Erasmus University Rotterdam
Erasmus School of Economics
Bachelor Thesis Econometrics

Testing the effectiveness of dropout methods for inferencing in data decentralized environments

Niek Drenth (585992)

Supervisor:	dr. Hakan Akyuz
Second assessor:	dr. Xiaomeng Zhang
Date final version:	2nd July 2024

Abstract

In decentralised data environments, data sharing could pose security risks and high transmission costs. This paper focuses on a situation with local heterogeneous datasets, and a regression neural network is trained on each network. The models are not allowed to share data or their parameters with the global model. A method is proposed to combine the local prediction of each model into a global prediction. This is done using uncertainty quantification (UQ) based on Monte Carlo dropout. Models with a high uncertainty are assigned lower weights in the final global prediction. This method outperforms the benchmarks and has low implementation costs. The original research is extended by adding a benchmark model that weighs local predictions on their distance to the mean of all local predictions. Further contributions are the addition of a weather station dataset where each local model is trained on data from a different weather station and the evaluation of 2 additional UQ methods, Gaussian dropout and DropConnect. It is found that the DropConnect method does improve the local models, however, that does not translate to better performance on the proposed method. Gaussian dropout has better performance on UQ and the local models, resulting in an improvement of the method in the original research.

1 Introduction

In 2013, 90% of the world's data was generated in the preceding 2 years (Data, 2013), signifying the growth of data collection in the world. With the rise of mobile phones and other IoT devices data has become very abundant. This aids the development of predictive models, these models require data to be shared to it for training. However, due to privacy- or data security- concerns this is not always feasible and the data is stored in separate locations. Research has been done on how to use these decentralized datasets to train the most accurate global model. Examples of this are federated learning (Zhang et al., 2021) and utilizing an ensemble of local models (Yu, Deng, Tang, Liu & Chen, 2019). Federated learning requires the parameters of the models to be shared with the global model iteratively, which can be unattractive due to high communication costs. The ensemble of local networks requires access to the local data which is not desirable in some use cases. Resulting in the following research question: *"How can a predictive model best be set up to account for decentralized data with restricted access?"* The restricted access in this case means that there is no access to the training data on the local nodes or the parameters of the local model and the amount of data transmissions between the global and local models should be limited. This question is further extended by the questions: *"How to best combine the predictions from the local nodes?"* and *"What other methods can be used for uncertainty quantification and do they work well?"* To answer these questions, Lee and Kang (2024) proposes a method not requiring local access or the transmission of parameter information. The method involves creating a local regression neural network (RNN) with dropout on each of the local nodes. The global model queries each of these local models multiple times and then makes a final prediction based on the variance of these predictions.

The contributions of this thesis are three-fold:

- Evaluating a different way of combining the local predictions by weighing local predictions based on their distance to the mean of all local predictions. This is called the global mean method.

- Testing additional dropout variants: DropConnect and Gaussian dropout.
- Evaluating the techniques on data from weather stations where each local model is trained on a unique weather station.

The global mean method works by having the final prediction be a weighted sum of the local predictions, where the weights are determined by the local model's distance to the mean of all of the local predictions. A larger distance results in a lower weight, with the furthest models being expected to have a weight close to zero.

DropConnect is a dropout method different to 'normal' dropout in that instead of the activations, the weights in the RNN are dropped out. DropConnect is a more generalized version of 'normal' dropout and thus in theory allows for better generalization. Gaussian dropout does not drop out weights entirely but adds noise generated by a Gaussian distribution to each weight in a forward pass. These methods are normally only used during training, however not in this paper as the randomness introduced is used to quantify the uncertainty of the predictions. In the replication, similar results are found as in the original paper by Lee and Kang (2024), where the proposed method has the best performance, followed by the other benchmark models utilizing uncertainty quantification (UQ). For the extensions, the distance to the global mean method has the worst performance out of all of the benchmark models. The dataset extension on the weather stations shows the quality of the methods, with the methods utilizing UQ having a considerably better performance than the other methods, in line with the results from the other datasets. Finally, the additional dropout methods both improve the local model's performance. The DropConnect dropout has a worse performance on the proposed method than the original research, in contrast to the Gaussian method which improves both the local model's performance and also the performance on the proposed method. Making it an improvement on the original research by Lee and Kang (2024)

The thesis is structured as follows. First, the Literature section explores different decentralized learning methods, ensemble methods, and uncertainty quantification methods. The Methods section outlines the proposed method and the uncertainty quantification methods based on MC, DropConnect and Gaussian dropout. It also shows the weight normalization and weight calculation methods. Then the experiments section gives some statistics on the datasets and how they are processed, after this it gives the hyperparameters, the structure of the local Regression Neural Network and how the training is performed. Following, the benchmark methods used to evaluate the proposed method are given. Then the results of the experiments are given. Finally, the conclusion and some suggestions for further research are given.

2 Literature

The paper considers a situation where data can not be shared between models. This is called decentralized learning. Decentralized learning contains two variants, Federated Learning and Ensemble Learning. Some background on these two methods is given in section 2.1. Finally, the proposed method utilizes uncertainty quantification (UQ) in the form of Monte-Carlo dropout to make the final prediction. Section 2.2 describes some UQ methods and delves deeper into the origin of dropout methods.

2.1 Decentralized learning

The literature on decentralized learning is mostly split between federated learning (FL) and ensemble learning (EL). In FL the local models iteratively share their model parameters to the global model and they are updated based on the global modal. Ensemble learning methods combine the outputs of other (base) models to make a final prediction.

2.1.1 Federated Learning

Zhao et al. (2018) show that federated learning reduces prediction accuracy by 55% per cent for highly skewed non-i.i.d. data if each node is only trained on one class. By just sharing 5%, the accuracy can be improved by 30%.

Requiring all local nodes to participate in training the global model is quite a strict assumption. Wu et al. (2023) create a new method called 'FedAMD' which works well with partial client participation. Their method is quicker and requires less data transmission than other techniques.

However, by transmitting the parameter data, the original data can be partially reconstructed, which makes FL less secure (Al-Huthaifi, Li, Huang, Gu & Li, 2023).

2.1.2 Ensemble learning

Ensemble learning methods combine the outputs of other (base) models to make a final prediction. This can be done by a static method, i.e. using the same combination every time. Dynamic methods do not take the same combination every time. They can be determined by a different model, such as a random forest or simple regression (Talukder et al., 2024). These methods are widely applied to many different fields, such as geology and healthcare (Yaghoubi, Yaghoubi, Khamees & Vakili, 2024; Yang, Lei & Zhang, 2024). However, training the second layer (the one that uses the outputs of the other models) requires access to the local data which is not feasible in situations where privacy issues can arise.

Finally Magureanu and Usher (2024) explore a method in classifying problems, where the final prediction is made by the local models forming a 'consensus'. This is done by having each local model 'vote' for the output and the models can be influenced by the votes cast. The authors do see possible applications to regression problems.

2.2 Uncertainty quantification and dropout methods

Uncertainty quantification (UQ) is commonly applied to obtain prediction intervals for, for example, engineering problems and situations where sensors may provide noisy data (Kabir, Khosravi, Hosen & Nahavandi, 2018). A few UQ methods are Bootstrapping, Bayesian, Mean-Variance estimation and Dropout (Kabir et al., 2018). Bootstrapping refers to a method where a network is repeatedly trained on different data, which is sampled with replacement from a larger pool of data. This results in every model being trained on (slightly) different data and makes the model more robust. This can then also be applied during inferencing for UQ.

Dropout was originally created to help regularize neural networks to prevent them from overfitting. This was first applied to speech and object recognition tasks and at the time has resulted in state-of-the-art results (Hinton, Srivastava, Krizhevsky, Sutskever & Salakhutdinov, 2012). The method randomly removes the activation of nodes, which prevents the model from overfitting on the data as different nodes are turned off during training. This method drops out entire nodes, other research has considered only dropping out some of the activations by setting the weight of those activations to zero, this method is called DropConnect and is a more generalized version of dropout (Wan, Zeiler, Zhang, Le Cun & Fergus, 2013). Finally, there is also Variational dropout, which is similar to the Gaussian dropout. Gaussian dropout adds noise to the weights by a Gaussian distribution with a variance that is set as a hyperparameter. Variational dropout is different in that it determines this variance while training the local model. This results in a parameter that is better than when it is set by hand (Kingma, Salimans & Welling, 2015).

3 Methods

3.1 Proposed method

Lee and Kang (2024) propose a method for training a model on decentralized data. The data is stored at local nodes and is not to be transmitted between nodes, or to a 'global' model. The method involves training a Regression Neural Network (RNN) on K local nodes, and then combining the local predictions $f_1, ..., f_K$ using a global model based on the predictive uncertainty of the local prediction using equation 1. $\omega(\mathbf{x}, f_k)$ is the weight assigned to f_k on query instance $\mathbf{x}$ and $\bar{f}_k$ is the mean of the multiple predictions generated by the Monte-Carlo (MC) dropout on local model k

$$\hat{y} = \frac{1}{\sum_{k=1}^{K} \omega(\mathbf{x}, f_k)} \sum_{k=1}^{K} \omega(\mathbf{x}, f_k) \bar{f}_k(\mathbf{x}), \tag{1}$$

Due to the definition of the weights the fraction before the sum will always sum up to one, so it can be simplified to equation 2. This is the final prediction of the proposed method.

$$\frac{1}{K} \sum_{k=1}^{K} \omega(\mathbf{x}_k; f_k) \cdot f_k \tag{2}$$

The prediction procedure starts with every local model being given query $\mathbf{x}$, then uncertainty quantification, uncertainty normalization and finally weight calculation is performed. The predictive means and their weights are combined using equation 2 to obtain the final prediction.

3.2 Uncertainty quantification

When predicting a single instance using the neural network, the model does not provide a way to determine the uncertainty for that prediction. The proposed method employs dropout to test Bayesian inference in the inference phase. Dropout is a method developed to be used in training for regularization of a neural network (Hinton et al., 2012). Nodes from the hidden layers of the network are randomly dropped with probability p.

This paper extends upon this dropout method by testing two additional methods which are outlined in section 3.5. What makes these methods attractive is their low implementation cost as they are compatible with all neural networks and don't require architectural modifications, the only extra cost is that each query has to be inferenced L times. Due to the stochastic nature, the predictions can vary. Equations 3 and 4 show how the L predictions are utilized to quantify the uncertainty.

$$\bar{f}_k(\mathbf{x}) = \frac{1}{L}\sum_{l=1}^{L} f_k^{(l)}(\mathbf{x}); \tag{3}$$

$$u(\mathbf{x}; f_k) = \frac{1}{L}\sum_{l=1}^{L}\left(f_k^{(l)}(\mathbf{x}) - \bar{f}_k(\mathbf{x})\right)^2. \tag{4}$$

Equation 3 is used to obtain the final individual prediction for local model k, equation 4 measures the variability of the L predictions and is used as the quantified predictive uncertainty of model k.

A higher predictive uncertainty indicates greater variability in the predictions. This implies that the neural network generating that prediction is less confident, and should thus be assigned a lower weight in the final prediction.

3.3 Uncertainty normalization

The uncertainty is normalized since different RNNs based on different data distributions can have different scales of uncertainty. To combat this issue, the uncertainties are normalized by using the predictive uncertainties of the validation subset (i.e. the sample variance of all predictions in the validation set). The normalized uncertainty is the predictive uncertainty divided by the mean of the uncertainties of the validation set for that local model. The mean of the uncertainties in the validation set is calculated using equation 5. Here D_k^{val} is the validation set of local model k.

$$\bar{u}_k = \frac{1}{|D_k^{\text{val}}|}\sum_{(\mathbf{x}_i, y_i)\in D_k^{\text{val}}} u(\mathbf{x}_i; f_k). \tag{5}$$

The final normalized uncertainty is obtained using equation 6

$$u'(\mathbf{x}, f_k) = \frac{u(\mathbf{x}, f_k)}{\bar{u}_k} \tag{6}$$

3.4 Weight calculation

The normalized predictive uncertainties are used to determine the weights in the proposed method. α is a hyperparameter that can be set to determine the variability of the weights. The formula used is as follows:

$$\omega(\mathbf{x}; f_k) = \exp\left(\frac{\alpha}{u'(\mathbf{x}; f_k)}\right). \tag{7}$$

Here x is the input query, f_i is the output of local node i for input x and $u^*(x; f_k)$ is the normalized predictive uncertainty of local model k on query x. α for the proposed method is set to 10. The weights are then used to generate the final prediction. This is done by taking

the weighted average of the weights and the predictions of each local prediction as shown in equation 2.

3.5 Extensions

The uncertainty quantification from Lee and Kang (2024) is extended by comparing two more dropout methods: DropConnect and Gaussian dropout. Furthermore, the selection methods are also extended by applying one new method: Distance to global mean.

3.5.1 DropConnect

DropConnect is a different way of regularizing neural networks to prevent them from overfitting. Instead of dropping out nodes, which is more commonly used in 'normal' dropout, DropConnect drops the weights instead of the entire node. This allows for more variability. Figure 1 gives a visual explanation of the difference between dropout and DropConnect. Here a 'normal', dropout and DropConnect neural network layer is shown. In the no-drop network, there is no dropout and everything functions as usual, v_i are the inputs which are connected to the nodes in the hidden layer, there we have the weight matrix W. In each node, we have the activations given by activation function $a(.)$, and the output of the hidden layer is then given by $\mathbf{r} = a(W\mathbf{v})$.

In the dropout network, a mask $m(.)$ vector is applied to the activations $a(.)$ and drops some of them out, this is done by multiplying the activation vector by the mask, which consists only of 1's and 0's, resulting in an output given by $\mathbf{r} = m * a(W\mathbf{v})$, where $*$ is the Hadamard product. This is also shown visually by the output of node 2 being removed by the mask.

Finally, the DropConnect network is different in that it drops out some weights instead of the activations, here the mask M is not a vector but a matrix of the same size as W. Again it consists of 1's and 0's. The final output is then given by $\mathbf{r} = a((M * W)\mathbf{v}$ where $*$ is again the Hadamard product.

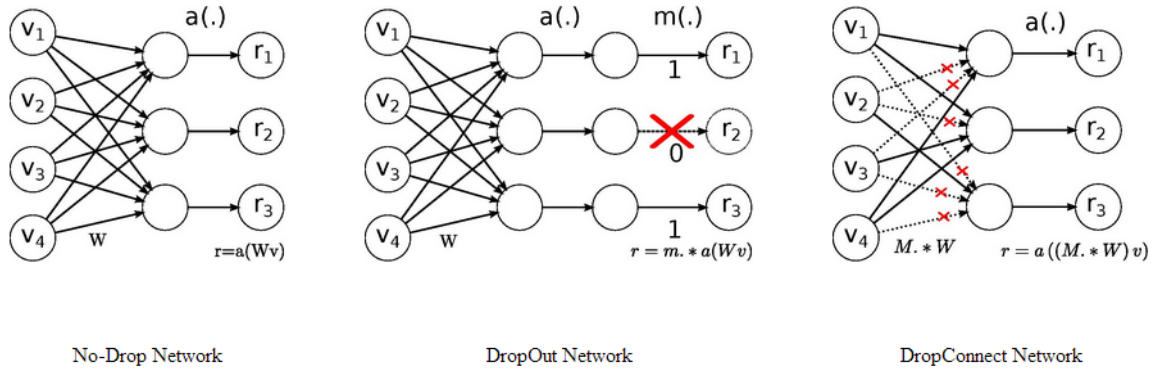

Figure 1: Explanation of DropConnect by Wan et al. (2013)

Similar to the 'normal' dropout in the proposed method, DropConnect is not turned off during inferencing to allow for uncertainty quantification which is performed as described in section 2.2

3.5.2 Gaussian Dropout

Gaussian dropout is tested as a final extension on the predictive uncertainties. Instead of completely dropping out weights or nodes (i.e. setting the weight to zero) in a forward pass, the Gaussian dropout adds noise generated from a Gaussian $N(1, \alpha)$ distribution, where $\alpha = \frac{p}{1-p}$. It adds the noise by multiplying the weights W by ϵ where $\epsilon \sim \mathcal{N}(1, \alpha)$, such that if h is the forward pass activation, the final activation becomes $\hat{h} = h * \epsilon$. In more general terms a mask M where $m_{ij} \sim \mathcal{N}(1, \alpha), \forall i, j$ is applied to weights W. This method adds noise to all of the weights, resulting in the output of the layer being obtained by $\mathbf{r} = a((M * W)\mathbf{v})$ ($*$ signifying the Hadamard product). The method was developed by Wang and Manning (2013).

3.5.3 Distance to global mean method

The final methodological extension adds a new method of combining the local predictions into a global prediction. The final prediction is the weighted sum of the predictions and the weights are generated by applying the softmax function on the distance to the global mean. The weight $\omega(x)$ is calculated using the following function, where f_i is the prediction of local model i on query x, $\bar{f}$ is the mean of all of the local predictions and the softmax function is defined as shown in equation 8

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}} \tag{8}$$

Applying this to $f_i(x)$ and $\bar{f}(x)$, we have:

$$\omega(x) = \text{Softmax}(f_i(x) - \bar{f}(x)) = \frac{e^{f_i(x) - \bar{f}(x)}}{\sum_K e^{f_k(x) - \bar{f}(x)}} \tag{9}$$

Which then makes the final prediction by using the weighted sum as given in equation 2

4 Experiments

4.1 Benchmark problems

To test the validity of the proposed method, the following public regression datasets are used: Bikesharing, Ctscan, Indoor- loc, and Telemonitoring from the UCI machine learning repository (Dua, Graff et al., 2017) and Compactiv, Cpusmall, Mv, Pole, and Puma32h from the KEEL dataset repository (Derrac, Garcia, Sanchez & Herrera, 2015). See table 1 for some descriptive data statistics. The data is separated into local subsets with heterogeneous data distributions to conform the problems to the decentralized environment explored in this paper. Lee and Kang (2024) describe the way the data is processed, first, the features are standardized to have zero mean and unit variance, subsequently, principal component analysis was performed to generate at most 10 uncorrelated features. This reduced redundant and noisy information in the original features which helps in separating the dataset into multiple local datasets (Reddy et al., 2020). After this, data standard scaling and k-means clustering are applied to scale the data and separate it into 10 different clusters. Clusters containing less than 5% of the data were not used due to lacking distinct characteristics and since neural networks do not function well under

low data circumstances. The regression neural networks (RNN) are trained on 80% of the data in each cluster. The remaining 20% is used as the test dataset to evaluate the generalization performance.

Table 1: Description of datasets in original research.

Dataset	No. instances	No. features	Domain	No. local nodes
Bikesharing	17,397	12	Bike Sharing System	9
Compactiv	8,192	21	Computer System	4
Cupsmall	8,192	12	Computer System	5
Ctscan	53,500	385	Computed Tomography	8
Indoorloc	19,937	525	Indoor Positioning System	6
Mv	40,768	10	Synthetic	10
Pole	14,998	26	Telecommunication	5
Puma32h	8,192	32	Robot Arm	10
Telemonitoring	5,875	20	Parkinson's Disease	7

4.1.1 Dataset extension

One additional dataset is considered as an extension to Lee and Kang (2024). It contains data from different weather stations during World War 2. For each weather station, there is daily data on minimum temperature, precipitation, snowfall and poor weather (i.e. storms, dust, thunder etc.). This is a smaller number of features than the other benchmark datasets, so therefore principal component analysis is not necessary to reduce the dimensionality of the data. The goal is to predict the daily maximum temperature. It should be noted that the minimum temperature is given to the model, therefore accurate results are expected. 10 weather stations are chosen, each in a different geographic location where every continent except North America and Antarctica (due to data limitations) is represented. The locations are picked by hand to make sure the locations in each continent are not too close to each other. One exception is made as 2 weather stations in Liberia are chosen, as those weather stations have a lot of data and there is still a good spread all over the world. On each weather station, a local model is trained as described in section 4.2. The requirement of each cluster or local model containing at least 5% of the total data is relaxed as the weather stations with fewer data should still have their own data generating process (DGP) in contrast to the k-means clustering performed on the other datasets, where a small cluster could indicate a DGP that is not clearly defined or separated from other clusters. Missing values in the dataset are set to zero, to ensure that there is enough data available to train the local models. After this processing, the results are obtained as shown in table 2.

Table 2: Location and amount of data of weather stations

Continent	Country	City	Datapoints
Africa	Ghana	Accra	1157
Africa	Liberia	Cape Palmas	935
Africa	Liberia	Fishermans Lake	1141
Asia	Japan	Osaka	61
Asia	Iran	Camp Parks	320
Europe	UK	Warrington	295
Europe	France	Paris	429
Europe	Iceland	Reykjavik	663
Oceania	Australia	Brisbane	415
South America	Guyana	Georgetown	1527

4.2 Experimental Settings

On each of the clusters in the benchmark problems, a regression neural network (RNN) is trained. This network uses feed-forward architecture, 3 hidden layers of 128 nodes and ReLU activation. Mean squared error (MSE) is used as the loss function. The RNN uses a dropout of 10% during training, meaning that each node has a 10% chance of not being used. By using this dropout, inferencing the same data point multiple times can result in different outputs, as different nodes get 'turned off' by the dropout. The training was done using the Adam optimizer, with a learning rate of 10^{-4}, weight decay factor of 10^{-5}, and mini-batch size of 64. If the validation loss does not improve in 20 successive epochs, or after 500 epochs, training is terminated. The hyperparameters of the proposed method are as follows. The hyperparameter α in the weighting function is set to 10. The Monte Carlo (MC) dropout during inferencing is set to 10%, equal to the dropout in training. Every query is predicted 20 times to estimate the uncertainty. Apart from these parameters, only the α parameter in the weighting functions for the dynamic selection methods has to be configured. A grid search is performed to select the best value out of $\{0.1, 1, 10\}$.

The experiment is repeated 10 times using different random seeds. The results are evaluated using RMSE and MAE on the test datasets.

The DropConnect and Gaussian dropout extensions use a dropout of $p = 0.1$. There are no other hyperparameters that have to be defined for the extensions. Lastly, in contrast to the replication, the extensions experiment is only run once due to computation time constraints.

4.3 Benchmark methods

The proposed method is compared to the following benchmarks.

- **Mean:** Takes the mean of all of the local predictions and uses that as the final prediction.
- **Median:** Takes the median of all local predictions as the final prediction
- **Dynamic selection using unnormalized predictive uncertainty (D-SEL(u):** Uses the local model that has the lowest PU as the final prediction.
- **Dynamic selection using normalized predictive uncertainty (D-SEL(u):** Similar to the previous method but uses the normalized PU to determine which local model to use

- **Dynamic ensemble using unnormalized predictive uncertainty(D-ENS(u'):** The proposed method where the normalized PU is swapped for the unnormalized PU. The final prediction is calculated by the following formula:

$$\frac{1}{N}\sum_{i=1}^{N} w(u; f_i) \cdot f_i \tag{10}$$

- **Oracle:** This model is used to obtain a reference to the possible performance without the data sharing restrictions, it gives the global model full access to the local data and is thus not feasible given the restrictions. It selects the local RNN based on the local node that follows the same data distribution as the given query $\mathbf{x}$. The final prediction is made as follows:

$$\hat{y} = \bar{f}_{k_*}(x), \quad \text{where } k_* = \underset{k\in\{1,\dots,K\}}{\operatorname{argmin}} \min_{(x_i,y_i)\in D_k} \|x - x_i\|. \tag{11}$$

4.4 Results and discussion

4.4.1 Replication results

Tables 3 and 4 show the mean absolute error (MAE) and root mean square error (RMSE) of the benchmark models and the proposed method. This is replicated from Lee and Kang (2024). In the tables, an asterisk(*) denotes a statistically significant outperformance of the superior method tested by a two-sample t-test at a 95% significance level. The sample standard deviations required for this calculation are given in the appendix.

Similar results to the original paper are found, there the proposed method had the best performance, achieving the lowest average ranking of the methods. In the replication, it is found that for the MAE the proposed method has the same average rank as the D-SEL(u') method, but outperforms it for the RMSE, where it had the lowest errors for all but one dataset. Therefore it is concluded that the proposed method does have the best results. Notably, the methods utilizing the normalized uncertainty have a considerably better performance than the ones using the unnormalized uncertainty, suggesting that this is a worthwhile improvement of the uncertainty quantification. Finally, for the Puma32h dataset, the results are similar for each method, indicating that this dataset does not consist of clusters of observations with different data-generating processes. In this case, the Oracle method also has poor performance, showing that the neural network on the local model has poor performance, this is known to hamper uncertainty quantification (Kendall & Gal, 2017).

Table 3: Performance of proposed method and baseline methods in terms of MAE (bold numbers indicating the best method and an asterisk denoting statistically significant values)

Dataset	**Oracle**	**Proposed**	**Baseline methods**			
		Method	**D-ENS**(u)	**D-SEL**(u')	**D-SEL**(u)	**Median**
Bikesharing	0.2362	**0.4472**	0.4981	0.4478	0.5007	0.5751
Compactiv	0.1025	0.1141	**0.1085**	0.1166	0.1086	0.1486
Cpusmall	0.1089	**0.2273***	0.2376	0.2356	0.2381	0.2399
Ctscan	0.0308	0.1311	0.2839	**0.1193**	0.2841	0.4917
Indoorloc	0.0454	0.0712	0.0842	**0.0670**	0.0840	0.3858
Mv	0.0154	0.0538	0.9359	**0.0329**	0.8540	0.3671
Pole	0.0705	0.1268	**0.1222**	0.1323	0.1226	0.2518
Puma32h	0.7786	0.7771	0.7942	0.7843	0.7944	**0.7734**
Telemonitoring	0.1744	0.5073	0.6566	**0.5069**	0.6607	0.6498
Average rank	–	2.0	3.11	2.0	3.67	4.11

Table 4: Performance of proposed method and baseline methods in terms of RMSE (bold numbers indicating the best method and an asterisk denoting statistically significant values)

Dataset	**Oracle**	**Proposed**	**Baseline methods**			
		Method	**D-ENS**(u)	**D-SEL**(u')	**D-SEL**(u)	**Median**
Bikesharing	0.3740	**0.7616**	0.8429	0.7969	0.8495	0.9132
Compactiv	0.1511	**0.1892***	0.2176	0.1971	0.2179	0.2060
Cpusmall	0.1511	**0.7075**	0.7677	0.7424	0.7675	0.7734
Ctscan	0.0570	**0.3597**	0.6978	0.4041	0.7006	0.7344
Indoorloc	0.0678	**0.1466**	0.1970	0.1655	0.1971	0.5564
Mv	0.0241	**0.1567**	1.3870	0.1600	1.3538	0.7689
Pole	0.1489	**0.3146**	0.3351	0.3570	0.3405	0.3934
Puma32h	0.9855	0.9881	1.0153	0.9963	1.0154	**0.9845**
Telemonitoring	0.2832	**0.7846***	1.0178	0.8550	1.0269	0.9094
Average rank	–	1.11	3.55	2.33	4.11	4.44

4.4.2 Extension results

Table 5 shows the performance of the different dropout techniques. The three methods are ranked based on their performance in the Oracle and proposed method. The Oracle method indicates the performance of the local neural network model. For the Oracle method, it is shown that the DropConnect and Gaussian dropout perform better than the normal dropout, indicating that these techniques are better at preventing the model from overfitting on the data. However, these results do not translate to the performance of the proposed method. It can be seen that the DropConnect technique performs a lot worse than the other techniques, even though it does perform well in the Oracle method. Finally, Gaussian dropout is the winner, with both a good performance in the local models and a good performance on the proposed method.

The full results are given in tables 10 and 11 in the appendix. There the performance of each dropout technique on each method of combining the predictions can be observed. Out of the 10 tested datasets, the best performing dropout method and global model combination belongs to the Gaussian dropout 5 times, normal dropout 3 times and 2 times to the DropConnect (for both MAE and RMSE), further highlighting the superior performance of the Gaussian dropout. The proposed method achieves an average rank of 1.5 and 1.2 in terms of MAE and RMSE

Table 5: Average rank in terms of MAE and RMSE

Dropout Technique	Oracle	Proposed
Normal dropout	2.7	1.6
DropConnect	1.7	2.6
Gaussian dropout	1.7	1.8

(a) Average rank in terms of MAE

Dropout Technique	Oracle	Proposed
Normal dropout	2.6	1.8
DropConnect	1.6	2.6
Gaussian dropout	1.7	1.6

(b) Average rank in terms of RMSE

respectively, further establishing it as the best global model which is in line with the results from Lee and Kang (2024).

The results of the global mean extension are also given in tables 10 and 11, here poor performance is obtained with it having the worst performance in all but one case. This suggests the information in the local models that have a prediction that is far from the mean of all of the models still contains valuable information for the final prediction, rejecting the idea that local models with erratic predictions should be assigned a lower weight in the final prediction.

Finally, tables 6 and 7 give the results on the weather stations dataset, which was an extension of the original research. Bold text indicates the best-performing global model and an asterisk indicates the best dropout technique and global model combination. This was an extension of the original research by Lee and Kang (2024). Due to the data on each local model coming from a different weather station, it is expected that a model trained on a cool climate performs poorly on a query from a warm climate. This is exactly what is observed, as the proposed method and D-Sel(u') perform very well, and the benchmark models based on the mean perform very poorly. Therefore it is also shown for this dataset that UQ helps in selecting the local predictions to use for the final prediction.

Table 6: Performance of select methods on dataset extension in terms of MAE

Dataset	**Method**	**Oracle**	**Proposed**	**D-Sel(u')**	**Glob. mean**	**Mean**
Weather	Dropout	0.0699	**0.0563***	0.0645	1.0225	0.2436
	Dropconnect	0.0682	**0.0907**	0.1147	1.0252	0.2109
	Gaussian	0.0681	**0.0582**	0.0751	1.0251	0.2389

Table 7: Performance of select methods on dataset extension in terms of RMSE

Dataset	**Method**	**Oracle**	**Proposed**	**D-Sel(u')**	**Glob. mean**	**Mean**
Weather	Dropout	0.0130	0.0261	**0.0220***	0.8746	0.2148
	Dropconnect	0.0098	0.0333	**0.0263**	0.8772	0.1872
	Gaussian	0.0097	0.0283	**0.0255**	0.8769	0.2148

5 Conclusion

In this thesis, the research from Lee and Kang (2024) is replicated and extended. Similar results to the original paper are found in the replication with the proposed method having the best performance, followed by the benchmark model using the normalized predictive uncertainty (D-sel(u')). It is shown that utilizing uncertainty quantification (UQ) and normalizing it is a good way of combining local predictions into a global prediction for situations with restricted data.

The extensions evaluate two different UQ methods, DropConnect and Gaussian dropout, an additional dataset and a new benchmark method called the global mean. These extensions do not violate any of the data restrictions imposed by the original paper. It is shown that the Gaussian dropout method is a better technique to be used for UQ as it has a better performance than the 'normal' dropout technique applied in the original paper. The DropConnect technique has better performance than 'normal' dropout on the local models (Oracle method) but has worse performance on the proposed method.

Furthermore, the quality of the UQ is highlighted by the results of the weather stations dataset. In the situation of local models trained on very different local data, the global models utilizing this UQ perform considerably better than the models not utilizing it.

Finally, a different way of combining the local predictions, which weighs each local prediction according to their distance to the mean of all local models, has a poor performance, being outperformed by all other methods that were tested.

For further research, Lee and Kang (2024) suggest exploring specific situations where the model does not perform well, such as low data and high noise environments. This is something that is also observed in this thesis, as there is some variation in which global model has the best performance. It could be analyzed what model performs well under what circumstances and if it is possible to decide on what global model to use based on the dataset that is being tested.

Furthermore, it would be illuminating to see how often the dynamic selection benchmark method picks the right local model (the model the predicted query is originating from) and how often the proposed method assigns a considerable weight the the correct model. This would highlight the contribution of the method and could provide valuable insight into where the methods may be improved. Wang and Manning (2013) have researched a dropout method that is not directly applied at the start of model training but introduced as the model is being trained. They have shown a better generalization performance and it would be interesting to see how this method performs when applied as the UQ method. Lastly, even more UQ methods could be tested, this could be done by trying additional dropout methods or modifying existing ones to better perform at inferencing. Particularly there is one paper by Mobiny et al. (2021) which suggests a different way of utilizing the DropConnect technique for inferencing which could also be applied in the restricted data situation.

References

Al-Huthaifi, R., Li, T., Huang, W., Gu, J. & Li, C. (2023, 03). Federated learning in smart cities: Privacy and security survey. *Information Sciences*, *632*. doi: 10.1016/j.ins.2023.03.033

Data, B. (2013). for better or worse: 90% of world's data generated over last two years. *SCIENCE DAILY, May*, *22*(3).

Derrac, J., Garcia, S., Sanchez, L. & Herrera, F. (2015). Keel data-mining software tool: Data set repository, integration of algorithms and experimental analysis framework. *J. Mult. Valued Logic Soft Comput*, *17*, 255–287.

Dua, D., Graff, C. et al. (2017). Uci machine learning repository.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.

Kabir, H. D., Khosravi, A., Hosen, M. A. & Nahavandi, S. (2018). Neural network-based uncertainty quantification: A survey of methodologies and applications. *IEEE access*, *6*, 36218–36234.

Kendall, A. & Gal, Y. (2017). What uncertainties do we need in bayesian deep learning for computer vision? *Advances in neural information processing systems*, *30*.

Kingma, D. P., Salimans, T. & Welling, M. (2015). Variational dropout and the local reparameterization trick. *Advances in neural information processing systems*, *28*.

Lee, Y. & Kang, S. (2024). Dynamic ensemble of regression neural networks based on predictive uncertainty. *Computers & Industrial Engineering*, 110011.

Magureanu, H. & Usher, N. (2024). Consensus learning: A novel decentralised ensemble learning paradigm. *arXiv preprint arXiv:2402.16157*.

Mobiny, A., Yuan, P., Moulik, S. K., Garg, N., Wu, C. C. & Van Nguyen, H. (2021). Dropconnect is effective in modeling uncertainty of bayesian deep networks. *Scientific reports*, *11*(1), 5458.

Reddy, G. T., Reddy, M. P. K., Lakshmanna, K., Kaluri, R., Rajput, D. S., Srivastava, G. & Baker, T. (2020). Analysis of dimensionality reduction techniques on big data. *Ieee Access*, *8*, 54776–54788.

Talukder, M. A., Islam, M. M., Uddin, M. A., Hasan, K. F., Sharmin, S., Alyami, S. A. & Moni, M. A. (2024). Machine learning-based network intrusion detection for big and imbalanced data using oversampling, stacking feature embedding and feature extraction. *Journal of Big Data*, *11*(1), 33.

Wan, L., Zeiler, M., Zhang, S., Le Cun, Y. & Fergus, R. (2013). Regularization of neural networks using dropconnect. , 1058–1066.

Wang, S. & Manning, C. (2013, 17–19 Jun). Fast dropout training. , *28*(2), 118–126. Retrieved from `https://proceedings.mlr.press/v28/wang13a.html`

Wu, F., Guo, S., Qu, Z., He, S., Liu, Z. & Gao, J. (2023). Anchor sampling for federated learning with partial client participation. In *International conference on machine learning* (pp. 37379–37416).

Yaghoubi, E., Yaghoubi, E., Khamees, A. & Vakili, A. H. (2024). A systematic review and meta-analysis of artificial neural network, machine learning, deep learning, and ensemble learning

approaches in field of geotechnical engineering. *Neural Computing and Applications*, 1–45.

Yang, J., Lei, X. & Zhang, F. (2024). Identification of circrna-disease associations via multi-model fusion and ensemble learning. *Journal of Cellular and Molecular Medicine*, *28*(7), e18180.

Yu, Y., Deng, J., Tang, Y., Liu, J. & Chen, W. (2019). Decentralized ensemble learning based on sample exchange among multiple agents. In *Proceedings of the 2019 acm international symposium on blockchain and secure critical infrastructure* (pp. 57–66).

Zhang, C., Xie, Y., Bai, H., Yu, B., Li, W. & Gao, Y. (2021). A survey on federated learning. *Knowledge-Based Systems*, *216*, 106775.

Zhao, Y., Li, M., Lai, L., Suda, N., Civin, D. & Chandra, V. (2018). Federated learning with non-iid data. *arXiv preprint arXiv:1806.00582*.

6 Appendix

6.1 Standard deviations of replication results

Table 8: Standard deviations of RMSE

Dataset	**Oracle**	**Proposed**	**Baseline methods**			
		Method	**D-ENS**(u)	**D-SEL**(u')	**D-SEL**(u)	**Median**
Bikesharing	0.2200	0.1395	0.1552	0.1386	0.1542	0.1122
Compactiv	0.0042	0.0081	0.0078	0.0194	0.0082	0.0120
Cpusmall	0.0027	0.0320	0.0230	0.0253	0.0230	0.0163
Ctscan	0.0029	0.0577	0.0302	0.0689	0.0300	0.0193
Indoorloc	0.0014	0.0213	0.0322	0.0274	0.0322	0.0694
Mv	0.0011	0.0304	0.0456	0.0433	0.0452	0.0483
Pole	0.0045	0.0328	0.0454	0.0368	0.0448	0.0278
Puma32h	0.0021	0.0033	0.0092	0.0047	0.0092	0.0021
Telemonitoring	0.0047	0.0404	0.0241	0.0522	0.0254	0.0365

Table 9: Standard deviations of MAE

Dataset	**Oracle**	**Proposed**	**Baseline methods**			
		Method	**D-ENS**(u)	**D-SEL**(u')	**D-SEL**(u)	**Median**
Bikesharing	0.1780	0.1320	0.1340	0.1450	0.1337	0.0947
Compactiv	0.0010	0.0035	0.0015	0.0037	0.0015	0.0131
Cpusmall	0.0014	0.0077	0.0064	0.0072	0.0065	0.0028
Ctscan	0.0013	0.0241	0.0180	0.0251	0.0179	0.0151
Indoorloc	0.0008	0.0059	0.0129	0.0057	0.0128	0.0345
Mv	0.0005	0.0108	0.0614	0.0097	0.0574	0.0272
Pole	0.0033	0.0112	0.0147	0.0118	0.0147	0.0200
Puma32h	0.0018	0.0026	0.0088	0.0037	0.0089	0.0018
Telemonitoring	0.0041	0.0324	0.0111	0.0429	0.0117	0.0245

6.2 Results of dropout extension

Table 10: Comparison of dropout method's performance on baseline and proposed methods in terms of MAE. (Asterisk indicating best overall method and bold the best-performing method per dropout technique)

Dataset	Method	Oracle	Proposed	D-Sel(u')	Glob. mean	Mean
Bikesharing	Dropout	0.2362	**0.4472**	0.4478	0.8587	0.5357
	Dropconnect	0.1768	**0.4970**	0.5083	0.7198	0.6113
	Gaussian	0.2123	**0.3981***	0.3990	0.9540	1.3240
Compactiv	Dropout	0.1025	**0.1141***	0.1166	0.4320	0.1491
	Dropconnect	0.1008	**0.1155**	0.1168	0.4289	0.1841
	Gaussian	0.1012	**0.1192**	0.1250	0.4303	0.1596
Cpusmall	Dropout	0.1089	**0.1141***	0.1166	0.5136	0.2420
	Dropconnect	0.1057	0.2221	0.2289	0.5120	**0.2154**
	Gaussian	0.1083	**0.2222**	0.2332	0.4994	0.2327
Ctscan	Dropout	0.0308	0.1311	**0.1193**	0.7032	0.4665
	Dropconnect	0.0316	0.1600	**0.1513**	0.7283	0.5462
	Gaussian	0.0235	**0.0928***	0.0734	0.7065	0.5595
Indoorloc	Dropout	0.0454	0.0712	**0.0670**	0.8333	0.3561
	Dropconnect	0.0455	**0.1069**	0.1070	0.8742	0.7811
	Gaussian	0.0433	0.0642	**0.0590***	1.0320	1.1240
Mv	Dropout	0.0154	0.0538	**0.0329**	0.7283	0.3789
	Dropconnect	0.0087	0.0601	**0.0373**	0.5410	0.5656
	Gaussian	0.0010	0.0435	**0.0245***	0.7013	0.6006
Pole	Dropout	0.0705	**0.1268**	0.1323	0.6888	0.2264
	Dropconnect	0.0584	**0.1151***	0.1217	0.5656	0.5295
	Gaussian	0.0615	**0.1503**	0.1543	0.5888	0.4860
Puma32h	Dropout	0.7786	0.7771	0.7843	0.7885	**0.7731**
	Dropconnect	0.7801	0.7888	0.8020	0.7885	**0.7720***
	Gaussian	0.7809	0.7792	0.7860	0.7880	**0.7729**
Telemonitoring	Dropout	0.1744	0.5073	**0.5069**	0.7717	0.6655
	Dropconnect	0.1179	**0.5504**	0.5758	0.8450	0.8318
	Gaussian	0.1543	0.4081	**0.3972***	0.8245	0.7653
Weather	Dropout	0.0130	0.0261	**0.0220***	0.8746	0.2148
	Dropconnect	0.0098	0.0333	**0.0263**	0.8772	0.1872
	Gaussian	0.0097	0.0283	**0.0255**	0.8769	0.2148

Table 11: Comparison of dropout method's performance on baseline and proposed methods in terms of RMSE. (Asterisk indicating best overall method and bold the best-performing method per dropout technique)

Dataset	**Method**	**Oracle**	**Proposed**	**D-Sel(u')**	**Glob. mean**	**Mean**
Bikesharing	Dropout	0.3740	**0.7616**	0.7969	1.1675	0.8404
	Dropconnect	0.2883	**0.8037**	0.8380	0.9537	0.8628
	Gaussian	0.3446	**0.6681***	0.7100	1.2743	1.5544
Compactiv	Dropout	0.1511	**0.1892**	0.1971	0.5105	0.2038
	Dropconnect	0.1436	0.1752	**0.1672***	0.5097	0.2303
	Gaussian	0.1532	**0.1939**	0.2127	0.5105	0.2146
Cpusmall	Dropout	0.1511	**0.7075**	0.7424	0.8950	0.7946
	Dropconnect	0.1479	0.7017	0.7455	0.8637	**0.6257***
	Gaussian	0.1508	0.7014	0.7625	0.8643	**0.6361**
Ctscan	Dropout	0.0570	**0.3597**	0.4041	0.8641	0.6960
	Dropconnect	0.0536	**0.3990**	0.4561	0.8967	0.7004
	Gaussian	0.0477	**0.2628***	0.2951	0.8757	0.7314
Indoorloc	Dropout	0.0678	**0.1466**	0.1655	0.9528	0.5035
	Dropconnect	0.0694	**0.2524**	0.2877	0.9839	0.8526
	Gaussian	0.0668	**0.1286***	0.1496	1.2391	1.4568
Mv	Dropout	0.0241	**0.1567**	0.1600	0.9165	0.7619
	Dropconnect	0.0140	**0.1684**	0.1942	0.7707	0.7942
	Gaussian	0.0154	**0.1159***	0.1268	0.8716	0.7965
Pole	Dropout	0.1489	**0.3146***	0.3570	0.8899	0.3599
	Dropconnect	0.1372	**0.3434**	0.3678	0.7864	0.6676
	Gaussian	0.1360	**0.3902**	0.4293	0.8049	0.6009
Puma32h	Dropout	0.9855	0.9881	0.9963	1.0146	**0.9841***
	Dropconnect	0.9863	0.9979	1.0114	1.0138	**0.9855**
	Gaussian	0.9874	0.9907	0.9976	1.0136	**0.9843**
Telemonitoring	Dropout	0.2832	**0.7846**	0.8550	0.9906	0.9242
	Dropconnect	0.2070	**0.8784**	1.0152	1.0863	1.0640
	Gaussian	0.2691	**0.6480***	0.7090	1.0723	0.9824
Weather*	Dropout	0.0699	**0.0563***	0.0645	1.0225	0.2436
	Dropconnect	0.0682	**0.0907**	0.1147	1.0252	0.2109
	Gaussian	0.0681	**0.0582**	0.0751	1.0251	0.2389

6.3 Code description

The code used was programmed is written in Python 3. The most important package used is PyTorch which has a lot of built-in functions for training and evaluating (regression) neural networks. The replication code was kindly made available by Lee and Kang (2024). This code was run to obtain the replication results. This took about 24 hours of computation time. Running is done by first initializing the models by running the Utils.py file and then running the Run.py file. The run.py file saves the results in a CSV file in the main folder and the model parameters in the model_parameters folder, each dataset's parameters are stored in a subfolder of model_parameters where the first dataset has folder '1', the second '2' etc. The folders may have to be created by hand in case of errors.

The extensions are built upon the provided code. The utils.py file is modified to change the neural networks to the desired structure. In particular by making the 'WeightDropoutLayer' ob-

ject which implements the dropConnect method and by defining the Gaussian dropout method. The final neural network is then obtained by initializing the neural network (RegNN in the code) with the layers specified using these layer objects.

Extension.py contains the code which runs the model with the modified neural networks. The code is mostly the same as Run.py, but it only runs one time instead of 10 times due to the computation time constraints. The file also implements the global mean method. Running the code saves CSV files for both the MAE and RMSE values in the 'extensioncsv' folder. In case of any errors, this folder may have to be made by hand.

Datasetextension.py runs the experiment with the extension weather station dataset and again very similar to run.py. Here the data preprocessing is changed to remove the k-means clustering and to give each weather station a unique local neural network. The results are obtained in the same manner as is done in extension.py. The data of this extension is stored in the data folder next to the datasets originally tested by Lee and Kang (2024).

Finally, the last file tablegen.py contains very simple code that reads the last 10 lines from the csv files generated by run.py. and outputs the means and standard deviation in a manner that is easily translated to LaTeX code, to ease the process of transferring the data to the tables.