Optimizing Selection of Training Samples for Robotics Learning Problems

Reza Mahjourian Peter Stone

December 2, 2011

1 Introduction

It is desirable to carry out learning tasks on the real robots, but at the same time learning on robots is costly as too many trials can cause wear and tear. Therefore, it is of value to reduce the number of training samples that are taken for a learning task. An efficient solution to this problem would be useful for reducing the cost of learning tasks such as motion prediction for a kick or walk.

Since for this project I needed to be able to evaluate the efficiency of the solutions, I tried to select test domains where a lot of experiments can be performed. The developed solutions are tested for function approximation and for predicting the outcome of a kick in UT's 3D simulation league software.

2 Problem Description

In this project we are considering supervised learning problems, where a set of training samples $D = (x_i, y_i), x_i \in X, y_i \in Y$ are used for learning a mapping $X \rightarrow Y$. For example, for learning the outcome of a parametrized kick, x_i could correspond to the values for the parameters to the kick engine and the relative position of the ball to the robot, and y_i could correspond to the distance travelled by the ball and the angle of its travel. Similarly, for predicting the velocity of a walk under different parameter values, x_i could correspond to the parameter values sent to the walk engine, and y_i could correspond to the velocity achieved by the walk.

Independent of the learning task at hand, one can consider the following objectives for optimizing the design of training samples:

Reducing the number of training samples We would want to use as few training samples as possible.

Reduce the weighted cost of training samples It makes sense to associate different costs to different training samples. For example, a walk sample that results in a fall is more costly and therefore undesirable. The estimated cost of a training sample should be taken into account for selecting new training samples.

Increase generalization power of the learning architecture Whether we use neural networks or statistical models to learn the mapping $X \rightarrow Y$, we would like to increase the generalization efficiency of the model by reducing the estimated prediction error for members in X.

Some learning problems include an optimization objective as well. For example, when improving a walk we are ultimately interested in finding an optimal point x_{opt} for which the corresponding value in Y is maximized. This project does not address optimization problems.

3 Related Work

One of the earliest applications of *Optimal Experiment Design (OED)* to the robotics problems is the work in [1] which uses OED to identify link masses and inertial moments of a robotic arm. Their results show that automatically generating training trajectories for learning results in a significant improvement over trajectories designed manually.

One of the earliest theoretical analyses on Active Learning is the work in [9]. MacKay's work, which builds on solutions in [5], discusses optimizing selection of data points for learning in a Bayesian framework. He considers three different objectives: 1) If one interpolation model is used, we may want to choose new data points such that they are *maximally informative*, that is it maximizes the information gain on the mapping that is being learned. 2) We might be interested in maximizing the the accuracy of the interpolant not globally, but in some limited region of interest that is more important to the problem. 3) If more than one candidate interpolant model are being considered, we might be interested in choosing data points that give us maximal information on determining which model performs best.

In [4], Cohn et al. demonstrate using active learning for optimizing the training set for a classification problem. The mapping that is to be learned is the membership of a subset of the points in the domain space in some set c. For example, in a two-dimensional plane, c might be the set of all points in a fixed rectangle. They try to design training samples that would allow the learner to identify the unknown boundaries of the rectangle.

Their solution is based on the concept of *region of uncertainty*, which corresponds to the areas of the domain space that are not determined by the existing information from previous training samples. Their solution draws points strictly from the region of uncertainty and avoids points that are already determined. Figure 1, used from [4], shows the region of uncertainty after some training samples.

Since computing the region of uncertainty is computationally intensive, they also propose using a neural network to approximate the region of uncertainty.

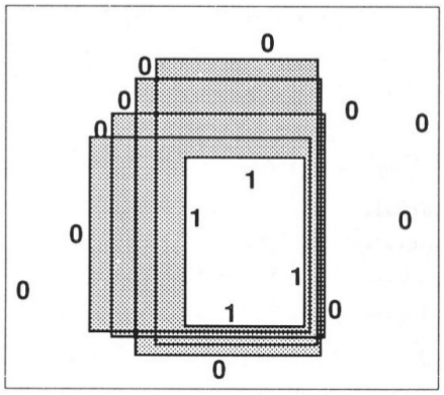

Figure 1: Region of Uncertainty [4]

The work in [3] focuses on improving the generalization of neural networks for a learning problem at the cost of extra computations. Their problem is formulated as choosing training samples that would minimize the mean squared error for prediction of values over the input domain. This work assumes all samples to have the same cost. It also does not address optimization problems.

Their solution is based on estimating the output variance of the network by computing the network's output sensitivity. They select the new sample such that it would minimize the updated output variance of the network.

They also make a distinction between learning problems with static constraints and dynamic constraints. With static constraints, the learner might pick any learning sample regardless of the current state of environment. With dynamic constraints, the set of training samples that are available are determined by the current state of the environment. For example, consider a robotic arm whose current state is specified by its joint angle and velocity and the current torque applied to the joint. In this case, the learner can not request a training sample with arbitrary new values for joint angle, velocity and torque.

The work in [6] discusses using active learning for two statistically-based learning architectures: Gaussian mixtures and locally weighted regression. They show that while the solutions for reducing the variance in neural networks are approximate, the techniques used for these two statistically-based models are both accurate and efficient.

As another application of Active Learning to machine learning, the work in [10] discusses using active learning to enhance the generalization behavior of Support Vector Machines (SVMs). They show that training an SVM on some carefully chosen subset of available samples performs better than training on all available samples. Using fewer training samples also results in a faster learning method.

In [8] Krogh discusses the idea of using neural network ensembles for improving the efficiency of training and also for active learning. Their solution is based on training a randomly initialized ensemble of neural networks, which have the same topology. By comparing the predictions of the networks in the ensemble, one can assess the degree of uncertainty in the model about various points in the input. Points that show a higher degree of variance in predictions are good candidates for sampling. Our solution is based on this approach. However, we expect to be able to deal with noise in sampling on real robots, and with varying sample costs, which are not discussed in their solution.

For many learning tasks on the real robot, we need to pay special attention to avoiding samples that can cause the robot to malfunction or undergo extra strain. For example, biped robots like the Nao are less stable and can sustain costly damage if they fall during an experiment. For this reason, it makes sense to assign different costs to different learning samples based on their outcome. This problem is not studied as extensively as active learning.

In [11] the authors consider the problem of predicting sample costs for active learning. The domain of their work is human-generated annotations on natural language text. They use regression learning algorithms for estimating the cost of annotating a sample. Each sample is represented by a set of numerical features, like number of words, paragraphs, and entities.

The work in [7] frames active learning as a reinforcement learning problem. The conventional active learning methods use greedy algorithms which select the best training sample at each point in time according to some metric. This work develops a non-greedy solution which tries to optimize long-term gains. Each sample point is associated with a reward, based on how much it can increase the prediction ability of the learner, and a cost, based on the amount of work that is involved in taking the sample. Their cost model is similar to the work in [11] and is designed for domains similar to text processing, where the cost of a sample can be determined externally.

I was not able to find solutions with a sample cost estimation model which would be suitable to the robotic learning tasks that we are interested in.

4 Active Learning

Figure 2 shows the conventional workflow for passive learning. Sample points are selected randomly from the input space. By trying the sample point in the experiment domain, one obtains a training sample, which can then be used to train the learner.

The objective of active learning is to select training samples that are maximally informative in terms of improving the prediction ability of the learner. Most active learning methods use some metric to estimate the variance of the learner's prediction on points in the input space and select points which are expected to have a higher degree of variance.

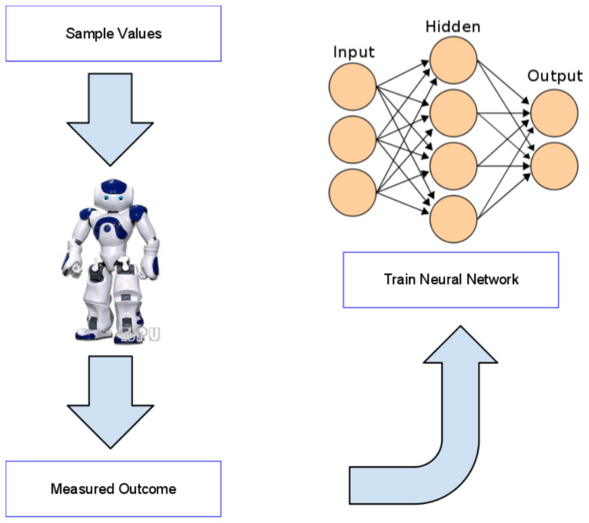

Figure 2: Passive Learning

4.1 An Analytical Approach

My first attempt at implementing an active learning solution was based on the work in [3]. Their solution is driven by trying to minimize the expected mean-squared error (MSE) of the learner's prediction over the entire input space. Their approach is based on analyzing the neural network that is used by the learner and estimating its behavior using a mathematical model.

They show that the expected MSE can be decomposed as an output variance term plus a bias term. Assuming an unbiased estimator, error can be minimized by minimizing the output variance. The output variance is computed based on the sensitivity of the neural network with respect to changes in the connection weights. Note that the network's output sensitivity can be measured for an individual point. However, their approach requires minimizing it over the entire input space. So they use an approximation to relate these two measures.

When a new sample based on point x is tried, a resulting output y is obtained. Adding the sample (x, y) to the training set will change the variance of the learner. They try to pick an optimal x such that it will maximize the reduction in the learner's variance over the entire input.

They use a series of approximations to justify reducing the problem of lowering MSE to picking an optimal point that maximizes the expected reduction in the average output sensitivity of the network. In the end, the problem is reduced to hill climbing on an objective function that is developed based on these approximations.

Even though they show that the approximations they employ are somewhat supported by their data, the performance of their solution does not seem convincing (See Figure 4 in [3]). In particular, the results show that sampling from a uniform grid outperforms their solution.

4.2 Using an Ensemble of Neural Networks

Figure 3 shows the schema of the solution that I have used for learning with active sampling. This approach is based on the work in [8]. A set of 5 neural networks with the same topology are used as the learner. The weights in the networks are initialized randomly. In each iteration, all the networks are trained on the past training samples obtained.

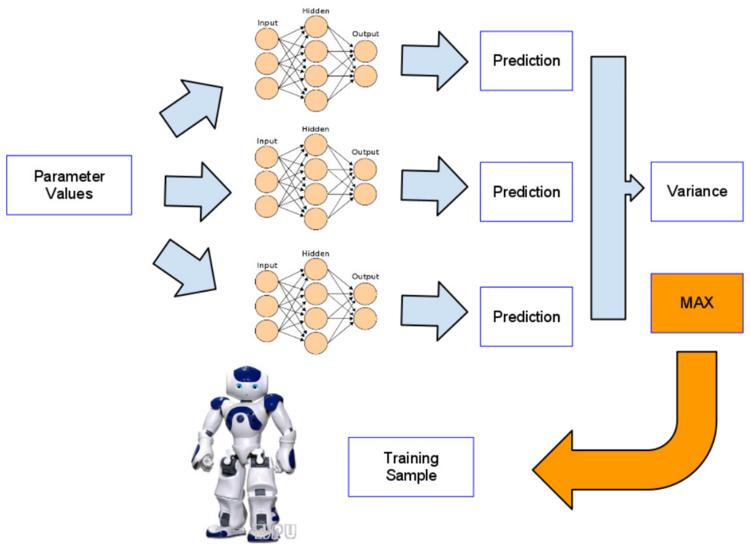

Figure 3: Active Learning with an Ensemble of Neural Networks

The networks are trained using backpropagation. When training each network, the samples are split into a training set, and a cross-validation set. The cross-validation set is used to decide when to stop training in order to prevent over-generalization over the samples.

In order to determine what point to sample from next, the networks are queried on a series of randomly generated points in the input space. For each point, the predictions of the networks are obtained and a variance of predictions is computed. The point with the highest variance is selected as the next sampling point.

This setup shares one of the two approximations use in Cohn's solution in that it assumes that reducing the variance of the learner will lead to a reduction in the error of the learner. However, it does not directly rely on the other approximation in Cohn's solution which is used to estimate the reduction in the learner's variance when sampling from a given point.

It's worth mentioning that this solution has the drawback that the networks have to be retrained from scratch on all training samples after each new sample is added. The initial random weights of the networks are stored and reloaded before each retraining. A potential solution for this issue would be to sample multiple points in each iteration. The number of samples per iteration can be increased proportionally to the current size of the training set. The analytical solution developed by Cohn shares this problem as well. At the same time, since for most robotics problems taking a sample requires a considerable amount of time, the extra computational cost of retraining can be justified.

4.3 Optimizing Average Sample Costs

Our solution for reducing the average sample costs for the training session is based on using a separate ensemble of neural networks to predict the sample costs. The networks in the second ensemble have the same topology as the networks in the ensemble used for value prediction, with the exception that they have a single output node used for estimating the sample cost. In each iteration, once the sample has been tried, the sample cost obtained from the trial is used to update a separate training set for the cost prediction ensemble.

Unlike the value ensemble, which is initialized randomly, the cost ensemble is pre-trained over a grid in the input space, so that the initial prediction of the cost ensemble is relatively uniform for all points. The training target for the points is set to be the lowest possible cost for samples. If the cost prediction ensemble is not initialized, the learner would become randomly biased towards sampling from some parts of the input space when taking the first few samples.

When selecting a new training sample, both the value prediction ensemble and the cost prediction ensemble are queried on random points in the input space. The metric I have used to balance the two potentially conflicting objectives of reducing the prediction error and reducing the average sample costs is:

$$\frac{\left(\frac{\text{point variance}}{\text{average variance in ensemble}}\right)^2}{\text{estimated cost}}$$

A point which has the highest value according to this metric is chosen. By changing this metric function one can give varying weights to minimizing the error and minimizing the average sample cost.

4.4 Objectives for Cost-Efficient Active Learning

Our cost-efficient selection metric seems arbitrary. In fact I tried a number of different metrics and they produced comparable results. Some reduced the average cost more and some reduced the MSE more.

One can also consider using cost-efficient objective functions for balancing these two requirements. For example, one can try to minimize MSE $\times$ average sample cost. However, it is not straightforward to design an appropriate sample selection metric which can lead to minimizing such a given objective function.

Moreover, coming up with an objective function can be tricky as well. For example, suppose a learner has been able to reduce the MSE to 0.02 with an average sample cost of 5. Such a result is probably more desirable than achieving an MSE of 0.01 with an average sample cost of 10, which would have the same objective value if we multiply MSE by average sample cost.

I tried to create a solution which can minimize an arbitrary given objective function. The idea I used was to replace the hand-written selection metric with a neural network. The network was expected to learn an optimal function over the available measurements form the value and cost ensembles to minimize the given objective. Example inputs that can be provided to this network are: the variance of the value network at the point that is being considered, the average variance in the value network, the predicted cost of of sample from that point, etc. Once a training sample is selected, the change in the value of the objective function is measured and the network is trained to learn about the outcome of its previous decision.

This solution didn't work well in our experiments. One hypothesis on why this didn't work is that the dynamics of the value and cost networks are constantly changing. The reduction in the value of the objective function that is experienced in one iteration may not be repeatable since the value and cost ensembles are adapting to the new samples. This is why I didn't feed this network with point coordinates of sample points. If the network learns that sampling from some point was rewarding, it would try to select that point again, which would be counter-intuitive for active learning.

A potential workaround to this problem may be to keep a limited training history for this network, so that it only considers the outcome of its recent choices. We can also use a weighted training set, which gives larger weights to recent samples. I tried to implement such a solution using an `ImportanceDataSet` in PyBrain, but for some reason the implementation didn't work.

It's also worth mentioning that minimizing any objective function that involves MSE is not a well-defined problem in this context. For robotics learning problems where sampling is costly and active learning is desired, one does not have access to an evaluation set which can be used to measure the MSE of the learner. Our best hope would be to develop and evaluate a solution to some learning problem and expect it to work on other problems which have similar value curves and cost functions.

This is why I considered developing and evaluating the solution in the 3D simulation software.

4.5 Active Learning for Robotics Problems

During my experiments, I realized that when working with the simulator, the noise in sampling can throw off the active learning algorithm. Active learning works based on the assumption that if there the variance in the ensemble is high for some point, taking a sample at that point will lead to a reduction in variance.

However, if the measurements in the simulator is noisy, sampling can lead to an increase in the variance of the ensemble. This is because the networks on the ensemble are trained on different subsets of the training samples. If two samples are taken from the same point and they come back with different outcomes, when they get assigned to different networks in the ensemble, they will drive up the variance in the learner. If such a situation occurs, which our selection metric will choose to sample from the same point again.

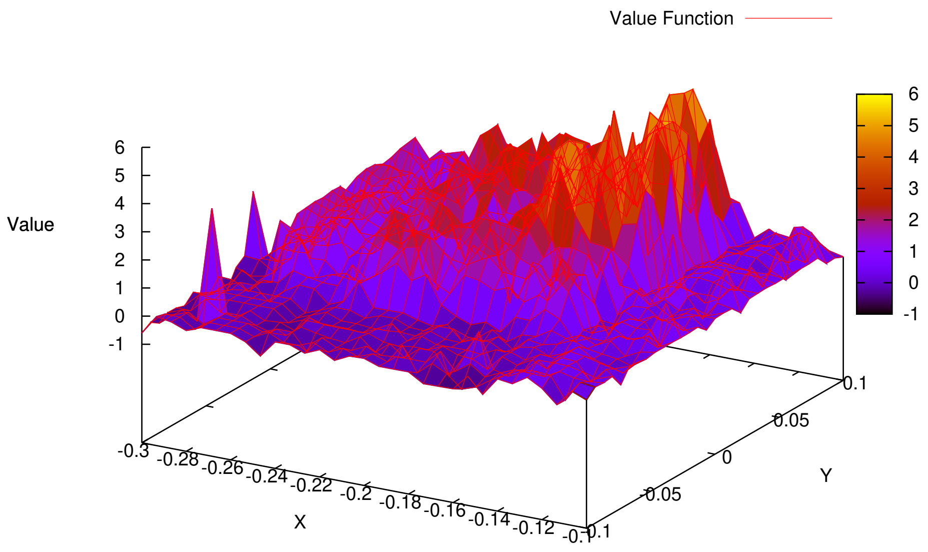

Figure 4: Kick Distance Measurements in the Simulator

Figure 4 shows the kick distance measurements obtained from the simulator for varying starting positions of the robot relative to the ball. The plot is averaged over 5 runs. As the plot shows, there is considerable noise in the outcome of the kick. This is while these measurements are made on modified version of the simulator that gives the agent access to ground truth about the position of the ball, which eliminates the noise due to vision or localization.

The solution that I developed to deal with this noise is to use the measurements from different trials of the sample in the simulator to estimate the input variance for the sample. Since the active learning algorithm is driven by variance, we would like to be able to isolate the variance that is due to the input from the variance that is caused by using different subsets of the training set for training each network.

Figure 5 shows how this setup works. When points are being sampled in the

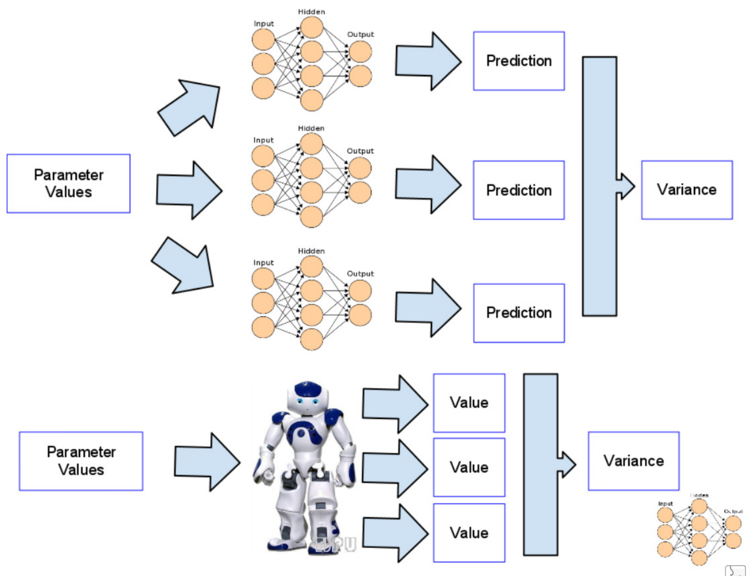

Figure 5: Active Learning with Noisy Samples

simulator, the variance from the different trials is also obtained. The variance is used to train a separate ensemble whose function is to predict the variance of the value function at each point in the the input space.

5 Experiments

I implemented the solutions in Python and using the PyBrain neural network library. I started with training a neural network to learn a 2D value function and a 2D cost function in order to gain insight about the right training parameters that need to be set for the neural networks. In particular, I tried to evaluate the impact of using cross-validation on training times and accuracy. I also realized through some initial failures that higher values for momentum, though making the training more quick, can prevent the networks from learning non-smooth functions, like a square wave function.

5.1 Function Approximation

I used the solutions developed for the project to train neural networks to approximate the function shown in Figure 6.

For the cost function, I used the step-like function shown in Figure 7. Giving a higher cost to the areas that lie on the boundary regions of the input space was motivated by the observation that parametrized behaviors are more likely to exhibit failure when extreme values are tried.

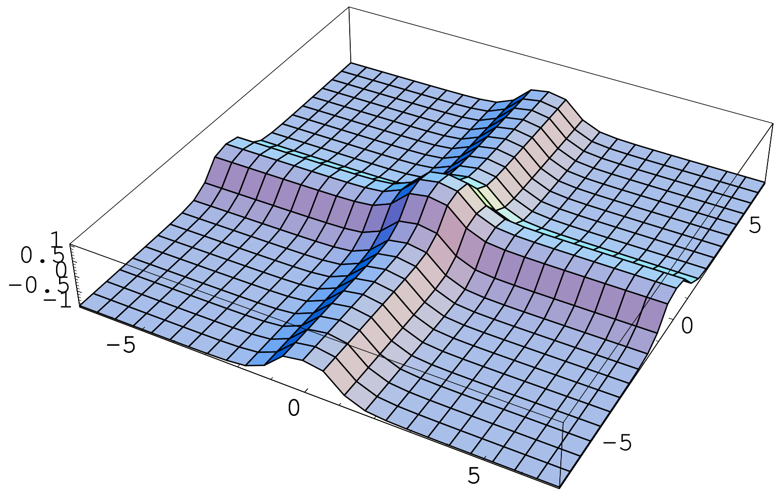

Figure 6: Value Function Used for Approximation Experiments

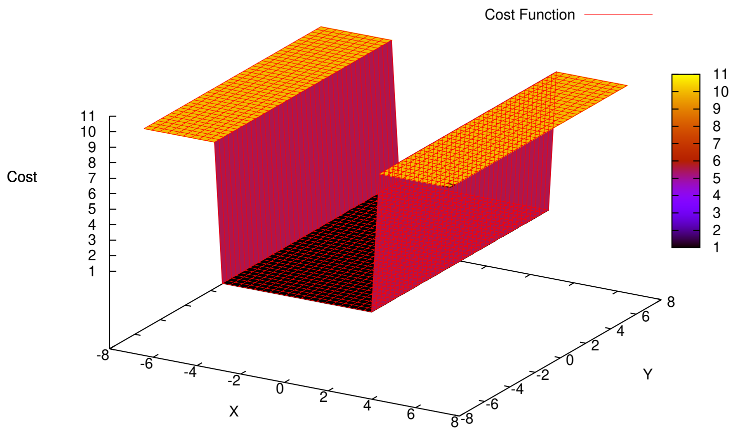

Figure 7: Cost Function Used for Approximation Experiments

I compared the efficiency of random sampling with active sampling using the ensemble of neural networks. I used each method to generate 40 samples. For all three methods, once 5 training samples were taken, I turned on cross-validation and early stopping to prevent the networks from over-generalizing based on the training samples.

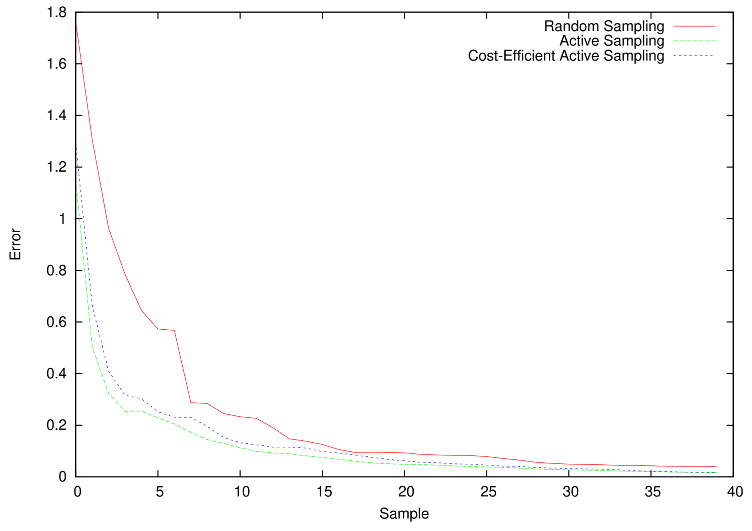

Figure 8: Mean-Squared Error values Measured over the Entire Input

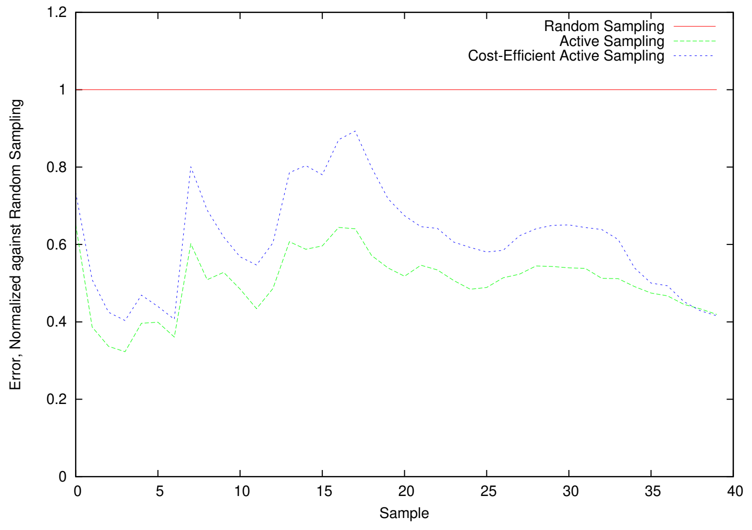

Figure 9: Mean-Squared Error values Normalized Against Random Sampling

Figure 8 shows the comparison of Mean-Squared Error values for each method as the number of samples increase. Active sampling and cost-efficient active sampling perform better than random sampling. Since the function that is being approximated is relatively smooth, random sampling can catch up with more training samples. Figure 9 shows the same plot normalized against the error of random sampling for easier comparison.

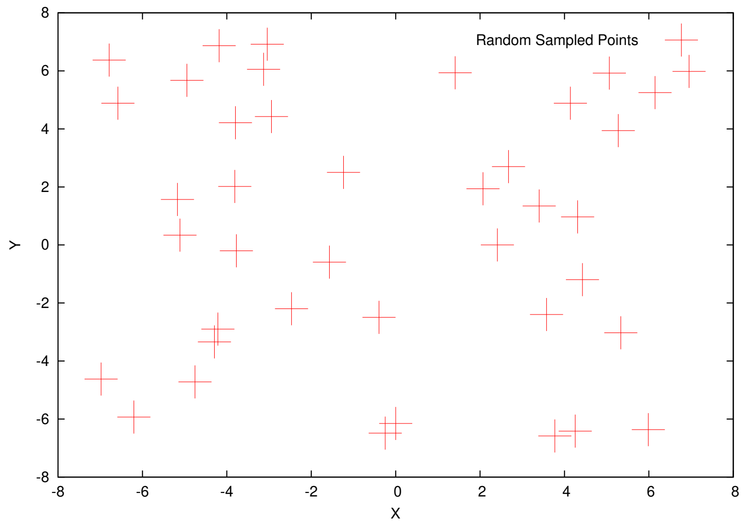

Figure 10: Random Sampled Points

Figure 10 shows the points that are selected by random sampling. As expected, they are distributed randomly around the input space. Figure 11 shows the point sampled by active sampling. There is more focus on the boundaries of the input and also on the areas that lie on the ridges and the peak in the original function. Active sampling usually starts sampling from the boundary regions. This is due to the fact that the networks in the ensemble are initialized randomly. Since the magnitude of the input signals to the networks are higher in the boundary regions, the outputs of the networks are more likely to have higher magnitudes as well.

Figures 12 and 13 show the plot of the learned approximation of the original function after 40 samples by random sampling and active sampling, respectively. The function learned by active sampling is closer to the original function.

Figure 14 shows the points sampled by cost-efficient active sampling. The pattern of the samples is somewhat similar to the points selected by active sampling, with the exception that fewer points are sampled from the high cost regions.

Figure 15 compares the average sample costs for the three methods. Random sampling achieves an average sample cost of 5.5, which is the average over the input space. Cost-efficient sampling achieves the lowest average cost. With active sampling, the average cost start slightly higher than average and then

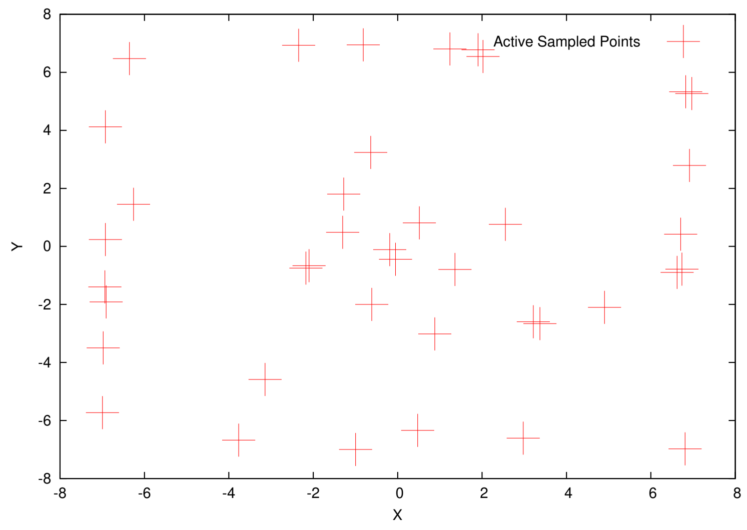

Figure 11: Active Sampled Points

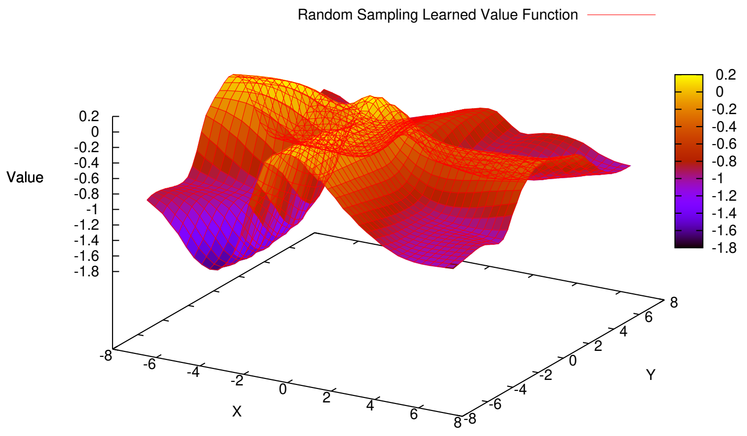

Figure 12: Approximation Learned by Random Sampling

goes down. This is because of the initial preference of active sampling towards the boundary regions.

In most robotics learning problems, there is some negative correlation between value and cost. A walk that fails would be assigned a low value and a

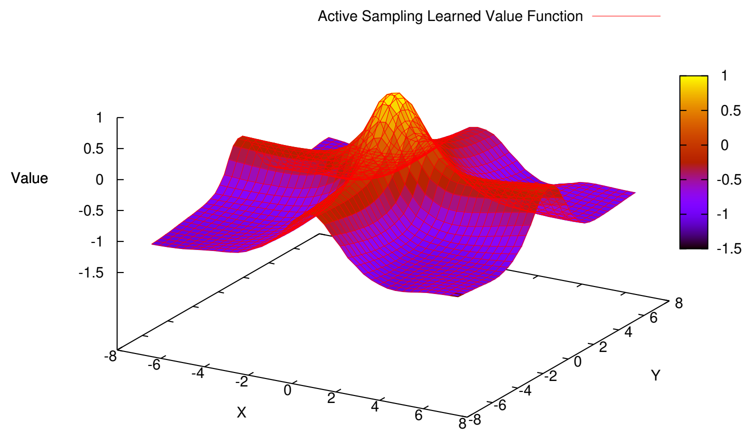

Figure 13: Approximation Learned by Active Sampling

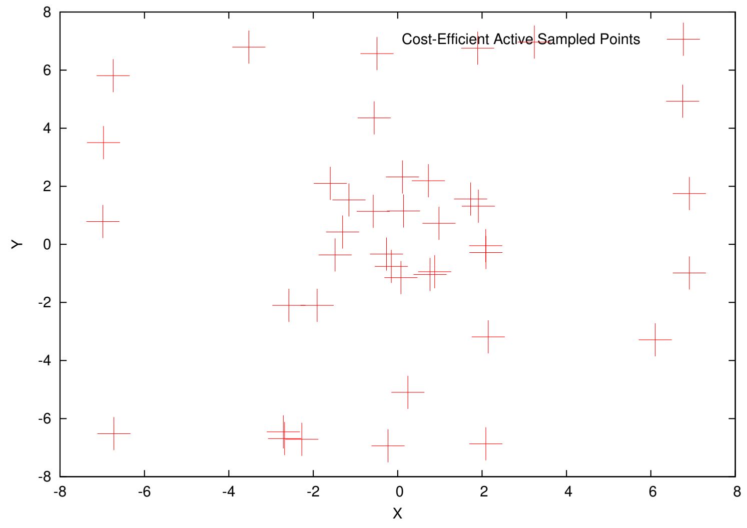

Figure 14: Cost-Efficient Active Sampled Points

high cost. So, generally the regions in the input space that have higher costs have lower values. For this reason, one can expect plain active learning to also do relatively well cost-wise in a learning task. This is partially evident from Figure 15, which shows a downward trend for the average samples costs of active

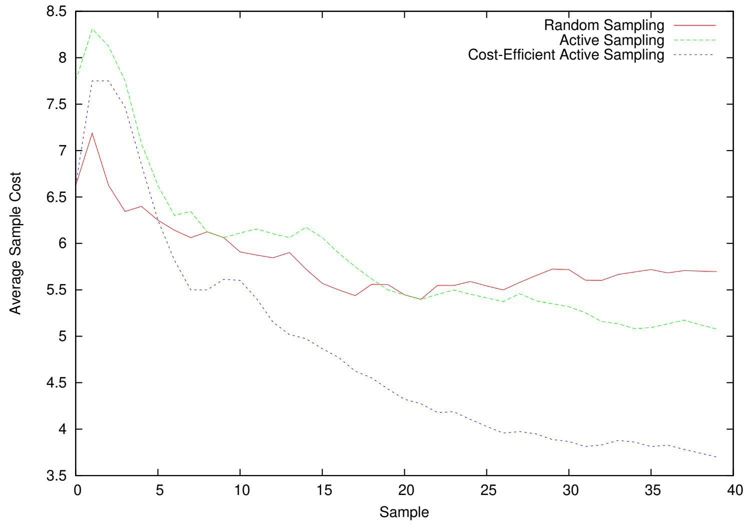

Figure 15: Average Sample Costs over the Entire Input

sampling. If we continued sampling, the average sample cost would go down, since active sampling is less likely to sample from the flat areas in the original function.

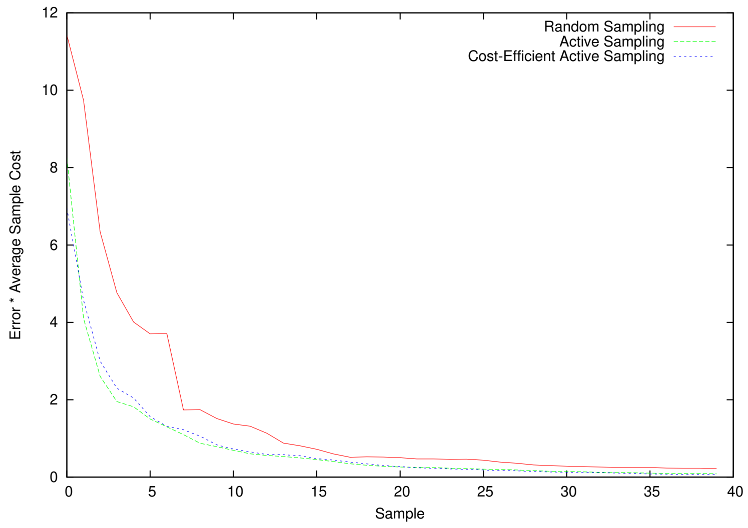

Figure 16: MSE * Average Sample Cost over the Entire Input

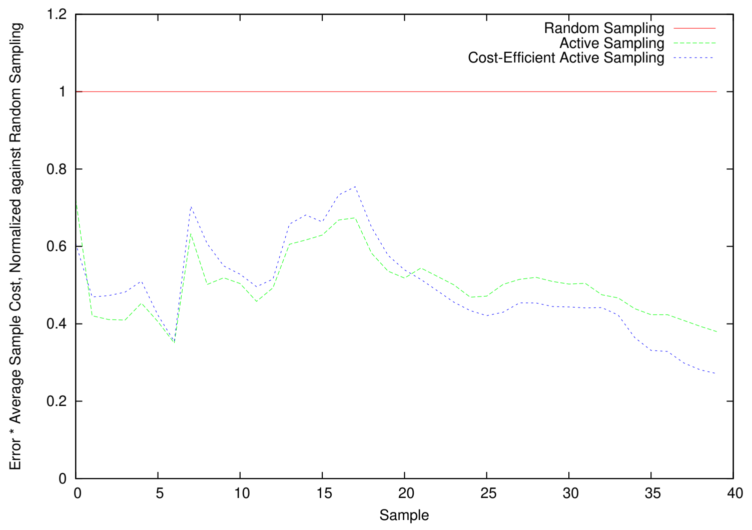

Figure 17: MSE * Average Sample Cost Normalized Against Random Sampling

Figure 16 compares the mean-squared error multiplied by the average sample cost for the three methods. Figure 17 shows the same plot normalized against random sampling. It seems cost-efficient active sampling is able to achieve a better value after it has gathered some information on the cost function. This learner's estimate on the cost function after 40 samples is shown in Figure 18. However, as discussed earlier, this particular objective function is probably not a good choice for optimizing learning.

Another interesting property of active sampling using the ensemble of networks and cross-validation is that it is robust to the choice of network topology. I experimented with reducing and increasing the number of nodes in the network and the solution could consistently do better than randomly sampling. Specially, increasing the number of nodes in the network widened the gap with random sampling. Generally, when the network has more nodes than needed, it might over-fit the samples and show a reduction in prediction ability. It seems using cross-validation is preventing this phenomenon from happening.

I also tried the solution for approximating a geometric function. I have also tried different cost functions, for example one that has 3 high cost circular regions in the input space. Active learning performed better than random sampling in these cases as well.

5.2 Learning Kick Parameters in the Simulator

For this experiment, I used one of the parametrized kicks developed for the simulation league software [2]. The code for the kick was slightly modified to be suitable for this experiment. In particular, the kick behavior was based on

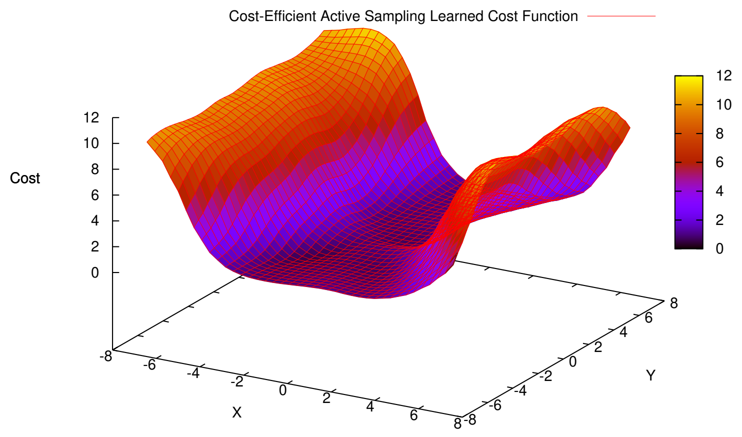

Figure 18: Prediction of Cost after 40 Samples

approaching the ball and repeating the kick 10 times and returning an average fitness value. It was modified to kick once and return the distance and angle of the ball's travel vector. It was also modified to read in the initial beam position from the parameters so that the initial relative position to the ball could be controlled.

For this experiment I chose to use two parameters specifying the initial coordinates of the robot relative to the ball as the input parameters. These parameters were chosen to make it easier to verify the behavior of the program, as the outcome of the kick would be easier to predict and verify. The implementation is flexible and can work with other choices of parameters. Also I started with two parameters so that I could visualize the outcome.

For gathering an evaluation dataset, I wrote a separate program to generate sample points on a 13×13 grid over the input space and to process the results that are generated after the simulation runs. For higher number of dimensions, the number of samples per dimension would be reduced to control the total number of sample points. The 961 samples were evaluated using the condor cluster in the department. A total of 5 runs were executed and the results were averaged.

Since active learning creates training samples in sequence, it wouldn't be possible to use condor to achieve speedups. Samples have to be run in the simulator one after the other.

The active learning solutions didn't do very well on this task. Judging from the shape of the evaluation function shown in Figure 4, I tried increasing the number of samples from 5 to 20, which didn't improve the results.

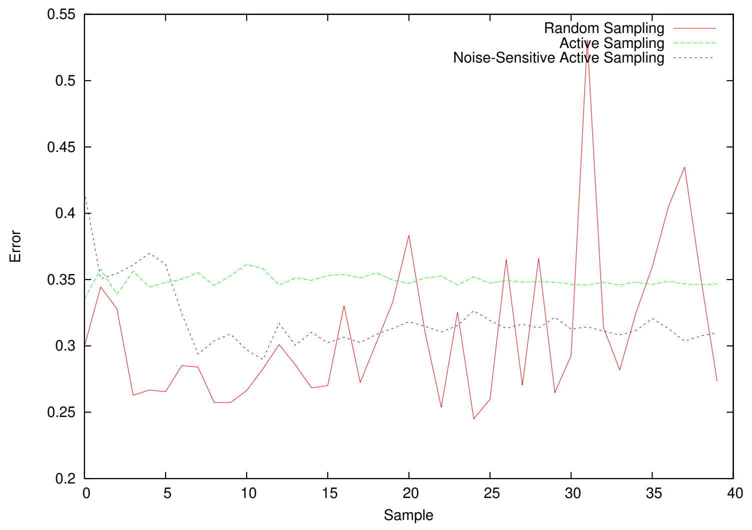

Figure 19: Mean-Squared Error for Kick Distance Prediction

The results are shown in Figure **??**. The plots shows the results from a single run. My implementation can use multiple processors and I was able to run 8 trials in parallel for the function approximation task. But it seems that running copies of the simulator in parallel on the same computer doesn't work as expected.

It's evident that the plain active learning algorithm can not handle the noise in the samples. Its error curve in the plot is almost flat, as it keeps sampling from one or two points in the corners of the input space. Using the variance prediction network improved the results, but still it didn't do better than random sampling.

One reason why this is happening might be that the regions of the input space that I have selected for this task contain mostly important points and few unimportant points, where the function has the same value. The surface of the kick distance function is non-uniform in the evaluation set in almost all places. So sampling from any point is likely to lower the mean-squared error over the evaluation set.

But what is troubling in this figure is that even for random sampling, which does not get trapped in some part of the input space, the mean-squared error doesn't seem to be decreasing with more samples. One guess is that the evaluation set that I have generated is not close to the real mean of the kick distance and angle values and therefore does not constitute a good ground for evaluating the functions produced by the sampling methods. It is also possible that there is an issue with the kick behavior that I have modified in the simulator and it is causing it to exhibit some erratic behavior.

6 Conclusions and Future Work

We developed an active learning solution based on an ensemble of neural networks. We also presented a solution for reducing the sampling costs in active learning for problems where the sample costs can not be predicted by an external model and have to be extracted by trying the sample in the real environment.

Even though not being able to reproduce the function approximation results in the simulator is a setback, it revealed that dealing with noise is key for active learning on robots. This is also in line with the general theme of this course. I wasn't able to find an active learning solution that discusses the issue of noise, so it seems such a solution would be valuable.

It would be interesting to verifying the active learning solution with variance prediction on a problem with controlled noise, so that we can isolate the issue of noise from any other potential issues. For example, we can try to approximate a function which has added noise generated by an overlayed function.

References

[1] B. Armstrong. On finding exciting trajectories for identification experiments involving systems with nonlinear dynamics. *The International journal of robotics research*, 8(6):28, 1989.

[2] S. Barrett, K. Genter, M. Hausknecht, T. Hester, P. Khandelwal, J. Lee, M. Quinlan, A. Tian, P. Stone, and M. Sridharan. Austin Villa 2010 standard platform team report. Technical Report UT-AI-TR-11-01, The University of Texas at Austin, Department of Computer Sciences, AI Laboratory, January 2011.

[3] D. Cohn. Neural network exploration using optimal experiment design. *Neural Networks*, 9(6):1071–1083, 1996.

[4] D. Cohn, L. Atlas, and R. Ladner. Improving generalization with active learning. *Machine Learning*, 15(2):201–221, 1994.

[5] V. Fedorov. Theory of optimal experiments. 1972.

[6] Z. Ghahramani, D. Cohn, and M. Jordan. Active learning with statistical models. *J. of Artificial Intelligence Research*, 4:129–145, 1996.

[7] A. Krause and C. Guestrin. Nonmyopic active learning of gaussian processes: an exploration-exploitation approach. In *Proceedings of the 24th international conference on Machine learning*, pages 449–456. ACM, 2007.

[8] A. Krogh and J. Vedelsby. Neural network ensembles, cross validation, and active learning. *Advances in neural information processing systems*, pages 231–238, 1995.

[9] D. MacKay. Information-based objective functions for active data selection. *Neural computation*, 4(4):590–604, 1992.

[10] G. Schohn and D. Cohn. Less is more: Active learning with support vector machines. In *MACHINE LEARNING-INTERNATIONAL WORKSHOP THEN CONFERENCE-*, pages 839–846. Citeseer, 2000.

[11] B. Settles, M. Craven, and L. Friedland. Active learning with real annotation costs. In *Proceedings of the NIPS Workshop on Cost-Sensitive Learning*, pages 1069–1078. Citeseer, 2008.