Systematic Determination of Optimal Activation Functions, Hidden Layer Count, & Number of Neurons

Toshan Doodnauth

Abstract

In order to determine the optimal configuration of hyperparameters that would result in the highest accuracy, this experiment employed a combination of 5 different activation functions, single, double, and triple hidden layers, each constructed of a varied number of neurons. The autoencoder model benchmarked was utilized to extract important features from protein sequences that would later be used for the decoding of a testing set of randomized protein sequences. The sequence length of the protein sequences was 43 characters. Each character described 1 of 20 amino acids, plus an additional character (dash line, '-') to account for gaps of indeterminate length in the sequence, giving a total of 21 possible characters. The sequences were then one-hot encoded by multiplying the sequence length (43) by the sequence depth (21) so that it was ensured that the model ran at its fastest capacity. The training, testing, and validation split was 70%, 20%, and 10%, respectively, and featured a total of 12,459 samples. Accuracy was measured by how exact the model performed in the reconstruction of the testing set of protein sequences. Overall, the model accuracy tended to increase proportionally to the number of neurons present in a hidden layer. However, experimental results dictated that as the number of hidden layers increased, the accuracy of the model decreased. Aside from ReLU, all the activation functions depicted a similar distribution across the hidden layer modifications and accuracy. Given that ReLU consistently had a significant underperformance, it could indicate that there may have been a large negative bias term that negatively affected the way ReLU processed gradients for its weights.

Introduction

Autoencoders are a type of artificial neural network that relies on an encoder and decoder structure to distill high dimensional information down to a simpler form where it can be reconstructed from a decoding function. A key component of ensuring the success of autoencoder models lies in their hidden dimension; the hidden dimension consists of a number of hidden layers containing a certain number of nodes (both amounts determined by the user), which are useful for extracting key features from the input data. The performance of an autoencoder can be influenced by the amount of nodes selected by the user, however there is a clear point where computational cost would dominate performance, and the model would take too long to complete. Targeting the point

where computational cost and model performance are balanced is imperative to ensuring the success of a model.Wanas et al. discuss an empirical approach to determining the optimal point of cost versus performance (919). They do this by relating the entropy of the hidden layer to numerical weights of the input and output data via the following equation:

$$H(x) = -\Sigma\,[p(x_k) * log(p(x_k))] \quad (1)$$

Where $p(x_k)$ is equal to the probability that pattern x_k appears in the layer under investigation

The formulaic manipulation is out of the scope of this paper, however the conclusion reached was that entropy in the hidden neurons must be maximized in order to achieve minimum entropy across the neural network. Intuitively, this will ensure that the classification ability of the hidden layer is used to its fullest extent.

In order to achieve the most optimal point of performance and cost, Wanas et al. ran different simulations of a three layer autoencoder (input, hidden, and output) with recording the performance of the model, and the entropy per number of nodes (920). After graphing and analyzing the results of the simulations, they determined that the point of convergence between efficiency and accuracy was the value of log(T), where T denotes the amount of total samples accounted for in the dataset (Wanas et al. 920-921). However, Wanas et al. also discussed that a greater number of nodes in the hidden layer seemed to proportionally scale with the accuracy outputted by a model, but at the cost of computational efficiency (921). Fundamentally, this coincides with the notion that more neurons will allow for a greater extraction of features from an input, resulting in the model attaining a greater accuracy. And although not addressed in Wanas et al.'s findings, too many neurons could result in the overfitting of a dataset. So once again, it is crucial to find a balance of optimal hyperparameters.

One detail that Wanas et al. neglected to consider was the application of their discovery in regard to stacked autoencoders. Stacked autoencoders are described similarly to the aforementioned three layer autoencoder, but differ in the number of hidden layers included in the model (Sun et al. 3). In addition to the number of hidden layers being included in this experiment, the model accuracy will also be modeled as a function of the other hyperparameters, included but not limited to epochs, batch size, activation functions, latent dimension, and learning rate.

Model Framework

The model used in this study consisted of three main components, an encoder-decoder class, a data module loader, and a segment to output the results of the trials. The encoder-decoder class followed the typical framework for artificial neural networks. Each contained a forward function, a weight initialization function, and a method to apply

linear layer(s).Weight initialization in the encoder-decoder structure followed the kaiming method, with weights sampled from a standard normal distribution. And in the output layer of the decoder, the last activation layer used, regardless of hyperparameters, was softmax. A model wrapper class was used to communicate the hyperparameters required for the encoder-decoder structure, with the tested inputs (hidden layer neurons, hidden layer dimension, and activation function) being entered from the command line. The hyperparameters kept constant were the number of epochs (750), the size of the latent dimension (40), the learning rate (2e-4), the batch size (100), and the weight initialization (kaiming, normal distribution). Continuing through the model, the data loader class was responsible for one-hot encoding the protein sequences by equating each possible character to a unique binary vector. Afterwards, the DataLoader module in pytorch was applied to the training, testing, and validation sets so that they could become iterable when it was time to run the model. Finally, once the model completed iterating through all of the datasets, it generated an output of the number of epochs, number of hidden layers, size of the latent dimension, the amount of neurons present in each hidden layer, the activation function used, and the test accuracy, as seen in Table 1.

Methods

To determine the optimal parameters for the hyperparameters of this model, the input protein sequences were first preprocessed from the FASTA file downloaded from the European Bioinformatics Institute. The preprocessing included a shortening of the sequence length, the exclusion of sequences that were 80% similar or higher (establishing generality in the model), a shuffling of the sequences to ensure randomization, and the splitting of the total sequences into the training (70%), testing (20%), and validation groups (10%). Upon conclusion of these preprocessing steps, the data was entered into the data module class where the training sequences were once again shuffled. This experiment employed high performance computing as a means for operating the model, and did so via the use of job scripts. Each job script contained instructions for executing a combination of hidden layer(s) and activation functions (i.e one job script for 40 neurons and ReLU, another for 40 neurons and sigmoid, etc.). A for loop was then used in the command line to run this job script file four times. An example of the raw output can be seen in Table 1. The experiment initially started out at 14 neurons because of Wanas et al.'s approximation that the most optimal point was log(# of total samples). Following this, the samples were increased by ~20 neurons per activation function. And,the average test accuracy was then computed for each combination as demonstrated in Table 2. This process was repeated multiple times until the accuracy of the final combination of hidden layer and activation function was determined.

Graphs and Tables

EPOCHS	NUM_H_LAYERS	LATENT_DIM	ENCODER_H_DIMS	ACTIVATION	TEST_ACC
750	1	40	14	None	0.52029751
750	1	40	14	None	0.51525812
750	1	40	14	None	0.51547276
750	1	40	14	None	0.51521146
750	1	40	14	relu	0.50483407
750	1	40	14	relu	0.50653253
750	1	40	14	relu	0.50422748
750	1	40	14	relu	0.51232782
750	1	40	14	leakyrelu	0.51836574
750	1	40	14	leakyrelu	0.52342379
750	1	40	14	leakyrelu	0.52816454
750	1	40	14	leakyrelu	0.51994288
750	1	40	14	sigmoid	0.51581806
750	1	40	14	sigmoid	0.51716189
750	1	40	14	sigmoid	0.51072268
750	1	40	14	sigmoid	0.52396506
750	1	40	14	tanh	0.52287319
750	1	40	14	tanh	0.51374631
750	1	40	14	tanh	0.50779237
750	1	40	14	tanh	0.50686849

Table 1. Sample Output of Autoencoder Model. Table 1 displays a portion of the total output by the model. Each combination of neurons in the hidden layer and the activation function was run for a total of 4 times, with an average taken at the end as seen below in Table 2.

EPOCHS	NUM_H_LAYERS	LATENT_DIM	ENCODER_H_DIMS	ACTIVATION	TEST_ACC
750	1	40	14	0.51655995	None
750	1	40	14	0.50698	relu
750	1	40	14	0.522474	leakyrelu
750	1	40	14	0.516917	sigmoid
750	1	40	14	0.51282	tanh
750	1	40	40	0.57039	None
750	1	40	40	0.543864	relu
750	1	40	40	0.567878	leakyrelu
750	1	40	40	0.572315	sigmoid
750	1	40	40	0.565629	tanh

Table 2. Final Processing of Output Data. Once every test accuracy was averaged from Table 1, the following data table was produced. Numbers and labels from this table were used to manufacture Figures 1-3 below.

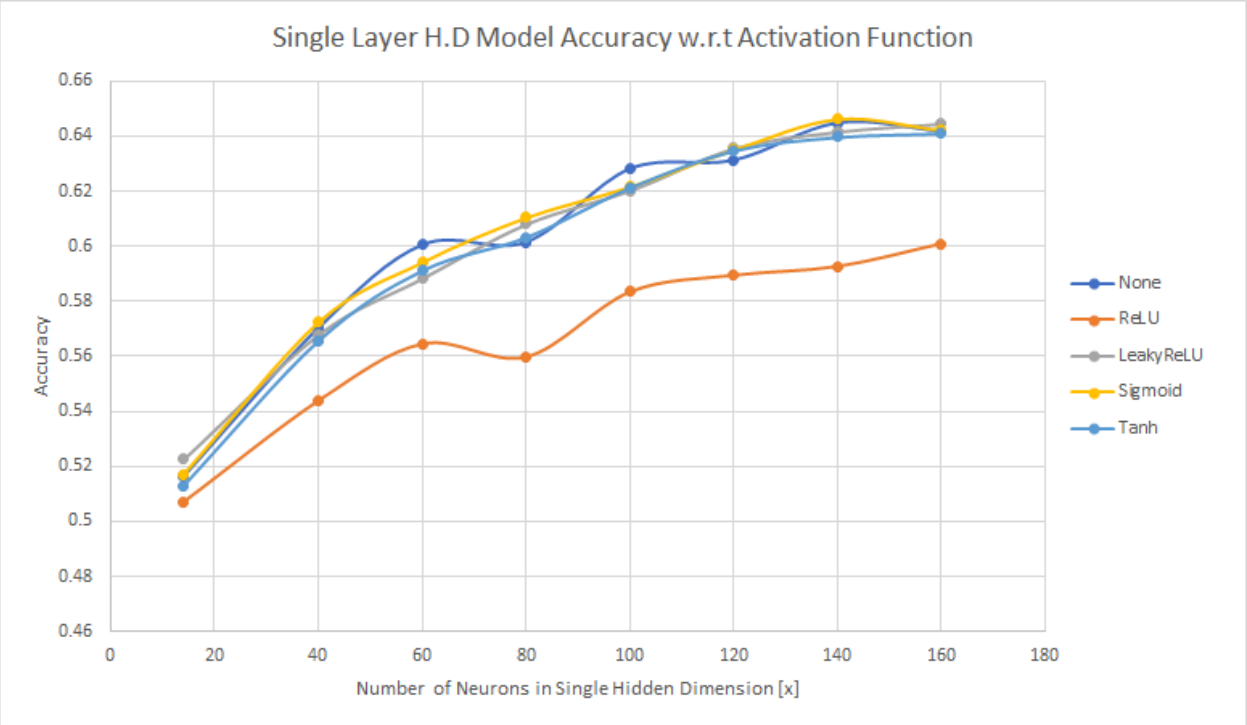

Figure 1. Relationship Between Single Hidden Layer & Accuracy. This visualization of the varying number of neurons in a single hidden layer versus the accuracy detailed the seemingly proportional relationship between neuron count and accuracy that was proposed by Wanas et al. While this proposal held true for some of the activation functions, there were noticeable dips preceding higher accuracies at the 80 and 120 position, as well as very slight differences (on the order of $1.0*10^{-3}$) for no activation function and the sigmoid activation function at the last two neuron counts. With the difference for both being so slight, it was possible that this could be attributed to the inherent variance of the model, especially seeing as the other activation functions increased on the same order of magnitude.

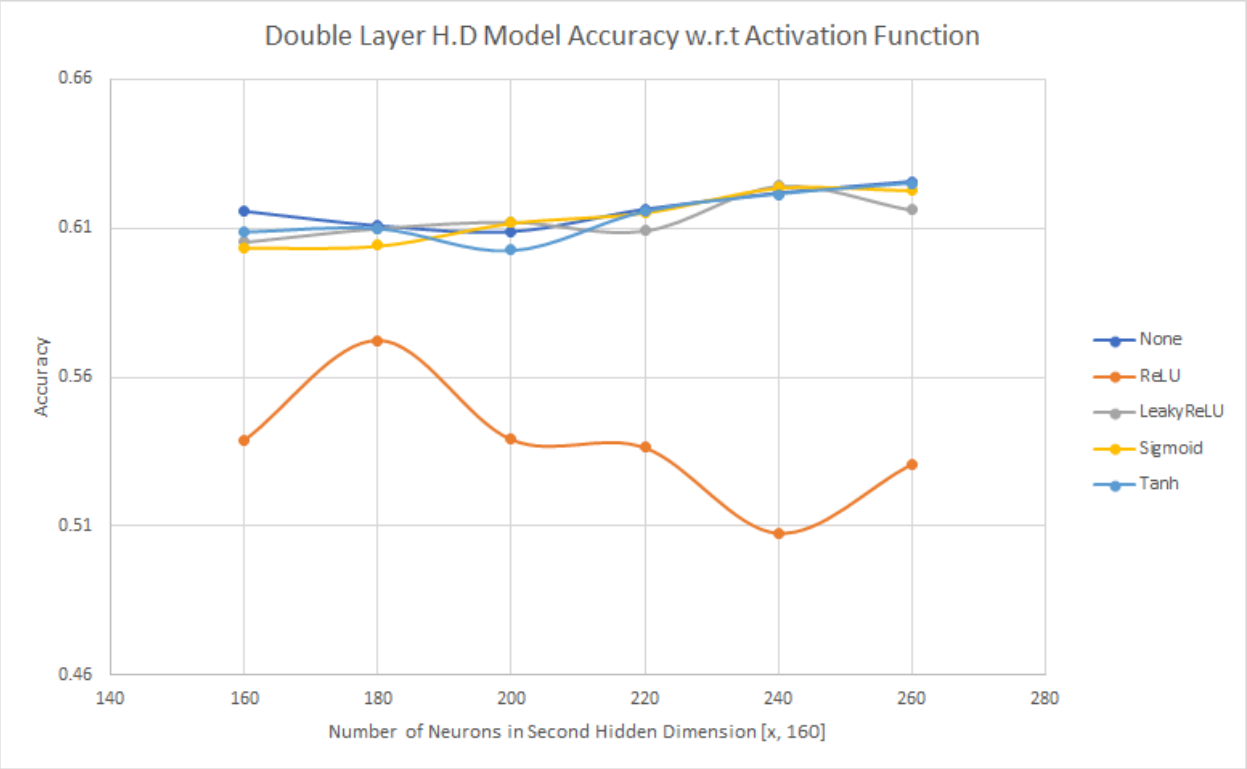

Figure 2. Relationship Between Double Hidden Layer & Accuracy. Following the research documented by Pierre Baldi and the University of California, the neuron count from the single hidden layer with the greatest accuracy overall accuracy (i.e 160) was used as the barrier between the latent dimension and the second hidden layer. Baldi specified that once the most optimal neuron count was found for a layer, changing that threshold would only result in a lower accuracy once another layer was added (41). Although not discussed in this paper, Baldi supported the claim by using linear algebra, projections onto the subspace, and eigenvectors (41).

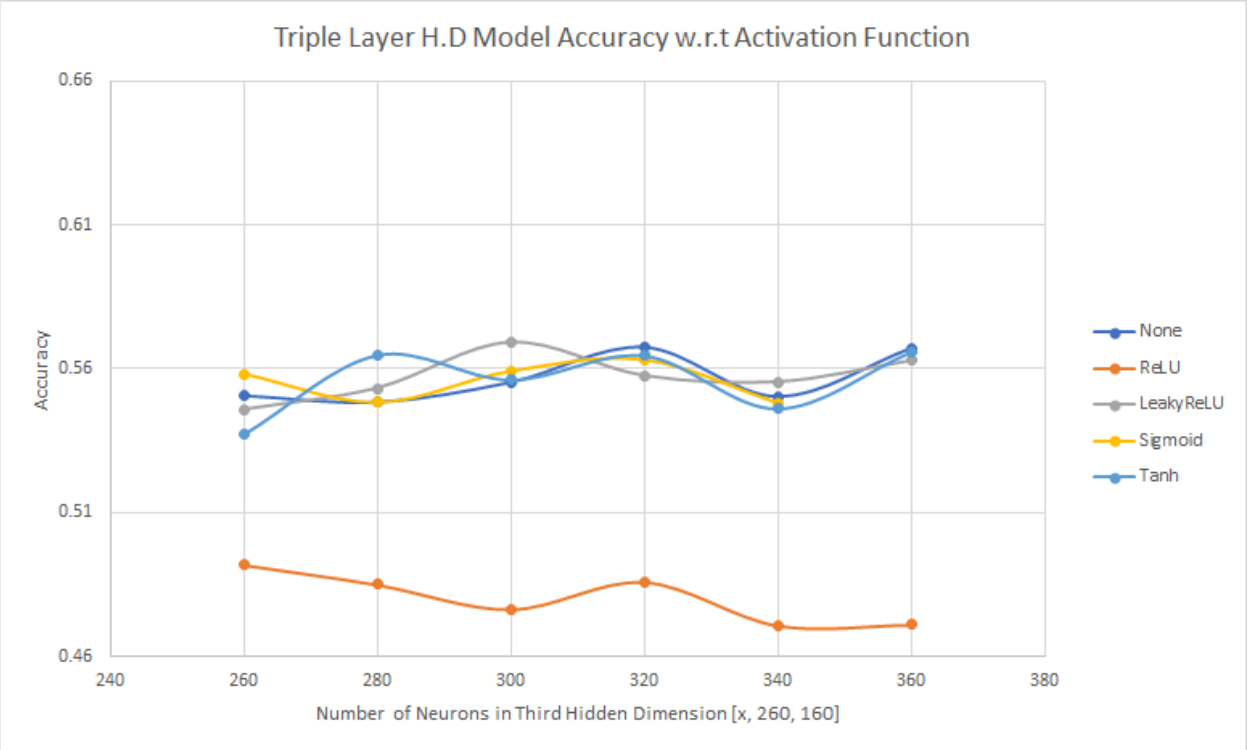

Figure 3. Relationship Between Triple Hidden Layer & Accuracy. This figure showcased the inferiority of a triple hidden layer model, as most of the accuracies were clustered around a single accuracy of 0.56, and saw very little to no gains.

Discussion

Analysis of the data present in the Graphs and Tables section revealed the relationship was that the accuracy was inversely proportional to the amount of hidden layers present in the model. By this, it was meant that the greater the number of hidden layers, the worse the model performed in terms of accuracy. As seen in Figure 3, the accuracies of all the activation functions barring ReLU were congregated around the 0.56 mark. In contrast, Figure 2 saw accuracies with a slight increasing trend stemming from the 0.61 mark. And Figure 1 depicted a large increase in accuracies up until the 140-160 range, where the trendlines rise to a fairly uniform pattern ultimately resulting in accuracies as high as 0.64. Additionally, a notable feature common to all the three figures was the underperformance of ReLU. Regardless of the dimensionality of the hidden layers, ReLU was consistently outperformed by a factor of at least 6.25% (calculated from the max point of ReLU per figure, and the averages stated earlier).

Beginning with the discrepancy displayed by the inversely proportional relationship between the number of hidden layers and the resulting accuracy, there are two likely scenarios that may have caused such an outcome. First, the learning rate of $2 \cdot 10^{-4}$

could have been set too high, and therefore influenced the model such that it over adjusted the weights that were initialized. The over approximation of the weights in the model could have consequences related to the loss gradient descent (Kuan and Hornik 1). Gradient Descent is an algorithm that attempts to minimize the cost function by cycling through many iterations until the most optimal weights are determined (Kuan and Hornik 1). The learning rate has a direct role in this algorithm as it determines how quickly the algorithm approaches the calculated weight. A smaller learning rate would ensure that the Gradient Descent algorithm is more likely to find the most optimal weight, however it would have a drastic increase in computational efficiency as it would increase the number of iterations the algorithm goes through. This ties into the relationship between accuracy and depth of the hidden layers by the idea that as the number of hidden layers increases, the more complex the optimization becomes. And therefore by having a relatively high learning rate, the determination of the most optimal weights proves to be more difficult which would indicate a lower accuracy (Kuan and Hornik 3).

The other possibility that could explain the drop in accuracy, with respect to the depth of the hidden layers, is a phenomenon known as the vanishing gradient problem. The vanishing gradient problem directly affects how well the model learns the input data. In this artificial neural network, the weights follow kaiming initialization and are back propagated through the model after a forward pass through one run of the model. In this backward propagation the gradient descent is applied, and the weights are updated by taking the difference of the current weight and the product of the learning rate and gradient, as seen in the following equation:

$$\omega_1 = \omega_1 - (l * g) \quad (2)$$

where:

ω_1 = Weight

l = Learning rate

g = Gradient

Furthermore, the gradient itself is calculated using the chain rule with respect to the weights present in the neural network (Amari 1). Therefore, it would correlate that as the number of weights increase in the neural network, the smaller the gradient would become since every number multiplied is less than 1 (i.e the product of two numbers less than 1 will be even smaller). Since the number of weights increases with the number of hidden layers present, this would result in a gradient that is extremely small. Moreover, the weights are updated following the process in Equation 2, which multiplies the value of the gradient by the learning rate (i.e $2 \cdot 10^{-4}$). This number produced is even smaller than that of the gradient, which when subtracted from the current value of the weight may bring a change to weight that is so small, it may as well be negligible. This is the vanishing gradient problem, and it would be a possible explanation for how the accuracy would decrease despite the increase in hidden layers.

In regards to the ReLU activation function performing significantly lower than the other functions (or lack thereof), ReLU can suffer from the same complication observed in the

vanishing gradient problem. ReLU as a function is mapped from [0, x], where x is defined by having a slope of 1, meaning that for any number less than or equal to 0, that number is set = 0. Known as the dying ReLU problem, this issue pertains to the neurons trained with ReLU only outputting a value of 0, and thus contributing nothing to the learning of the model as the weight is perpetually stuck at 0. Further proof of the dying ReLU problem can be seen in its counterpart, LeakyReLU. LeakyReLU aims to combat the dying ReLU problem by introducing the bounds as [0.01x, x]. The importance of this change is that it removes the issue of all negative values being set to 0, and thereby giving the model a chance to recover during gradient descent (Lu et al. 2). In this scenario, it can be seen that due to this lower bound being set as 0.01x, the model indeed did recover, and would explain the difference in accuracy of the two similar activation functions. There are two main ways that the dying ReLU problem occurs, one is when the learning rate is set too high and the other is in the inclusion of a large negative bias term (Lu et al. 2). The learning rate being set too high would follow the points mentioned earlier in the discussion about the gradient descent algorithm, while the large negative bias term could result in another parameter having to be investigated. A change in these two parameters could solve this complication and feasibly increase the accuracy of ReLU.

Future Considerations

Although the focus of this project was on the hidden dimension and its performance with various activation functions, this experiment did not alter all of the possible hyperparameters due to the time constraints associated with the computation of those values. In the future, this optimization could be applied to not just the different hyperparameters (learning rate, batch size, epochs, latent dimension size, etc.), but also the different types of autoencoders. Variational autoencoders have an extremely similar framework to regular autoencoders, but only differ in the inclusion of a sampling step that makes the model generative. So, by testing the hyperparameters in the same fashion that this experiment did, it could have implications for generating a better output as a result of optimization.

Conclusion

The goal of this paper was to experimentally determine the optimal configuration of hyperparameters that would result in the highest accuracy. This experiment employed a combination of 5 different activation functions, single, double, and triple hidden layers, each constructed of a varied number of neurons. The data quantitatively

concluded that the highest accuracy was achieved in a single hidden layer with 160 neurons, with either no activation function, or an activation function of Sigmoid, Tanh, or LeakyReLU. The increase in hidden layers and number of neurons displayed a lower accuracy than what was initially hypothesized, but could be explained by factors affecting the Gradient Descent algorithm, whether that be in the learning rate or the actual calculation of the gradient. In addition, the underperformance of the ReLU activation function was perceived to be caused by the dying ReLU problem, and possible fixes for that could be found in altering the learning rate or the associated bias term. For all the hyperparameters that were assessed in this study, the results indicated that model accuracy tended to increase proportionally to the number of neurons present in a hidden layer. However, experimental results dictated that an increase to the number of hidden layers resulted in the overall decrease of the accuracy.

References

Amari, S, et al. “Backpropagation and Stochastic Gradient Descent Method.” *Neurocomputing*, Elsevier, 14 Aug. 2003, www.sciencedirect.com/science/article/abs/pii/092523129390006O.

Baldi, Pierre. “Autoencoders, Unsupervised Learning, and Deep Architectures.” *JMLR*, University of California, Irvine, 2012, proceedings.mlr.press/v27/baldi12a/baldi12a.pdf.

Kami, K. “Pfam PF00018 SH3 Domain .” *InterPro*, EMBL-EBI, www.ebi.ac.uk/interpro/entry/pfam/PF00018/. Accessed 10 Dec. 2023.

Kuan, Chung-Ming, and Kurt Hornik. “Convergence of Learning Algorithms with Constant Learning Rates | IEEE ...” *IEEE Transactions on Neural Networks*, IEEE, Sept. 1991, ieeexplore.ieee.org/abstract/document/134285.

Lu, Lu, et al. “Dying Relu and Initialization: Theory and Numerical Examples.” *arXiv.Org*, Massachusetts Institute of Technology, 21 Oct. 2020, arxiv.org/abs/1903.06733.

N. Wanas, G. Auda, M. S. Kamel and F. Karray, "On the optimal number of hidden nodes in a neural network," Conference Proceedings. IEEE Canadian Conference on Electrical and Computer Engineering (Cat. No.98TH8341), Waterloo, ON, Canada, 1998, pp. 918-921 vol.2, doi: 10.1109/CCECE.1998.685648.

Sun, Yanan, et al. "An experimental study on hyper-parameter optimization for stacked auto-encoders." 2018 IEEE congress on evolutionary computation (CEC). IEEE, 2018.