BIRLA INSTITUTE OF TECHNOLOGY SCIENCE, PILANI

A Project report on

Modelling Neutron Star Parameters using Bayesian Neural Networks

21st Dec, 2020

Authors:

Nikhil P.S. Bisht

(2017B5A70610G)

Utsav A. Murarka

(2017B5A70854G)

Project Mentors:

Dr. Kinjal Banerjee

Dr. Tuhin Malik

Academic year 2020-2021

Acknowledgment

We would like to express our gratitude towards Dr. Kinjal Banerjee, the project mentor for his constant guidance. We would also like to thank Dr. Tuhin Malik, for his support during the entirety of the project, providing us with the relevant dataset and critically analysing the outcome of the various models we build.
Lastly, We would like to express our sincere thanks to BITS Pilani K.K. Birla Goa Campus, for providing us the opportunity to work on this project.
Thanks a lot.

Abstract:
We try to estimate Neutron Star parameters such as Neutron Star Mass, Radius, Tidal Deformability by implementing various Machine Learning (Multiple Single Target Regressors, Multi-target Regression, variational Inference) and Deep Learning Techniques (Artificial neural Network, bayesian Neural Network) which will help replace the traditional method of calculating such parameters mathematically and save computational time and complexity.

Contents

List of Figures **5**

1 Introduction **6**

2 Dataset **8**

2.1 Features of the dataset 8

2.2 Data Statistics and Feature Selection 9

3 Machine Learning Approach **13**

3.1 What is ML? 13

3.2 Models 13

3.3 The Reverse Problem 19

4 Artificial Neural Network Approach **21**

4.1 What is ANN 21

4.2 Models 21

5 Bayesian Neural Network Approach **22**

5.1 What is BNN 22

5.2 Models 23

5.2.1 Approach 1: Self Implementation of BNN 23

5.2.2 Approach 2: Implementation using Tensorflow libraries (All 7 Inputs) 26

5.3 Output 29

6 Conclusion **29**

References **30**

List of Figures

1 Correlation matrix 9
2 Case 1 pair Plot of Features 10
3 Case 2 pair Plot of Features 11
4 Case 3 pair Plot of Features 12
5 Actual v/s Predicted values, Random Forest, CASE-1 14
6 Actual v/s Predicted values, Random Forest, CASE-2+CASE-3 14
7 Multiple single target vs multi-target regression. Each target y_j is learned by a model, where inputs are shown as incoming arrows 15
8 Actual v/s Predicted values and standard deviations, Dot product + White Noise Kernel 16
9 Actual v/s Predicted values and standard deviations, Gaussian Kernel 17
10 Actual v/s Predicted values and standard deviations, Matern Kernel 18
11 Actual v/s Predicted values and standard deviations, Rational Quadratic Kernel 18
12 Actual v/s Predicted values of EOS Parameters 20
13 Loss curve for BNN with 10 nodes in 2 Hidden layers each 26
14 Actual Vs Predicted values for various outputs 26
15 Loss curve for BNN-19 27
16 Predictions for all three outputs with errorbars of 2σ 28
17 Parameters used for sampling 29

1 Introduction

The compact stars such as Neutron Stars (NSs), observed as pulsars, are believed to contain matter up to few times nuclear saturation density in its core. The NSs present one of the densest forms of matter in the observable universe. They are the ideal cosmic laboratories to shed light directly or indirectly on different theories of physics as well as on the physics beyond the standard scenario. To explain and understand the extreme properties of such stars, one needs to connect different branches of physics including low energy nuclear physics, QCD under extreme conditions, general theory of relativity (GR) etc. The internal structure of the neutron star (NS) and its properties, such as mass, radius, quadrupole deformation and moment of inertia, depends on the hydrostatic equilibrium between the inward gravitational pull of matter and the outward neutron degeneracy pressure. If we assumed the correctness of GR, to understand the internal structure of NS predominantly, one needs the theory of the behavior of matter at extreme conditions, i.e., the theory of infinite nuclear matter equation of state (EOS). The EOS is conventionally defined as energy (or pressure) as a function of density, over a wide range of densities.

The energy per nucleon at a given density $\rho = \rho_n + \rho_p$ with ρ_n and ρ_p the neutron and proton densities, respectively, and asymmetry $\delta = \left(\rho_n - \rho_p\right)/\rho$, can be decomposed, to a good approximation, into the EoS for symmetric nuclear matter $e(\rho, 0)$, and the density dependent symmetry energy coefficient $S(\rho)$:

$$e(\rho, \delta) \simeq e(\rho, 0) + S(\rho)\delta^2$$

Expanding the isoscalar contribution until fourth order and the isovector until third order we obtain for the isoscalar part $e(\rho, 0)$:

$$e(\rho, 0) = e\left(\rho_0\right) + \frac{K_0}{2}x^2 + \frac{Q_0}{6}x^3 + \mathcal{O}\left(x^4\right)$$

and for the isovector part $S(\rho)$:

$$S(\rho) = J_0 + L_0 x + \frac{K_{\mathrm{sym},0}}{2}x^2 + \mathcal{O}\left(x^3\right)$$

where $x = \frac{\rho-\rho_0}{3\rho_0}$ and $J_0 = S\left(\rho_0\right)$ is the symmetry energy at the saturation density. The incompressibility K_0, the skewness coefficient Q_0, the symmetry energy slope L_0 and its curvature $K_{\mathrm{sym},0}$ evaluated at saturation density are defined. The key nuclear matter parameters

(NMPs) of the EOS are: K_0, Q_0, J_0, L_0 and $K_{\mathrm{sym},0}$ We can construct large number of EOS database as a point in the seven dimensional space of NMPs using multivariate Gaussian distribution (MVGD) the pa— rameters being $e_0, \rho_0, K_0, Q_0, J_0, L_0,$ and $K_{\mathrm{sym}\,,0}$. Symbolically, the 'ith' EOS is written as

$$\begin{aligned}\mathrm{EOS}_i &= \left\{e_0, \rho_0, K_0, Q_0, J_0, L_0 \text{ and } K_{\mathrm{sym},0}\right\}_i \\ &\sim N(\mu, \mathbf{\Sigma})\end{aligned}$$

where μ designates the mean value of the parameters and $\mathbf{\Sigma}$ is the co-variance matrix. The diagonal elements of Σ represent the variance or the squared error for the parameter set p. The off-diagonal elements of Σ are the covariance between different parameters and measure the correlations among them. Once the values of all the seven NMPs are given, the 'ith' EOS can be calculated either in Taylor expansion mode (as discussed previously) or in mean field formalism. This EOS can be employed to calculate the various properties of neutron stars such as its maximum mass, radius and tidal deformability by solving the NS structure equation. Hence we able to calculate correlations of NMPs with neutron star properties [2]

2 Dataset

2.1 Features of the dataset

The dataset looks as follows:

e(0)	**rho(0)**	**K(0)**	**Q(0)**	**J(0)**	**L(0)**	**Ksym(0)**
-15.8689	0.1611	224.7651	238.4824	35.3659	50.6193	-110.152
-16.19	0.1613	234.2391	114.2765	36.5871	52.8665	-76.1168
-15.7564	0.1656	232.0309	190.543	34.3259	44.7315	-67.61
-16.0897	0.1635	259.2558	172.773	32.5705	57.5269	-84.8941

NS mass	**Rmax**	**R14**	**Lambda10**	**Lambda14**	**Lambda18**	**Vs**	**Qsym**
2.1004	10.2722	12.1612	2488.672	333.8504	49.0337	0.9998	423.7202
2.1769	10.6051	12.3266	2791.131	395.8141	64.0643	0.9998	180.5748
2.1513	10.4025	11.9278	2479.163	351.7705	56.3647	0.9995	385.6867
2.2118	10.8058	12.6223	3119.921	441.2771	73.6572	0.9991	236.2859

Table 1: A sample dataset (Only first 4 rows)

where the columns are as follows:

- e(0): EoS for symmetric nuclear matter
- rho(0): saturation density
- K(0): incompressibility
- Q(0): skewness coefficient
- J(0): symmetry energy at the saturation density
- L(0): symmetry energy slope
- Ksym(0): curvature of L evaluated at saturation density
- NS mass: Neutron Star Mass
- Rmax: Max NS radius
- Lambda: Tidal Deformability

2.2 Data Statistics and Feature Selection

The data has the following statistics:

	e(0)	rho(0)	K(0)	Q(0)	J(0)	L(0)	Ksym(0)
count	6651	6651	6651	6651	6651	6651	6651
mean	-16.002175	0.159942	230.384131	295.312487	32.143682	64.906808	-61.936726
std	0.193909	0.003827	14.739531	74.165665	2.792884	16.207698	69.637652
min	-16.884400	0.142900	160.648100	6.439000	23.366100	1.600100	-233.634200
25%	-16.060600	0.158800	226.717450	272.300150	30.208050	53.788950	-114.351750
50%	-16.000000	0.160000	230.000100	299.999900	32.138700	64.540400	-67.199500
75%	-15.956900	0.161100	234.959400	310.581700	34.090800	75.821250	-15.963700
max	-15.229400	0.178400	301.493200	591.237900	40.983500	118.984900	183.091900

	NS mass	Rmax	R14	Lambda10	Lambda14	Lambda18	Vs	Qsym
count	6651	6651	6651	6651	6651	6651	6651	6651
mean	2.081123	10.728979	12.740520	3301.218472	435.371914	63.351133	0.795189	294.707876
std	0.100450	0.560741	0.768753	953.949644	131.629958	25.576188	0.149925	223.345938
min	1.812600	9.121300	7.811700	124.296900	20.291100	7.232300	0.060800	-657.885400
25%	2.021550	10.349650	12.234650	2623.298550	338.138150	44.582550	0.684800	157.368050
50%	2.105200	10.704400	12.744600	3212.461200	425.727700	62.398800	0.833500	273.327100
75%	2.141100	11.114050	13.244100	3888.418500	519.801200	80.664750	0.921700	404.932050
max	2.448400	12.733000	15.992900	8612.172200	990.928900	173.354500	0.999800	1787.546600

Table 2: Data statistics

And the following Correlation matrix:

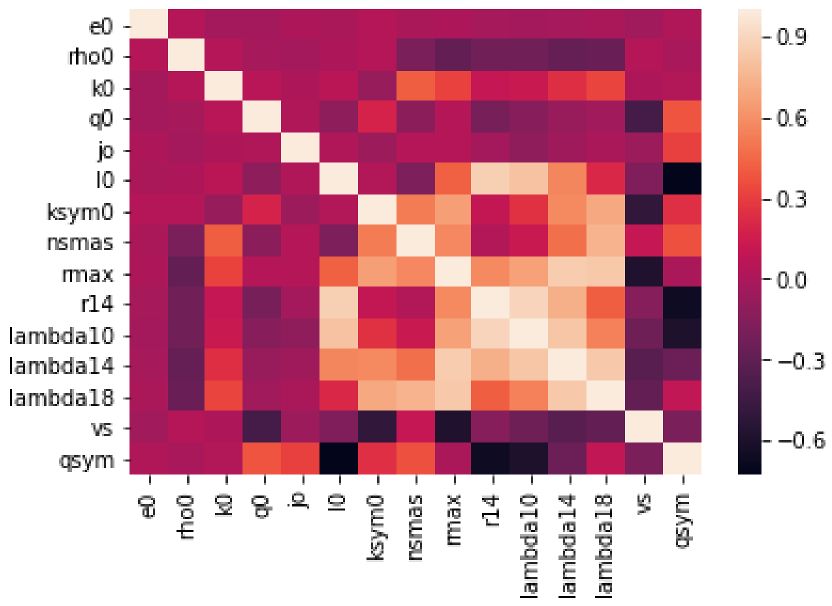


Figure 1: Correlation matrix

And the following pair Plots:

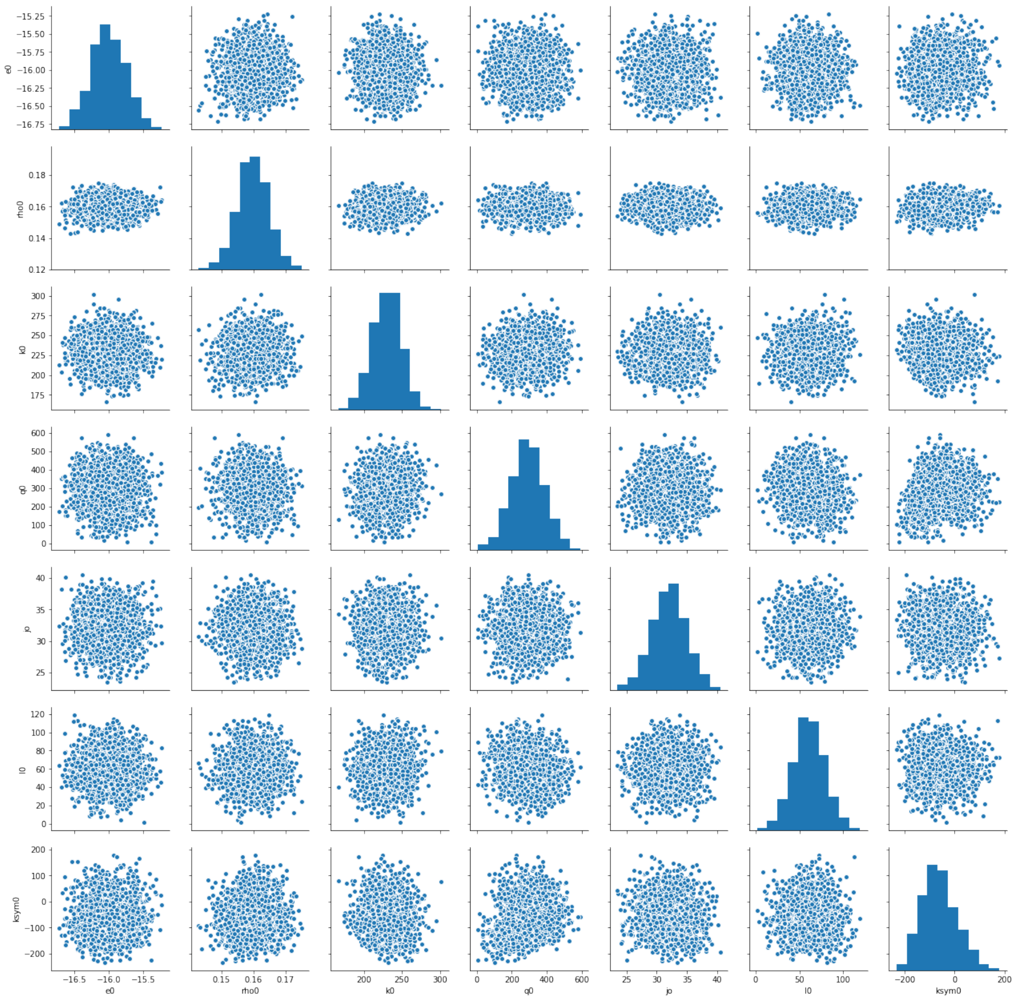


Figure 2: Case 1 pair Plot of Features

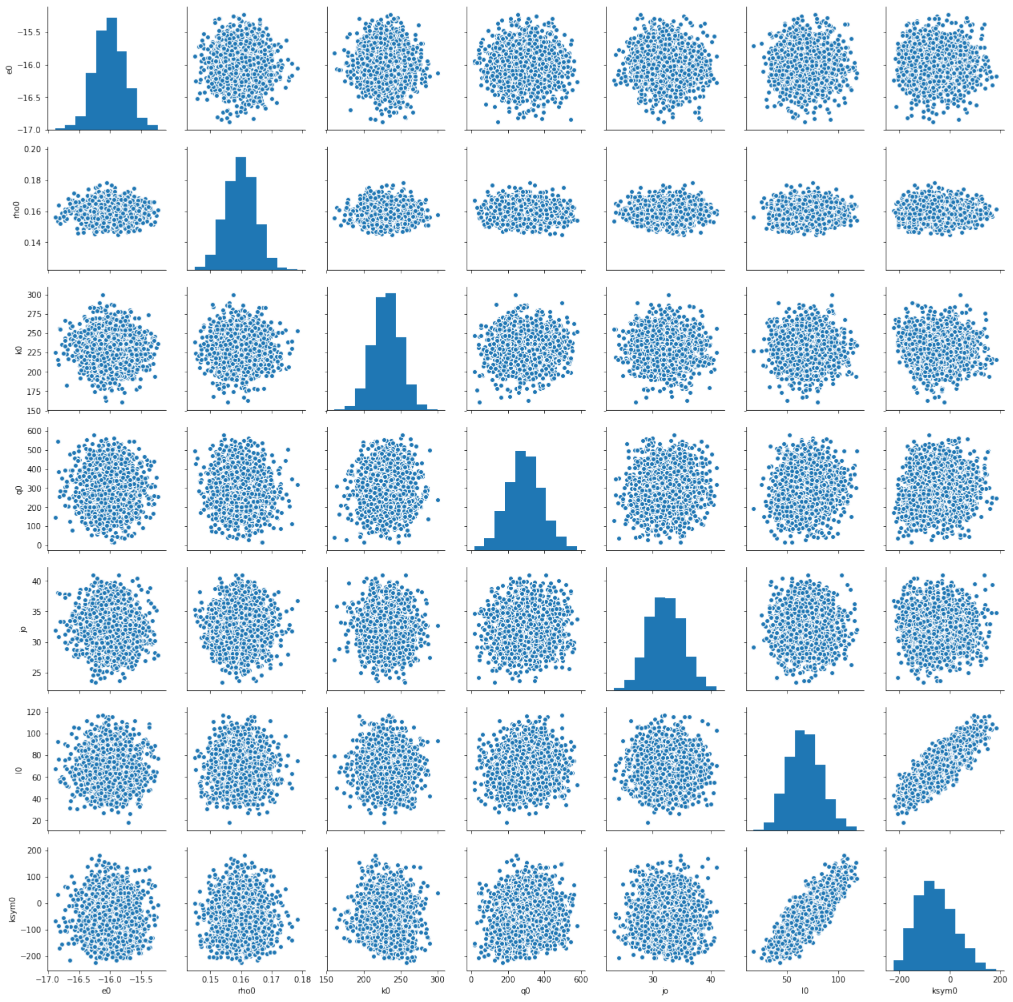


Figure 3: Case 2 pair Plot of Features

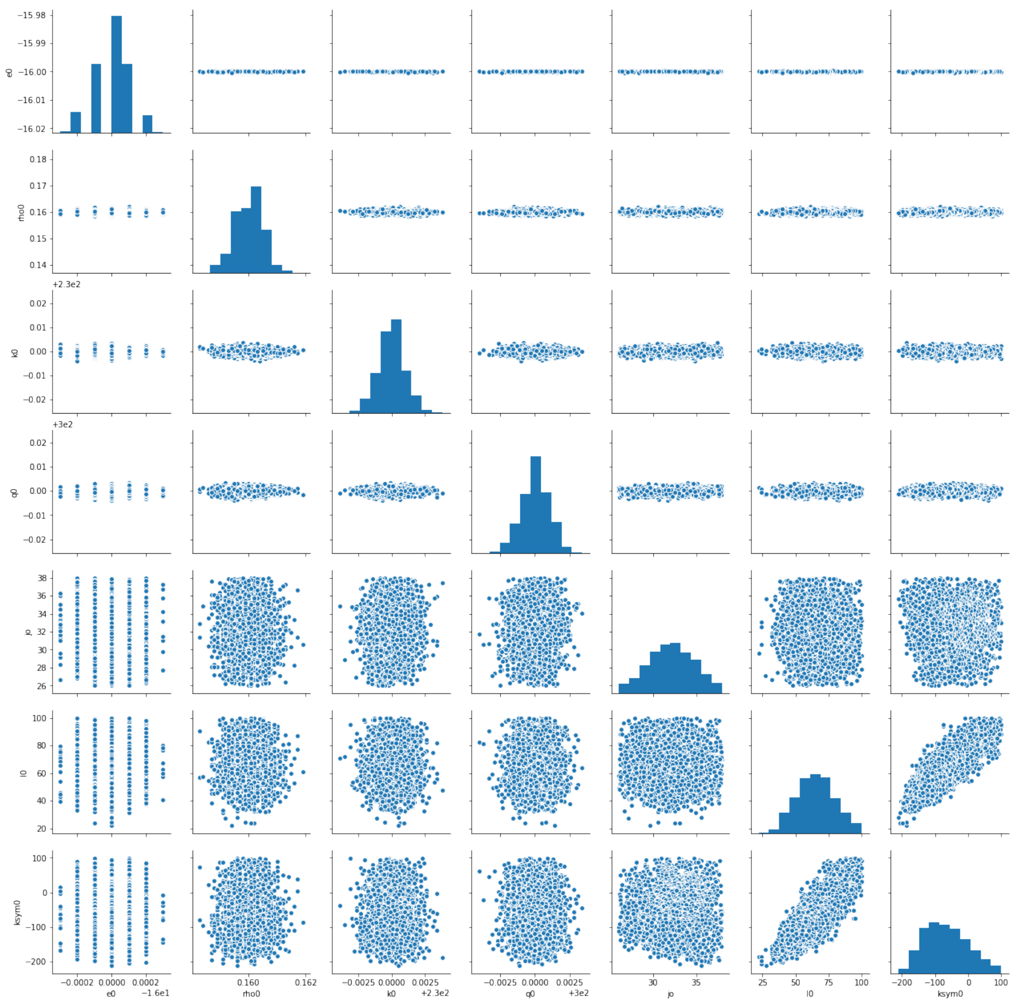


Figure 4: Case 3 pair Plot of Features

3 Machine Learning Approach

3.1 What is ML?

Machine Learning (ML) is a technique to built statistical models based on data. The problem we are trying to solve is a multi-target regression problem. A multi-target regression problem is a problem in which there are several target variables which depend on various features, and also might be inter-dependent among each other.
Usually, ML algorithms are designed to fit to only one target variable given a set of features as input. So, to solve multiple target regression problems [1], there are two approaches, either we can fit a separate model for each target variable, or we can try to connect these models such that the correlations in the output variables are also incorporated.

3.2 Models

As discussed earlier, the input to the models are the EOS parameters: $e_0, \rho_0, K_0, Q_0, J_0, L_0$ and $K_{\text{sym},0}$, and the models predict the NMPs: NS mass, r14 and Lambda14. Therefore, the model $(\mathcal{M})$ should be of the form:

$$\{NSMass, r14, Lambda14\} = \mathcal{M}(e_0, \rho_0, K_0, Q_0, J_0, L_0, K_{\text{sym},0})$$

There are two kinds of models for any multi-target regression problem:

- Multiple Single Target Regressors
- Multi-target Regression

Multiple Single Target Regressors

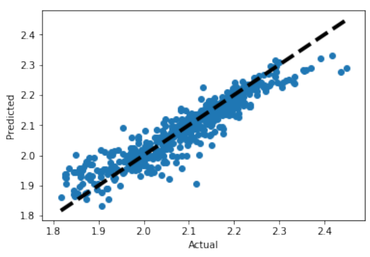
This strategy involves fitting an independent regressor for each target variable. So, we will fit one model for each of the 3 outputs i.e. NSMass, r14, Lambda14.

$$\begin{aligned} NSMass &= \mathcal{M}_1(e_0, \rho_0, K_0, Q_0, J_0, L_0, K_{\text{sym},0}) \\ r14 &= \mathcal{M}_2(e_0, \rho_0, K_0, Q_0, J_0, L_0, K_{\text{sym},0}) \\ Lambda14 &= \mathcal{M}_3(e_0, \rho_0, K_0, Q_0, J_0, L_0, K_{\text{sym},0}) \end{aligned}$$

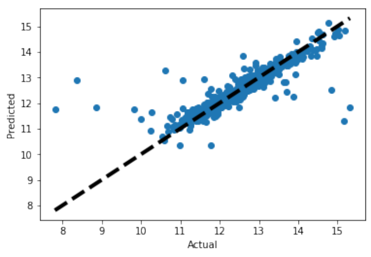
$\mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3$ are chosen to be Random Forest models because they gave the least RMS error.

Results (On CASE-1 data):

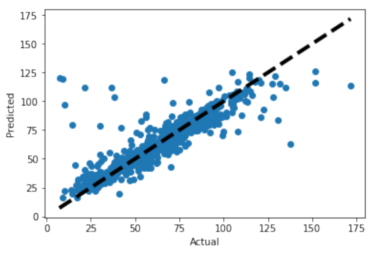
Output	RMSE
NSMass	0.0044
r14	0.4874
Lambda14	14.3772



(a) NSMass



(b) r14

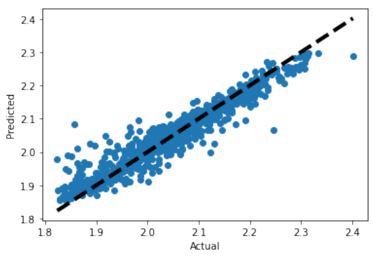


(c) Lambda14

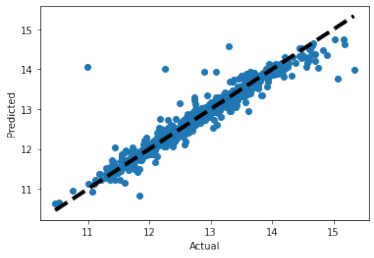
Figure 5: Actual v/s Predicted values, Random Forest, CASE-1

Results (On CASE-2+CASE-3 data):

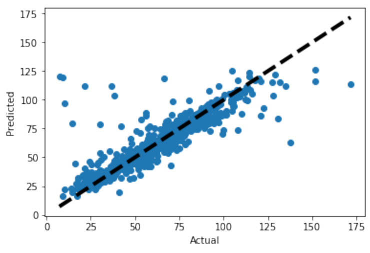
Output	RMSE
NSMass	0.0262
r14	0.1655
Lambda14	25.9824



(a) NSMass



(b) r14



(c) Lamnda14

Figure 6: Actual v/s Predicted values, Random Forest, CASE-2+CASE-3

Multi-Target Regression

The drawback of multiple singe target regressors is that, they assume mutual independence of the output variables. However, usually that is not the case. Therefore, we need a technique to exploit the correlation among the outputs while training the model. The most commonly used technique is called **Regressor Chaining** [4][5]. In regressor chaining, the prediction of one regressor is used as an input to the next regressor. In this way, regressors are chained to exploit the correlation among the output variables.

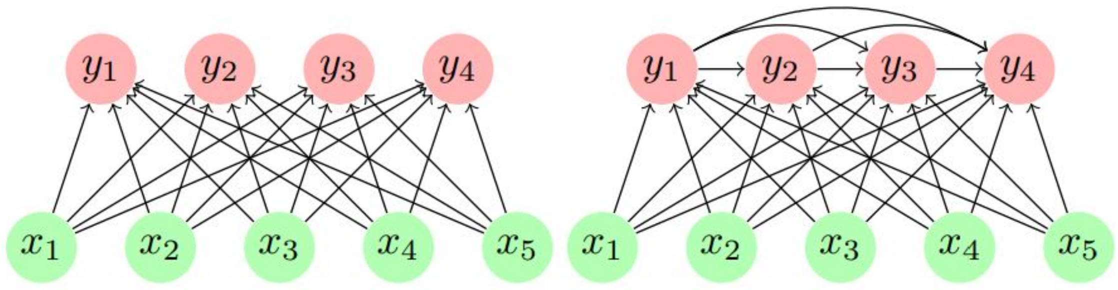


Figure 7: Multiple single target vs multi-target regression. Each target y_j is learned by a model, where inputs are shown as incoming arrows

However, the order in which these regressors are chained in important and may affect the output. For 3 output variables, as in our case, there are total 3! = 6 chains possible. The model $\mathcal{M} = \{\mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3\}$ for the chain $(NSMass \to r14 \to Lambda14)$ is shown below.

$$\begin{aligned} NSMass &= \mathcal{M}_1(e_0, \rho_0, K_0, Q_0, J_0, L_0, K_{\text{sym},0}) \\ r14 &= \mathcal{M}_2(e_0, \rho_0, K_0, Q_0, J_0, L_0, K_{\text{sym},0}, NSMass) \\ Lambda14 &= \mathcal{M}_3(e_0, \rho_0, K_0, Q_0, J_0, L_0, K_{\text{sym},0}, NSMass, r14) \end{aligned}$$

Individually, $\mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3$ are chosen to be **Support Vector Regressors**[6] because they gave the least RMS error.

Results (On CASE-1 data):

Chain Order	$NSMass_{RMSE}$	$r14_{RMSE}$	$Lambda14_{RMSE}$
$NSMass \to r14 \to Lambda14$	0.0472	0.4681	75.2071
$NSMass \to Lambda14 \to r14$	0.0472	0.4875	76.2368
$r14 \to NSMass \to Lambda14$	0.0477	0.4865	76.4140
$r14 \to Lambda14 \to NSMass$	0.0512	0.4648	76.4571
$Lambda14 \to NSMass \to r14$	0.0514	0.4875	76.2348
$Lambda14 \to r14 \to NSMass$	0.0514	0.4871	76.2349

The RMSE of Lambda14 is very large in this case, therefore this is not the best result possible.

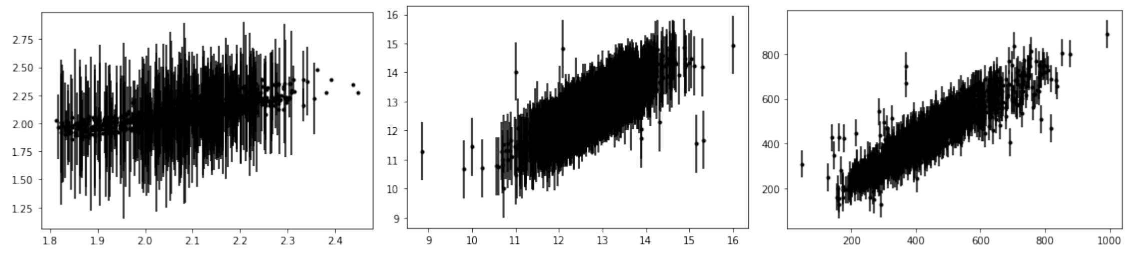
Variational Inference

Apart from predicting the NMPs, we are also interested in predicting the uncertainty in the values of the outputs. Therefore, **Gaussian Process Regression** is used to fit the data for error estimation. Gaussian process regression (GPR) is a bayesian approach, which means, it estimates the posterior probability using a specified prior and the evidence (training data). The prior is of the Gaussian Process is specified using a covariance kernel function. The output depends on the choice of the covariance kernel function.

Following are the results obtained for various kernels.
The RMSE reported is considering the mean value of the prediction. The error bars shown in the graph are the standard deviation predicted by the model.

1. Dot Product + White Noise Kernel

Output	RMSE
NSMass	0.0631
r14	0.3844
Lambda14	58.3936



(a) NSMass (b) r14 (c) Lambda14

Figure 8: Actual v/s Predicted values and standard deviations, Dot product + White Noise Kernel

2. Gaussian Kernel

$$k(x_i, x_j) = exp(-\frac{d(x_i, x_j)^2}{2\sigma^2})$$

Output	RMSE
NSMass	0.2872
r14	2.2907
Lambda14	157.8810

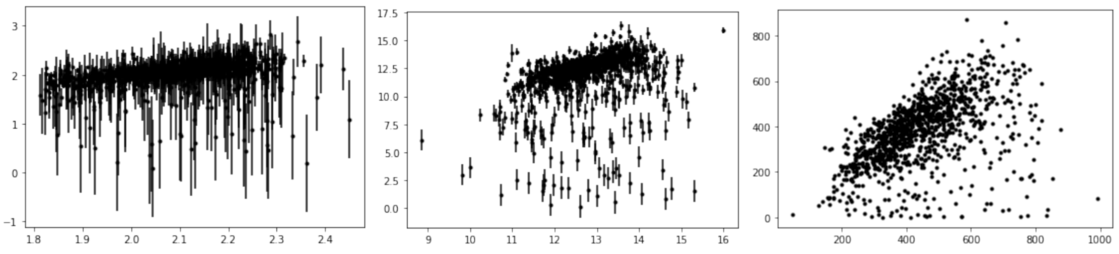

(a) NSMass (b) r14 (c) Lambda14

Figure 9: Actual v/s Predicted values and standard deviations, Gaussian Kernel

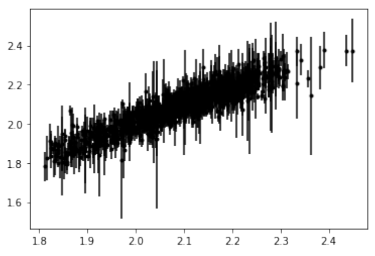
3. Matern Kernel

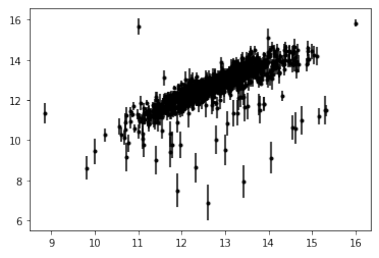
$$k(x_i, x_j) = \frac{1}{\Gamma(\nu)2^{\nu-1}}\Big(\frac{\sqrt{2\nu}}{l}d(x_i, x_j)\Big)^{\nu} K_{\nu}\Big(\frac{\sqrt{2\nu}}{l}d(x_i, x_j)\Big)$$

where,

K_ν is the modified Bessel Function, and Γ is the gamma function ν value is taken to be 1.5

Output	RMSE
NSMass	0.0444
r14	0.6279
Lambda14	87.8872


(a) NSMass


(b) r14

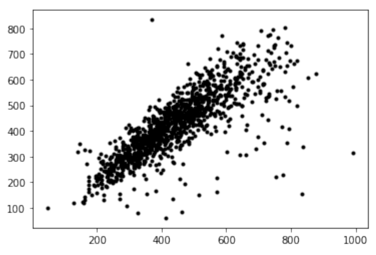

(c) Lambda14

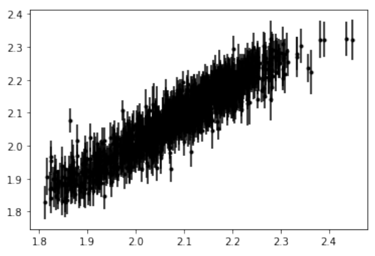
Figure 10: Actual v/s Predicted values and standard deviations, Matern Kernel

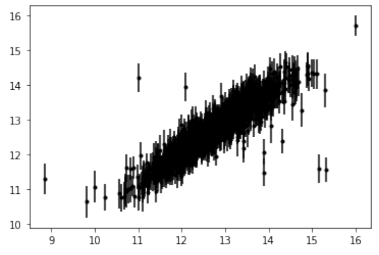
4. Rational Quadratic Kernel

$$k(x_i, x_j) = \Big(1 + \frac{d(x_i, x_j)^2}{2\alpha l^2}\Big)^{-\alpha}$$

where, ”α is taken to be 1.

Output	RMSE
NSMass	0.0399
r14	0.3733
Lambda14	61.0257


(a) NSMass


(b) r14

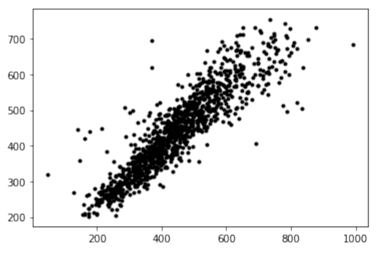

(c) Lambda14

Figure 11: Actual v/s Predicted values and standard deviations, Rational Quadratic Kernel

Therefore, the rational quadratic kernel works best in terms of better mean prediction as well as less deviation.

3.3 The Reverse Problem

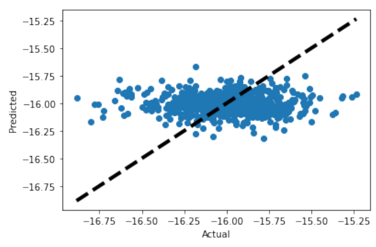
An attempt was also made to solve the reverse problem, i.e. training models on NMPs to estimate EOS parameters. Here, instead of only 3 NMPs, all 6 NMPs have been used for training.

7 different Random Forest models were fit to the data :

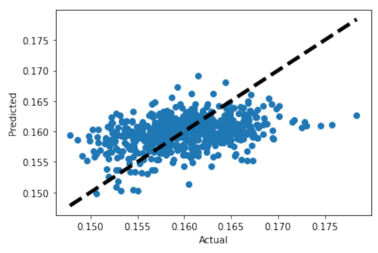
$$\begin{aligned}
e(0) &= \mathcal{M}_1(NSMass, Rmax, r14, Lambda10, lambda14, lambda18, vs, qsym)\\
rho(0) &= \mathcal{M}_2(NSMass, Rmax, r14, Lambda10, lambda14, lambda18, vs, qsym)\\
K(0) &= \mathcal{M}_3(NSMass, Rmax, r14, Lambda10, lambda14, lambda18, vs, qsym)\\
Q(0) &= \mathcal{M}_4(NSMass, Rmax, r14, Lambda10, lambda14, lambda18, vs, qsym)\\
J(0) &= \mathcal{M}_5(NSMass, Rmax, r14, Lambda10, lambda14, lambda18, vs, qsym)\\
L(0) &= \mathcal{M}_6(NSMass, Rmax, r14, Lambda10, lambda14, lambda18, vs, qsym)\\
Ksym(0) &= \mathcal{M}_7(NSMass, Rmax, r14, Lambda10, lambda14, lambda18, vs, qsym)
\end{aligned}$$

Results: (using CASE-2 data)

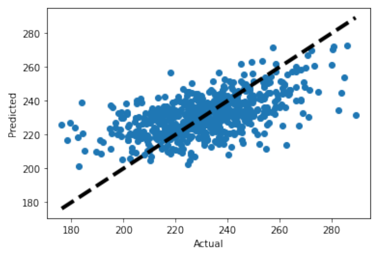
Output	RMSE
e(0)	0.2655
rho(0)	0.0045
K(0)	15.7981
Q(0)	76.2037
J(0)	2.4891
L(0)	5.4772
Ksym(0)	27.9631



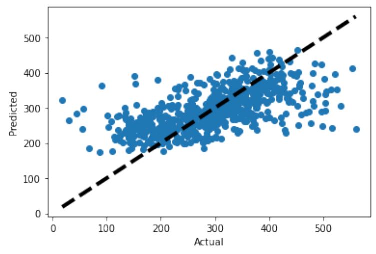
(a) e(0)



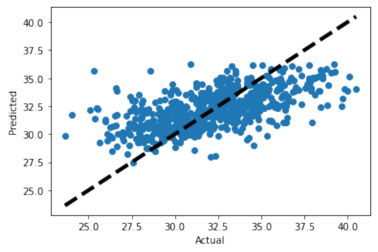
(b) rho(0)



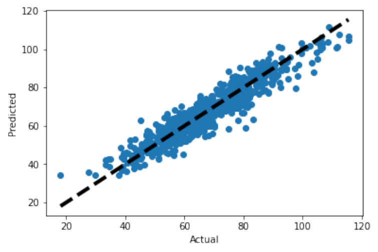
(c) K(0)



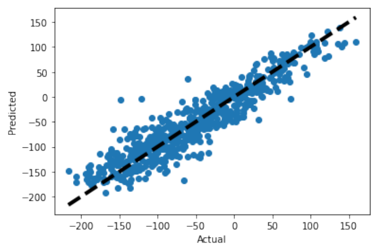
(d) Q(0)



(e) J(0)



(f) L(0)



(g) Ksym(0)

Figure 12: Actual v/s Predicted values of EOS Parameters

Therefore, it can be seen that only the trend 2 of the EOS parameters viz. L(0) and Ksym(0) could be captured.

4 Artificial Neural Network Approach

4.1 What is ANN

Artificial Neural Networks (ANN) are mathematical models comprised neurons, each of which are associated with a weight and a bias. Together, the neurons can model many complex relationships, and provide accurate answers. Neural networks inherently support multi-target regression, because the output layer can have more than one neurons which make the final prediction.

4.2 Models

We trained around 400 ANN models by keeping 1 or 2 Hidden layers and varying the number of nodes in each hidden layer from 1 to 20 (the upper limit 20 was calculated as to prevent over-fitting given the size of the dataset).
We used ReLu for the Hidden layer, and linear for the output layer, MSE was taken for Loss and Adam was taken as the optimizer.
Out of the 400 models, we filtered out the best models based on RMS and R^2 values, while also comparing the True Vs Predicted Pearson Coefficient value between the L(0) and Lambda 1.4 features and between Ksym(0) and Lambda 1.4. We also did several testings where we trained and tested on various combinations of case 1, case 2 and case 3, thereby training close to 2000 models. Out of all, some of the best ones are as follows:

Model	**RMS**	**RMS_Mass**	**RMS_Rad**	**RMS_Lam**	**R2**
ANN(2HL-14,15 Nodes)	14.531461	0.022595	0.206077	25.168376	0.952384
ANN(2HL-14,19 Nodes)	29.283586	0.025433	0.364181	50.719345	0.872441
ANN(2HL-14,20 Nodes)	14.128842	0.023468	0.207352	24.470983	0.951466
ANN(2HL-14,20 Nodes)	3.512546	0.012410	0.050077	6.083690	0.989099
ANN(2HL-14,20 Nodes)	28.680133	0.029109	0.361099	49.674127	0.870699
ANN(2HL-15,19 Nodes)	29.327132	0.031020	0.342073	50.794922	0.871234
ANN(2HL-16,17 Nodes)	28.922670	0.025599	0.351684	50.094293	0.877417
ANN(2HL-16,18 Nodes)	3.870370	0.012464	0.062257	6.703376	0.987797
ANN(2HL-16,19 Nodes)	29.055961	0.032494	0.338020	50.325255	0.871125
ANN(2HL-17,15 Nodes)	13.658629	0.021909	0.200474	23.656581	0.955626

Model	**PC-L,Lam14 (Original)**	**PC-L,Lam14 (predicted)**	**%error(L_Lam)**	**PC-Ksym,Lam14 (Original)**	**PC-Ksym,Lam14 (predicted)**	**%error(Ksym_Lam)**
ANN(2HL-14,15 Nodes)	0.830533	0.832506	0.002375	0.856680	0.860173	0.004078
ANN(2HL-14,19 Nodes)	0.562694	0.568485	0.010292	0.581096	0.626745	0.078558
ANN(2HL-14,20 Nodes)	0.830533	0.835639	0.006147	0.856680	0.862004	0.006215
ANN(2HL-14,20 Nodes)	0.909419	0.907856	0.001719	0.974911	0.973922	0.001014
ANN(2HL-14,20 Nodes)	0.562694	0.572383	0.017218	0.581096	0.618385	0.064171
ANN(2HL-15,19 Nodes)	0.562694	0.566525	0.006807	0.581096	0.619200	0.065572
ANN(2HL-16,17 Nodes)	0.562694	0.577687	0.026645	0.581096	0.630053	0.084250
ANN(2HL-16,18 Nodes)	0.909419	0.907775	0.001808	0.974911	0.974195	0.000735
ANN(2HL-16,19 Nodes)	0.562694	0.577750	0.026756	0.581096	0.623972	0.073785
ANN(2HL-17,15 Nodes)	0.830533	0.830641	0.000130	0.856680	0.861551	0.005686

5 Bayesian Neural Network Approach

5.1 What is BNN

A Bayesian neural network (BNN) refers to extending standard networks with posterior inference. Standard NN training via optimization is (from a probabilistic perspective) equivalent to maximum likelihood estimation (MLE) for the weights.

For many reasons this is unsatisfactory. One reason is that it lacks proper theoretical justification from a probabilistic perspective: why maximum likelihood? Why just point estimates? Using MLE ignores any uncertainty that we may have in the proper weight values. From a practical standpoint, this type of training is often susceptible to overfitting, as NNs often do.

One partial fix for this is to introduce regularization. From a Bayesian perspective, this is equivalent to inducing priors on the weights (say Gaussian distributions if we are using L2 regularization). Optimization in this case is akin to searching for MAP estimators rather than MLE. Again from a probabilistic perspective, this is not the right thing to do, though it certainly works well in practice.

The correct (i.e., theoretically justifiable) thing to do is posterior inference, though this is very challenging both from a modelling and computational point of view. BNNs are neural networks that take this approach. In the past this was all but impossible, and we had to resort to poor approximations such as Laplace's method (low complexity) or MCMC (long convergence, difficult to diagnose). However, lately there have been some super-interesting results on using variational inference to do this [1], and this has sparked a great deal of interest in the area.

BNNs are important in specific settings, especially when we care about uncertainty very much. Some examples of these cases are decision making systems, (relatively) smaller data settings, Bayesian Optimization, model-based reinforcement learning and others

5.2 Models

5.2.1 Approach 1: Self Implementation of BNN

I implement and train a Bayesian neural network with Keras following the approach described in [3].

A neural network can be viewed as probabilistic model $p(y \mid \mathbf{x}, \mathbf{w})$. For classification, y is a set of classes and $p(y \mid \mathbf{x}, \mathbf{w})$ is a categorical distribution. For regression, y is a continuous variable and $p(y \mid \mathbf{x}, \mathbf{w})$ is a Gaussian distribution.
Given a training dataset $\mathcal{D} = \left\{\mathbf{x}^{(i)}, y^{(i)}\right\}$ we can construct the likelihood function

$$p(\mathcal{D} \mid \mathbf{w}) = \prod_i p\left(y^{(i)} \mid \mathbf{x}^{(i)}, \mathbf{w}\right) \tag{1}$$

which is a function of parameters $\mathbf{w}$. Maximizing the likelihood function gives the maximimum likelihood estimate (MLE) of $\mathbf{w}$. The usual optimization objective during training is the negative log likelihood. For a categorical distribution this is the cross entropy error function, for a Gaussian distribution this is proportional to the sum of squares error function. MLE can lead to severe overfitting though.
Multiplying the likelihood with a prior distribution $p(\mathbf{w})$ is, by Bayes theorem, proportional to the posterior distribution

$$p(\mathbf{w} \mid \mathcal{D}) \propto p(\mathcal{D} \mid \mathbf{w})p(\mathbf{w}) \tag{2}$$

Maximizing $p(\mathcal{D} \mid \mathbf{w})p(\mathbf{w})$ gives the maximum a posteriori (MAP) estimate of $\mathbf{w}$. Computing the MAP estimate has a regularizing effect and can prevent overfitting. The optimization objectives here are the same as for MLE plus a regularization term coming from the log prior.

Both MLE and MAP give point estimates of parameters. If we instead had a full posterior distribution over parameters we could make predictions that take weight uncertainty into account. This is covered by the posterior predictive distribution

$$p(y \mid \mathbf{x}, \mathcal{D}) = \int p(y \mid \mathbf{x}, \mathbf{w})p(\mathbf{w} \mid \mathcal{D})d\mathbf{w} \tag{3}$$

in which the parameters have been marginalized out. This is equivalent to averaging predictions from an ensemble of neural networks weighted by the posterior probabilities of their parameters $\mathbf{w}$.

Unfortunately, an analytical solution for the posterior $p(\mathbf{w} \mid \mathcal{D})$ in neural networks is untractable. We therefore have to approximate the true posterior with a variational distribution $q(\mathbf{w} \mid \theta)$ of known functional form whose parameters we want to estimate. This can be done by minimizing the KullbackLeibler divergence between $q(\mathbf{w} \mid \theta)$ and the true posterior $p(\mathbf{w} \mid \mathcal{D})$. As shown in Appendix, the corresponding optimization objective or cost function is

$$\mathcal{F}(\mathcal{D}, \theta) = \mathrm{KL}(q(\mathbf{w} \mid \theta) \| p(\mathbf{w})) - E_{q(\mathbf{w}|\theta)} \log p(\mathcal{D} \mid \mathbf{w}) \tag{4}$$

This is known as the variational free energy. The first term is the Kullback-Leibler divergence between the variational distribution $q(\mathbf{w} \mid \theta)$ and the prior $p(\mathbf{w})$ and is called the complexity cost. The second term is the expected value of the likelihood w.r.t. the variational distribution and is called the likelihood cost. By rearranging the KL term, the cost function can also be written as

$$\mathcal{F}(\mathcal{D}, \theta) = E_{q(\mathbf{w}|\theta)} \log q(\mathbf{w} \mid \theta) - E_{q(\mathbf{w}|\theta)} \log p(\mathbf{w}) - E_{q(\mathbf{w}|\theta)} \log p(\mathcal{D} \mid \mathbf{w}) \tag{5}$$

We see that all three terms in equation 2 are expectations w.r.t. the variational distribution $q(\mathbf{w} \mid \theta)$. The cost function can therefore be approximated by drawing samples $\mathbf{w}^{(i)}$ from $q(\mathbf{w} \mid \theta)$.

$$\mathcal{F}(\mathcal{D}, \theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left[\log q\left(\mathbf{w}^{(i)} \mid \theta\right) - \log p\left(\mathbf{w}^{(i)}\right) - \log p\left(\mathcal{D} \mid \mathbf{w}^{(i)}\right) \right] \tag{6}$$

In the following example, we'll use a Gaussian distribution for the variational posterior, parameterized by $\theta = (\mu, \sigma)$ where μ is the mean vector of the distribution and σ the standard deviation vector. The elements of σ are the elements of a diagonal covariance matrix which means that weights are assumed to be uncorrelated. Instead of parameterizing the neural network with weights $\mathbf{w}$ directly we parameterize it with μ and σ and therefore double the number of parameters compared to a plain neural network.
A training iteration consists of a forward-pass and and backward-pass. During a forward pass a single sample is drawn from the variational posterior distribution. It is used to evaluate the approximate cost function defined by equation 6 . The first two terms of the cost function are data-independent and can be evaluated layer-wise, the last term is data-dependent and is evaluated at the end of the forward-pass. During a backward-pass, gradients of μ and σ are calculated via backpropagation so that their values can be updated by an optimizer.
Since a forward pass involves a stochastic sampling step we have to apply the so-called re-parameterization trick for backpropagation to work. The trick is to sample from a parameter-

free distribution and then transform the sampled ε with a deterministic function $t(\mu, \sigma, \varepsilon)$ for which a gradient can be defined. Here, ε is drawn from a standard normal distribution i.e. $\varepsilon \sim \mathcal{N}(0, \mathbf{I})$ and function $t(\mu, \sigma, \varepsilon) = \mu + \sigma \odot \varepsilon$ shifts the sample by mean μ and scales it with σ where $\odot$ is element-wise multiplication.
For numeric stability we will parameterize the network with ρ instead of σ directly and transform ρ with the softplus function to obtain $\sigma = \log(1 + \exp(\rho))$. This ensures that σ is always positive. As prior, a scale mixture of two Gaussians is used

$$p(\mathbf{w}) = \pi\mathcal{N}\left(\mathbf{w} \mid 0, \sigma_1^2\right) + (1 - \pi)\mathcal{N}\left(\mathbf{w} \mid 0, \sigma_2^2\right) \tag{7}$$

where σ_1, σ_2 and π are hyper- parameters i.e. they are not learned during training.
Our model is a neural network with two DenseVariational hidden layers, each having 20 units, and one DenseVariational output layer with one unit. Instead of modeling a full probability distribution p(y|x,w) as output the network simply outputs the mean of the corresponding Gaussian distribution. In other words, we do not model aleatoric uncertainty here and assume it is known. We only model epistemic uncertainty via the DenseVariational layers.
The network can now be trained with a Gaussian negative log likelihood function as loss function assuming a fixed standard deviation (noise). This corresponds to the likelihood cost, the last term in equation 6.
When calling model.predict we draw a random sample from the variational posterior distribution and use it to compute the output value of the network. This is equivalent to obtaining the output from a single member of a hypothetical ensemble of neural networks. Drawing 500 samples means that we get predictions from 500 ensemble members. From these predictions we can compute statistics such as the mean and standard deviation. In our example, the standard deviation is a measure of epistemic uncertainty.

For a normal BNN with 2 Hidden layers and 10 nodes in each layer, the loss value was of the order of 10^9. To reduce it, we play around with different set of hyperparamters (seen in equation 7). Finally, a value of $\sigma_1 = 0.25$ and $\sigma_2 = 0.3$ with $\pi = 0.999$ seemed to give the lowest loss values. For such a case, the BNN model gives the loss curve as shown in the Figure.
As can be seen, the loss curve is highly non linear and the actual and predicted values are very off, thus, we need some better means to implement our BNN.

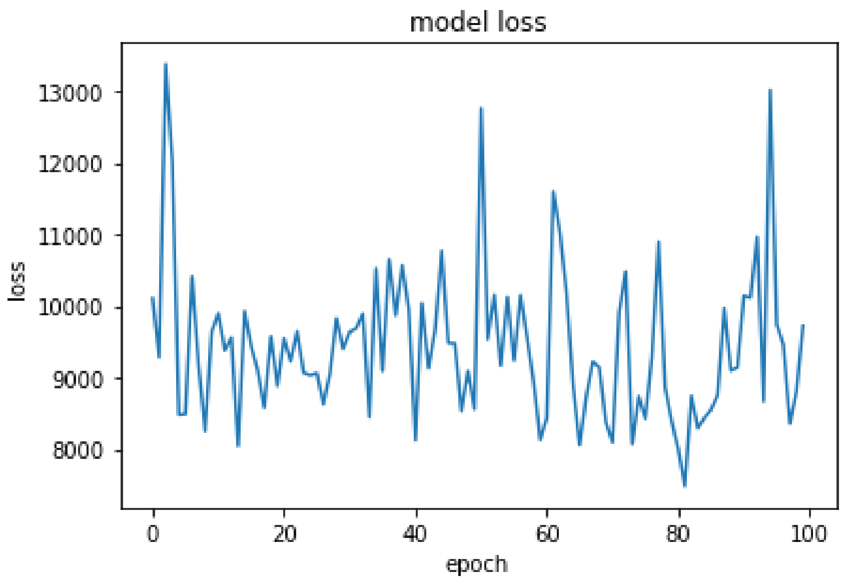


Figure 13: Loss curve for BNN with 10 nodes in 2 Hidden layers each

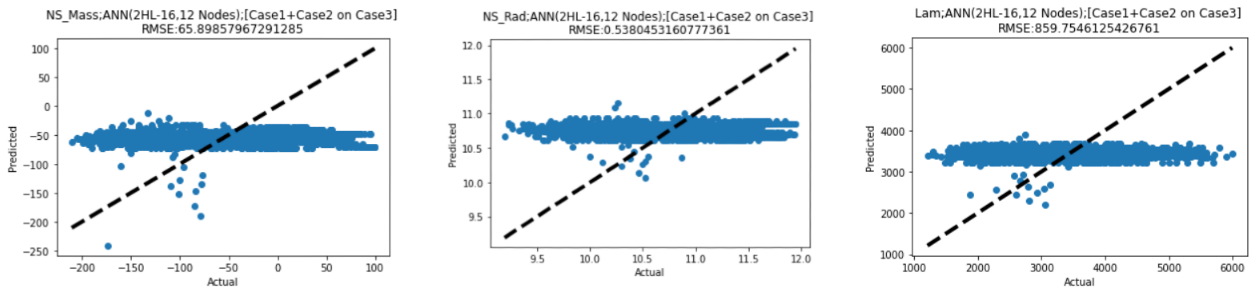


Figure 14: Actual Vs Predicted values for various outputs

5.2.2 Approach 2: Implementation using Tensorflow libraries (All 7 Inputs)

We first use standard scalar to scale the entire input data. TensorFlow offers a dataset class to construct training and test sets. We shall use 70% of the data as training set. The sets are shuffled and repeating batches are constructed.

To account for aleotoric uncertainty, which arises from the noise in the output, dense layers are combined with probabilistic layers. More specifically, the mean and covariance matrix of the output is modelled as a function of the input and parameter weights. The first hidden layer shall consist of x nodes, the second one needs three nodes for the means plus seven nodes for the variances and covariances of the three-dimensional (there are three outputs) multivariate Gaussian posterior probability distribution in the final layer. This is achieved using

the params_size method of the last layer (MultivariateNormalTriL), which is the declaration of the posterior probability distribution structure, in this case a multivariate normal distribution in which only one half of the covariance matrix is estimated (due to symmetry). There is also an early stopping condition implemented to make sure the loss doesn't go negative.
Once that is done, we run the model for 40 epochs (early stopping implemented) along with a validation check at each epoch. We train 20 such models (with x nodes in the first layer ranging from 1 to 20) and the loss curve for one such model is in the figure.

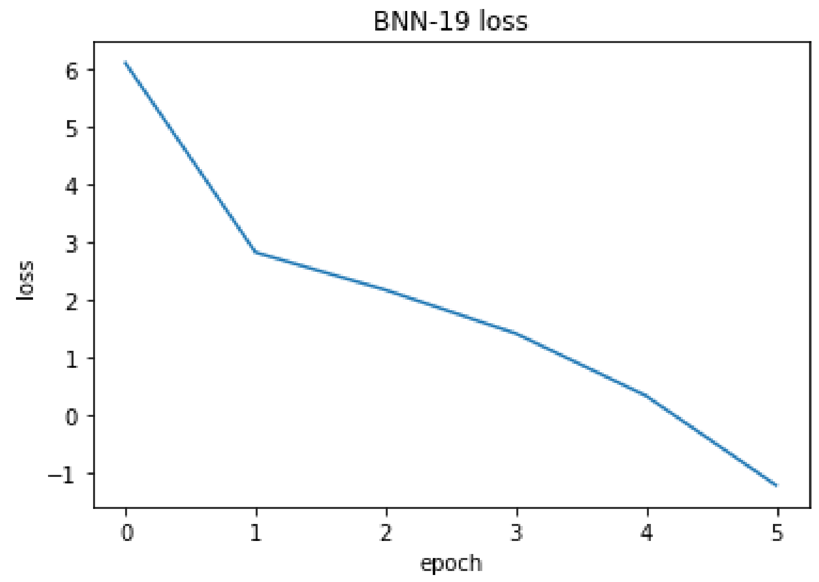


Figure 15: Loss curve for BNN-19

Since it is a probabilistic model, a Monte Carlo experiment is performed to provide a prediction. In particular, every prediction of a sample x results in a different output y, which is why the expectation over many individual predictions has to be calculated. Additionally, the variance can be determined this way. Finally we use RMS and R^2 values to select the best models out of the 20 trained and the results are as follows:

Model	**RMS**	**RMS NS_Mass**	**RMS NS_Rad**	**RMS Lam**	**R2**	**PC (Lam_1.6 vs Ksym(0) (True,Pred)**	**sigma_2 NSM**	**sigma_2 NSR**	**sigma_2 NSL**
BNN-19	9.859321	0.058202	0.130934	17.076243	0.847003	(0.9324, 0.9281)	0.94	0.97	0.98
BNN-9	11.155777	0.048876	0.171477	19.321549	0.869391	(0.9221, 0.9435)	0.91	0.97	0.98
BNN-16	11.228783	0.058024	0.203776	19.447669	0.810475	(0.9090, 0.9201)	0.94	0.98	0.90
BNN-10	11.247332	0.059860	0.141286	19.480346	0.827740	(0.9475, 0.9492)	0.95	0.98	0.97
BNN-20	13.406316	0.056145	0.229794	23.219215	0.811083	(0.8971, 0.9025)	0.95	0.98	1.00
BNN-6	14.160746	0.065759	0.173254	24.526431	0.825604	(0.9009, 0.9241)	0.87	0.96	0.94

Here, the PC is the Pearson Coefficient between Lam_1.6 and Ksym(0) variables, the first value being true and the second value being the predicted one. Another metric is the

sigma_2 metric which is the percentage of data points whose actual value lies within 2σ of the predicted mean (and σ here being the variance.)
The we use the best of these models to predict our outcome, the outcome (with errorbars of 2σ) are as follows:

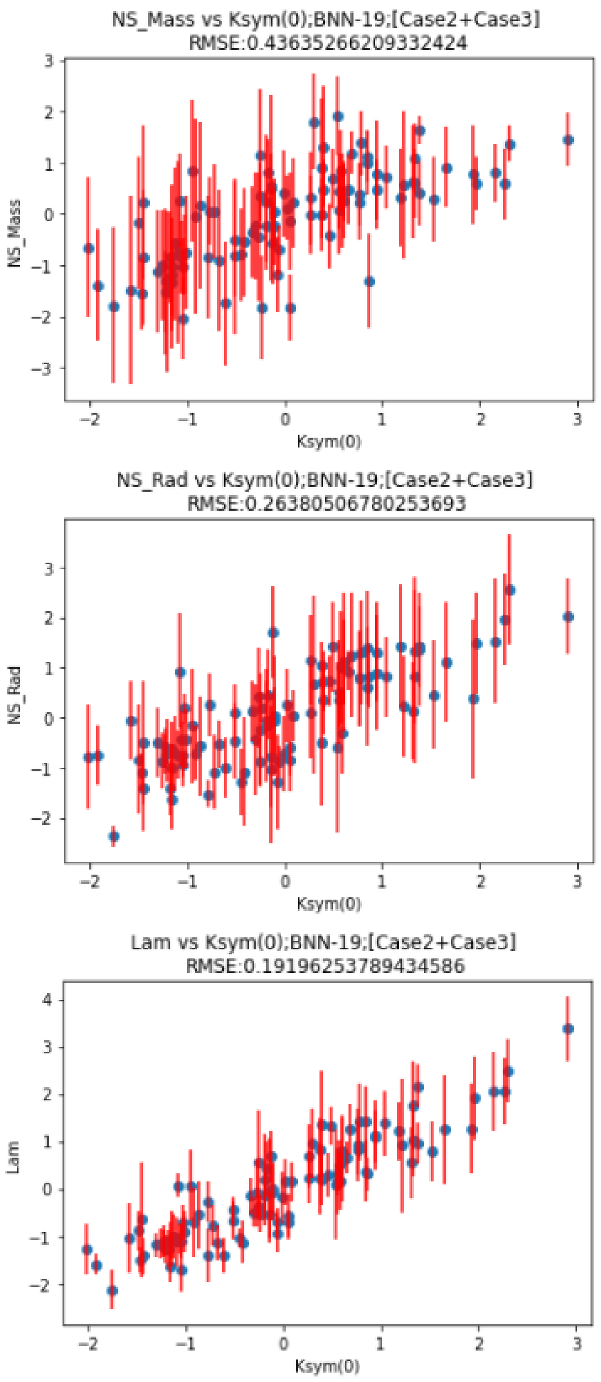


Figure 16: Predictions for all three outputs with errorbars of 2σ

5.3 Output

We finally use the BNN model seen earlier to predict our final outcomes. Using all three cases and training on the entire data and keeping track of the scaling parameters used to scale the input data, which was generated from sampling from a MVGD with parameters:

	MVGD	
	$\overline{P}_i$	σ_{P_i}
e_0	-16.0	0.25
ρ_0	0.16	0.005
K_0	230.0	20
Q_0	-300	100
J_0	32.0	3
L_0	60.0	20
$K_{\mathrm{sym},0}$	-100.0	100

Figure 17: Parameters used for sampling

We sample 1000 inputs and for those, the outputs we get are:

	e(0)	**rho(0)**	**K(0)**	**Q(0)**	**J(0)**	**L(0)**	**Ksym(0)**	**NS Mass**		**NS Radius**		**Lambda 1.4**	
								Mean	Std_dev	Mean	Std_dev	Mean	Std_dev
0	-15.852569	0.159284	223.523908	349.689957	31.398190	66.711028	-38.969833	2.081290	0.018933	12.853865	0.321992	464.762643	56.112333
1	-15.976395	0.156089	242.838571	408.672947	29.973709	69.757225	-111.086834	2.007281	0.051408	12.728095	0.300416	380.670717	52.328973
2	-15.623846	0.158483	212.709710	219.167159	30.416085	81.705726	-94.363594	2.043819	0.055812	13.527555	0.458907	445.714137	47.878349
3	-16.029494	0.162671	223.932621	385.995336	35.320064	67.216773	67.728655	2.064833	0.015013	12.568495	0.351118	502.446809	58.601324
4	-15.883499	0.164560	229.522794	156.780237	27.299334	54.896736	-123.075430	2.009268	0.047281	12.466829	0.294770	347.859190	39.868851

6 Conclusion

In conclusion, one can successfully model and estimate Neutron Star parameters using Machine Learning and Deep Learning Techniques (Code available in [7]) and obtain results within 0.001% error margin, which is equivalent to calculating the parameters using theory. Given more dataset, one can explore the entire 7-parameter space and produce better results for input with different types of means and variances for sampling. one also needs to carry out further analysis on the rigidity of such models and how they will behave when their results are compared to actual observations. Only then will the usefulness of such models be apparent.

References

[1] Borchani, Hanen and Varando, Gherardo and Bielza, Concha and Larra - A Survey on Multi-Output Regression, September 2015, https://doi.org/10.1002/widm.1157

[2] Tuhin Malik, B. K. Agrawal, Constança Providência, J. N. De, Unveiling the correlations of tidal deformability with the nuclear symmetry energy parameters, https://arxiv.org/abs/2008.03469

[3] Charles Blundell and Julien Cornebise and Koray Kavukcuoglu and Daan Wierstra, Weight Uncertainty in Neural Networks, https://arxiv.org/abs/1505.05424

[4] Jesse Read, Luca Martino, Probabilistic Regressor Chains with Monte Carlo Methods, https://arxiv.org/pdf/1907.08087.pdf

[5] Gabriella Melki, Alberto Cano, Vojislav Kecman, Sebastián Ventura, Multi-target support vector regression via correlation regressor chains, Information Sciences, Volumes 415–416, 2017, Pages 53-69, ISSN 0020-0255, https://doi.org/10.1016/j.ins.2017.06.017. (http://www.sciencedirect.com/science/article/pii/S0020025517307946)

[6] Blog - http://people.vcu.edu/ acano/MTR-SVRCC/

[7] Github Repository for the project - https://github.com/AstroDnerd/NS-ML-on-Nuclear-Matter–KB