Generative Neural Networks for modelling the inverse function of an Artificial Neural Network .

Mohammad Aaftab V[1] under the supervision of Mansi Sharma[2]

Abstract— **This paper proposes a neural Network Architecture for modelling the Inverse function of a Neural Network or for even modelling the inverse function of a Neural Network partially.**

Where a Regular Neural Network is able to predict a target value based on given feature vectors, The Architecture proposed in this paper is able to generate a feature vector using a predetermined target value such that when it is fed to the forward Neural Network, the target value predicted will be very close to the predetermined target value, this this architecture is able to map inverse function of the function represented by a Neural Network.

This paper also proposes a way for a given Neural Network to explore a given local space for a longer period of time, thus enabling it to find local optimum of a function aside from just being able to find global optimum, this is made possible by using a class of functions this paper proposes known as Constraint Functions.

I. INTRODUCTION

Neural Networks are known to be great function approximators [1] for a given data.This paper tries to introduce a new Neural Network Architecture, that can map not just the forward function (i.e. the function from input to output), as is done in usual Artificial Neural Networks but also the inverse of the function represented by ANN i.e. the function that maps from output or target to input.

The problem of finding the inverse of a Neural Network is extremely important, in various engineering problems, especially the optimisation ones [2]–[7].

Inverse problems in imaging and computer vision are being solved with the help of Deep Convolution Neural Networks [8]–[11]. Generative Adversarial Networks [17] have also been used extensively in solving inverse problems [12]–[16].

The primary use case of the proposed architecture is to find the input feature vector that corresponds to a desirable value of target vector i.e., this architecture can generate feature vector based on a given Neural Network and given target vector value such that, when the generated feature vector is passed into the forward Neural Network, it outputs the target vector very near to the predetermined target value.

Apart from simply being able to generate a feature vector corresponding to a target value, this architecture can also generate feature vectors subject to constraints like fixed range for a few or all of the feature values or fixed value for a few or all of the feature values.

[1]Mohammad Aaftab V is a final Year B.Tech Student at IIT Madras, India. aaftaabv@gmail.com

[2]Mansi Sharma is with the Department of Electrical Engineering, IIT Madras, India mansisharma@smail.iitm.ac.in

This architecture is also able to identify to an extent, the feature importance of each feature.This is done by generating multiple feature vectors that correspond to a single target value, and the feature that has the least variation amongst the generated feature vectors is the most important feature in determining the function in that range, similarly the most varying feature has the least feature importance.

II. RELATED WORK

The problem of developing Generative Neural Networks has been explored well, especially in the recent years.Our proposed methodology is not related to or inspired from any of these works.In this section, we explore few such papers and their methods.

Chu-Teh Chen et al. [18] describes a general-purpose inverse design approach using generative inverse design networks along with active learning.This paper introduces a framework named Genertive Inverse Design Networks (GIDNs).This framework has two DNNs namely, the predictor and the generator.The weights and baises of the predictor are learned using gradient decent and backpropogation while training on a given dataset.In the designer, the weights and biases are adapted from the predictor and set as constants.Initial Gaussian distribution features are fed into the designer as inputs and the optimized design features are generated as outputs.Active learning takes place in the feedback loop, where the optimized designs or features are verified and added to the dataset and predictor is trained on the new more robust dataset. The designer inverse model in this paper optimizes the initial inputs or designs based on an objective function via gradient descent with weights and biases of all layers kept as constants and only the input layer values are learned.

This approach is similar to our approach in that we also use a pre-trained Oracle Neural Network to model the forward function and while training the Generator Neural Network, this Oracle Neural Network is constant and used to judge the correctness of a given feature vector under a given target value,but we differ from the approach in this paper, in that where we use an additional Generator Neural Network to generate features from latent data whose size is tunable.This modelling of features from latent data using an additional Neural Network means that complex inverse functions can be modelled, i.e. as the complexness of the inverse function increases, we can increase the size of the generator Neural Network along with the size of latent data, all these additional trainable weights and biases help in getting a better approximation of the complex inverse function.

Ziqi Yang et al [19] published a paper on Neural Network Inversion in Adversarial Settings.This paper mainly talks about generating the training data that was used in training a Neural network using a black box access to the forward Neural Network.Our paper also uses a black box access to the forward Neural Network, but the use case, training procedure and Architecture are completely different. While this paper mainly focuses on generating the Exact training data that was used while training the forward Neural Network, our paper shows how to generate data corresponding to any desired output.

Tahersima et al [20] have published a paper describing a method to design Integrated Photonic Power Splitters using Deep Neural Networks to model the inverse relation.This paper deals along the same lines as ours, in that it also tries to use ANNs to find feature vector that corresponds to a given target value, but it is different in that it is built for a very specific application of designing Integrated Photonic power Splitters whereas, our paper gives an architecture that can be generalized and used to any data.

This paper models inverse function by simply flipping the input and target variables and training it as a general Neural Network, which does not work for all data especially ones where a lot of features determine only a small number of target variables.

Our paper overcomes this problem by having a different procedure and Neural Network Architecture to model the inverse function than the general procedure and architecture used by ANNs today to model forward function for a given data.

Zang et al [21]. wrote a paper describing Multivalued Neural Network for Inverse Modeling and application to Microwave filters.This paper also attempts to find the feature vector corresponding to a required target value and it focuses mainly on the existence of non-unique feature vectors corresponding to a given target value.This paper overcomes this problem partly, by generating multiple sets of feature vectors at a time for a given target vector.

III. Proposed Network

This paper is not directly related to any of the above works and attempts to propose an architecture that is entirely different to those of it's predecessors.

This paper proposes a 2 tier Neural Network Architecture, for finding the inverse of a Neural Network. While traditionally, inverse of a Neural Network was obtained by simply reversing the input and output features of a data and training the Neural Network. This method allows for modelling the whole inverse function at a time, but the problem arises when the forward data consists of a lot of input features determining a few output features.

In such a data, to model inverse function using ANN, we should be able to produce many output features based solely on a few input features and directly training a Neural Network with the new large number of output features will not result in a good model because there is not enough information contained in a few input features.

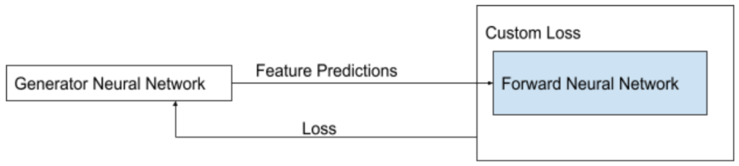
Hence, this paper tries to mitigate the above mentioned issue by using random variables as inputs to the generator network, this way if the inverse function requires more number of outputs (i.e. the forward Neural Network has a large number of features) then we can counteract the lack of input variables in the traditional approach by having large number of random features as input, which act as latent data and then training the Generator neural network which now contains enough complexity right from the input layer to be able to model the inverse ANN function effectively.

The Architecture proposed is as follows, given a trained Neural Network, we will create a Generator Neural Network that takes a specific number of input features (this number is a hyper-parameter and can be increased or decreased to fit the complexity of the Inverse function) and has some hidden layers and the number of output neurons will be equal to the number of features of the trained Neural Network, with each neuron corresponding to one input feature.

The training of Generator Neural Network is done via gradient descent to minimize a custom loss function.The custom loss function will be calculated as follows, once we get an output from generator Neural Network, we will pass it into the forward Network and get the output from forward Neural Network.The custom loss value is the mean squared error between the obtained output and a predetermined output of our choice.By minimizing this custom loss function, we can get as output from the Generator Neural Network, the exact values of the input features such that when they are modelled using the forward Neural Network, the output will be very close to our desired value.

Thus, we have successfully arrived at the values of input features corresponding to a desired value of output features.We can generate multiple such input feature vectors at a time by passing many rows into the Generator Neural Network and optimising all of them at the same time. Thus, we have found multiple input feature vectors that all correspond to a same or similar value of a output vector.The input feature that has least variation among these multiple solutions is the feature that has the most importance and similarly the feature that has the highest variation is the feature of least importance.

The Complete Training Process is show below:



What is possible with this Architecture and Custom Loss Function:

- First and foremost, we can model the inverse function to a given Neural Network by trying to produce the values of feature vectors that correspond to a required output in the Forward Neural Network.
- We can also model only a part of the inverse function

of an ANN, i.e. we can fix one or more input feature values and then can train the generator to produce only the remaining input feature vectors.In this case, we will have the number of neurons in the generator output layer equal to the number of input features that are not fixed and in the custom loss function, to get the output from forward neural network, we will add the fixed values of input features along with the predicted input features from Generator Neural Network and send it into the forward Neural Network and then minimize this loss function.

- We can also constraint the values of one or more generated feature values.This is accomplished by using a family of functions introduced in this paper known as Constraint Functions.

A. *Constraint Functions*

Activation functions in Neural Networks such as ReLu, Tanh serve various purposes ranging from introducing non-linearity to the modelling process to avoiding gradient vanishing etc.

This paper proposes one such family of functions named, Constraint functions, which serve the purpose of constraining the search/exploration space in a neural network. These are similar to activation functions in that they are applied to each output from a neuron, but different in that they are conditional and they help the Neural Network minimize the loss while still exploring in a prescribed space.

These constraint functions help us maintain the outputs in a certain range.They are extremely modifiable and have to be made based on the neural network and data. One such example function is described in the Experiments sections below.

IV. Experiments and Results

A. *DataSet Preparation*

A custom Dataset was made out of a polynomial exponential function of 4 variables. The function is given below.

$$y = 9x_1^{0.87} + 8.97x_1^{0.02} + 0.876x_3^{0.12} + 2.9876x_4^{0.987}$$

10,000 points were sampled from this function randomly as training data and 1000 points were randomly sampled as testing data.

B. *Experimental Settings*

For all Experiments, this data was used to train a forward function mapping Neural Network.After this various Generator Neural Network(and an appropriate Custom Loss function) were used to carry out various experiments.

The Learning process for Generator in all experiments was same and was as described above.

C. *Experiments and Results*

1) Modelling the Inverse Function: The Sole purpose of this experiment was to simply find feature vectors corresponding to a required output, in this case the output was 1900.i.e., the goal of this experiment was to find the feature vector values (i.e. the x1,x2,x3,x4 values to be put in in the above equation) such that when they are modelled using the above mentioned function, the output should be very close to 1900.Exactly getting 1900 is not guaranteed because of the error in modelling the forward Neural Network for the above equation and also because that function may never reach 1900 for any value by it's own nature, hence the Generator Network tries to only minimize the error between predicted output and required one, it cannot make the error zero altogether.

After training, the Generator Function using Custom Loss Function as described already, the Generator Neural Network was able to generate values of x1,x2,x3,x4 such that when modelled using the above equation, we get an output of 1935.3740 .

The values generated are shown below:

$x_1 = 269.9371, x_2 = 0.000, x_3 = 148.5608, x_4 = 273.7796$.

Hence, this Experiment was able to arrive at the values of the feature vectors that correspond to an output of our choice.

2) Finding the feature vector with one feature set to constant: This experiment was designed to show that this method can navigate a subset of the inverse function as well.Here, the Generator Network was tasked to generate x1,x2,x3 and x4 was set as constant as a value of 10.Once the Generator Neural Network has generated x1,x2,x3, they are sent into a custom loss function where these values along with 10 are sent into the Forward Neural Network to get the predicted output and the final loss is calculated as a Mean Squared Error between predicted output and the required output.The Generator Neural Network then tries to minimize this error via gradient descent.

The results are as follows: The Generator Neural Network was able to produce a feature vector such that the corresponding output is 2040.8271 and the feature values generated are:

$x_1 = 4.9872e+02, x_2 = 1.3327e-01, x_3 = 2.0957e+02, x_4 = 10.00$

These experiments were done as a proof of concept of the capabilities of this method, the actual results like how close to 1900 we can get and so on, can be improved by using better generator architectures, novel optimizers and training for longer and so on.

3) Finding the feature vector with 3 features desired to be set in the range of 1 to 100 and one feature set to constant.: This experiment was designed to show that this method can navigate a subset of the inverse function as well as explore in a prescribed subspace.Here, the Generator Network was tasked to generate x1,x2,x3 and x4 was set as constant as a value of 100 and the remaining 3 were desired to be set between 0 and 100 such that they output a value as close to 1900 as possible.

The theoretical maximum value achievable for this function with features within the range of 0 to 100 is 787.34, still the Generator was asked to go as close to 1900 as possible to arrive at the maximum value possible within this range.

This Constraining of the feature values is made possible by a family of functions this paper introduces known as Constraint Functions. Constraint functions follow the general template as shown below:

The Constraint Function template:

$$\begin{cases} x/c_1 & UpperBound \leq x \\ x * c_2 & x \leq LowerBound \end{cases}$$

Where c_1 and c_2 are constants that have to be decided after checking the general range in which the Generator Neural Network is trying to optimize, i.e. c_1 has to set so that when the output from Generator without constraint functions is divided by c_1, it will approximately fall in the range we require.

For, instance, in this experiment, the Generator, before using Constraint Functions was giving outputs in the range of 200-500, hence to make these outputs finally fall approximately in the range of 1 to 10, c_1 was initially intuitively set as 2, but upon noticing the generator output exploration range further and tweaking the value a few times, the final c_1 used was, and similarly c_2 was experimentally arrived at as , and the final output was

This function was arrived at experimentally as discussed above and is shown below.

The Constraint Function:

$$\begin{cases} x/10 & 10 \leq x \\ x * 50 & x \leq 1 \end{cases}$$

This constraint function can be applied to any layer neurons (or all layer neurons, if necessary to maintain the gradients almost uniform across layers) and the hyper parameters of the neural networks and the constraint functions can be fine-tuned there after.However, in this experiment the constraint function was used only on the output layer neurons.

Once the Generator Neural Network has generated x1,x2,x3,in the required range, they are sent into a custom loss function where these values along with 100 are sent into the Forward Neural Network to get the predicted output and the final loss is calculated as a Mean Squared Error between predicted output and the required output.The Generator Neural Network then tries to minimize this error via gradient descent.

All the constraint function can do is make the Generative Neural Network search in the required space for a longer period of time, thus enabling us to explore our required space deeply.This essentially implies that for a function where global minimum lies in a range outside prescribed space, the Generator will eventually learn to optimise itself and go out of the constrained range and into the optimal global minimisation zone of custom loss function.Thus, by using constraint functions and stopping optimisation after features go beyond prescribed range, we can arrive upon at local optimal values.

The results are as follows: The Generator Neural Network was able to produce a feature vector such that the corresponding output is 525.5496, which can be interpreted to be close to the maximum value of the given function attainable such that the features all range in between 0 and 100. And the feature values generated are:

$x_1 = 100.1898, x_2 = 0.0000, x_3 = 9.7433, x_4 = 100.00$

The theoretical maximum value achievable for this function with features within the range of 0 to 100 is 787.34, which is not very far from the value obtained in results.

Thus this architecture can be used to impose constraints on one or more of the features and arrive upon at the corresponding local optima of the function in the feature space bounded by the constraints on the features themselves.

V. Conclusions

This paper proposes a new, powerful Neural Network Architecture,to generate feature vectors corresponding to a required target variable, i.e. model inverse of Neural Network.The concepts of Custom Loss function, Constraint Function also being proposed in this paper, are the pillars to this process.

Apart from just being able to model the inverse function, this architecture can also model partial inverse function and it can also find the local optimum and corresponding features in a given space using the constraint function.

VI. Further Work

A. Synthetic Dataset Generation:

So far, we have tried to generate feature vectors that are able to model a particular fixed target value. A Synthetic Dataset (which has properties very near to the original dataset) can be made by generating feature vectors that model the target value distribution similar to the distribution in the original dataset. This can be achieved by modifying the custom loss function. This new custom loss function should take a generated sequence of feature vectors as input, model that using the Forward Neural Network, which gives out the predicted target value(s). Now, the loss function should return the mean squared error between the predicted target value(s) and a random target value(s) present in the original dataset. We can as an alternative, return the largest Mean squared error among the mean squared errors of predicted target value(s) and every target value(s) in the original dataset.

Though this synthetic dataset can be very accurate, preliminary experiments do not show any increase in the Forward Neural Network performance when the synthetic dataset is added to original dataset and a new Forward Neural Network is trained.So, Synthetic dataset generation is an application of the architecture described in this paper to be worked more upon.

Synthetic Dataset Generation using the described architecture can be really helpful in classification tasks where a lot of class imbalance is present and we want to create new data that belong to a class with less data.

APPENDIX

Appendixes should appear before the acknowledgment.

ACKNOWLEDGMENT

The preferred spelling of the word ÒacknowledgmentÓ in America is without an ÒeÓ after the ÒgÓ. Avoid the stilted expression, ÒOne of us (R. B. G.) thanks . . .Ó Instead, try ÒR. B. G. thanksÓ. Put sponsor acknowledgments in the unnumbered footnote on the first page.

References are important to the reader; therefore, each citation must be complete and correct. If at all possible, references should be commonly available publications.

REFERENCES

[1] Kurt Hornik, Maxwell Stinchcombe, Halbert White, Multilayer feed-forward networks are universal approximators, Neural Networks, Volume 2, Issue 5, 1989, Pages 359-366, ISSN 0893-6080

[2] Oscar May, Luis J. Ricalde, Bassam Ali, Eduardo Ordoñez López, Eduardo Venegas-Reyes and Oscar A. Jaramillo (October 19th 2016). Neural Network Inverse Modeling for Optimization, Artificial Neural Networks - Models and Applications, Joao Luis G. Rosa, IntechOpen, DOI: 10.5772/63678. Available from: https://www.intechopen.com/books/artificial-neural-networks-models-and-applications/neural-network-inverse-modeling-for-optimization

[3] ”J.A. Hernández, D. Colorado, O. Cortés-Aburto, Y. El Hamzaoui, V. Velazquez, B. Alonso, Inverse neural network for optimal performance in polygeneration systems, Applied Thermal Engineering, Volume 50, Issue 2, 2013, Pages 1399-1406, ISSN 1359-4311.

[4] Pandian M.E. Phd, Jaganatha. (2015). Artificial neural network based inverse model control of a nonlinear process. 10.1109/IC4.2015.7375581.

[5] Nour Hattab, Mikael Motelica-Heino. Application of an inverse neural network model for the identification of optimal amendment to reduce Copper toxicity in phytoremediated contaminated soils. Journal of Geochemical Exploration, Elsevier, 2014, 136, pp.14-23. ff10.1016/j.gexplo.2013.09.002ff. ffinsu-00869148f

[6] J. Wang, ÒFundamentals of erbium-doped fiber amplifiers arrays (Periodical styleÑSubmitted for publication),Ó IEEE J. Quantum Electron., submitted for publication.

[7] Krasnopolsky V.M. (2009) Neural Network Applications to Solve Forward and Inverse Problems in Atmospheric and Oceanic Satellite Remote Sensing. In: Haupt S.E., Pasini A., Marzban C. (eds) Artificial Intelligence Methods in the Environmental Sciences. Springer, Dordrecht.

[8] arXiv:1710.04011 [eess.IV]

[9] arXiv:2005.06001v1 [eess.IV]

[10] Housen Li et al 2020 Inverse Problems 36 065005

[11] Wang, F., Eljarrat, A., Müller, J. et al. Multi-resolution convolutional neural networks for inverse problems. Sci Rep 10, 5730 (2020). https://doi.org/10.1038/s41598-020-62484-z

[12] Hailin Ren, Pinhas Ben-Tzvi, Learning inverse kinematics and dynamics of a robotic manipulator using generative adversarial networks, Robotics and Autonomous Systems, Volume 124, 2020, 103386, ISSN 0921-8890.

[13] Dan, Y., Zhao, Y., Li, X. et al. Generative adversarial networks (GAN) based efficient sampling of chemical composition space for inverse design of inorganic materials. npj Comput Mater 6, 84 (2020).

[14] M. Lenninger, ‘Generative adversarial networks as integrated forward and inverse model for motor control’, Dissertation, 2017.

[15] arXiv:1611.05644 [cs.CV].

[16] arXiv:1905.11672v4 [cs.CV].

[17] arXiv:1406.2661 [stat.ML].

[18] Chen CT, Gu GX. Generative Deep Neural Networks for Inverse Materials Design Using Backpropagation and Active Learning. Adv Sci (Weinh). 2020;7(5):1902607. Published 2020 Jan 9.

[19] arXiv:1902.08552 [cs.CR].

[20] Tahersima, M.H., Kojima, K., Koike-Akino, T. et al. Deep Neural Network Inverse Design of Integrated Photonic Power Splitters. Sci Rep 9, 1368 (2019).

[21] C. Zhang, J. Jin, W. Na, Q. Zhang and M. Yu, ”Multivalued Neural Network Inverse Modeling and Applications to Microwave Filters,” in IEEE Transactions on Microwave Theory and Techniques, vol. 66, no. 8, pp. 3781-3797, Aug. 2018, doi: 10.1109/TMTT.2018.2841889.