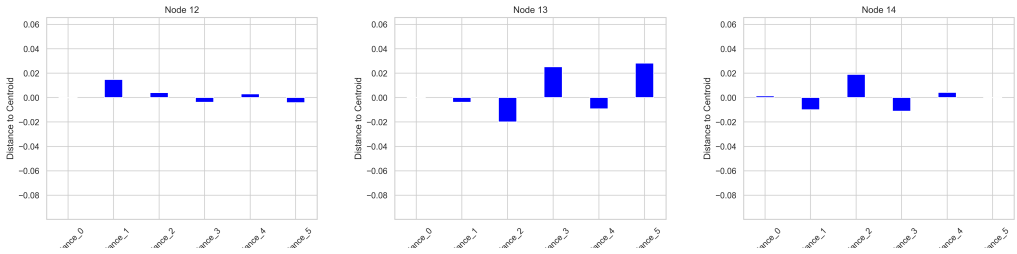
(a) (b) (c)

Figure 25: Regression Coefficients for Augmented Neurons 6, 7, and 8



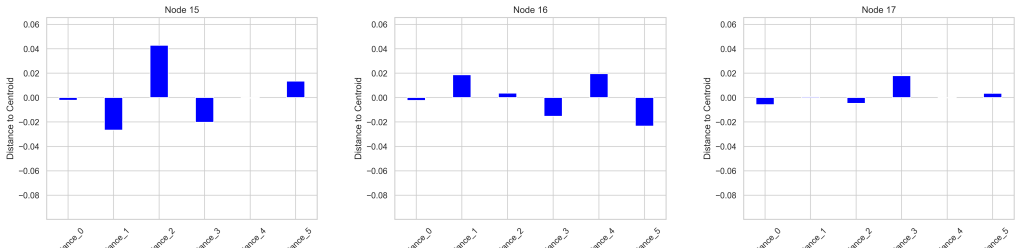
(a) (b) (c)

Figure 26: Regression Coefficients for Augmented Neurons 9, 10, and 11



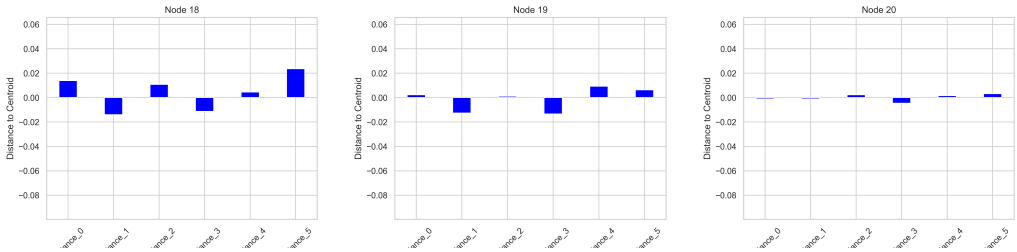
(a) (b) (c)

Figure 27: Regression Coefficients for Augmented Neurons 12, 13, and 14



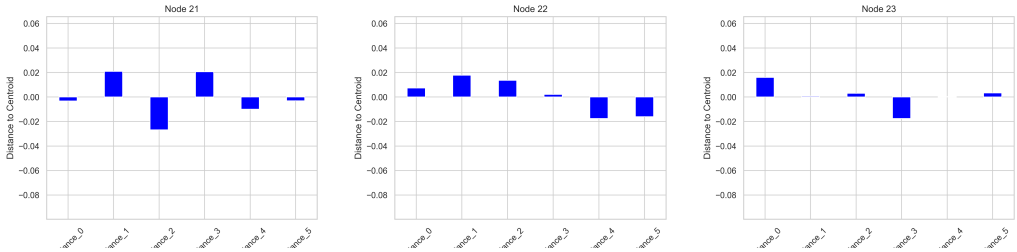
(a) (b) (c)

Figure 28: Regression Coefficients for Augmented Neurons 15, 16, and 17



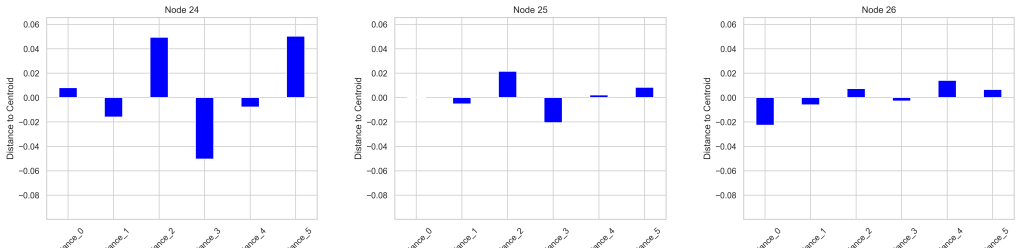
(a) (b) (c)

Figure 29: Regression Coefficients for Augmented Neurons 18, 19, and 20



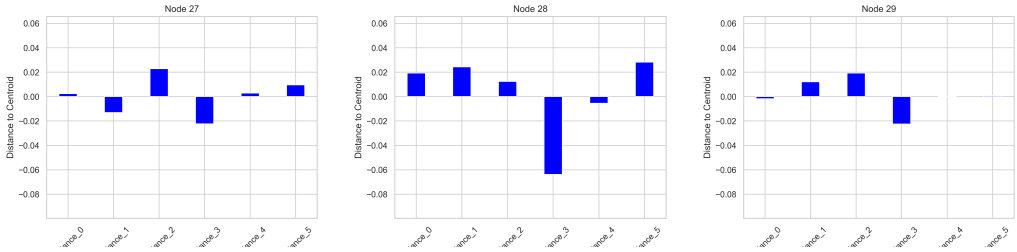
(a) (b) (c)

Figure 30: Regression Coefficients for Augmented Neurons 21, 22, and 23



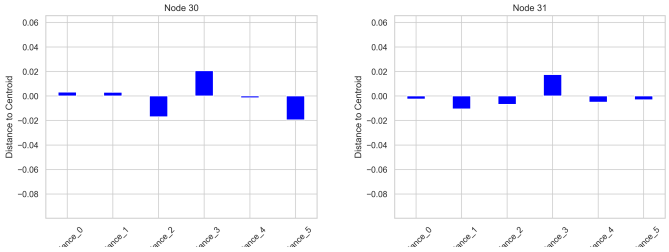
(a) (b) (c)

Figure 31: Regression Coefficients for Augmented Neurons 24, 25, and 26



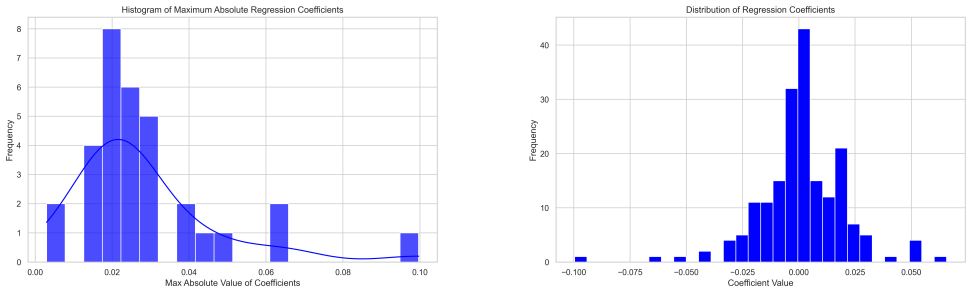
(a) (b) (c)

Figure 32: Regression Coefficients for Augmented Neurons 27, 28, and 29



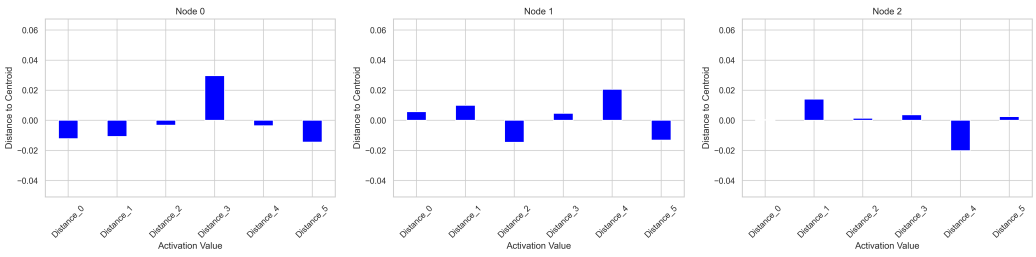
(a) (b)

Figure 33: Regression Coefficients for Augmented Neurons 30 and 31



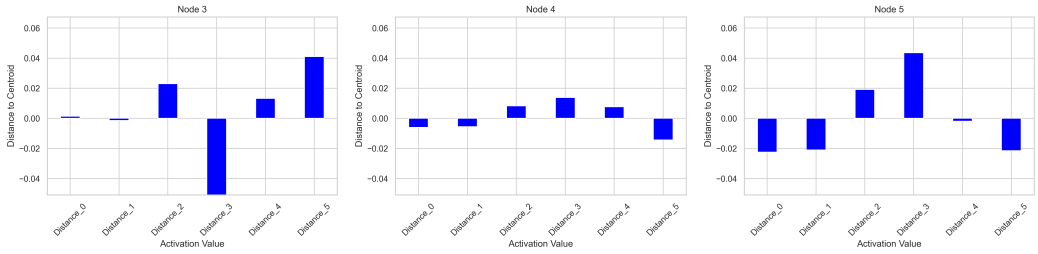
(a) Augmented Coefficient Histogram (b) Augmented Histogram of All Coefficients

Figure 34: Augmented Regression Coefficients for All Regression



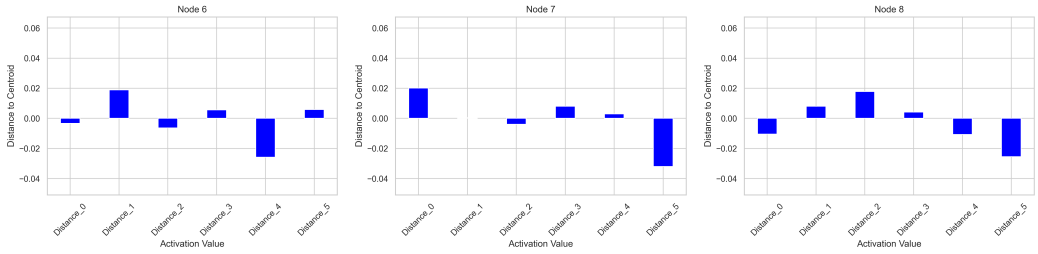
(a) (b) (c)

Figure 35: Full Input Regression Coefficients for Augmented Neurons 0, 1, and 2



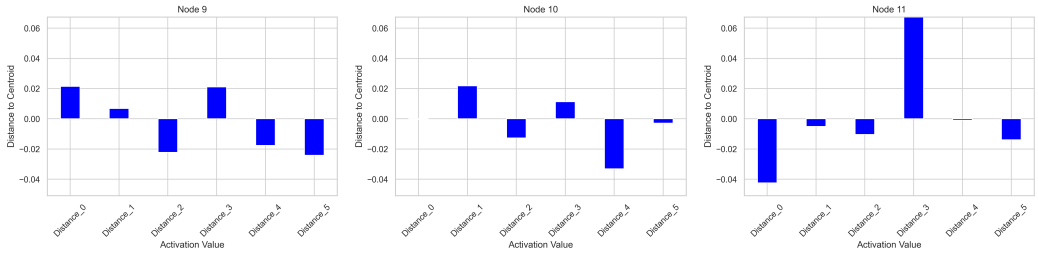
(a) (b) (c)

Figure 36: Full Input Regression Coefficients for Augmented Neurons 3, 4, and 5



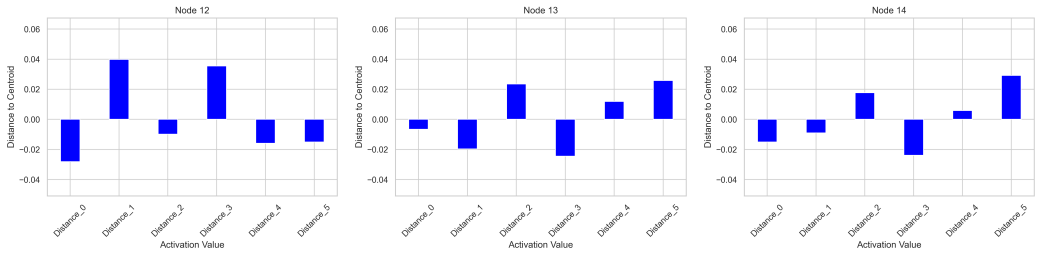
(a) (b) (c)

Figure 37: Full Input Regression Coefficients for Augmented Neurons 6, 7, and 8



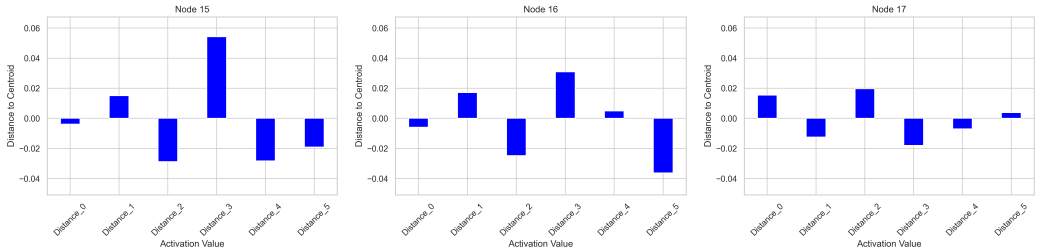
(a) (b) (c)

Figure 38: Full Input Regression Coefficients for Augmented Neurons 9, 10, and 11



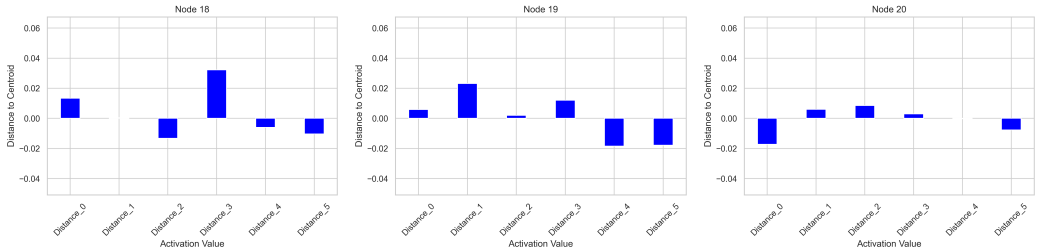
(a) (b) (c)

Figure 39: Full Input Regression Coefficients for Augmented Neurons 12, 13, and 14



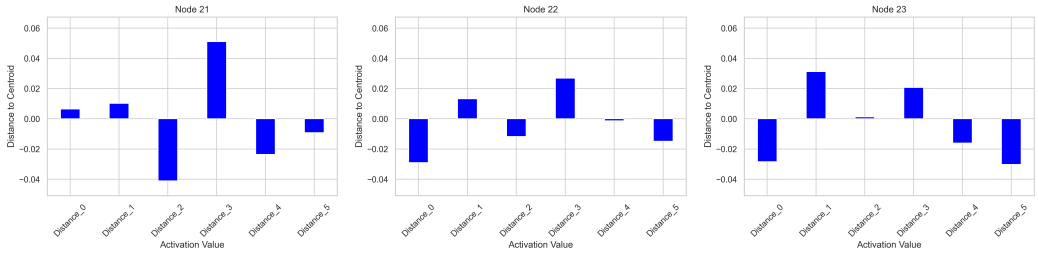
(a) (b) (c)

Figure 40: Full Input Regression Coefficients for Augmented Neurons 15, 16, and 17



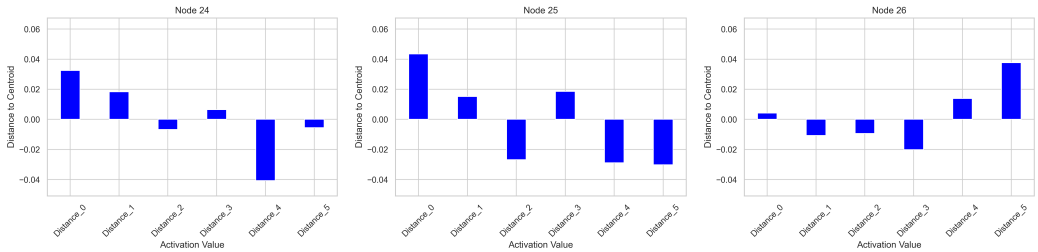
(a) (b) (c)

Figure 41: Full Input Regression Coefficients for Augmented Neurons 18, 19, and 20



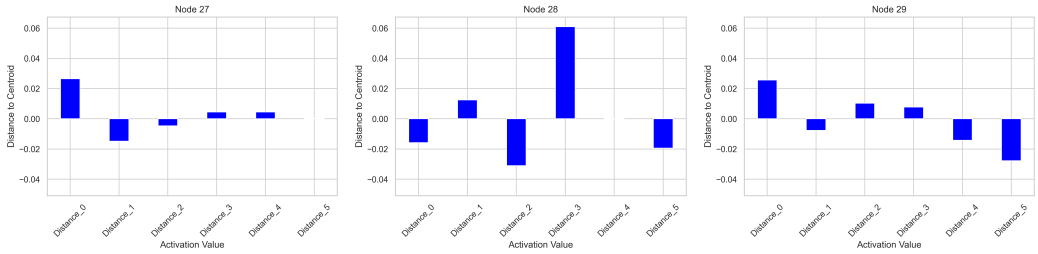
(a) (b) (c)

Figure 42: Full Input Regression Coefficients for Augmented Neurons 21, 22, and 23



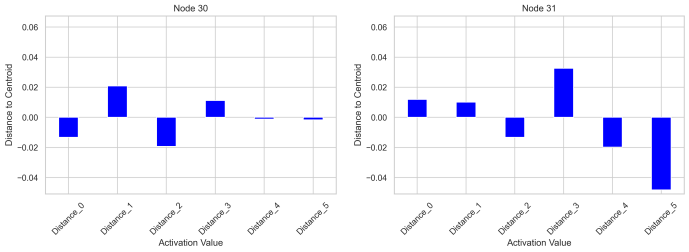
(a) (b) (c)

Figure 43: Full Input Regression Coefficients for Augmented Neurons 24, 25, and 26



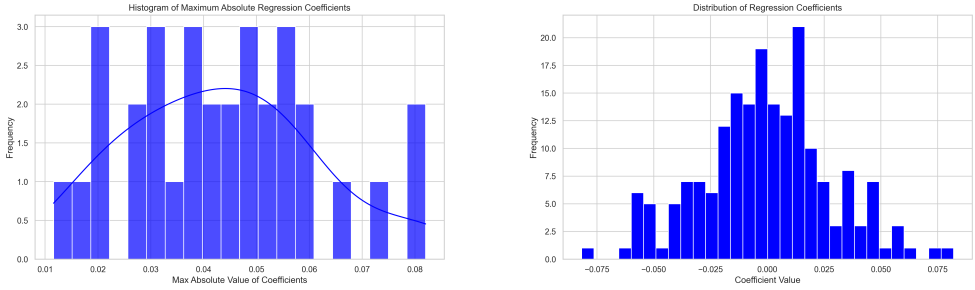
(a) (b) (c)

Figure 44: Full Input Regression Coefficients for Augmented Neurons 27, 28, and 29



(a) (b)

Figure 45: Full Input Regression Coefficients for Augmented Neurons 30 and 31



(a) Full Input Coefficient Histogram (b) Full Input Histogram of All Coefficients

Figure 46: Full Input Regression Coefficients for All Regression

Table 4: Results for Removing Feature Influence by Different Methods

	Method	Validation Loss	Loss % Change	Importance Reduction %
Feature 0				
	amplitude_reduction	1.5313	1.90	-100.00
	stochastic_fine_tuning	1.5212	1.23	-65.93
	inversion_fine_tuning	1.5035	0.05	-22.73
	simple_retraining	1.5177	0.99	-100.00
Feature 1				
	amplitude_reduction	1.5306	1.85	-100.00
	stochastic_fine_tuning	1.5208	1.20	-67.40
	inversion_fine_tuning	1.5049	0.14	-21.84
	simple_retraining	1.5220	1.28	-100.00
Feature 2				
	amplitude_reduction	1.5307	1.86	-100.00
	stochastic_fine_tuning	1.5213	1.23	-68.57
	inversion_fine_tuning	1.5043	0.10	-23.48
	simple_retraining	1.5175	0.98	-100.00
Feature 3				
	amplitude_reduction	1.5298	1.80	-100.00
	stochastic_fine_tuning	1.5221	1.28	-68.39
	inversion_fine_tuning	1.5036	0.05	-25.63
	simple_retraining	1.5246	1.45	-100.00
Feature 4				
	amplitude_reduction	1.5325	1.98	-100.00
	stochastic_fine_tuning	1.5223	1.30	-68.70
	inversion_fine_tuning	1.5042	0.09	-23.88
	simple_retraining	1.5250	1.48	-100.00
Feature 5				
	amplitude_reduction	1.5308	1.86	-100.00
	stochastic_fine_tuning	1.5216	1.25	-73.66
	inversion_fine_tuning	1.5042	0.09	-24.75
	simple_retraining	1.5172	0.96	-100.00
Feature 6				
	amplitude_reduction	1.5308	1.87	-100.00
	stochastic_fine_tuning	1.5223	1.30	-65.87
	inversion_fine_tuning	1.5037	0.06	-23.23
	simple_retraining	1.5199	1.14	-100.00
Feature 7				
	amplitude_reduction	1.5307	1.86	-100.00
	stochastic_fine_tuning	1.5239	1.41	-66.78
	inversion_fine_tuning	1.5057	0.19	-22.88
	simple_retraining	1.5216	1.25	-100.00
Feature 8				
	amplitude_reduction	1.5306	1.85	-100.00
	stochastic_fine_tuning	1.5223	1.30	-66.80
	inversion_fine_tuning	1.5041	0.09	-18.94
	simple_retraining	1.5209	1.21	-100.00
Feature 9				
	amplitude_reduction	1.5311	1.88	-100.00
	stochastic_fine_tuning	1.5222	1.29	-65.39
	inversion_fine_tuning	1.5036	0.05	-21.85
	simple_retraining	1.5202	1.16	-100.00

Method	**Average Runtime (seconds)**
Amplitude Reduction	3.5191
Stochastic Fine Tuning	3.1223
Inversion Fine Tuning	3.1770
Simple Retraining	16.9567

Table 5: Average runtime across all removed PCA components for each method