PROJECTILE PATH TRACE

Sachin Vernekar
Cisco Systems, India

1. Data Exploration

First and the foremost step is to plot the sequences for x and y direction separately.

Looking at the plot Figure 1 for time-index vs x, it is pretty much a linear relation, hence a linear regression is the way to go. And there seems to be no significant number of outliers, hence no need to do any further tests for outliers or do any preprocessing for it. Plot also suggests that slope of line segment is different for each line segment, hence should depend on 1st 2 points in the sequence that are given. Since 1st point is 0, it is excluded from consideration as a feature.

Equation of straight line passing through origin is,

$$y = mx$$

Here m is dependent on 1st point, x(t1) in the projectile path. So it evident that equation has to be of the form

$$x(t) = x(t1) * t$$

Plot for y is very much a parabola, hence linear regression should suffice.

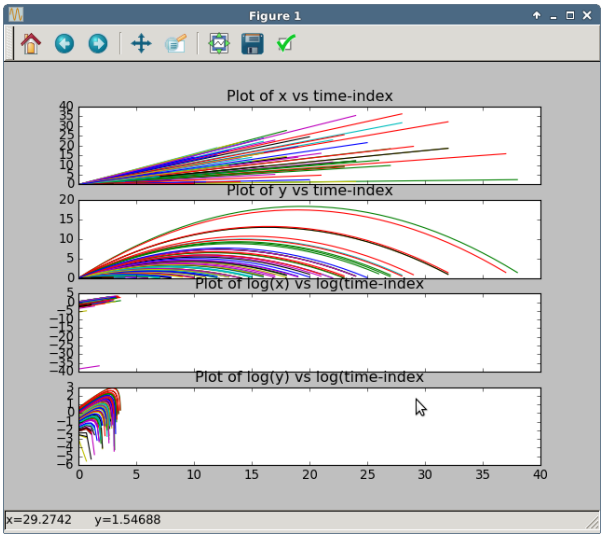

Figure 1. Plot of projectile paths for different launch angle and velocity.

2. Training

Training was done using **keras** [2] package in python. Horizontal distance x and vertical distance y were trained separately. There are 3 different versions of training, namely, linear regression on non-transformed output/input, linear regression on log-transformed x, LSTM. I skipped scaling input and output as the variance or range of data isn't large. If needed, a MinMax scaler from sklearn package can be used. In all the training methods, to evaluate training for its reliability, 2 sets of data were created with 70:30 training and validation split, with each set having non-overlapping validation data. Validation samples are sampled randomly from dataset. Mean-squared-error (MSE) was monitored for early_stopping, and percentage RMSE (PRMSE), is taken as a criterion to weigh the models in final ensemble of 2 models.

$$PRMSE = RMSE * 100/mean(output)$$

2.1. Linear Regression

The features taken are as follows to predict x(t):

$$\{x(t1), t, x(t1) * t\}$$

After training with **adam** optimizer (lr=0.001), both the train and validation mean square errors reach almost close to 0 as can be inferred from Figure 2. If we see the feature importance, the most prominent(or the only prominent, hence other features can be excluded) feature is *x(t1)*t*, which is consistent with the assumption made by looking at Figure 1.

To train for y(t), the features are as follow:

$$\{t, 0.01 * t^2, y(t1), y(t1) * 0.1 * t\}$$

Both train and validation errors are close to 0. Features, 2 and 4 have highest importance, which is consistent with parabolic graph of y.

2.2. Linear Regression - log transformed x

Just to try out, trained with log transformed input and output. The features taken are as follows to predict log(x(t)):

$$\{log(t), log(x(t1))\}$$

Here too, both train and validation errors are close to 0, and both features have equal importance. Tried log-transformation for y as well, but results aren't good and it is expected for a parabolic curve.

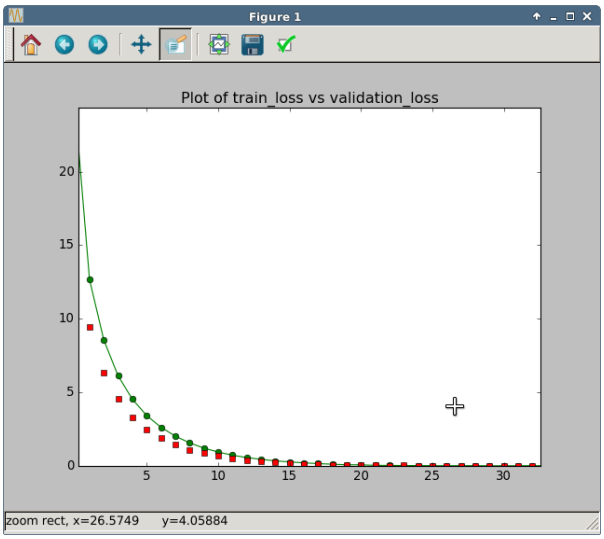

Figure 2. Plot of train–loss vs validation-loss

2.3. LSTM

For general sequential data in question, if the plot of data doesn't convey much, I would go for an RNN. Hence I have trained an LSTM network, which also gives a very small PRMSE. LSTM is generally used to train long sequences where there is possibility of vanishing gradients if highest Eigen value of weight matrix is small. But in the current case, the sequences aren't that long, hence simple RNN can also be used. Keras provides LSTM package to implement sequence learning for sequences with varying length by padding 0s to data.

To train for x, a look_back (window_size-1 in the code) of 2 is used, for which padding will not be necessary as we have first 2 values of any projectile path. To train for y, look_back is 5, and hence padding is necessary. look_back values and number of memory cells in LSTM were optimized by trying over a range of values. PRMSE values for both x and y training are close to 1, which is acceptable, but aren't as good as those obtained from linear regression. I also tried multi-dimensional LSTM, but results were not any better (not sure if it's due to limitation on keras package as I expected it to do better.)

One unique aspect of this problem is that, this is a case of multiple sequence learning, and these sequences are somehow related! Generally in multiple sequence learning problems, if the sequences are unrelated, we simply go for implementing different LSTMs for different sequences. Keras doesn't provide a way to train multiple sequences that are related in a single LSTM network.

Stateful LSTMs are not used as there are multiple sequences and individual sequence lengths are not as large.

3. Results

Values of x and y for a projectile launched at an angle of 45 degrees and 10m/s velocity are saved in results file for different approaches used and when compared with actual results deduced from projectile motion equations, they are in close sync as can be inferred from Table 1.

Table 1. Models and validation PRMSE

Model	X	Y
Lin Reg	close to 0	close to 0
Lin Reg - log	close to 0	close to 0
LSTM	0.83	1.21

4. Question - Answers

This section discusses unanswered questions from the assignment.

- Can your model predict projectiles launched at arbitrary angles and velocities equally well (or badly)?

From the PRMSE values, it is quite clear, if the test data falls in the range of training data used, predictions will be as accurate as with validation set. This can further be supported with n-fold cross validations if needed.

- What assumptions does your model make?

Important one is that, horizontal and vertical distances - x and y are not related. Other assumptions are general assumptions, that test data also falls in the range of training data (but for linear regression model - it seems not necessary) Other assumption is that the problem is solvable :) that is there is some hypothesis function that fits the data.

- Will your approach/model change if we hadn't told you that data was from projectile?

Everything depends on how the plot of data looks like, or results obtained through some simple exploratory analysis of data (usually I use t-SNE to do that). Hence as data looked very simple to fit using linear regression, I went for it. Else as explained previously under LSTM section, I would go for an RNN. I would have also not traine for x and y independently. I would have tried to train then together (the idea isn't concrete yet). I

- Did you refer to any relevant literature while solving this problem?

Yes, revised a bit of LSTM to handle cases where there are multiple sequences to learn that are related. Tried to implement it in keras without much luck. Otherwise, kaggle experience helped me a lot.

- If you were given enough time, how would you improve the model?

I would try to find a solution to training of multiple sequence that are related (keras lacks this feature). I would also implement a stacked LSTM model to see if accuracy gets any better. I would also see if I can chose validation set any better, may be by uniformly picking samples from the distribution.

References

[1] https://github.com/sachinvernekar/projectilePathTrace
[2] https://github.com/fchollet/keras
[3] https://github.com/sachinruk/PyData_Keras_Talk/
[4] http://scikit-learn.org/stable/
[5] https://github.com/danielfrg/tsne
[6] https://github.com/danielfrg/tsne