Predicting Asteroid Diameter Using Artificial Neural Networks

Cain Farnam

1 Introduction

There are over 150 million asteroids orbiting around the sun. These rocky satellites range anywhere from a few meters in diameter, to several kilometers. It can be extremely difficult to measure the diameter of an asteroid, as they are generally quite some distance away from our own plant. A NASA subsidiary keeps track of all of the asteroids discovered, but often times the diameter is not calculated. The goal of this paper is to provide a deep neural network that predicts the diameter of an asteroid based on other variables, much of which are more easily obtained.

2 Data

feature	observations	data type	feature	observations	data type
full name	839736	object	diameter	137681	object
a	839734	float64	extent	18	object
e	839736	float64	albedo	136452	float64
G	119	float64	rot per	18796	float64
i	839736	float64	GM	14	float64
om	839736	float64	BV	1021	float64
w	839736	float64	UB	979	float64
q	839736	float64	IR	1	float64
ad	839730	float64	spec B	1666	object
per y	839735	float64	spec T	980	object
data arc	823947	float64	neo	839730	object
condition code	838743	object	pha	822814	object
n obs	839736	int64	moid	822814	float64
H	837042	float64			

Table 1: Data structure of asteroid data set

The data set comes from the Jet Propulsion Laboratory (JPL) of California Institute of Technology which is an organization under NASA. The original data set contains over 800,000 asteroids with 27 features. The majority of the observations; however, lack the diameter. Since we will be creating a supervised learning model, we will only be looking at the observations that do contain the diameter.

2.1 Data Cleaning

Table 1 shows the structure of the data obtained from JBL. Only 137,681 observations contain the diameter, so the rest of the observations will not be used to train the model. The following data cleaning processes were also taken:

- Dropped arbitrary features (asteroid name and condition code)
- Dropped features containing too many missing values
- Encoded categorical features
- Filled remaining missing values with feature mean

The cleaned data set has a total of 137,680 observations and 16 remaining features that can be used for training.

2.2 Summary Statistics and Correlation

Table 2 shows the summary statistics for each remaining numerical feature.

feature	a	e	i	om	w	q	ad
mean	2.81	0.15	10.35	169.83	181.90	2.40	3.23
std	1.52	0.08	6.84	102.71	103.56	0.52	2.9
min	0.63	0.00	0.02	0.00	0.00	0.08	1.00
25%	2.54	0.09	5.12	82.33	91.94	2.07	2.86
50%	2.75	0.14	9.39	160.44	183.66	2.36	3.17
75%	3.09	0.19	13.74	256.28	271.76	2.69	3.47
max	389.15	0.98	170.32	359.99	360.00	40.47	772.20
feature	per y	data arc	n obs	H	diameter	albedo	moid
mean	4.88	8908.70	659.41	15.18	5.48	0.13	1.42
std	25.53	6147.38	581.88	1.40	9.37	0.11	0.51
min	0.50	1.00	5.00	3.20	0.00	0.00	0.00
25%	4.04	6266.00	214.00	14.40	2.77	0.05	1.08
50%	4.56	7497.00	483.00	15.30	3.96	0.08	1.38
75%	5.44	9652.00	958.00	16.10	5.74	0.19	1.70
max	7676.74	72684.00	9325.00	29.90	939.40	1.00	39.51

Table 2: Summary statistics for features

Table 3 show the linear correlation between the diameter and the predictor variables. It doesn't appear that the data has high correlation, with the largest coefficient being approximately 0.57.

feature	correlation	feature	correlation
diameter	1.000000	ad	0.093622
H	0.568580	i	0.052827
data arc	0.493088	e	0.049180
n obs	0.386325	per y	0.049053
moid	0.333046	neo Y	0.036215
q	0.330317	pha Y	0.019629
a	0.145027	w	0.002909
albedo	0.107493	om	0.001191

Table 3: Correlation between diameter and features

3 Methods & Analysis

3.1 Pre-Processing

Before I began training our model, I split the data into a train and test sets (80:20 split) and separated the targets from the predictors. Because the predictor variables have such a wide range of values and variability, I normalized the predictors using a standard scaler. Since there there are quite a few observations, I decided to use a hold-out validation set to observe the performance during the training process. I made the validation set 25,000, roughly the same amount as the test set.

3.2 Model Architecture

Since the diameter of an asteroid is a continuous variable, my deep neural network is a regression model. The input layer will be the predictor variables and the output layer should be the predicted diameter. The model uses the following architecture:

- Loss function: mean square error (mse)
- Optimizer: ADAM (learning rate = 0.01)
- Metric: mean absolute error (mae)
- Hidden layers (activation): dense (rectified linear unit function)
- Output layer (activation): one unit (none)

The hyper-parameters that I considered for the tuning process were:

- batch size
- number of hidden layers
- size of hidden layers
- number of dropout layers
- number of epochs

3.3 Model Development

The goal is to over-fit the data and then scale down until we achieve the optimal model that generalizes the data. The first model needs to be large enough that it over-fits the train set.

First model:

- batch size: 64
- number of hidden layers: 4
- size of hidden layers: 128, 128, 64, 64
- dropout layers: none
- epochs: 50

Figure 1 shows the train and validation loss of our first model. There is quite a bit of noise in both the train and validation loss, but it definitely is over-fitting on the train set.

3.4 Hyper-parameter Tuning

The next step is to begin tuning the hyper-parameters until our model performs as well as possible, without over-fitting or under-fitting.

My first goal during the tuning process was to eliminate the noise in the training loss. The noise was likely a consequence of the small initial batch size. The variability between batches was too high. So on the second attempt, I increased the batch size and re-ran the same model.

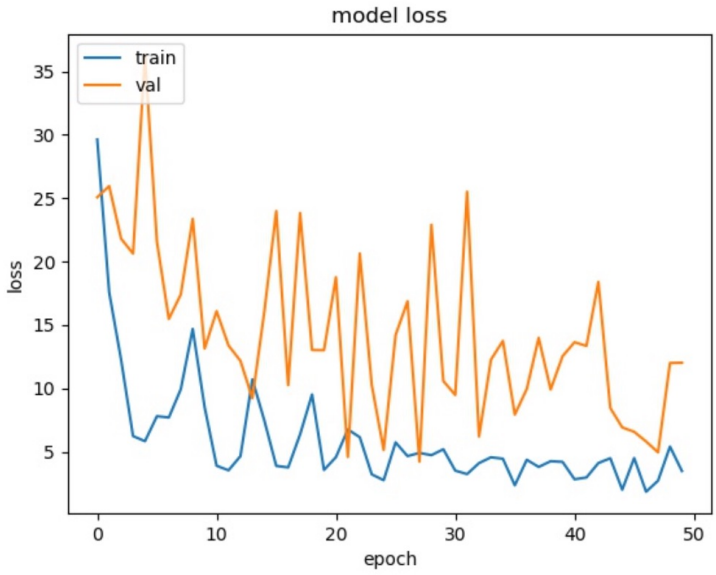

Figure 1: Train loss vs. validation loss of first model

Second Model:

- batch size: 512
- number of hidden layers: 4
- size of hidden layers: 128, 128, 64, 64
- dropout layers: none
- epochs: 50

Figure 2 shows the loss from our second model. Increasing the batch size greatly reduced the noise, as I had hoped for, but it doesn't appear to have reduced the over-fitting.

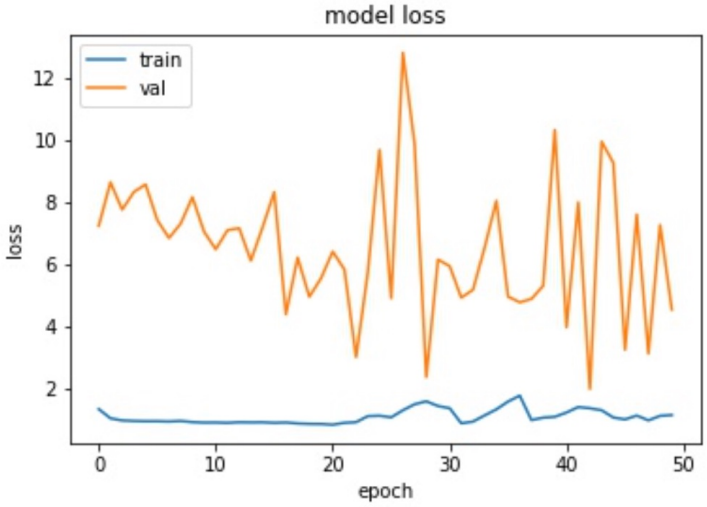

Figure 2: Train loss vs. validation loss of second model

With the noise issue taken care of, I began to downsize the model until it no longer over-fit the train set. The first adjustment I made was to the actual size of the hidden layers. I halved each of their output size and trained the model again.

Third model:

- batch size: 512
- number of hidden layers: 4
- size of hidden layers: 64, 64, 32, 32
- dropout layers: none
- epochs: 50

The third model loss is shown in Figure 3. It did manage to reduce the over-fitting, but not nearly enough. I also noticed that the model begins converging very early, around 10 epochs or so.

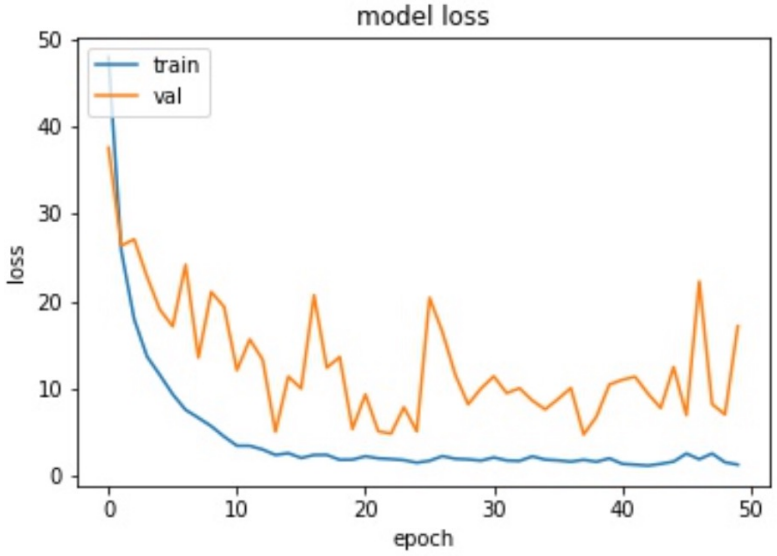

Figure 3: Train loss vs. validation loss of third model

Since the train loss is converging pretty early on, I decided to reduce the model size again. This time I actually removed a hidden layer entirely and reduced the size of the last hidden layer.

Fourth model:

- batch size: 512
- number of hidden layers: 3
- size of hidden layers: 64, 32, 16
- dropout layers: none
- epochs: 50

The results from the fourth model are plotted in Figure 4. The over-fitting was reduced again, but still prevalent. The train loss did take longer to converge though, which was expected.

Rather than reducing the model size on the fifth training run, I added a dropout layer between each hidden layer. I chose a relatively small dropout rate of 0.2.

Fifth model:

- batch size: 512
- number of hidden layers: 3
- size of hidden layers: 64, 32, 16
- dropout layers: 3 (rate: 0.25)

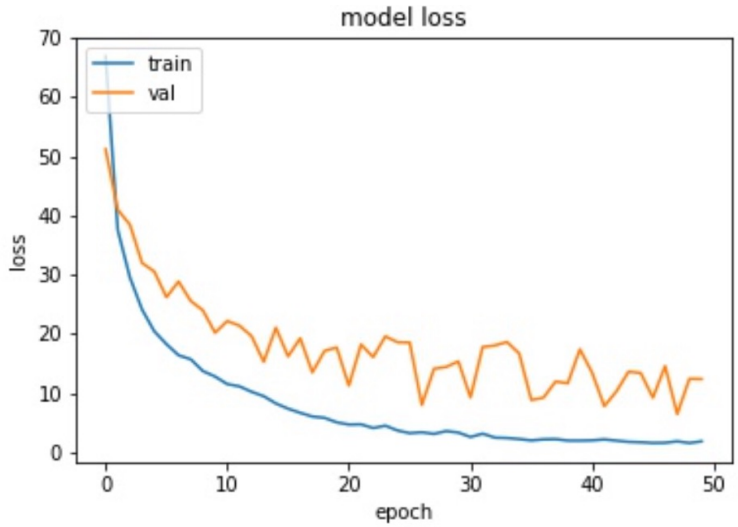

Figure 4: Train loss vs. validation loss of fourth model

- epochs: 50

As you can see in Figure 5, the dropout layers reduced the amount of over-fitting drastically. The validation loss follows the train loss very closely. The model performance begins to converge around 30 epochs.

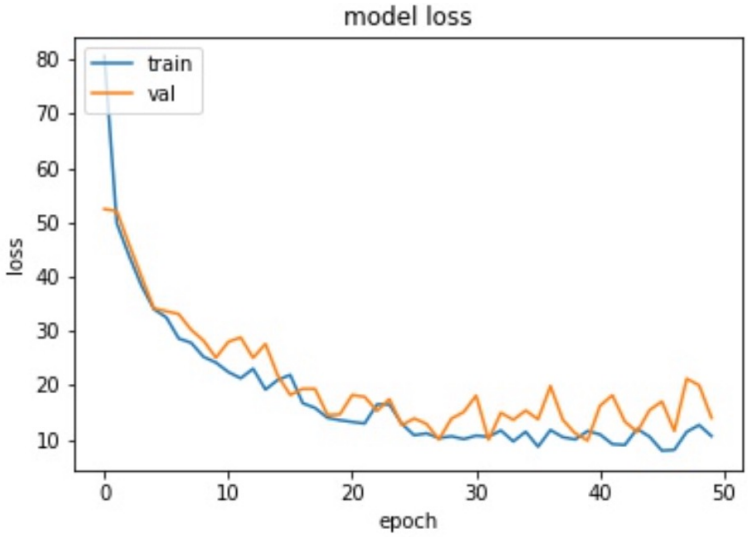

Figure 5: Train loss vs. validation loss of fifth model

For the last training cycle, I reduced the number of epochs to 30, as that was were the model began tapering off.

Final model:

- batch size: 512
- number of hidden layers: 3
- size of hidden layers: 64, 32, 16
- dropout layers: 3 (rate: 0.25)
- epochs: 30

The tuned model performed very well. After evaluating on the reserved test set, I got the following values:

- $mae = 0.969$
- $R^2 = 0.895$

3.5 Feature Engineering

The original model uses only the features provided by JPL. To increase model performance, I performed feature engineering to create new representations of the data. The new features that I decided to use were the log transforms. I took the log of every numeric feature and added it to the predictor set. I also transformed the diameter, so the new target is the log of diameter. The top 10 correlated features are shown in Table 4. The correlation in the data after the engineering is significantly higher than the previous predictor set.

feature	correlation	feature	correlation
log(diameter)	1.000000	log(q)	0.543708
log(H)	0.835468	log(moid)	0.528661
H	0.832306	q	0.522391
log(a)	0.563592	moid	0.521080
log(per y)	0.563592	data arc	0.519099

Table 4: Correlation with log(diameter)

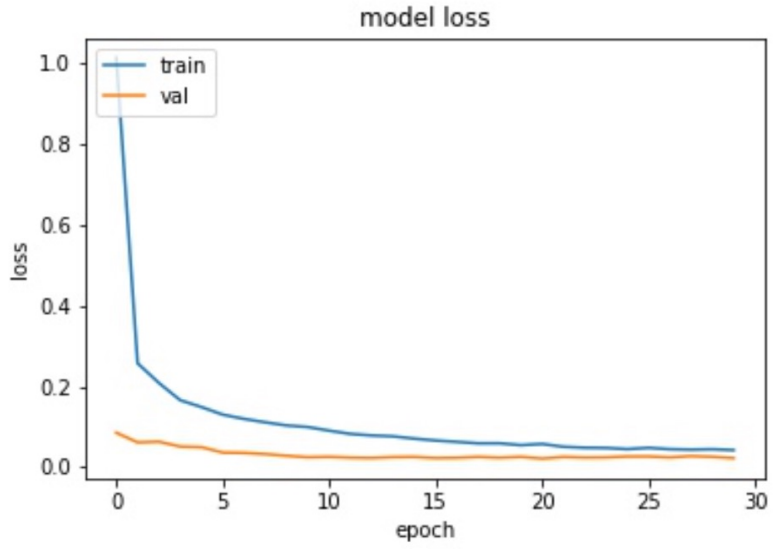

Figure 6: Train loss vs. validation loss of original model w/ feature engineering

For the first training cycle, I used the already tuned model. The loss is shown in Figure 6. As expected, the model loss converges almost immediately. The model is unnecessarily complex for the engineered data. I simplified the model by reducing the number of hidden layers, decreasing the size of the hidden layers, and removed the dropout layers. I also noticed the lack of any noise in the loss, so I decreased the batch size as well. The model loss is shown in Figure 7.

Simplified model:

- batch size: 64
- number of hidden layers: 2
- size of hidden layers: 16, 8
- dropout layers: none
- epochs: 10

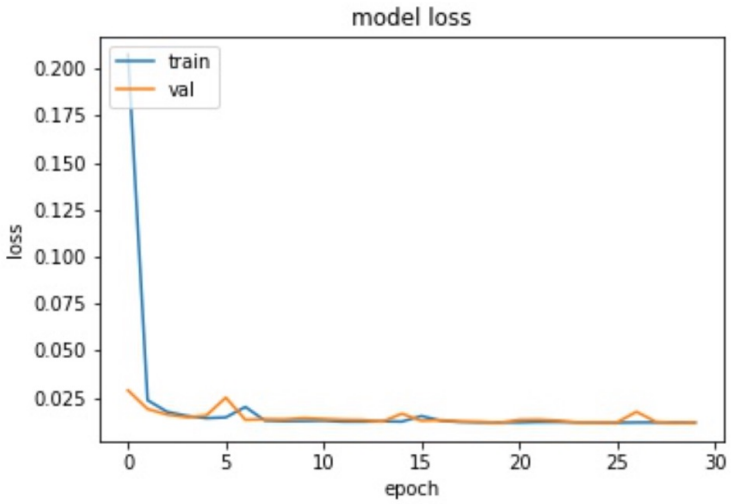

Figure 7: Train loss vs. validation loss of simplified model w/ feature engineering

Even after simplifying the model, the loss converges very quickly (around 4 epochs). I re-trained the same model again, but only with 4 epochs.

Simplified model w/ less epochs:

- batch size: 64
- number of hidden layers: 3
- size of hidden layers: 64, 32, 16
- dropout layers: 3 (rate: 0.25)
- epochs: 4

I evaluated the model on the test set and got the following values:

- $mae = 0.078$
- $R^2 = 0.967$

4 Discussion

The original model I developed performed reasonably well, with $R^2 = 0.895$. After feature engineering, I was able to create a much more simple model that performed even better, with $R^2 = 0.967$. This goes to show how important feature engineering is to deep learning, and more generally, machine learning as a whole.

References

Chollet, Francois. 2018. *Deep Learing with Python.* Manning.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning.* The MIT Press.

n.d. https://ssd.jpl.nasa.gov/sbdb_query.cgi.