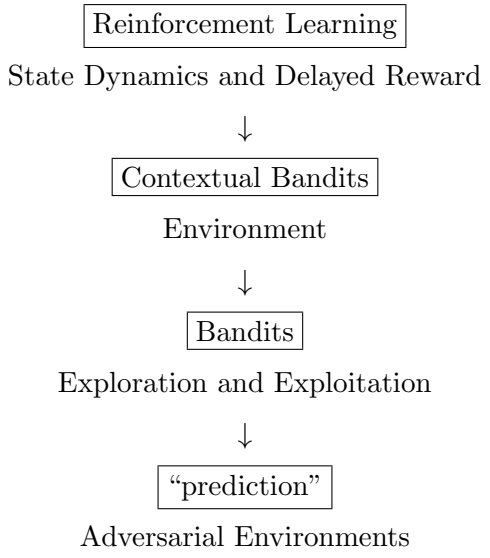
EE290S - Lecture 2 Scribe Notes

Jessie Liu
Alex Beltran

28 August 2018

1 Overview

Reinforcement learning can be viewed as a progression of models as more players and dynamics are added. We can break this down by moving through the following diagram:



Without state dynamics (where an action made by an agent can change the state of the environment), reinforcement learning becomes contextual learning. Further, if we remove the factor of the environment, contextual learning becomes a multi-armed bandit problem. Finally, with only adversarial environments, we are left with predictions.

2 Bias/Variance Tradeoff

One of the most fundamental measures of accuracy in machine learning are the errors made on the training and test sets. The disparity between these is best explained by the bias/variance tradeoff. In these types of errors, we are concerned with the error in the predicted outputs, $\widehat{y}$ as a function of model complexity. Consider a general model (h_d) of d dimensions trained on a random subset of the training data, (x_i, y_i) with noise η_t:

$$y_t = h_d(x_t) + \eta_t$$

The mean squared error can be broken down into two components, the squared bias of the model and the variance of the model:

$$\mathbb{E}[(h_d(x_t) - y_t)^2] = \mathbb{E}[\mathbb{E}[h_d] - f(x_t)]^2 + Var[h_d(x_t)] + \sigma^2$$

The tradeoff comes from the amount of data and the amount of noise observed. As we increase the amount of data, we increase the bias of our model while decreasing the variance. High bias can lead to underfitting while high variance can lead to overfitting. There will always be some amount of irreducible error, σ^2, but the error caused by the bias-variance tradeoff can be mitigated through regularization efforts to prevent overfitting as we gain more data. Some common forms of regularization include adding a regularizer to the cost function, as in ridge or LASSO regression. In the case of neural networks, we may also use dropout to regularize, where node outputs of layers are randomly set to 0 at a specified rate. In a way, these methods of regularization to prevent overfitting, try to seek a sparse solution (using the ℓ^1 norm in LASSO, for example).

3 Universal Function Approximators

In machine learning, there exist various universal function approximators. That is, there are several methods that, with the right parameters, are able to approximate any function. Each method has various amounts of parameters that will influence the model complexity, and consequently the runtime complexity.

1. Linear Models with pre-specified universal features
    (a) Polynomials
    (b) Fourier bases
    (c) Wavelets
    (d) Dictionaries
2. Neural Networks with non-linear activation functions

3. Trees and additive trees

4. Nearest neighbors models

It is additionally important to note the difference between model parameters and hyperparameters. Parameters are attributes of the model that are trainable, such as the weights in a neural network or the coefficients in linear regression. Hyperparameters are variables that help in the optimizing of the model and are often manually specified. These include variables such as the learning rate in gradient descent or the number of k-nearest neighbors.

4 Approximation/Estimation Tradeoff

Given a model architecture and some data (x,y) there exists a model that "best" captures the relationship between x and y. However, it is unrealistic to learn this "best" model in a real-world setting as we only have finite data available for training. Instead there is some "learned" model that is computed from available data and our goal is for the "learned" to be as close to the "best" model as possible. It is also desirable that the "best" model to closely follow the true model. To introduce some notation, we have denoted two different y's predicted from each model:

$$x \rightarrow \boxed{\text{Best}} \rightarrow y^*$$

$$x \rightarrow \boxed{\text{Learned}} \rightarrow \widehat{y}$$

If we evaluate these metrics for some data (x, y) we can calculate:

$$\textit{prediction error} = y_{true} - \widehat{y} = (y_{true} - y^*) + (y^* - \widehat{y})$$

And so our prediction error can be described as a sum of our approximation error ($y_{true} - y^*$) and estimation error ($y^* - \widehat{y}$). The approximation error is controlled by hyperparameters, e.g. choosing a more appropriate learning rate for gradient descent in a neural network.

4.1 An Example with Hooke's Law

In the simplest terms, approximation error can be described as how closely the "best" model of a pre-specified type can predict the data. Estimation error then describes how close the "learned" model is to this "best" model.

In order to illustrate this point, consider the following. We have a spring for which we want to determine its spring constant. To do so, we take measurements of the force required to displace the spring different lengths. From physics, we know that the data should follow Hooke's Law, which states that displacement and force exhibit a linear relationship governed by the spring constant. Suppose that we aren't

aware of the linear relationship and explore the modeling space with a variety of polynomial models.

Let's first visualize our data. We have synthetically created this data using an example spring constant, Hooke's Law, and have added Gaussian noise. Our training data spans displacements up to 5 meters. In Fig. 1, we see that our data does visually follow a linear relationship, with the noise adding some variability. We can then take our x and y data and fit a third order polynomial. We fit two models here, our best model using all the available training data and our learned model using a subset of the training data. Fig. 1 shows the predictions using these two models on the range that we trained on (0-5 m) as well as new displacements values (up to 8 m). At first glance, there doesn't seem to be much of a difference between these two models, but a closer look can show up the approximation/estimation tradeoff for both the training and testing scenario.

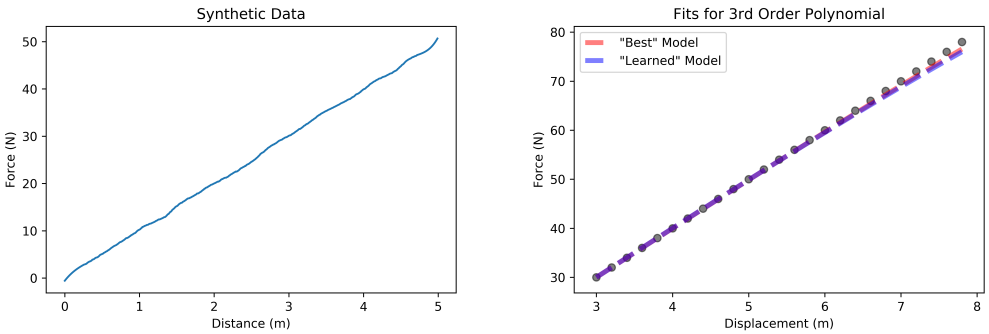


Figure 1: Left, visualization of a synthetic dataset, created by using an example spring constant with the addition of normal Gaussian noise, and third order polynomial fits. Right, visualization of predictions made by a third order polynomial model using either all of the available data ("best" model) or using a subset of the available data ("learned" model).

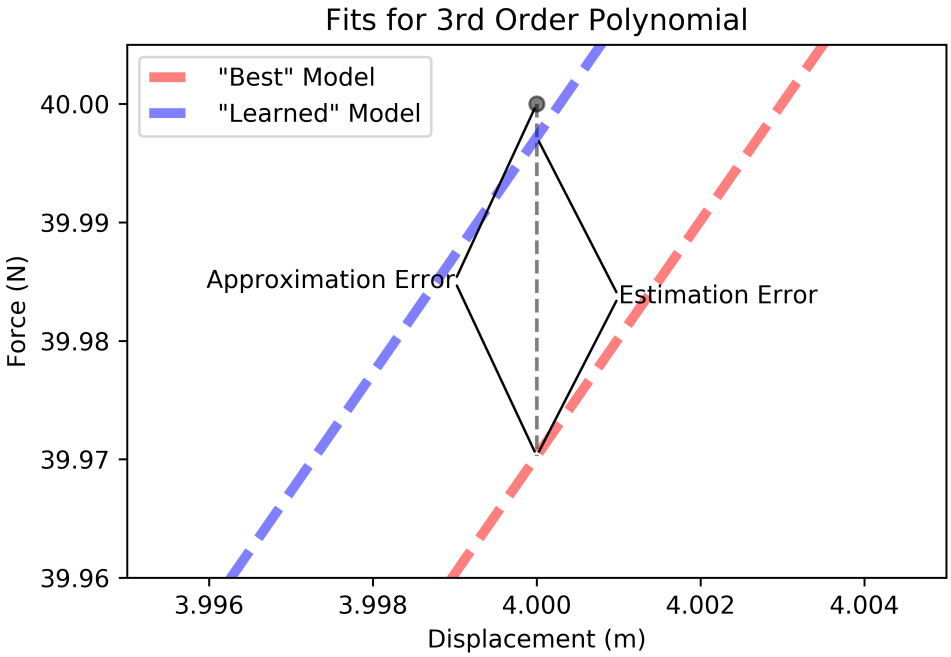


Figure 2: Close-up of the difference between the "best" and "learned" model fits on a data point within the range of the training data. For this particular point, approximation error is greater than estimation error.

In Fig. 2, we look closer at the predictions made by the best and the learned model in relation to the true data (which we have plotted as a black point). We can see here the approximation error (the difference between the best model and the true data) and the estimation error (the difference between the best and learned models).

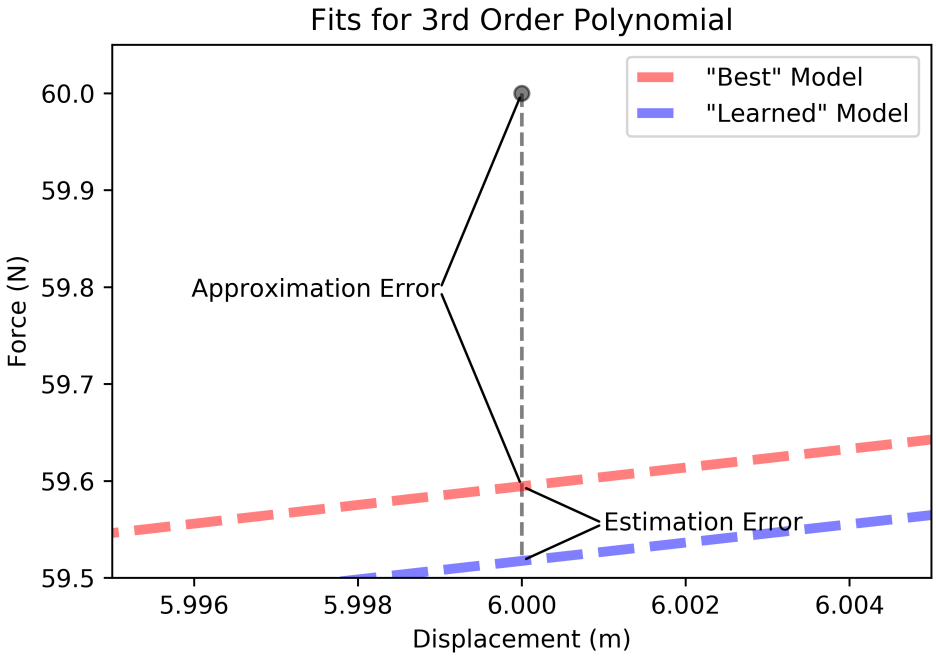


Figure 3: Here, we can see that on completely new data, the "best" model performs better than the "learned" model. This is most likely due to the amount of training data.

In Fig. 3, we look at predictions on a data point from the test set for both the best and learned models. Here, we see that approximation error is greater than estimation error. We also note that the the best model outperforms the learned model in predicting the force for this particular point. We can use this observation to motivate a closer study of bias/variance and approximation/estimation.

We can then study the effects of model order on both the approximation-estimation and training-test error tradeoffs. We fit several order polynomial models on training data and calculate the training and test error (train and test sets were created via an 80/20 split). In Fig. 4, we can clearly see that as we continue to increase model order, we increase overfitting. As model order increases, we see that the training error decrease and the test error increases. In other words, increasing model order decreases our bias while increasing the variance of our predictions.

Figure 4: Although increasing the model order seems to decrease the training error, we can see that it simultaneously is increasing the testing error. This is a sign of overfitting, with high model orders, the model may begin to fit noise as opposed to fitting the true underlying data.

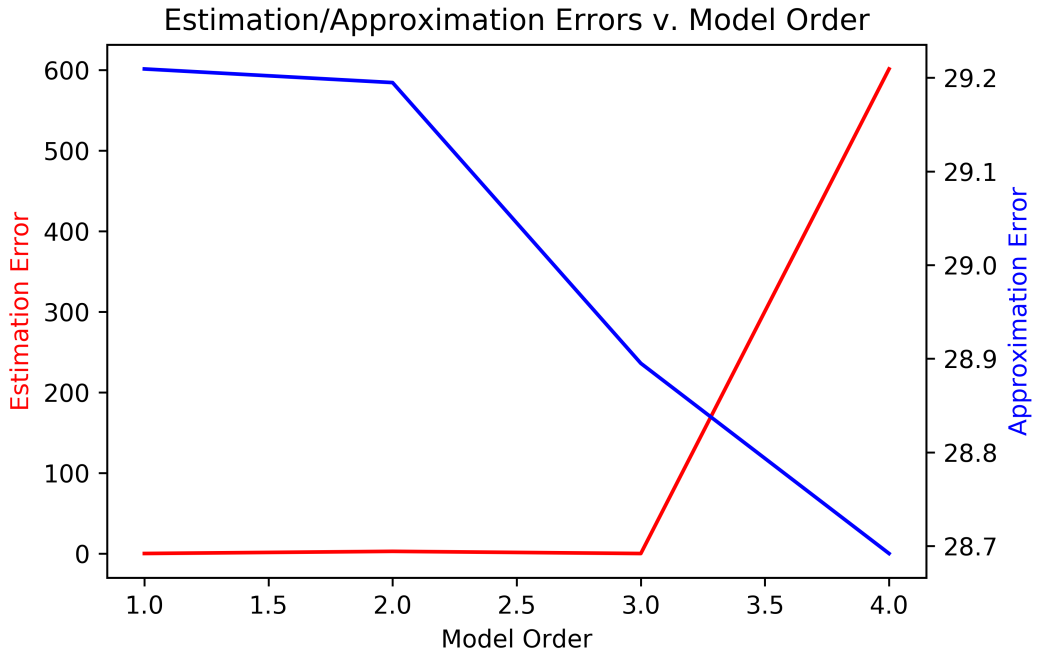


Figure 5: As we increase the model order and overfit to the data, our ability to approximate gets better (i.e. approximation error decreases). However, our estimation error (difference between models using some versus all of the data) increases. This demonstrates that higher model orders aren't able to generalize as well as lower model orders.

We can additionally look at the approximation and estimation errors as we increase model order. In Fig. 5, we see that there is a similar tradeoff observed. In both Fig. 4 and Fig. 5, we can see the tradeoffs and considerations that must be made when choosing how to model data.

4.2 When is learning easier?

Approximation/estimation can be described from an SNR perspective where we have a linear model with Gaussian noise:

$$y = wx + N$$
$$N \sim \mathcal{N}(0, 1)$$

If we suppose that we have n samples (x_i, y_i), then the ordinary least squares (OLS) solution for $\widehat{w}$ is given by:

$$\widehat{w} = \frac{\sum_{n=1}^{n} x_i y_i}{\sum_{n=1}^{n} x_i^2} = w + \frac{\sum_{n=1}^{n} x_i N_i}{\sum_{n=1}^{n} x_i^2}$$

It then follows that the difference between the w and $\widehat{w}$ is a random variable as described by:

$$w - \widehat{w} \sim \mathcal{N}(0, \frac{1}{\sum_{n=1}^{n} x_i^2})$$

As we average over more points, the variance of the noise becomes smaller. Extrapolation is difficult but interpolation is much easier; if we are able to better characterize the noise in our data, we can achieve a truer model.

5 Approximation/Estimation Beyond Model

Beyond the idea of model order, we can consider the strength of a prior on the approximation/estimation tradeoff. By prior, we mean some description of what the model weights should look like. For example, if we were to take a simple linear model we could enforce a Gaussian prior:

$$y = wx + N$$
$$Prior : w \sim \mathcal{N}(0, \sigma^2)$$

Imposing a prior can be useful in many domains if there is previous knowledge that can help guide what we expect to find or if there are phenomenons that are known to govern the underlying processes that we wish to model. For example, one might prefer a prior that attempts to enforce low energy consumption for a physical system. By injecting this prior knowledge into our model, we can coerce it into a form that agrees with the domain.