International Journal on Artificial Intelligence Tools
© World Scientific Publishing Company

POLYNOMIAL REGRESSION WITH AUTOMATED DEGREE: A FUNCTION APPROXIMATOR FOR AUTONOMOUS AGENTS

DANIEL STRONGER and PETER STONE

Department of Computer Sciences, The University of Texas at Austin
C0500, 1 University Station
Austin, TX 78712, United States of America
{stronger,pstone}@cs.utexas.edu

In order for an autonomous agent to behave robustly in a variety of environments, it must have the ability to learn approximations to many different functions. The function approximator used by such an agent is subject to a number of constraints that may not apply in a traditional supervised learning setting. Many different function approximators exist and are appropriate for different problems. This paper proposes a set of criteria for function approximators for autonomous agents. Additionally, for those problems on which polynomial regression is a candidate technique, the paper presents an enhancement that meets these criteria. In particular, using polynomial regression typically requires a manual choice of the polynomial's degree, trading off between function accuracy and computational and memory efficiency. Polynomial Regression with Automated Degree (PRAD) is a novel function approximation method that uses training data to automatically identify an appropriate degree for the polynomial. PRAD is fully implemented. Empirical tests demonstrate its ability to efficiently and accurately approximate both a wide variety of synthetic functions and real-world data gathered by a mobile robot.

1. Introduction

In order for an autonomous agent to robustly interact with its environment, it is valuable for it to be able to learn the relationships between relevant environmental variables. For example, as a mobile robot moves over different terrains, it may need to learn a new function from its motion commands to the corresponding velocities for each new surface it encounters. Similarly, a software travel agent may need to learn how the prices of different airline tickets vary with respect to time.

These relationships can be represented by function approximators. A function approximator learns the relationship between a dependent variable and a set of independent variables on the basis of a series of training examples. In this paper, we consider learning functions of one variable. Even in cases where an agent may not have access to explicit training examples, function approximation systems can sometimes be used. For example, a legged robot can use function approximators to simultaneously learn one model of its walking actions and one of its visual sensor, without labeled training data for either model [1].

An autonomous agent needs its function approximator to satisfy certain constraints that may not be necessary in a traditional supervised learning setting. In particular, it must be:

- **Flexible:** Widely applicable without tuning parameters.
- **Efficient:** With respect to computation time and memory.
- **Robust:** Able to handle varying amounts of random noise.

These criteria are described more precisely and motivated in the following section.

2. Motivation

For an agent to be truly autonomous, its function approximator must be *flexible.* Specifically, since any parameters it has cannot be manually tuned for each new function, they must have pre-set values that are effective over the entire range of potential functions.

The function approximator must also be *efficient.* An autonomous agent has limited time and memory to allocate to many different computational problems, and since it may need to approximate many different functions, each one must take up as little time and space as possible. This constraint implies that the method should take less time for simpler target functions. For space, the algorithm must use an amount of storage space that remains bounded as the amount of input data increases. This requirement is necessary in light of the virtual "firehose of experience" [2] to which agents in the real world may be subject.

Finally, agents often receive data through noisy sensors, and there can be random noise in the dynamics of the world. The function approximator must be *robust* to these effects, differentiating between random noise and genuine variation in the target function.

There are many methods in common use for approximating functions. However, we are not aware of any current method that is able to satisfy all three of the constraints mentioned above. For example, the Nearest Neighbor method [3] stores all of the data points that are encountered, and evaluates a given input based on the test points it is closest to in input space. This method does not satisfy the efficiency constraint. Although there are enhancements to Nearest Neighbor [4] that enable it to store only some of the points, doing so normally introduces parameters that need to be manually tuned.

Two other common function approximation methods are backpropogation of a neural network with a fixed topology [5,6] and radial basis functions (RBFs) [7,8]. These methods do not satisfy the flexibility constraint, as there are a number of parameters that must be set manually, such as the learning rate and the number of hidden nodes in the network, or the kernel widths. More specifically for back-propagation, a network with any *fixed* number of hidden nodes will be unable to fit sufficiently complex functions, despite the fact that bounded continuous functions can be approximated arbitrarily closely by a neural network with one layer of hidden nodes [9,10]. Similarly for RBFs, arbitrary functions can be approximate, but only when the kernel widths can be individually specified [11,8].

One particularly straightforward function approximator is *polynomial regression.*

Polynomial regression has some drawbacks which lead statisticians to often favor other methods, such as cubic splines [3]. First, as the polynomial degree increases, the regression curve can sometimes overfit the data and oscillate wildly between the data points. Furthermore, the regression can become numerically unstable, especially if the degree is high or the function domain is not centered at zero. However, in the presence of enough data that overfitting is not a concern, and when the degree and domain of the regression can be restricted to limit the numerical instability, the simplicity and efficiency of polynomial regression can make it the method of choice. As outlined above, some problems faced by autonomous agents indeed have these properties. Other successful uses of polynomial regression include modeling a visual sensor [12] and speech recognition [13].

Even in situations where polynomial regression can be useful, it still relies on a manually chosen degree that is crucial to the function approximator's effectiveness. If the degree is too high, the regression can suffer from the drawbacks mentioned above, and in the context of autonomous agents, high degree polynomial regression carries a significant computational expense. On the other hand, if the degree is too low, it will not be able to represent the complexity of the function being learned. This reliance on the degree causes standard polynomial regression to violate the flexibility constraint discussed above.

This paper presents a technique, Polynomial Regression with Automated Degree (PRAD), that augments polynomial regression so as to satisfy all three constraints. PRAD is fully implemented and empirically validated on both synthetic data from a wide range of function complexities and noise magnitudes, and data collected from an autonomous mobile robot.

3. Finding the Polynomial Degree

Polynomial regression entails an inherent trade-off between accuracy and efficiency. As the degree of the polynomial increases, the fit grows in accuracy (up to a point), but the time and space needed increases as well. This section presents a method to automatically identify an appropriate degree for polynomial regression based on the training data.

PRAD takes pairs (x_i, y_i) and identifies the best fit polynomial $P(x) = \alpha + \sum_{k=1}^{d} \beta_k x^k$, where d is the degree of the polynomial. The best fit polynomial is defined to be the one that minimizes the total squared error $\sum_{i=1}^{n} [P(x_i) - y_i]^2$, where n is the number of data points encountered thus far. The best fit polynomial is computed by maintaining $3d+2$ sums, namely $\sum_{i=1}^{n} x^k$, where k ranges from 0 to $2d$, and $\sum_{i=1}^{n} yx^k$, where k ranges from 0 to d. These sums can be used to produce a function estimate after each data point [14].

PRAD finds the appropriate degree by starting with a first degree (linear) polynomial and continually monitoring the fit to see if the degree needs to be increased. If so, the regression is restarted with the degree incremented by one. To restart the regression, the sums are all reinitialized to zero. Another option would be to maintain a large number of sums from the beginning, depending on the relative importances of data efficiency and space efficiency. The degree continues to be incremented until a satisfactory fit is found.

In order to determine whether or not a particular degree is satisfactory, PRAD compares a *global prediction error* and a *local noise estimate*, two values that are continually maintained. A high global prediction error indicates a poor fit, suggesting that the polynomial degree should perhaps be increased. However, such an error might also be accounted for by a large amount of random noise in the observed data, in which case increasing the degree will not help. This comparison does not explicitly take resource efficiency into consideration, but it finds the lowest degree polynomial that achieves a satisfactory fit, and thereby implicitly balances computational costs against accuracy.

PRAD computes a best fit polynomial after each data point is encountered. This running function estimate is used to compute an error for each data point. These errors are accumulated into a root mean square error, which we denote as C. C is PRAD's global prediction error.

We assume that y_i can be expressed as $f(x_i)+\omega_i$, where f is the function PRAD is trying to approximate and ω_i is zero-mean random noise. The expected squared error, C^2, is $E([\hat{f}(x)-y]^2)$, where $\hat{f}$ is the current estimate of f. Since the noise is zero-mean and independent from the function error, this error can be decomposed as:

$$C^2 \approx E([\hat{f}(x)-y]^2) = E([\hat{f}(x)-f(x)]^2) + E(\omega^2) \tag{1}$$

PRAD tries to minimize the first term on the right, which represents the function estimate inaccuracy. The second term, which we denote by D^2, is unavoidable, since the noise generates error even in the presence of a perfect fit to the function. Therefore, it is desirable to be able to distinguish between these two sources of noise. PRAD makes this distinction by estimating D^2, the variance of the random noise, directly.

To estimate D^2, PRAD partitions the input space into a pre-set number (r) of equal small intervals. Each data point (x, y) is then classified according to which interval x falls into. The variance of y over the points in any given interval is a good estimate of the noise in the data at each point, because the variation in the underlying function is limited over such a short interval. PRAD maintains the variance of y over each interval and considers their average to be the mean squared noise, D^2. This method requires knowing the function domain, but if it is not known, the robot can first estimate the domain from a small amount of data and use that estimate.

From (1), we know that $C \geq D$. The ratio between these quantities is a measure of the extent to which the prediction error, C, can be explained by the random noise, D. Therefore, to determine whether it is necessary to increment the polynomial degree, PRAD compares C and D using a threshold ratio, $\theta > 1$. If $C > D\theta$, there is a significant amount of error that cannot be explained by random noise in the data. In that case, PRAD increments the degree by one.

The ratio between C and D is a good indicator of when to increase the polynomial's degree, but care must be taken to ensure that C and D have stabilized before this metric is used. The following measures ensure this goal. First, when very few data points have been taken into account, the resulting best fit polynomials can be very chaotic (or undefined). Thus PRAD does not begin accumulating the root mean square prediction error C until a sufficient number of distinct inputs have been encountered. This threshold amount is set to $3(d+1)$ (based on the idea that $d+1$ points are required to determine a d-degree polynomial uniquely). Furthermore, for inputs that are either the highest or lowest input seen so far, their prediction error is discarded because of the unpredictable nature of polynomial regression outside of its range of training inputs. Finally, to ensure that C and D are based on a sufficiently representative sample of the data, we prohibit incrementing the degree until a threshold number of prediction errors have been counted. This threshold is equal to $3r$.

Pseudocode for PRAD is depicted in Algorithm 1. The running function estimate, $\hat{f}(x)$, is computed after every data point. The *INITIALIZE* routine sets all of the sums used in computing the regression to zero. As each training pair (x, y) is *RECEIVED* by PRAD, the *INCORPORATE* routine updates all the stored sums appropriately.

PRAD uses four constants as summarized in Table 1. Their suggested values indicated in the table were chosen without extensive experimentation and are used in all of the experiments reported in this paper. These parameters replace a single parameter in conventional polynomial regression: the polynomial's degree. Nevertheless, we argue that PRAD is actually a significant improvement with respect to the flexibility constraint. In particular, one setting for these four variables is able to handle a wide range of function complexities and amounts of noise, whereas no one setting for the degree can do that. This hypothesis is supported empirically in the following section. Furthermore, the same setting of the constants is also effective in the robot experiments.

Notably, the method described here is not sensitive to the scale of the data. If all of the x or y are multiplied by a constant scaling factor, PRAD will settle on the same degree for the polynomial. This property contributes to the algorithm's ability to handle a wide range of functions and noise levels.

Constant	**Value**
Number of Intervals (r)	20
Degree Incrementing Threshold (θ)	1.3
Error Usability Threshold	$3(d+1)$
Interval Accumulation Threshold	$3r = 60$

Algorithm 1 Pseudocode for PRAD

```
Given: $\theta$, $r$, $Domain_{min}$, $Domain_{max}$
$degree \leftarrow 1$
loop
  $ShouldIncrementDegree = false$
  INITIALIZE $\hat{f}(x)$                    \\Running function estimate
  $n \leftarrow 0$                           \\Number of data points
  $ECounter \leftarrow 0$                    \\Number of prediction errors counted
  $ESquareSum \leftarrow 0$                  \\Total squared prediction error
  for $i = 1$ to $r$ do
    $ICounter_i \leftarrow 0$                \\Number of points in each interval
    $ISum_i \leftarrow 0$                    \\Sums used to compute interval variances
    $ISquareSum_i \leftarrow 0$
  end for
  while $ShouldIncrementDegree = false$ do
    $n \leftarrow n + 1$
    RECEIVE $(x, y)$
    if ($x$ is not the largest or smallest $x$ so far) AND
    (have received $\geq 3(degree + 1)$ different values of $x$) then
      $Error \leftarrow y - \hat{f}(x)$
      $ECounter \leftarrow ECounter + 1$
      $ESquareSum \leftarrow ESquareSum + Error^2$
      $C \leftarrow \sqrt{ESquareSum/ECounter}$
    end if
    INCORPORATE $(x, y)$ into $\hat{f}(x)$
    $i \leftarrow \lceil \frac{r(x - Domain_{min})}{(Domain_{max} - Domain_{min})} \rceil$
    $ICounter_i \leftarrow ICounter_i + 1$
    $ISum_i \leftarrow ISum_i + y$
    $ISquareSum_i \leftarrow ISquareSum_i + y^2$
    $Variance_i \leftarrow \frac{(ISquareSum_i - ISum_i^2/ICounter_i)}{ICounter_i - 1}$
    $D \leftarrow$ root mean of defined values of $Variance_i$
    if ($ECounter > 3r$) AND ($C > D\theta$) then
      $ShouldIncrementDegree \leftarrow true$
    end if
  end while
  $degree \leftarrow degree + 1$
end loop
```

4. Experimental Validation

This section sets out to verify that PRAD satisfies the criteria set forth in the introduction. Namely, it should be *efficient*, *flexible* and *robust* to noise. More precisely, just one setting of the algorithm's numerical parameters should be able to efficiently handle a wide range of different functions and amounts of random noise. To verify conclusively that these constraints are met requires extensive testing in a wide variety of domains. Here we verify that PRAD is effective in at least some cases, and present evidence that all constraints are met by implementing PRAD and testing it both on a varied suite of simulated test data and on a robotic scenario that has been addressed previously with polynomial regression.

4.1. *Simulated Data Results*

We have tested PRAD on a suite of synthetic test data designed specifically to represent a wide range of function complexities and amounts of noise. In this setting, PRAD produces an improvement over polynomial regression with any fixed degree. PRAD also compares favorably against another popular function approximation method, natural cubic splines with uniformly distributed knots. All tests were performed on a 2.00GHz Pentium 4 processor with 512MB of RAM.

The suite of test data varied along two dimensions: i) the *complexity* of the underlying function, and ii) the amount of random *noise* added to the function values. Each dimension varies over a range of ten values, yielding 100 test trials.

For each point, x is chosen randomly with a uniform distribution over the domain interval, which is set to $(-1, 1)$. The complexity of the function is represented by the variable a, which ranges over the integers from 1 to 10. In terms of a, the test function is:

$$f(x) = \sin\left(\frac{a\pi}{4}\left(x - \frac{1}{2}\right)\right) \tag{2}$$

The higher a is, the more complex the function is over the domain $(-1, 1)$. Note that a sine wave is used instead of a polynomial, to demonstrate the flexibility of polynomial regression. The sine wave is centered at $x = 1/2$ so that the function is not strictly odd or even, since an odd or even function would diminish the importance of polynomials of the opposite parity degree.

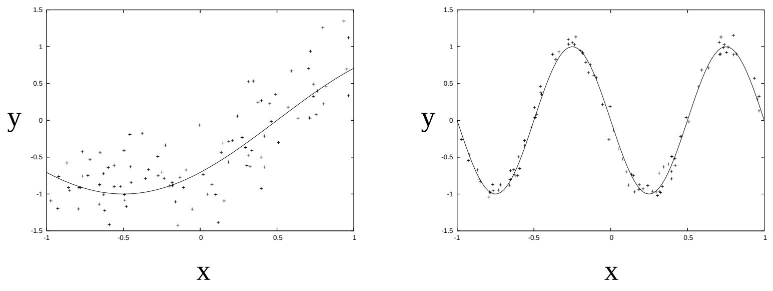

Fig. 1. Functions and 100 data points for two combinations of a and b, $a = 2$ and $b = 8$ (left), and $a = 8$ and $b = 2$ (right).

The amount of random noise is represented by the variable b, which also ranges from 1 to 10. A value of b causes a zero-mean, gaussian-distributed random offset with standard deviation $b/20$ to be added to $f(x)$ for each data point. Figure 1 depicts the function and 100 data points produced by each of two combinations of a and b. It is important to remember that the data presented to our simulated agent arrives as a stream. In general, the agent does not see all of the points at once, but rather must incorporate the new data efficiently as it comes in.

In each test trial, PRAD encounters 2000 data points. To evaluate PRAD, we compute the prediction error after the polynomial degree has stabilized. To measure this, PRAD's ability to increment the degree is turned off after 800 points. In tests where it was not turned off, it was very rare for the degree to increase beyond this point. The squared prediction errors after the 1000th point are then averaged to obtain the mean square error for that trial. This ensures that the measured errors are based on a significant amount of data with the selected degree.

The trials are also evaluated by the amount of time that they take. These are the amounts of time that a single run of PRAD takes to process all 2000 data points. Finally, in each trial the final degree used by PRAD is recorded. Each of the 100 data settings is tested 100 times, and the average errors, times, and degrees are used to evaluate the method. The results are shown in Figure 2.

A few general patterns are evident in the data. First, the average error is consistently quite close to the magnitude of the random noise, $b/20$. Therefore (as indicated by (1)), PRAD's function approximations are very close to the actual functions being estimated. In addition, the average degree used to represent the function increases as the function complexity increases. This pattern indicates that PRAD is adapting the polynomial degree as needed to represent the complexity of the function. Furthermore, for a given function complexity, as the amount of noise increases, the degree decreases slowly. PRAD uses slightly lower degrees in these cases because the noise masks some of the variation in the function. Finally, note that the time taken is highly correlated with the degree used, so that only as much time is used as needed for the complexity of the function being modeled.

For comparison, the same tests were run on polynomial regression with a fixed degree. For each degree from one to ten, the errors and times were measured for all 100 test cases. One straightforward measure of an algorithm's performance is its average time and error over all 100 trials. These averages are shown in Table 2.

Note that a baseline amount of error is 0.275, since that is the average of the magnitudes of random noise that are used. By comparison, PRAD's overall average error is 0.310, with an average degree of 4.20 and time of 111 ms. The lowest fixed degree that is able to attain PRAD's average error is 7, which takes 209 ms, or 88% more time.

A good measure of the function approximator's fidelity is the *excess error*, defined as the difference between the amount of average error and the magnitude of the noise. The average excess error is equivalent to the average overall error minus 0.275, so it behaves identically to the average error shown in Table 2. However, the

a	*b*	**Err.**	**Deg.**	**ms**		*a*	*b*	**Err.**	**Deg.**	**ms**
1	1	0.06	1.25	47		2	1	0.05	2.95	81
1	2	0.11	1.03	43		2	2	0.12	2.15	66
1	3	0.15	1.01	42		2	3	0.16	2.04	64
1	4	0.20	1.01	42		2	4	0.21	2.05	64
1	5	0.25	1.00	42		2	5	0.26	2.03	64
1	6	0.30	1.00	42		2	6	0.33	1.64	56
1	7	0.35	1.01	42		2	7	0.39	1.40	50
1	8	0.40	1.01	42		2	8	0.45	1.25	48
1	9	0.45	1.01	42		2	9	0.49	1.19	46
1	10	0.50	1.00	42		2	10	0.54	1.11	44
3	1	0.07	3.40	91		4	1	0.05	4.00	104
3	2	0.12	3.18	86		4	2	0.10	4.00	104
3	3	0.18	2.44	72		4	3	0.15	4.00	103
3	4	0.23	2.22	68		4	4	0.21	3.93	101
3	5	0.27	2.13	66		4	5	0.29	3.12	86
3	6	0.32	2.10	65		4	6	0.35	2.49	73
3	7	0.37	2.06	64		4	7	0.40	2.22	68
3	8	0.42	2.04	64		4	8	0.44	2.13	66
3	9	0.46	2.00	63		4	9	0.49	2.11	65
3	10	0.51	2.01	63		4	10	0.54	2.13	66
5	1	0.09	4.10	106		6	1	0.11	5.10	128
5	2	0.13	4.09	106		6	2	0.14	5.07	127
5	3	0.17	4.03	105		6	3	0.18	5.07	127
5	4	0.22	4.06	106		6	4	0.22	5.04	126
5	5	0.26	4.02	104		6	5	0.29	4.82	121
5	6	0.31	4.03	105		6	6	0.34	4.62	117
5	7	0.36	4.00	103		6	7	0.40	4.41	112
5	8	0.43	3.87	100		6	8	0.45	4.31	110
5	9	0.50	3.62	95		6	9	0.50	4.20	108
5	10	0.57	3.34	89		6	10	0.56	4.05	104
7	1	0.13	5.77	144		8	1	0.06	7.00	175
7	2	0.16	5.46	136		8	2	0.11	6.99	176
7	3	0.20	5.37	135		8	3	0.17	6.82	170
7	4	0.24	5.14	128		8	4	0.26	6.04	150
7	5	0.29	5.15	129		8	5	0.31	5.74	144
7	6	0.33	5.07	127		8	6	0.36	5.39	135
7	7	0.38	5.08	127		8	7	0.41	5.31	133
7	8	0.43	5.05	126		8	8	0.46	5.12	128
7	9	0.48	5.03	126		8	9	0.50	5.13	129
7	10	0.53	5.01	125		8	10	0.55	5.07	127
9	1	0.12	7.01	176		10	1	0.13	8.03	204
9	2	0.15	7.02	177		10	2	0.17	7.91	201
9	3	0.18	7.00	176		10	3	0.21	7.83	198
9	4	0.23	7.01	176		10	4	0.26	7.72	196
9	5	0.27	6.99	175		10	5	0.31	7.58	191
9	6	0.33	6.92	173		10	6	0.37	7.35	185
9	7	0.39	6.81	170		10	7	0.41	7.30	184
9	8	0.46	6.56	163		10	8	0.46	7.21	181
9	9	0.53	6.05	151		10	9	0.52	6.99	175
9	10	0.59	5.81	146		10	10	0.58	6.78	172

Fig. 2. Performance of PRAD under different test conditions. All values are averaged over 100 runs. The columns represent the complexity of the function, the amount of noise, and the average error, polynomial degree, and time in milliseconds.

flexibility constraint requires the function to be accurately modeled in *each* of the settings it encounters. This ability is represented by the *maximum* amount of excess error achieved over all of the test scenarios. Figure 3 compares the maximum excess error and time spent by PRAD with those of constant degree polynomial regression.

The figure shows that PRAD attains a qualitatively better combination of excess error and time expended than polynomial regression with any of the fixed degrees

Degree	**Avg. Err.**	**Max. Excess**	**Time(ms)**
1	0.653	0.664	**39**
2	0.536	0.635	**62**
3	0.487	0.615	**84**
4	0.441	0.600	**110**
5	0.350	0.514	138
6	0.327	0.356	171
7	**0.289**	0.196	209
8	**0.281**	**0.083**	250
9	**0.277**	**0.024**	341
10	**0.276**	**0.004**	398
PRAD	**0.310**	**0.094**	**111**

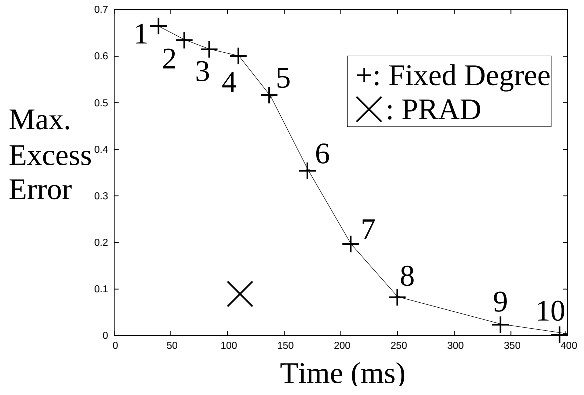

Fig. 3. The maximum excess errors and times attained by polynomial regression with a fixed degree, compared to PRAD.

tried. In particular, the lowest degree to attain the excess error achieved by PRAD was 8, which takes 125% more time than PRAD.

To test PRAD further, we compare it with a powerful alternative to polynomial regression: natural cubic splines. These are piecewise cubic curves that have a continuous second derivative [3]. For example, cubic splines can be used to identify kinetic parameters of a simulated or actual network of chemical reactions [15]. The transition points between the cubics are known as knots, and the question of how many knots there should be and where they should be placed has received much attention in the literature [16,17]. We are not aware of a solution to this problem that satisfies the constraints set forth in the introduction. For comparison purposes, we have implemented a straightforward approach to knot selection, spacing them uniformly throughout the interval. Since the method requires there to be at least 3 knots, we have tested each number of knots from 3 to 10. Each number of knots was tested on all 100 test cases, and the average errors, maximum excess errors, and average times were recorded. These values are shown in Table 3.

# of Knots	**Avg. Err.**	**Max. Excess**	**Time(ms)**
3	0.529	0.630	**64**
4	0.464	0.623	**83**
5	0.421	0.580	**108**
6	0.313	0.380	135
7	**0.289**	0.174	167
8	**0.280**	**0.063**	202
9	**0.277**	**0.025**	241
10	**0.276**	**0.010**	339
PRAD	**0.310**	**0.094**	**111**

PRAD outperforms any of the fixed settings tried for the cubic splines. The lowest number of knots that attains a better average error is 7, which takes 50% more time than PRAD. Furthermore, the lowest number that attains a better maximum excess error is 8, taking 82% more time than PRAD.

These results should not be taken as an indication that PRAD dominates cubic splines, or for that matter any other function approximator, in all cases. Rather, they indicate that for the subest of cases that polynomial regression is an appropriate method, PRAD can provide a more general, less brittle, solution.

4.2. *Mobile Robot Results*

The true test of PRAD's effectiveness is its performance on real autonomous agent tasks. In this section we present our implementation of PRAD on a mobile robot learning about itself and its environment. For these experiments, the algorithm and constants used were exactly the same as the ones used for the simulated data described above, namely the algorithm specified in Algorithm 1 and the constants in Table 1. The fact that no additional parameter tuning was necessary lends credence to PRAD's robustness.

The experimental setup used here is based on previous work training a mobile robot to simultaneously calibrate its action and sensor models [1]. In this work, a robot that is equipped with a camera uses polynomial regression to learn a model of a visual sensor. In the robot's camera image, the vertical size of a fixed landmark is taken as a visual sensor input. The sensor model is a function from this input to the robot's distance from the landmark. The robot and the landmark, a color-coded cylindrical beacon, are depicted in Figure 4. While the robot walks forwards and backwards facing the landmark, it uses estimates of its location (based on its action model), combined with corresponding sensor readings, to provide training data for the sensor model. In that work, the robot simultaneously learned its action model, a function from a parameterized range of action commands to their resultant velocities, also using polynomial regression.

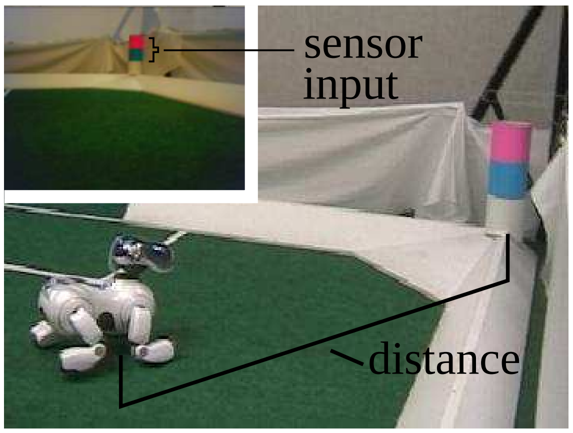

Fig. 4. A mobile robot with a fixed landmark. The inset depicts the landmark as seen through the robot's camera.

Previously, both regressions relied on a manually chosen degree for the polynomial. We chose 3 for the sensor model and 4 for the action model. These degrees were chosen based on the rough intuition that those were the amounts of complexity needed to represent those functions. However, the manually chosen degrees limit the method's applicability. Here we apply PRAD to the learning of the sensor model, demonstrating PRAD's effectiveness on real-world data. PRAD is implemented and tested on the Sony Aibo ERS-7 robot, shown in Figure 4. All processing is performed on the robot's 576 MHz processor in real time, including vision and motion processing.

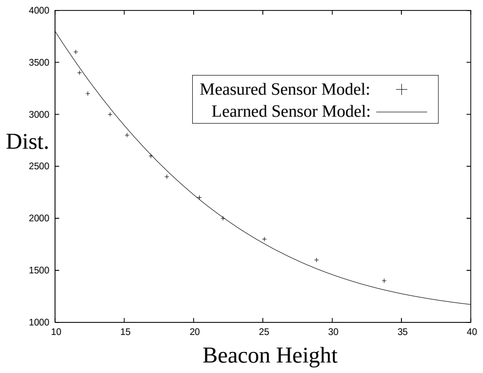

Fig. 5. The measured sensor model and a learned cubic model

The sensor model learned by PRAD is a function from the beacon's height in the image to the robot's distance from the beacon. In the scenario used to test PRAD, an action model is already known, but not a sensor model. In this situation, an appropriate polynomial degree for the sensor model would still be a priori unknown, making this scenario a realistic test case for PRAD. The distances used in the training are based on a pre-learned action model, which is a fourth degree polynomial mapping a parameterized set of actions to corresponding velocities. These estimated velocities are integrated over time to yield a continual distance estimate. The training examples for PRAD have the beacon's height as an input and the corresponding estimated beacon distance as an output. As PRAD learned a sensor model, the robot walked forwards and backwards via a sequence of randomly chosen action commands.

Each trial run lasted five minutes on the robot, and PRAD was tested over 15 trials. During the trials, PRAD was allowed to increment the degree at any time, but it never did so after two and a half minutes had passed. The fact that PRAD was able to settle on a degree every time demonstrates the method's stability. Furthermore, considering that each time the robot is booted up, it takes about 27 seconds to initialize, we consider this learning time of five minutes to be qualitatively quite short. Nevertheless, randomness in the training data caused some variation in the degree settled on by PRAD. The average degree chosen was 3.33, with a standard deviation of 1.29. This degree corroborates our earlier estimate that a third degree polynomial was roughly the amount of complexity needed to learn the sensor model. Figure 5 depicts a typical cubic sensor model learned by PRAD. The measured data is obtained by manual measurements compared with the robot's reported beacon heights.

The learned sensor models are evaluated by comparing them to the measured data. However, although the training outputs correspond to relative distances from the beacon, the robot has no access to its absolute distance. Thus we can expect the learned sensor model to match the measured one only up to an additive constant. To evaluate the learned model, we computed the shifting constant that minimizes the mean squared error. On average, the root mean square error between the shifted learned sensor model and the measured sensor model was 10.1 ± 3.4 cm. Compared to the distance range of 240 cm, the error is 4.2 percent.

These experiments demonstrate that PRAD can be effective in the context of an autonomous agent in the real world. It successfully negotiates the data obtained from the robot's noisy sensors and effectors and settles on an appropriate polynomial degree.

5. Related Work

PRAD can be seen as a type of stepwise regression algorithm. Such an algorithm adds to a set of basis functions for the regression, one function at a time. PRAD is in this category because incrementing the degree of the polynomial by one is equivalent to adding another exponent to a set of basis functions. One common method for stepwise regression involves adding new terms based on their computed statistical significance [18]. When this method is applied to polynomial regression, it can fail to find the correct degree because there are situations where incrementing the degree by one produces no improvement but where further increases can yield a significant improvement. PRAD circumvents this problem by increasing the degree until the fit is roughly as good as possible, based on an estimate of the amount of random noise. Other stepwise regression methods include MARS [19] and CIPER [20], which continually add multivariable polynomials to a basis set with the goal of minimizing the prediction error.

PRAD chooses a value for the degree of the polynomial to yield a small error using as little time and space as possible. Other methods for automatically choosing a parameter for a function approximator include the Akaike Information Criterion [21], the Bayes Information Criterion [22], and cross-validation [3]. However, these methods optimize parameters strictly in terms of prediction error, focusing on trying to make efficient use of a limited amount of data. PRAD differs from all of these methods in that it finds a good balance between prediction accuracy and resource efficiency in accordance with the amount of noise and complexity of the target function.

6. Conclusion

We have developed, implemented, and tested an enhancement to polynomial regression, PRAD. This technique is designed to satisfy three specific constraints. First, the algorithm uses only as much time and space as is necessary to represent the function being estimated. Second, PRAD is robust to varied amounts of random noise. Finally, it has no parameters that need to be manually tuned for each situation.

PRAD has been implemented and tested on both simulated test data and data gathered from a mobile robot. The experimental results with simulated data bear out that it can improve over polynomial regression with any fixed degree, according to the above constraints, and also that it can be more effective than natural cubic splines for some fixed settings of the spline knots. Furthermore, using the exact same algorithm and numerical constants, PRAD was able to automatically detect the appropriate degree for learning a sensor model on a mobile robot. These experiments suggest that PRAD is well-suited to learning in the context of an autonomous agent.

Acknowledgements

The authors thank Kurt Dresner, Peggy Fidelman, Nate Kohl, and Greg Kuhlmann for helpful comments on an earlier draft. This research was supported in part by NSF CAREER award IIS-0237699 and ONR YIP award N00014-04-1-0545.

References

1. Stronger, D., Stone, P.: Towards autonomous sensor and actuator model induction on a mobile robot. Connection Science **18** (2006) 97–119 Special Issue on Developmental

Robotics.
2. Kuipers, B.: Consciousness: Drinking from the firehose of experience. In: Proc. of the National Conference on Artificial Intelligence (AAAI-05). (2005)
3. Hastie, T., Tibshirani, R., Friedman, J.: The Elements of Statistical Learning. Springer-Verlag, New York (2001)
4. Morring, B., Martinez, T.: Weighted Instance Typicality Search (WITS): A nearest neighbor data reduction algorithm. Intelligent Data Analysis **7** (2003)
5. Rumelhart, D.E., McClelland, J.L.: Parallel distributed processing: exploration in the microstructure of cognition. MIT Press, Cambridge, MA (1986)
6. Haykin, S.: Neural Networks. Prentice-Hall, Inc., Upper Saddle River, New Jersey (1999)
7. Bishop, C.M.: Neural networks for pattern recognition. Oxford University Press, Oxford, England (1995)
8. Mitchell, T.: Machine Learning. McGraw-Hill (1997)
9. Cybenko, G.: Approcimation by superpositions of a sigmoidal function. Mathematics of Control, Signals, and Systems **2** (1989) 303–314
10. Hornik, K., Stinchcombe, M., White, H.: Multilayer feedforward networks are universal approximators. Neural Networks **2** (1989) 359–366
11. Hartman, E.J., Keller, J.D., Kowalkski, J.M.: Layered neural networks with Gaussian hidden units as universal approximations. Neural computation **2** (1990) 210–215
12. Sridharan, M., Kuhlmann, G., Stone, P.: Practical vision-based monte carlo localization on a legged robot. In: IEEE International Conference on Robotics and Automation. (2005)
13. Deng, L., Aksmanovic, M., Sun, X., Wu, C.F.J.: Speech recognition using hidden markov models with polynomial regression functions as nonstationary states. IEEE Transactions on Speech and Audio Processing **2** (1994)
14. Weisstein, E.: Least squares fitting – polynomial. MathWorld – A Wolfram Web Resource (2005)
15. Madar, J., Abonyi, J., Roubos, H., Szeifert, F.: Incorporating prior knowledge in cubic spline approximation - application to the identification of reaction kinetic models. Industrial and Engineering Chemistry Research **42** (2003) 4043–4049
16. He, X., Shen, L.X., Shen, Z.W.: A data-adaptive knot selection scheme for fitting splines. IEEE Signal Processing Letters **8** (2001) 137–139
17. Miyata, S., Shen, X.: Adaptive free-knot splines. Journal of Computational & Graphical Statistics **12** (2003)
18. Neter, J., Wasserman, W., Kutner, M.H.: Applied Linear Statistical Models: Regression, Analysis of Variance, and Experimental Designs. Irwin, Homewood, IL (1985)
19. Friedman, J.: Multivariable adaptive regression splines (with discussion). Annals of Statistics **19** (1991) 1–141
20. Todorovski, L., Ljubic, P., Dzeroski, S.: Inducing polynomial equations for regression. In: Proc. of the Fifteenth European Conference on Machine Learning, Pisa, Italy (2004)
21. Akaike, H.: Information theory and an extension of the maximum likelihood principle. In: Proc. Second International Symposium on Information Theory, Budapest (1973)
22. Schwarz, G.: Estimating the dimension of a model. Annals of Statistics **6** (1978) 461–464