Key Properties for Model Generalization in Regression: A Case Study on Initialization Methods

Kelyan Hangard
Semester project hosted by the Intelligent Maintenance and Operations Systems (IMOS) Lab

Spring semester 2024

Abstract

This project explore different factors influencing model generalization in regression tasks, focusing particularly on the role of neural network weight initialization techniques and their impact on generalization performance. Specifically, the study explores whether weight initialization can mitigate the rank collapse phenomenon in regression models. Initially, the project assesses uncertainty models, finding that they generally outperform traditional regression approaches. The primary focus then shifts to analyzing various weight initialization methods. Five distinct initialization strategies were rigorously tested across nine different regression datasets. The findings indicate that random and orthogonal initializations significantly enhance performance on unseen data compared to standard PyTorch, Xavier, and other methods such as He, Variance Scaling, or Zero initialization. An interesting outcome is that weights initialized orthogonally showed significantly lower correlations among themselves than those initialized by any other method. Furthermore, spectral analysis of these weights revealed a unique, uniform distribution of singular values centered around two, contrasting with the exponential decay distributions observed in all other initialization methods. Even if a clear link between weight correlations, spectral characteristics of weights and embeddings, and overall model performance was not found, these insights are profoundly intriguing and potentially valuable for a broad range of applications involving regression tasks on unseen data.

1 Introduction

Regression with neural networks is widely used in various areas today, from technology and social media to automotive, healthcare, finance, retail, and manufacturing. Despite the widespread application, regression techniques are often less studied compared to classification. It's critical for regression models to accurately predict on new, unseen data, particularly when this data might not align with the training distribution. Models typically struggle with predictions on such out-of-training-distribution data. Improving the generalization capabilities of regression neural networks is essential, especially in a fast-changing world where test sets may vary greatly due to new market conditions, environmental shifts, or evolving user behaviors. This paper explores the impact of different neural network weight initialization techniques on the generalization performance of regression models.

In both regression and classification, neural networks are subject to the rank collapse effect [1] during training. This effect occurs when the network's feature representations converge to a lower-dimensional space, causing the model's weight matrix to lose full rank. As a result, some rows or columns of the weight matrix can be expressed as a linear combination of others. This issue can reduce the model's ability to capture complex patterns and negatively impact its performance, especially on tasks that require predicting on unseen data. Mitigating the impact of rank collapse on regression performance is crucial. We hypothesize that reducing weight correlations and promoting more uniform singular values in the weights and embeddings will enhance generalization performance. This is because it increases the diversity of the learned features, ensures that each weight is important, and maintains a balanced embedded space that fully utilizes the variance of the input data.

2 Materials and Methods

2.1 Data

To conduct this analysis, we initially applied various weight initialization methods to the Wine Dataset [2], which is divided into two subsets: one containing 1,600 red wines and another consisting of 5,000 white wines. Each subset has 12 features, such as acidity, sugar content, pH, alcohol level, etc., alongside a quality rating ranging from 3 to 9 that our models aim to predict. The feature distributions is very different between the red and white wine datasets. We conducted experiments by training our model only on the red wine data and then testing it on the white wine data, and vice versa, to observe the effects of this distribution difference and to measure our model's ability to adapt to unseen data.

The second dataset that we used is the Rotated MNIST dataset, an adaptation of the well-known MNIST dataset [3] designed for regression tasks. Each digit image has been rotated, and the model's task is to predict the angle of rotation. For testing, we used a set of 1,000 out of training distribution angles that were not included in the training set, which consists of 7,000 images.

We then expanded our analysis by incorporating five additional datasets. The first is the Boston Housing Dataset [4], which uses data from the U.S. Census Service to explore housing characteristics in the Boston, MA area. Next, we analyzed the Concrete Strength Dataset [5], a collection of 1,000 records detailing the composition of concrete mixes. This dataset includes various ingredients and their proportions, as well as the age of the concrete, to predict its compressive strength. Another dataset employed is the Energy Efficiency Dataset [6], containing 800 records that evaluate the heating and cooling load requirements of buildings based on building parameters, aiming to assess their energy efficiency. We also used the Power-Plant dataset [7], this dataset consists of 9,568 data points from a Combined Cycle Power Plant (CCPP) collected over a six-year period from 2006 to 2011. The dataset captures the plant's operations under full load conditions with the goal of predicting the net hourly electrical energy output (EP). Finally, we utilized the Yacht Dataset [8], which contains 300 lines. This dataset's objective is to predict the residuary resistance of sailing yachts at the initial design stage, an important factor for evaluating shape performance and estimating required propulsive power. Key inputs include basic hull dimensions and boat velocity.

2.2 Libraries

Our analysis was conducted using Python, with scripts and Jupyter notebooks serving as our primary tools. For data manipulation, we employed standard frameworks including Pandas and NumPy, complemented by tqdm for efficient progress bar management during data processing. Data visualization was facilitated by Matplotlib and Seaborn, while mathematical computations and specific metrics were calculated using the Math and SciPy libraries. The creation and manipulation of neural networks were primarily executed in PyTorch, sometimes using TensorFlow. PyTorch's initialization functions made it straightforward to define and apply various weight initialization methods.

Regarding uncertainty models, two distinct models have been tested which are SCOD and Deep evidential. SCOD which stands for Sketching Curvature for Efficient Out-of-Distribution Detection [9] is a framework designed to improve the deployment of Deep Neural Networks (DNNs) by equipping them with the ability to detect out-of-distribution (OoD) inputs effectively. SCOD operates in two phases:

- **Offline:** It utilizes matrix sketching techniques to create a low-rank approximation of the Fisher information matrix from a trained DNN and its training data. This matrix helps identify influential directions in the network's weight space, which are crucial for making predictions.
- **Online:** SCOD estimates uncertainty by measuring how predictions change when inputs are perturbed in directions orthogonal to those identified as influential. This allows SCOD to assess whether new inputs might be out of the training distribution.

The framework is architecture-agnostic, meaning it can be applied across various DNN architectures. It has shown to be computationally efficient and performs comparably or better than existing methods in detecting OoD inputs, thereby enhancing the reliability and safety of AI systems in real-time decision-making environments.

The second method used is deep evidential regression [10], this framework consist of training deterministic neural networks to estimate both a continuous target and its associated evidence, enabling the learning of aleatoric and epistemic uncertainties. The approach involves using evidential priors over the Gaussian likelihood function and training the network to infer hyperparameters of this distribution. Furtermore, the method incorporates training regularization to align predictions with correct outputs. Deep evidential does not require sampling during inference or out-of-distribution examples for training, enhancing its efficiency and scalability.

2.3 Models

We used the same neural network structure for all weight initialization methods and all datasets except the Rotated MNIST dataset which is an image dataset that requires a convolutional neural network. The chosen model is a simple multi-layer perceptron (MLP) structured to perform both encoding of input data into an embedded representation and regression from that embedded representation to an output. ReLU is used in the encoder to introduce non-linearity after the first and second linear layers. The model includes a total of 3 linear layers. Including the activation layers, there are 5 layers in total. We selected this model first because it is a regression model previously used in the SCOD (Sketching Curvature for Efficient Out-of-Distribution Detection) paper for the wine dataset and other datasets. This choice allows us to directly compare our results with those reported in the SCOD paper. Furthermore, we wanted to examine the impact of rank collapse and explore strategies to mitigate this effect. The rank collapse effect tends to be more pronounced in smaller models, which led us to choose a simple model structure. Regarding the model we used on the Rotated MNIST dataset, it is a convolution neural network (CNN) using PyTorch's nn.Sequential. It consists of several layers configured to process image data. Here is how the network is structured:

- Convolutional Layers: Three layers are used, progressively increasing the depth of feature maps from 1 to 16, then 16 to 32, and maintaining 32 in the final convolutional layer. These layers help extract increasingly complex features from the input image.
- Activation Functions: ReLU functions follow each convolutional layer to introduce non-linearity, enabling the model to learn complex patterns in the data.
- Pooling Layers: Two MaxPooling layers with a 2x2 filter reduce the spatial dimensions after the first and second convolutional layers, helping to decrease the number of parameters and control overfitting.
- Flattening Layer: Converts the 3D output of the final convolutional layer into a 1D vector to feed into the fully connected layers.
- Fully Connected Layers: Two linear layers transform the flattened output to 10 dimensions and subsequently compress it to 2 dimensions for the final output.

This model has been selected for the same reason than the sequential model: because it's a relatively simple network and also because it is the same convolution network that has been used in the SCOD paper.

2.4 Weight Initialization

Several weight initializations have been tested across the project. First, weights were initialized with tensorflow.keras.initializers for tensorflow models, and then we switched to use exclusively pytorch models using torch.nn.init.

The first weight initialization tested is xavier initialization also known as Glorot initialization (nn.init.xavier_uniform_ in pytorch), we considered this initialization as a kind of baseline to compare with other weight initialization methods. The main goal of this Xavier initialization is to keep the scale of the gradients approximately the same in all layers of a deep neural network. Xavier initialization sets a layer's weights from a uniform distribution between (a,a) where $a = \sqrt{\frac{6}{n_{in} + n_{out}}}$ with n_{in} being the number of input units to the layer and n_{out} the number of output units. The gain is specifically calculated for the ReLU activation function using the pytorch function gain=nn.init.calculat_gain('relu') since we are using relu activation function.

We also tested the random weight initialization, also known as normal weight initialization using torch.nn.init.normal_. Normal initialization assigns weights randomly over a fixed interval, distributing them uniformly across a specified range and has a fixed variance. Xavier initialization adapts the range based on the layer's dimensions to ensure variance remains constant, whereas normal initialization uses a fixed range that does not change with layer size or depth and does not ensure that variance stay constant accross layers. We decided to take zero as the mean of the normal distribution and one as the standard deviation.

He initialization were also used. He initialization, also known as He normal or He uniform, is a method specifically designed for layers using ReLU activation functions, this technique adjusts the initial weight settings to solve the problems that typically arise when training deep neural networks with ReLU activations, such as the vanishing gradient problem. Weights are initialized from a normal distribution with mean zero and variance $\sigma^2 = \frac{2}{n_{in}}$ where n_{in} is the number of input units in the weight matrix. This variance scaling ensures that the variance of the inputs is roughly equal to the variance of the outputs, maintaining a balance in the activation distribution across the layers. The nonlinearity parameter has been set to "relu" and mode has been set to "fan_in"

Orthogonal weight initialization (nn.init.orthogonal_ in Pytorch) is a technique used in training deep neural networks where the weights of each layer are initialized as orthogonal matrices. The concept of orthogonal initialization is based on the idea that rows (or columns) of a weight matrix are orthogonal to each other. This means that the dot product between any pair of different rows (or columns) is zero, and the dot product of a row (or column) with itself is one. The main benefit of this is that it preserves the length of vectors during transformations, which helps in maintaining the norm of the gradient as it is backpropagated through the network.

Variance Scaling initialization is very similar to He initialization and initialize the weights of each layer with a distribution (normal or uniform) with a mean of zero and a variance of $\sigma^2 = \frac{scale}{fanmode}$ where:

- Scale is a hyperparameter that defines the scaling factor. Common choices include 1 (LeCun initialization), 2 (He initialization for ReLU)
- Fan mode: Can be fan-in, fan out, or average. Choosing fan-in maintains variance in the forward propagation, fan-out maintains it in the backward propagation, and average considers both. We decided to chose "fan_in".

Finally Zero initialization has been tested as well, Zero weight initialization refers to setting all the initial weights of a neural network to zero. This approach, while straightforward, generally results in suboptimal and problematic behavior during the

training process of most network architectures. The weight initialization is quite simple, every weight in the network is initialized to exactly zero, which means every neuron starts with the same bias and weight values.

3 Results and Discussion

3.1 Uncertainty Model Performances

To explore various methods to enhance the generalization capabilities of regression models, we investigated two uncertainty models discussed earlier in the 2.2 section. These models are particularly promising for their potential to perform well on test sets containing unseen data, including out-of-distribution points. To perform a comparison between these uncertainty models and more traditional approaches, we employed four identical neural network models each with two hidden layers, as outlined in the 2.3 section. One network was integrated with SCOD, another utilized the deep evidential regression framework, and the remaining two networks were enhanced with L1 and L2 regularization, respectively. Additionally, we developed a classifier regressor model that combines classification and regression losses during training, using its regression output for predictions.

We conducted ten training and tests for each model using the wine dataset, training on the red wine dataset and testing on the white wine dataset. The performance of each model was assessed using the Root Mean Square Error (RMSE), which measures the difference between the model predictions and the actual values on the test set. Findings from these experiments are presented in the table 1 below:

Method	RMSE	NLL
SCOD	0.896	4.198
Deep Evidential Regression	0.938	4.926
L1 - Regression	0.947	-
L2 - Regression	0.952	-
Classifier - Regressor	1.0164	-

Table 1: Comparative performance of various regression models

As illustrated in table 1, the two uncertainty models outperform the conventional models, despite utilizing the same network architecture, hyperparameters, and training and testing conditions. These findings are intriguing and lead us to several hypotheses regarding why these uncertainty models achieve better performances. The focus of the project then shifted to weight initialization, so we didn't explore further robust investigation with additional datasets, but here are our hypotheses:

- **SCOD:** By creating a low-rank approximation of the Fisher information matrix, SCOD identifies the most influential directions in the network's weight space, which are crucial for prediction. During operation, SCOD assesses how perturbations orthogonal to these directions affect predictions, enabling it to effectively detect inputs that are OoD. This is vital in real-world applications where new, unseen data may not necessarily follow the same statistical distribution as the training data.
- **Deep Evidential Regression:** This model learns not just to predict a continuous target but also to quantify the uncertainty of its predictions through aleatoric (data inherent) and epistemic (model uncertainty) components. By understanding both types of uncertainty, the model can better handle situations where the data might not strictly conform to the patterns seen during training.

3.2 Weight Initialization

3.2.1 Performances

We conducted an initial comparison of weight initialization performances, focusing on Random Weight Initialization and Xavier Weight Initialization. Each model, including Deep Evidential, SCOD, L1 model, L2 model, and Classifier Regressor, was trained

10 times on the red wine dataset and tested 10 times on the white wine dataset. We analyzed the averaged Root Meam square errors on the figure 1 below. The results indicate that Random Weight Initialization generally outperforms Xavier Weight Initialization across most models. These promising findings suggest further investigation. To validate these results, it would be beneficial to analyze additional datasets and explore more weight initialization methods to determine if this pattern is general or is specific to the wine dataset. That is exactly what we did in the next part of the project.

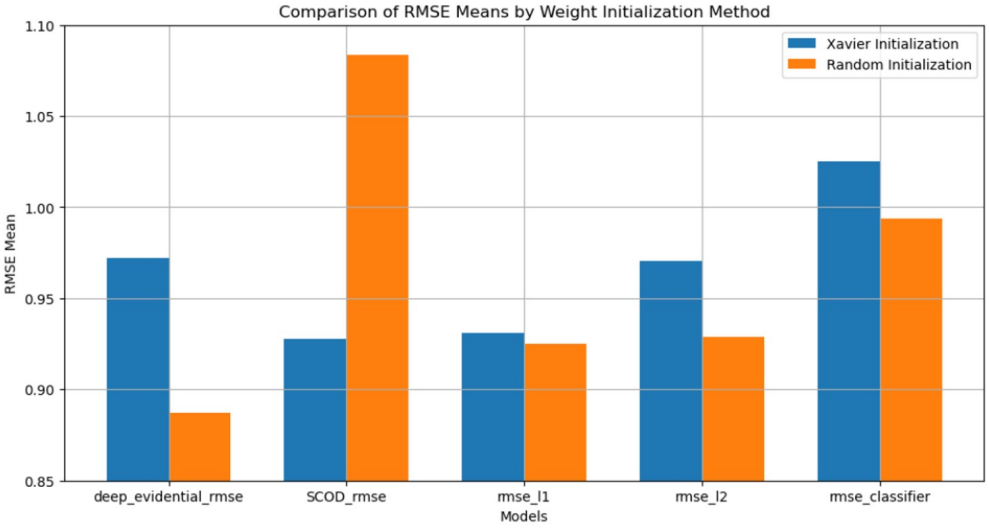


Fig. 1: Bar plots of RMSE averaged over 10 samples per weight initialization - model combination. Each model and weight initialization method have been trained on the red wine dataset and tested on the white wine dataset.

We then continue our analysis with L1 and L2 models on the Rotated MNIST Dataset, as detailed in the annex (see Figure B4), we again observe that Random Weight Initialization outperforms Xavier Weight Initialization. We then extended our examination to other weight initialization methods on the Wine dataset, as illustrated in the annex (see Figure B5). This additional analysis led us to exclude Zero Weight Initialization from further consideration due to its performances which are significantly lower compared to all other weight initialization methods.

To further analyze trends in weight initialization and confirm our intuition, we applied five different weight initialization methods as defined in 2.4: Random, Orthogonal, Xavier, Pytorch Init, and He. Each weight initialization method was trained and tested five times across each of the datasets listed in 2.1, which include Red Wine, White Wine, Rotated MNIST, Boston Housing, Concrete Strength, Energy-Efficiency, Power-Plant, and Yacht. We split each dataset into training and testing sets, conducted five training and testing cycles for each initialization method, and computed the average Root Mean Squared Error (RMSE) to obtain statistically significant results. These results are detailed in the table 2 and figure 2.

Our findings confirm the initial trend, which is very interesting: Random Weight Initialization and Orthogonal Weight Initialization outperform Xavier, PyTorch Init, and He Weight Initialization in regression tasks, especially in predicting unseen data. This distinction is clearly visible and is crucial for selecting an appropriate weight initialization method in regression applications.

However, while differences are significant, confidence intervals illustrated in Figure 2 show some overlaps, which indicate that we need to be cautious with these results. However, the upper bound of the confidence interval for Orthogonal Initialization is comparable to the mean of the Xavier initializations, still suggesting a significant difference. These analyses were conducted across a wide range of datasets with multiple tests averaged, supporting the reliability of our conclusions.

Weight Initialization	RMSE	RMSE std
Random	3.157	0.053
Orthogonal	3.188	0.052
Xavier	3.243	0.062
Pytorch Init	3.251	0.055
He	3.490	0.083

Table 2: Extensive comparison of weight initialization methods across eight different regression datasets, RMSE averaged among five samples per weight initialization - dataset combination.

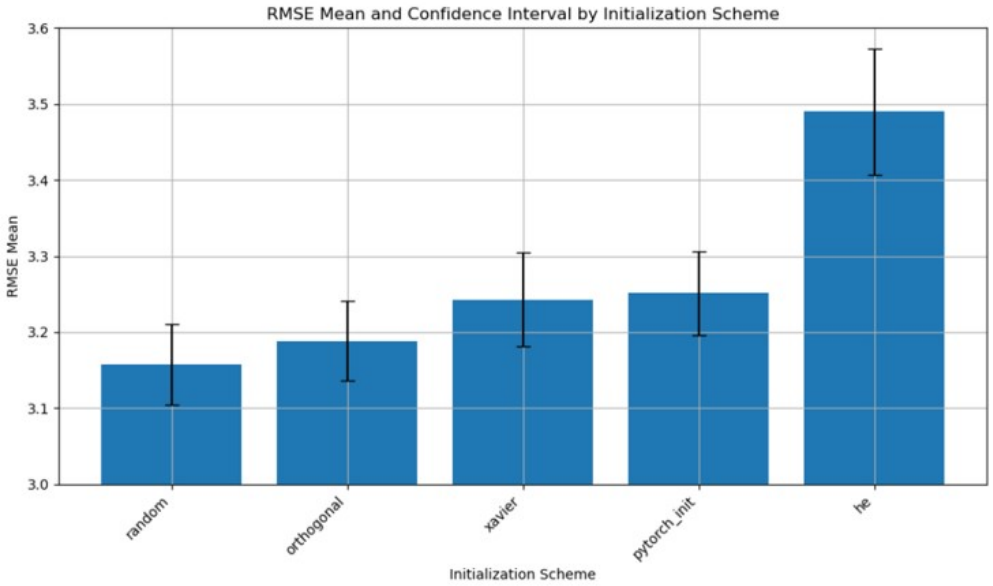


Fig. 2: Bar plots of RMSE averaged over five samples per weight initialization - dataset combination. Tested have been conducted on a L2 regression model and the confidence interval corresponds to two times the standard deviation.

This performance gap may be related to our earlier hypothesis (see section 1): Random and Orthogonal weight initialization methods might be less prone to rank collapse due to their greater diversity in weight assignments and lower correlation among weights and embeddings. We will delve deeper into this hypothesis in the following section by exploring whether there is a definitive link between model performance, weight correlations, and the uniformity of weights and embeddings' singular values.

3.2.2 Weights and embeddings analysis

Let's now explore the underlying causes of the observed performance differences seen in the previous section. Following the same training of each neural network as detailed before, we computed the average measure of weight correlations across all linear and convolutional layers of the models, using cosine similarity as the metric. Cosine similarity quantifies the similarity between two vectors, A and B, based on the angle between them. It is computed using the following equation:

$$CosineSimilarity(A, B) = \frac{A \cdot B}{||A||||B||} \tag{1}$$

After computing and averaging weight correlations for each weight initialization method, we can observe results in the table 3 below or on the graphic 3:

Weight Initialization	Weight Correlation	Weight Correlation std
Orthogonal	0.401	0.000
He	0.429	0.001
Xavier	0.431	0.000
Random	0.436	0.001
Pytorch Init	0.441	0.001

Table 3: **Comparison of weight correlations after training with respect to the weight initialization method used.**

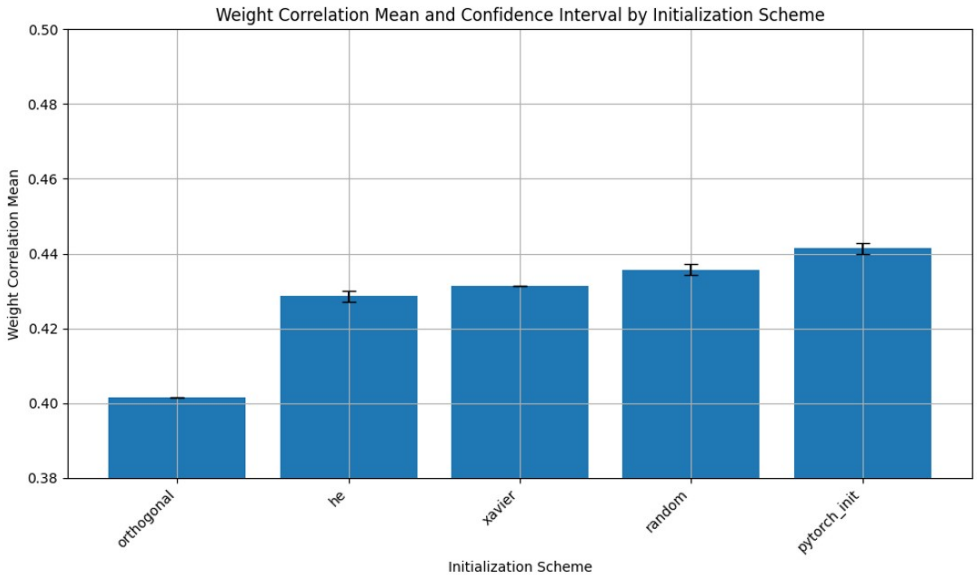


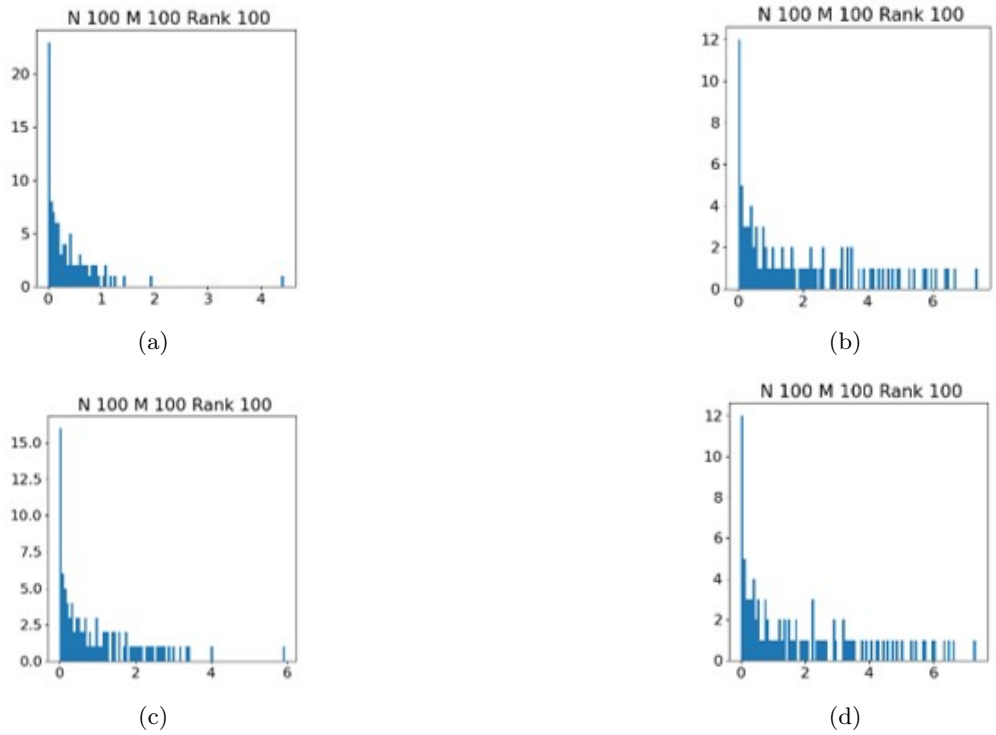
Fig. 3: **Bar plots of averaged weight correlations taken with 5 samples per weight initialization - dataset combination. Tested have been conducted on a L2 regression model and the confidence interval corresponds 2 times the standard deviation. Weight correlations have been computed using cosine similarity.**

As illustrated in Figure 3, our initial hypothesis does not find clear support, as there appears to be no direct correlation between weight correlations and performance. For instance, while random weight initialization showed the best performance in earlier tests, it exhibits the second-highest weight correlations here. And vice-versa, He weight initialization, which performed the poorest, now shows the second-lowest weight correlations.

However, an intriguing observation can be noticed regarding orthogonal weight initialization, which yields a significantly lower weight correlation (0.401) compared to the others, which range between 0.429 and 0.441. This finding aligns with the nature of orthogonal initialization, where weight matrices are set such that each row (or column) is orthogonal to the others. This configuration ensures that each weight vector is independent and contributes equally to the output variation. This independence, preserved even after training, helps reduce correlations among weights. Such outcomes highlight the potential benefits of using orthogonal weight initialization to mitigate rank collapse.

Now, let's look at the spectrum analysis of the weights. Using the same testing environment as explained previously, let's now compute the average of the singular value decomposition of trained weights for each weight initialization methods trained respectively on each of the datasets described before.

As showed in Figure 4, the weight initializations from PyTorch, He, Random, and Xavier all result in a similar singular value decomposition, characterized by an exponential decay distribution. This pattern indicates that most singular values are near zero, with only a few larger values (approaching 6) accounting for the majority of



(a) (b) (c) (d)

Fig. 4: Histograms of the weights singular value decomposition after training of pytorch initialization (a), he weight initialization (b), random weight initialization (c) and xavier weight initialization (d). Each x axis represents the values of the singular values and y axis represents the frequency of these singular values.

the variance in the trained weights. Such a distribution is indicative of rank collapse, where a limited number of parameters carry most of the variance, proving the most important of the model, while the majority are redundant and can be expressed as linear combinations of other parameters.

Despite these insights, the spectrum analysis does not establish a clear correlation between model performances and the distribution of trained weight singular values. The weight initializations, from Random (which had the best performance) to He (which had the poorest), all share the exponential decay distribution, as shown in Figure 4. We are then unable to validate our hypothesis and link the analysis of trained weight singular values to model performances based on the available data.

On the other hand, Figure 5 illustrates that neural networks initialized with Orthogonal Initialization show a distinctly different singular value decomposition. The distribution appears normally shaped, centered around two, unlike the exponential decay observed with other initializations. This unique distribution is quite uniform, with almost all singular values contributing to the variance. This pattern suggests that rank collapse may be less present with Orthogonal Initialization, aligning with our previous findings which highlighted low weight correlation and superior RMSE performance with this method. This observation is particularly interesting, as it supports the ability of Orthogonal Initialization in mitigating rank collapse effects.

Finally, we also explored the embeddings' singular value decomposition, using the same testing framework as before but focusing on the network's trained embeddings. Here, "embeddings" refer to the embedded representation of the input data after it has been processed through the encoder part of the neural network, it is a 100-dimensional feature representation of the original input before this input is feed to a regressor that will map this 100 dimension to dimension 1. After training, we extracted these embeddings for each weight initialization and performed a singular value decomposition. Each eigenvalue was normalized by dividing it by the sum of all eigenvalues,

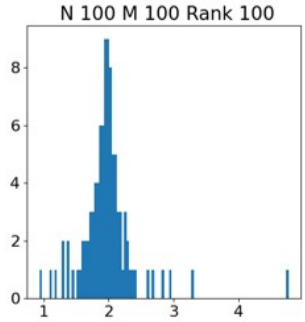


Fig. 5: Histogram of the trained weights singular value decomposition of orthogonal weight initialized models. X axis represents the values of the singular values and y axis represents the frequency of these singular values.

and the results were visualized in a scree plot, which plots the values of these singular values against their indices.

Figure 6 shows the scree plot of the embeddings' singular values for the Xavier initialization. Accross the different initializations, all weight initializations resulted in similar distributions, as detailed in the annex (see Figure C6). This similarity across all plots makes it impossible to establish any definitive correlations between the embeddings' spectrum analysis and model performance, as all distributions share the same general shape.

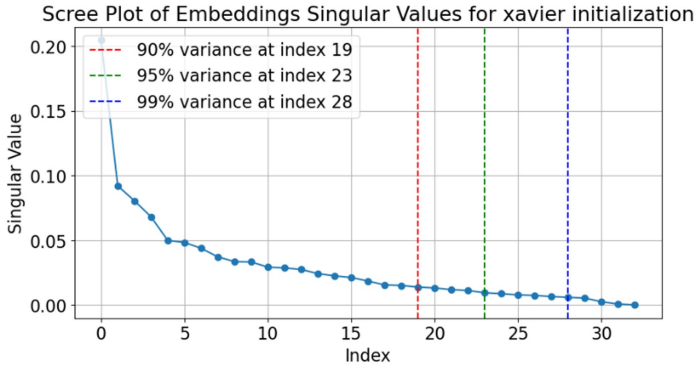


Fig. 6: Scree plot of the singular value decomposition of xavier weight initialization.

In order to distinguish the weight initialization techniques, we calculated the number of singular values required to explain 90%, 95%, and 99% of the variance. These thresholds are represented by red, green, and blue lines, respectively, on the scree plots, as illustrated in Figure 6. For each weight initialization method, we analyzed and compared the indices at which these percentages of variance were captured. The results are summarized in bar plot format in Figure 7, highlighting the variances explained by different numbers of singular values across the methods.

As shown in Figure 7, it remains challenging to establish a definitive link between the embeddings' spectrum analysis and performance outcomes. However, it is interesting that models with orthogonal weight initialization generally require a larger portion of their embeddings to explain the variance, as indicated by the higher indices. This observation aligns with previous results suggesting the robustness of orthogonal initialization. Yet, this finding must be approached with caution, as showed in the graph 7, differences are small and the overall distribution shapes do not vary significantly as observed in weight spectrum analysis. Then we cannot draw broad conclusions from this analysis alone.

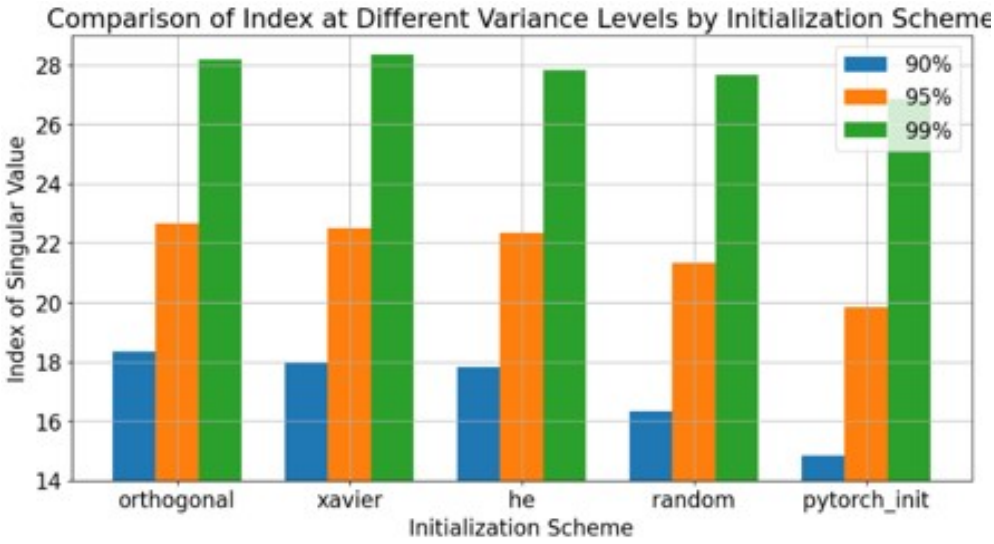


Fig. 7: **Analysis of Singular Value Indices Required to Explain 90% (in green), 95% (in orange), and 99% (in red) of Embeddings Variance**

4 Conclusion and further researches

In summary, this project resulted in valuable insights into improving the performance of regression neural networks, particularly in terms of their ability to generalize to unseen and out-of-distribution data. We hypothesized that poor performance on unseen data could be attributed to rank collapse, and that models with less correlated weights and more uniformly distributed weights and embeddings might perform better. Although our analyses did not definitively confirm this hypothesis due to the absence of a clear link as mentioned in the hypothesis, the findings were still very interesting and are summarized as follows:

- **Enhanced Performance with Uncertainty Models:** Uncertainty models like SCOD and Deep Evidential Regression appear to outperform conventional models such as L1 and L2 regularization in regression tasks involving predicting on unseen data.
- **Better performances of Random and Orthogonal Initializations:** Extensive testing revealed that Random and Orthogonal weight initializations yield better results than the more commonly used Xavier or default PyTorch initializations for regression tasks when it comes to predict on unseen data.
- **Orthogonal Initialization Mitigates Rank Collapse:** Orthogonal initialization not only performs exceptionally well but also demonstrates significantly lower weight correlations and a distinctly uniform weight spectrum. These characteristics suggest that Orthogonal initialization can enhance network performance and potentially reduce the effect of rank collapse.

These outcomes are both intriguing and promising, particularly the better performances of Orthogonal and Random weight initialization methods compared to others. In an environment where regression problems are everywhere across various industries yet less explored than classification tasks, these insights are very valuable. They suggest that simply selecting an appropriate weight initialization technique could significantly enhance a model's performance and its ability to generalize to unseen data.

This would be interesting to go further in this project by exploring the following areas:

- **Expand Testing of Uncertainty Models:** Confirm or challenge the current trend by applying uncertainty models to additional datasets.
- **Broader Analysis Across Datasets and Models:** Conduct similar studies using more complex models and a wider variety of datasets to assess the generalizability

of the findings across different types of regression models, not just the simpler ones investigated thus far.

References

[1] Noci, L., Anagnostidis, S., Biggio, L., Orvieto, A., Singh, S.P., Lucchi, A.: Signal propagation in transformers: Theoretical perspectives and the role of rank collapse. Advances in Neural Information Processing Systems **35**, 27198–27211 (2022)

[2] Aeberhard, S., Forina, M.: Wine. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5PC7J (1991)

[3] LeCun, Y., Cortes, C., Burges, C.: Mnist handwritten digit database. ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist **2** (2010)

[4] Harrison Jr, D., Rubinfeld, D.L.: Hedonic housing prices and the demand for clean air. Journal of environmental economics and management **5**(1), 81–102 (1978)

[5] Yeh, I.-C.: Concrete Compressive Strength. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5PK67 (2007)

[6] Tsanas, A., Xifara, A.: Energy Efficiency. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C51307 (2012)

[7] Tfekci, P., Kaya, H.: Combined Cycle Power Plant. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5002N (2014)

[8] Gerritsma, O.R. J., Versluis, A.: Yacht Hydrodynamics. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5XG7R (2013)

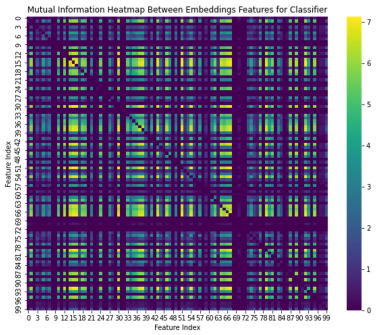
[9] Sharma, A., Azizan, N., Pavone, M.: Sketching Curvature for Efficient Out-of-Distribution Detection for Deep Neural Networks (2021)

[10] Amini, A., Schwarting, W., Soleimany, A., Rus, D.: Deep Evidential Regression (2020)

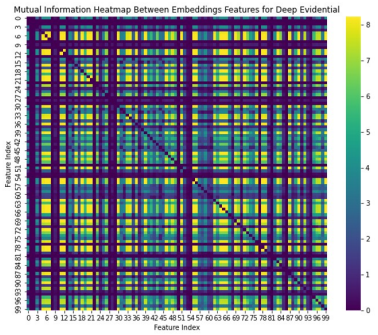
Appendix A Mutual Informations

Appendix B Weight Initialization Performances

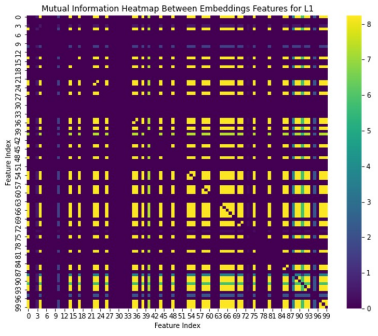
Appendix C Embeddings Spectrum Analysis



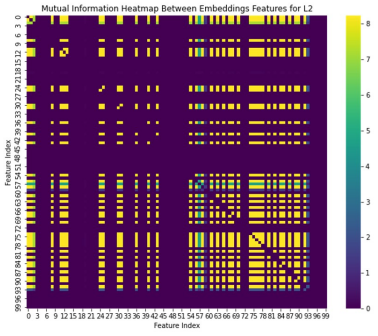
(a)



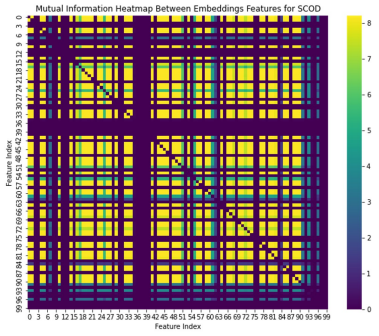
(b)



(c)

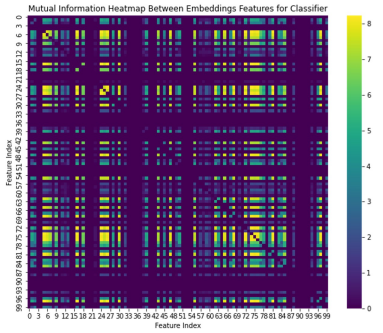


(d)

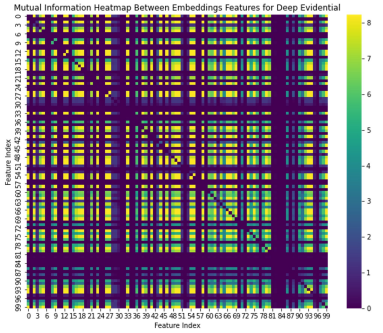


(e)

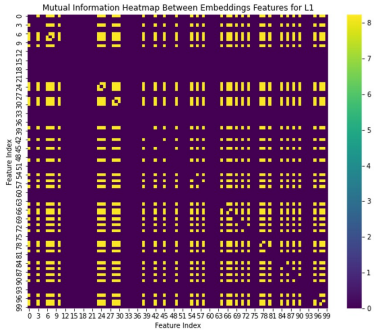
Fig. A1: **Mutual Information Heatmaps of trained embeddings from models initialized with Xavier weight initialization of a classifier-regressor model (a), deep evidential regression model (b), l1 model (c), l2 model(d), SCOD model (e)**



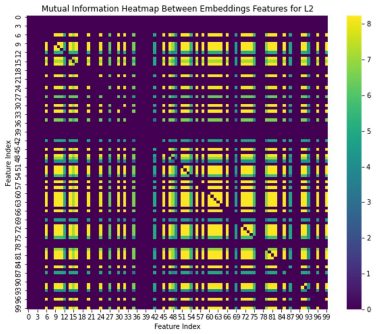
(a)



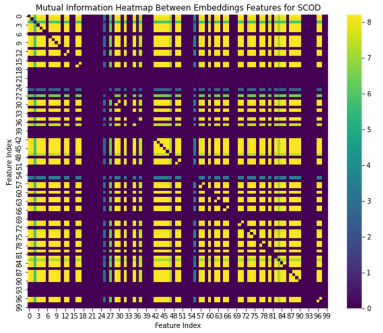
(b)



(c)

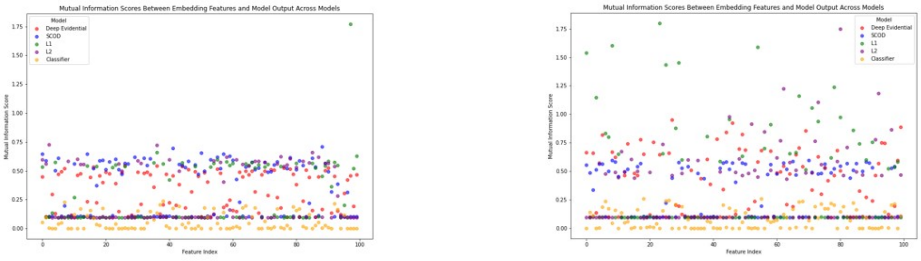


(d)



(e)

Fig. A2: **Mutual Information Heatmaps of trained embeddings from models initialized with Random weight initialization of a classifier-regressor model (a), deep evidential regression model (b), l1 model (c), l2 model(d), SCOD model (e)**



(a) (b)

Fig. A3: **Mutual information scores between embedding features and model output across models initialized with Xavier weight initialization (a) and Random weight initialization (b).**

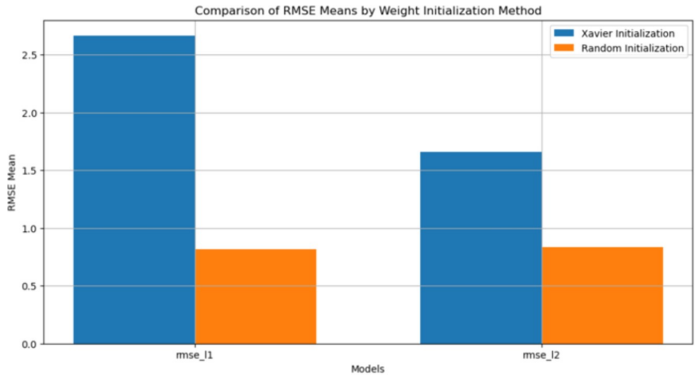


Fig. B4: **Bar plot representing performances of L1 and L2 models on the Rotated MNIST Dataset comparing Xavier and Random weight initialization. Ten models have been trained on regular angles and tested on the out of distribution angles and the average have been taken.**

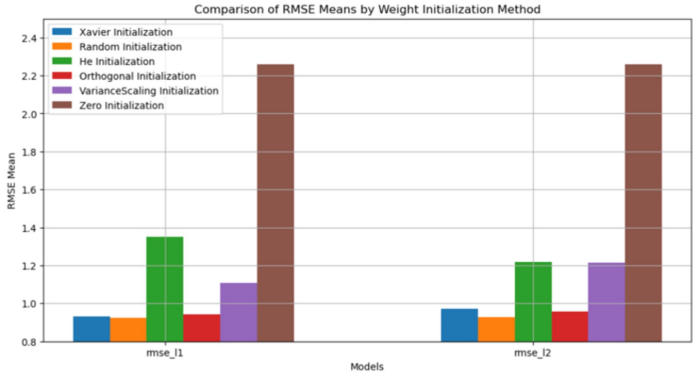


Fig. B5: **Bar plot representing performances of L1 and L2 models on the Wine Dataset comparing Xavier, Random, He, Orthogonal, VarianceScaling and Zero weight initialization. Ten models have been trained on the red wine and tested on the white wine dataset and the average RMSE on the test set have been taken.**

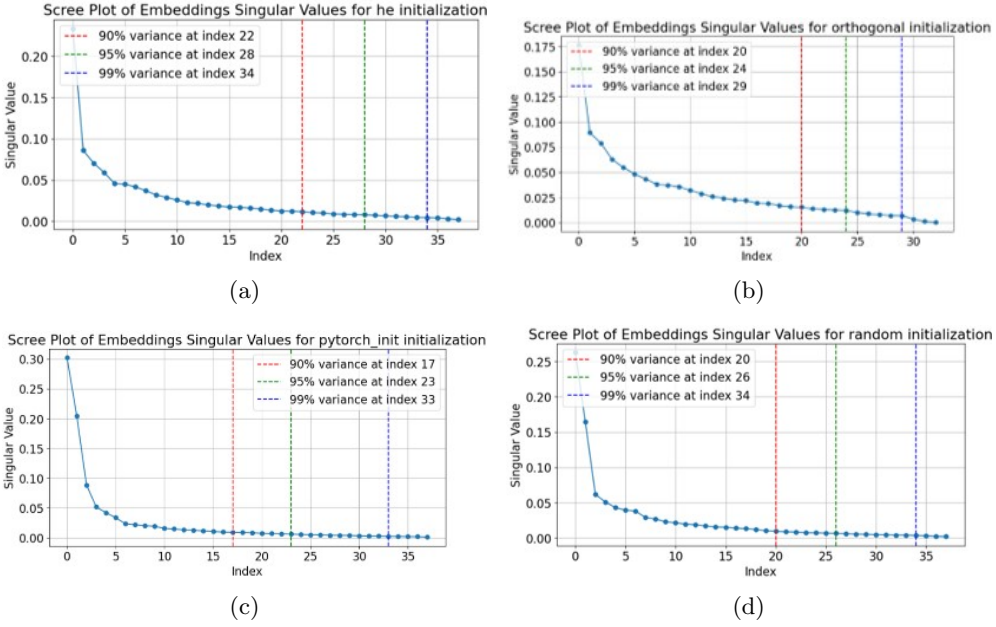


Fig. C6: **Scree plot of the singular value decomposition of He initialization (a), orthogonal initialization (b), pytorch initialization (c) and random initialization (d).**