Decoding the relation between shape and learning of MLPs

1. Abstract

This report examines the effect of changing the widths of layers in a Multi Layer Perceptrons (MLPs) on its learning on structured datasets. While deeper layers can give the model an opportunity to learn higher level features, this report will try to explore the possibility of changing the size of the hidden layers in an MLP to mimic certain distributions. That combined with datasets of different shapes and sizes, this report will try to postulate thumb rules or trends that one can expect to see when training their models on similar structured datasets.

2. Introduction

Deep Neural Networks (MLPs) have been an important milestone in the journey of AI as they have been able to achieve high performance on a large variety of datasets and problems. However, when it comes to explaining the model, they are often termed as black-box and are nearly impossible to interpret in most cases. This, in part, has bloomed into a sub-domain of research in Artificial Intelligence and Machine Learning, aimed at constructing methods to explain existing models as well as construct interpretable models from scratch.

In high-risk domains, such as law enforcement or self-driving cars, managing the uncertainty in MLPs is a crucial requirement prior to deployment. In domains where fast-deployment is crucial, such as ML applications on smartphones, MLPs face a different challenge: since part of the advantage of MLPs is reduced need for feature selection (earlier layers tend to take over that task), overall that can increase the computational cost of forward or backward propagation. Because of this inherent property of MLPs, the training phase of model development is interlaced with managing lot of uncertainty through heuristics and trial error, where the flexibility of the model has to be balanced with the computational cost of the same. Developments in higher end dedicated and integrated GPUs on mobile devices has allowed mobile-app developers to make use of ML on a larger scale.

However, tasks that deal with heavy data such as image and video processing still tend to be too expensive and active work is being put into converting pre-trained state-of-the-art models into smaller and smaller models for mobile deployment (for example, tiny-YOLO [3]). This has been achieved in literature with methods such as teacher-student knowledge distillation [2] [10], whereby a pre-trained computationally expensive model can be used to train a smaller model. This can be thought of as discarding redundant parameters and only keeping the ones that are absolutely crucial (with some loss in generalization or accuracy).

In this paper, we aim to explore the heuristic of minimizing parameters for achieving high performance as well as better understand how learning takes place in MLPs, and what factors can affect it. In particular, we look at MLP models of different shapes/architectures and how they affect the training phase on different types of datasets. With this, we gain some intuition about which shape of model architectures perform well for a particular type of dataset and will help bring some clarity to the training phase.

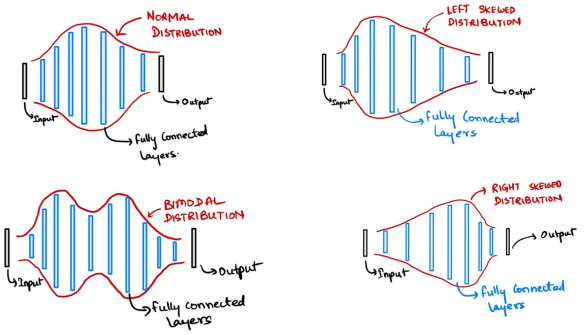
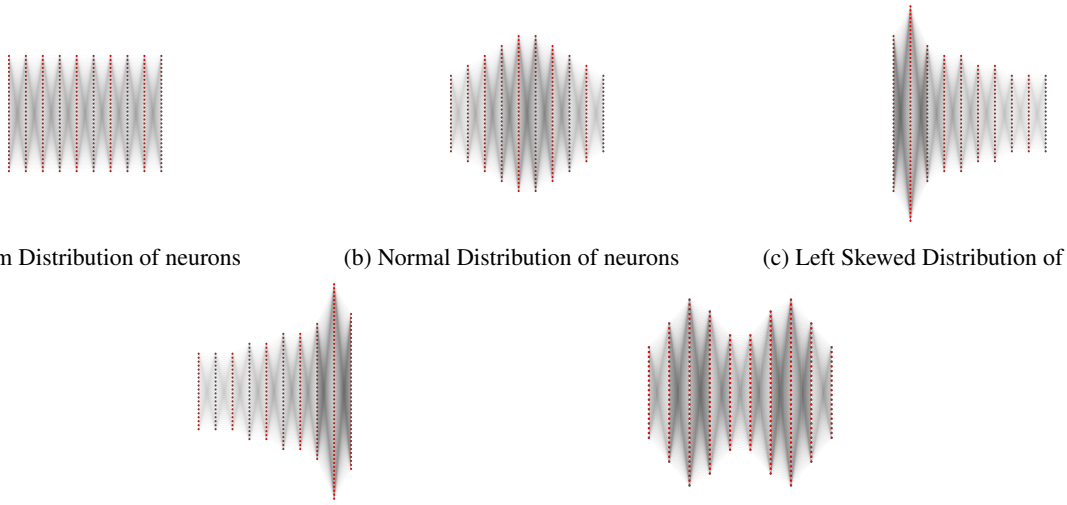


Figure 1: Example MLP Architectures of different shapes based on different distributions

3. Related Work

A lot of work has been done in trying to answer the question - How many number of layers and how many number of neurons should my model have? In his article, Dr. D Stathakis [7] proposes using a novel fitness function to find the most accurate and best solution while Stephen Trenn [9], in his paper, tries to approximate smooth multivariate functions along with the model and then use derived formulae to calculate the minimum required number of hidden units/neurons for each approximation. Lu, et al. (2017) [5] have explored the theoretical estimations of how the width of neural networks affect their abilities as universal approximators. There is also work on how the depth of neural networks affect their abilities to approximate, from the limitations of shallow networks such as [1] proving that certain 3-layer neural networks cannot be approximated by any 2-layer neural networks (within limits on the width) to experimental results, to papers like [8] that show how powerful additional depths can be at approximating very large shallow depth neural networks. We aim to add to this literature by exploring how arrangement of hidden units, ceteris paribus, can affect learning of the model.



(a) Uniform Distribution of neurons (b) Normal Distribution of neurons (c) Left Skewed Distribution of neurons

(d) Right Skewed Distribution of neurons (e) Bimodal Distribution of neurons

Figure 2: Hidden layer configurations for different model architectures based on different distributions

4. Approach

We will analyse the effect of changing the shape of the MLP Architecture on learning and prediction accuracy. By shape of the Neural Network, we mean the distribution of neurons across a fixed number layers and comparable parameter size (number of neurons). Some neural networks can have larger hidden layers at the beginning and some can have them towards the middle or the end (refer to Figure 1 and 2). We will train these model architectures on different datasets and analyze the model learning over time. For a fixed number of layers (L), fixed number of Neurons (N), we define architectures that follow the below mentioned architecture shape:

4.1. Problem Statement

Structured datasets can be categorized into fixed types based on the number of features, number of output classes and the size of the data. Certain model shapes might learn faster on one particular type of data than another and our goal is to understand this correlation.

We will analyse the change in loss and accuracy on one particular dataset (application) using a model having a fixed number of layers and parameters and we only change the distribution of neurons in each of these layers. We also plan on doing this exercise for multiple datasets (applications).

We will define the number of neurons per layer for an architecture based on skewed bell curve equations, while making sure the total sum of neurons are comparable to each other. For all models, we will keep all the hyperparameters except the learning rate and weight scale constant across our models including the initialization seed (these will be allowed to vary to optimize the models, as necessary).

4.2. Datasets

We ran our experiments on classification datasets available in the open source domain. We chose datasets from openml.org as it has numerous open source datasets and invites users to train models on them and compete for accuracy. We chose datasets from this website that provide us variety in terms of number of input features, number of classes and number of examples. We also ensured that some of the datasets are blind and some are know. So we can evaluate the learning phase objectively without any bias. We finalized 2 datasets having the following metrics:

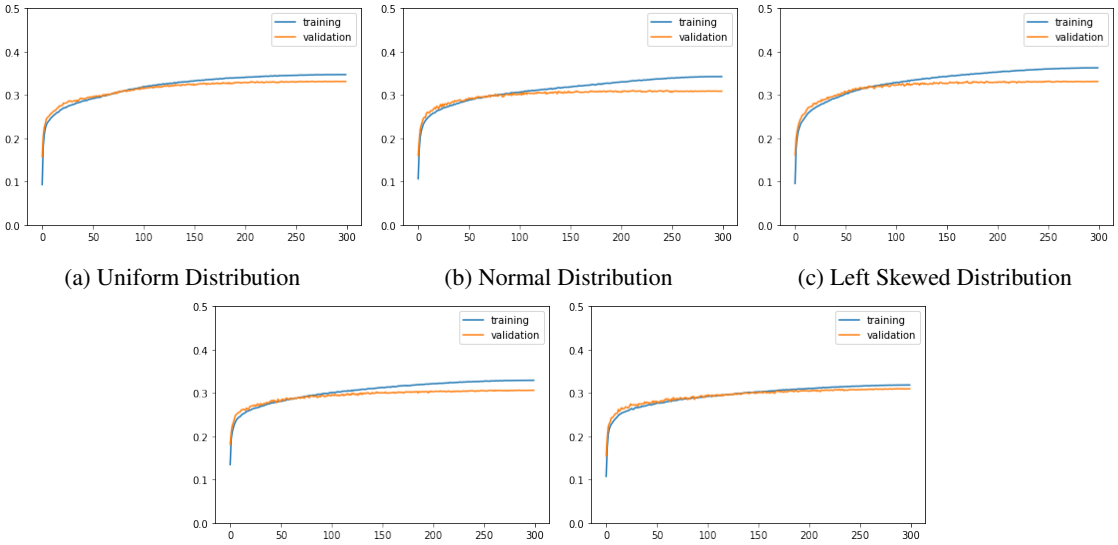
Name	Number of Features	Number of Classes	Number of Rows
cnae-9	856	9	1,080
Helena	27	100	65,196

This choice of datasets provide us with good coverage of the types of datasets that ML engineers see in general practice. We have both lean and broad datasets in terms of their (number of features/number of classes) ratio and also, the number of examples in the datasets vary drastically giving us an opportunity to examine over-fitting carefully

5. Technical Approach

5.1. Model Architectures

In order to create any MLP, we first had to decide N and P. We choose N = 10 for all datasets as 10 layers would give



(a) Uniform Distribution (b) Normal Distribution (c) Left Skewed Distribution

(d) Right Skewed Distribution (e) Bimodal Distribution

Figure 3: Training and Validation accuracy Curves for training models with different distributions on Helena Dataset

us enough scope to vary number of neurons per layer in order to mimic a distribution while still not being very deep. Then, for each dataset, we trained a 10 layer MLP with variable N and chose a value where the networks really starts to struggle in order to reach a baseline accuracy. This baseline accuracy was decided based on the average validation accuracy that a 10 layer, dense MLP with N=3000 (normally distributed neurons) can achieve on the same dataset.

Name of dataset	Number of layers (L)	Number of Neurons (N)	Baseline Accuracy
cnae-9	10	300	90%
Helena	10	300	30%

Once, we have finalized N and P for each dataset, we created MLP architectures based on the following distributions (Figure 2):

- Uniform distribution
- Normal distribution
- Left Skewed distribution
- Right Skewed distribution
- Bimodal distribution

Dropouts and Batch Normalization were not used in training. Dropouts were not included in order to keep the entire process deterministic and eliminate any stochastic behaviour. We employed ReLU activation functions for all but the output layer. As ReLU quashes the outputs between 0 and 1, it will be easier for us to later compare the neuron activations across different models, which we will see in the Results section.

Categorical Cross Entropy was used as our loss function:

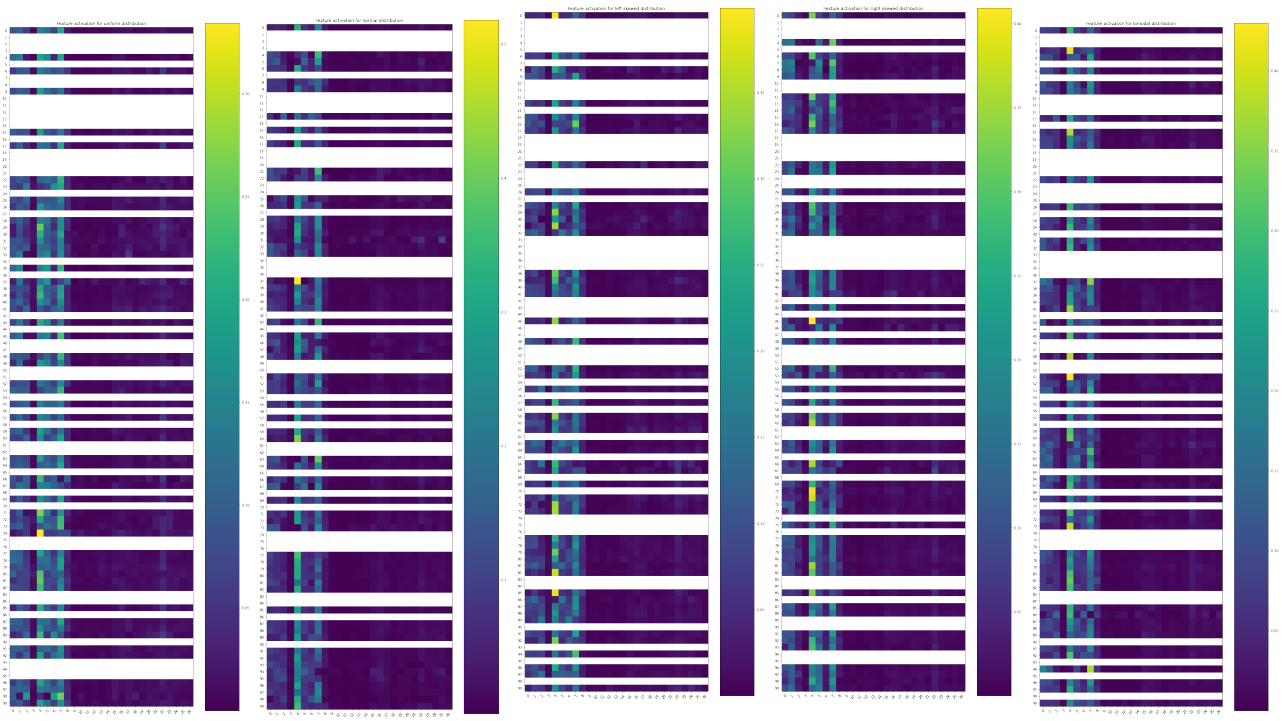
$$Loss = -\sum_{n}^{\text{batch}} \sum_{c}^{M} y_{n,c} \cdot \log(s_{n,c})$$

where $y_{n,c} = 1$ if the true class of the nth observation is c, $s_{n,c}$ is the score output of the neural network for the nth observation belonging to the cth class, and M is the number of classes.

5.2. Training and hyper-parameter tuning

The dataset was split into 3 chunks of Train, Val and Test. Train and Val dataset were used in the training process, while Test was used to evaluate the model after training. For training the models, we used either SGD with momentum or Adam as the optimizer. Whichever optimizer performed best for getting the baseline accuracy for a dataset is used throughout the experiment for that dataset.

The learning rate was obtained by using the LR Finder method [6]. We used an implementation of it on PyTorch in the open source domain. CosineAnnealing [4] scheduler was used along with the optimizer to update the learning rate. CosineAnnealingLR comes inbuilt with Pytorch. The reason we used these two optimization techniques was to ensure we follow a scientific method for all of the models and these two techniques are flexible enough to adapt themselves to different architectures. The Learning Rate



(a) Uniform Distribution (b) Normal Distribution (c) Left Skewed Distribution (d) Right Skewed Distribution (e) Bimodal Distribution

Figure 4: Input feature activations as calculated on correct predictions from the Helena test dataset

Finder cycles through different learning rates and provides us a region of steepest gradient descent and the CosineAnnealingLR starts from the learning rate found by the Learning Rate Finder and gradually decreases the learning rate to 0 over the epochs using a cosine function.

The models were trained for 150-200 epochs and their learning rate, accuracies and losses were monitored throughout.

5.3. Evaluation metrics

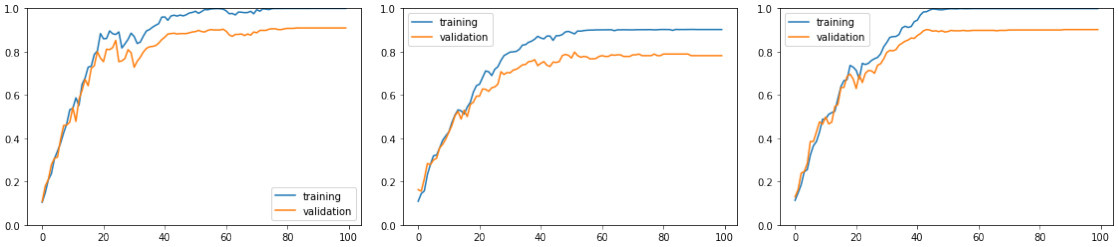
We compared the loss and accuracy of training and validation sets against epochs to measure the rate of convergence for the models. The final accuracy on the test sets was compared as part of the overall performance of an architecture. The model's accuracy on Test dataset was also be evaluated using a confusion matrix. We also explored the activation of neurons for each class on the Test dataset to examine any patterns in their learning.

6. Experiments

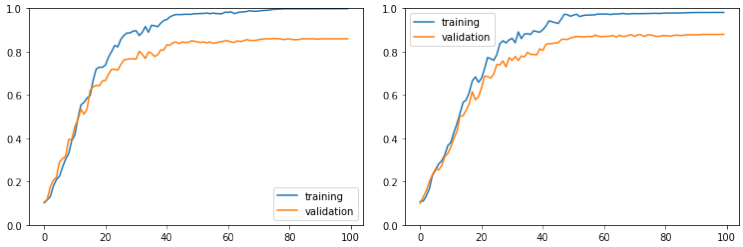
6.1. Helena Dataset

We started training models for number of layers (L) = 10 and number of Neurons (N) = 4000 for the Helena dataset. Then we trained a model having Normal Distribution and neurons and call the accuracy achieved as baseline accuracy for Helena Dataset. The baseline came out to be 32%. We considering a rounded value of baseline accuracy as we are only concerned with the broad value and not the exact one. Then we started reducing P (total number of neurons in the MLP) till we reach a value where the network starts to struggle to achieve baseline accuracy which came out to be 300 neurons.

Having finalized the value of N and P, we trained different MLP models for each distribution under consideration. All the models manage to reach very close to the benchmark. From the training graphs shown in Figure 3, it can be concluded that MLP with uniform distribution and left skewed distribution tend to learn slower than other but at the same time, they manage to keep the over-fitting under check. It can be seen that the models start to over-fit somewhere around 75 epochs but the over-fitting in the MLPs with uniformly distributed neurons and left skewed distribution of neurons is still learning as the val accuracy keeps on increasing and they manage to converge to the baseline accuracy more than any other distributions. Keeping the neuron distribution as uniform or increasing the number of neurons very quickly in the initial layers seems to affect the learning in a positive manner. Also, it can be concluded from the training graphs that the bimodal distribution does

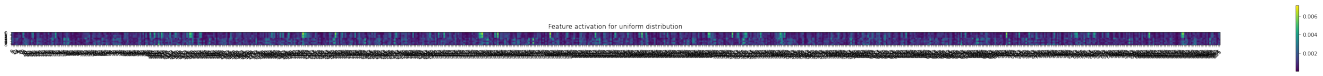


(a) Uniform Distribution (b) Normal Distribution (c) Left Skewed Distribution

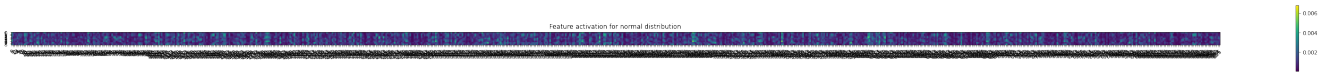


(d) Right Skewed Distribution (e) Bimodal Distribution

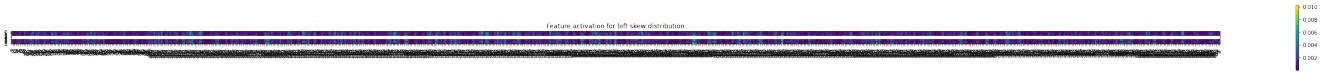
Figure 5: Training and Validation accuracy Curves for training models with different distributions on Cnae-9 Dataset



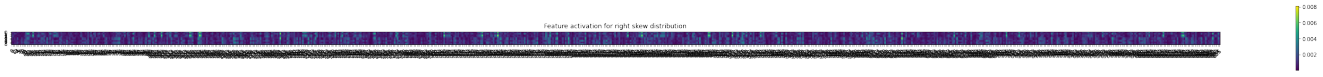
(a) Uniform Distribution



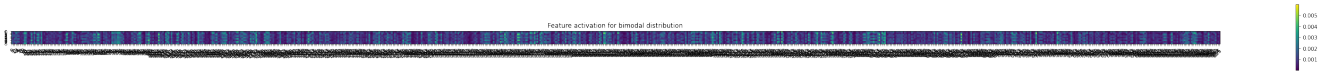
(b) Normal Distribution



(c) Left Skewed Distribution



(d) Right Skewed Distribution



(e) Bimodal Distribution

Figure 6: Input feature activations as calculated on correct predictions from the Cnae-9 test dataset

the best job when it comes to keeping the over-fitting in check. We found that architectures did not tend to have an impact on the rate of convergence as they all would learn very fast in the first 10-20 epochs, and then slowly keep training over 300 epochs.

On testing the models, we find that the learnings of each of these models has been significantly different even though they all manage to converge to a common baseline of 32%. We plotted the activation of input features for the correct predictions (Refer Figure 4) in the test dataset to realise that the all the models have learnt differently from the training dataset. In the figure, the while empty rows represent the classes for which that model got all the test examples wrong. It can be concluded that some models were predicting wrongly on on some class while others are not. A carefully designed ensemble using a MLP of each of the distribution could help boost the topline accuracy significantly. Also, the skewed distributions have larger activations for

some of the initial features in the input while uniform and normal distributions have some kind of regularizing effect on the activations of the input features.

On closer inspection of neuron activation (Refer Figure 9), we see that the bimodal distributed model have stronger activations in the neurons belonging to the middle layers where we reduce the size of the hidden layers. Also, the uniform distributed model has activations of stronger magnitude in the initial layers much more than any other model while the uniform distributed model has a smoother transition from heavier activations in initial layers to lower activations in the last layers. We can derive general trends like a gradual increase in the number of neurons (like uniform distribution) can allow the layers to pass more information down the model and also that if we constrict the size of layers (like the middle layers in bimodal), it can help activate the neurons in the deeper sections of the models.

6.2. CNAE-9 Dataset

With cnae-9 dataset, we were able to achieve state-of-the-art test accuracy of $> 90\%$ with a FC model with 5 hidden layers to have a benchmark to work against, with (N) = 2500. This worked very well with the default initialization of parameters. Next, to get to similar depth models as Helena, we trained a model with (L) = 10 and (N) = 5000 and were able to achieve baseline accuracy with Uniform distribution of Neurons. (As a side note, we found that with deep neural networks, Xavier Random Initialization far succeeded PyTorch's default initialization for fully connected layers: Kaiming Uniform Intializations.) Similar to the previous dataset, we conducted experiments with (L) = 10 layer networks and eventually reduced the number of neurons = 300. We found the convergence rate was highest for Left-skewed distributions, followed closely by Uniform and Bimodal distributions, then Normal and finally by Right-skew distribution which took over 70 epochs. Given that cnae-9 dataset has 856 input features, our understanding is that compressing that many features down to sub-10 neurons on the first layer would give high importance to the first layer's output while also making it more difficult for the information in a sparse dataset, such as cnae-9, to pass through to next layers. Baseline accuracy was achieved very closely by models with larger initial layers such as Left-skewed and Uniform distributions, with Bimodal following close suit. On the other hand, models that had small initial layer sizes such as Right-skewed distribution suffered, with Normal distribution consistently under performing and achieving sub-80% accuracy. Our initial hypothesis of why Normal distribution suffered even more than Right-skewed is that perhaps having multiple smaller layers in the former (at the start and at the end of the architecture) would hamper the learning more than having fewer of the smaller layers. To test this further, we ran the experiments again with (N) = 200 as can be seen in Figure 7 and Figure 8 and we found again that Normal and Right skewed distribution would suffer the most, but Right-skewed distribution would face significant overfitting. This clues us into supporting our hypothesis: larger number of small layers tend to hamper learning, even on training dataset, while small layers in general hamper generalization.

Confusion matrices did not provide any additional insight into how the models performed differently because of the architecture. They corroborated what the Accuracy vs Epoch graphs suggest: Uniform, Left-skewed, distributions would do well at generalizing and training while Right skewed and Normal distributions would suffer, with Bimodal distribution doing somewhere between the two ends of the spectrum.

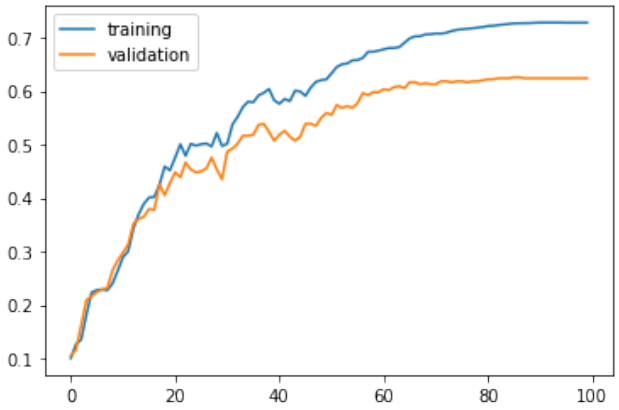


Figure 7: Normal Distribution (N) = 200

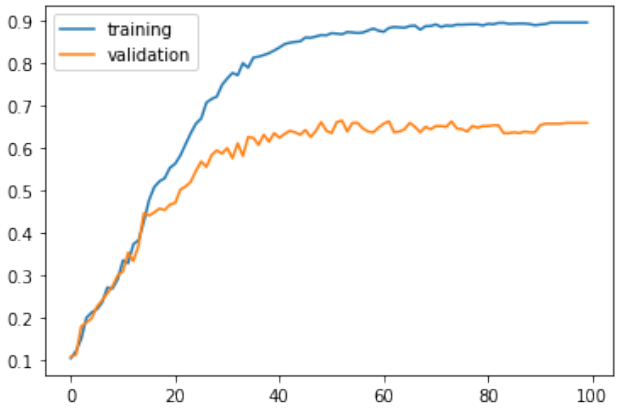


Figure 8: Right Skewed Distribution (N) = 200

7. Conclusion

In this paper, we have confirmed several heuristics that we intuitively use when creating models. We saw that sud-

den changes in size of layers (such as right-skew distribution in cnae-9) tend to significantly negatively affect the performance of the model. At the same time, this difference need not be large, at least in case of sparse datasets (left-skew distribution in cnae-9) as the size of layer seems to be offset by the depth of the network (we achieved baseline accuracy of shallow network with smaller deeper networks). We also found that networks with very little neurons (all uniform and left-skew distributions for both datasets) high performance can be achieved. We can see this as a future expansion of our work by comparing how depth and width of models can vary convergence and performance.

Our favorite finding, and a surprising one as well, was how the normal distribution and right-skewed distribution models differed. They both had similar small initial layers, with the former increasing the size of layers faster than the latter. We, thus, expected that Normal distribution may tend to have higher performance. However, we found that in fact both had similar lower accuracy. Additionally, right-skewed distribution models allowed for overfitting, which meant they were not limited by the ability to learn but rather by the ability to generalize. We would like to further explore this specific relationship between the number of smaller layers and number of neurons per layer. We believe, this can give us insight into how information traverses through a deep MLP. This can be especially interesting in the context of models such as Autoencoders that make use of small layers.

References

[1] Ronen Eldan and Ohad Shamir. The power of depth for feed-forward neural networks, 2016.

[2] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015.

[3] Ivan Khokhlov, Egor Davydenko, Ilya Osokin, Ilya Ryakin, Azer Babaev, Vladimir Litvinenko, and Roman Gorbachev. Tiny-yolo object detection supplemented with geometrical data. *CoRR*, abs/2008.02170, 2020.

[4] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts, 2017.

[5] Zhou Lu, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang. The expressive power of neural networks: A view from the width, 2017.

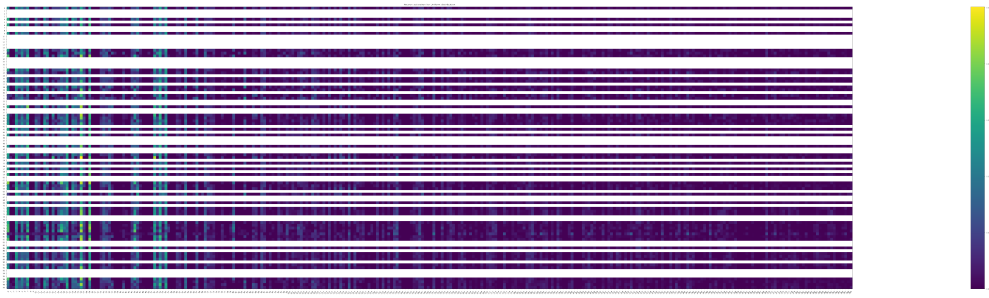
[6] Leslie N. Smith. Cyclical learning rates for training neural networks, 2017.

[7] D. Stathakis. How many hidden layers and nodes? *International Journal of Remote Sensing*, 30(8):2133–2147, 2009.

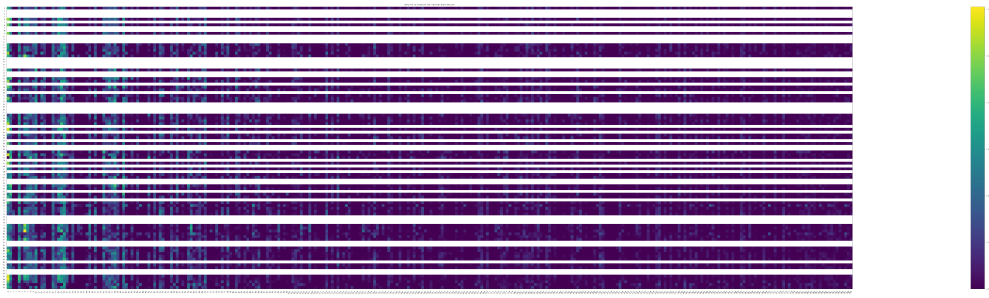
[8] Matus Telgarsky. Benefits of depth in neural networks, 2016.

[9] Stephan Trenn. Multilayer perceptrons: Approximation order and necessary number of hidden units. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, 19:836–44, 06 2008.

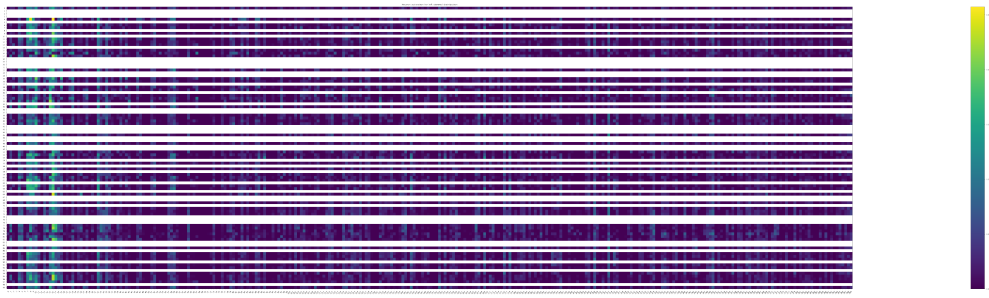
[10] Lin Wang and Kuk-Jin Yoon. Knowledge distillation and student-teacher learning for visual intelligence: A review and new outlooks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, page 1–1, 2021.



(a) Uniform Distribution

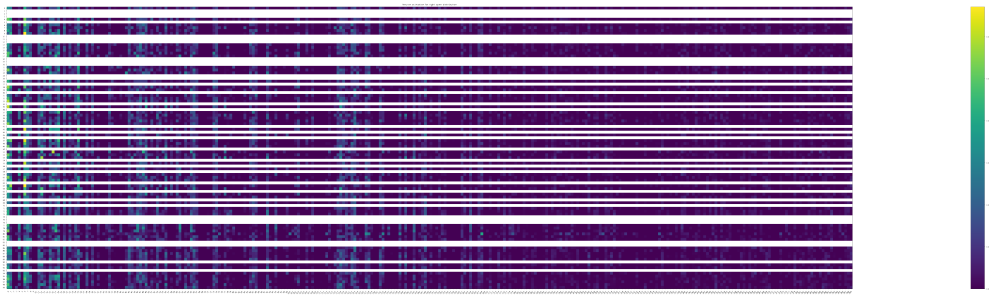


(b) Normal Distribution



(c) Left Skewed Distribution

(d) Right Skewed Distribution



(e) Bimodal Distribution

Figure 9: Neuron activations as calculated on correct predictions from the Helena test dataset

(a) Uniform Distribution

(b) Normal Distribution

(c) Left Skewed Distribution

(d) Right Skewed Distribution

(e) Bimodal Distribution

Figure 10: Neuron activations as calculated on correct predictions from the Cnae-9 test dataset