Using Gaussian Processes to Optimize Expensive Functions.

Marcus Frean and Phillip Boyle

Victoria University of Wellington, P.O. Box 600,
Wellington, New Zealand
marcus@mcs.vuw.ac.nz
http://www.mcs.vuw.ac.nz/∼ marcus

Abstract. The task of finding the optimum of some function $f(\mathbf{x})$ is commonly accomplished by generating and testing sample solutions iteratively, choosing each new sample $\mathbf{x}$ heuristically on the basis of results to date. We use Gaussian processes to represent predictions and uncertainty about the true function, and describe how to use these predictions to choose where to take each new sample in an optimal way. By doing this we were able to solve a difficult optimization problem - finding weights in a neural network controller to simultaneously balance two vertical poles - using an order of magnitude fewer samples than reported elsewhere.

1 Introduction

One potentially efficient way to perform optimisation is to use the data collected so far to build a predictive model, and use that model to select subsequent search points. In an optimisation context, this model is often referred to as a *response surface*. This method is potentially efficient if data collection is expensive relative to the cost of building and searching a response surface. In many such cases, it can be beneficial to use relatively cheap computing resources to build and search a response surface, rather than incur large costs by directly searching in the problem space. A case in point is the construction of robotic control systems.

In the response surface methodology [1] we construct a response surface and search that surface for likely candidate points, measured according to some criterion. Jones [2] provides a summary of many such methods and discusses their relative merits. As a simple example, consider a noiseless optimisation problem where, given an intial set of samples, we proceed as follows:

1. Fit a basis function model to the data.
2. Find an optimum point of the model and call this point $\mathbf{x}_{\text{new}}$
3. Sample the problem at $\mathbf{x}_{\text{new}}$, and add the result to the current data set.
4. Repeat until satisfied or until is no satisfactory progress is being made.

This is a poor general purpose optimisation algorithm, and for several reasons. One symptom is that it rapidly becomes stuck in one region of the search space, wasting further samples there despite already having good information about it from previous samples.

A more sophisticated search method might attempt to capture regularities about the nature of the search space (rather than merely fitting the existing data), and then use that model more sensibly than simply suggesting the highest predicted point for the next sample. The tendency to *explore* unchartered territory and collect new information about the problem's structure once local territory has been mapped should be an emergent property of a good search algorithm, not a heuristic to be wired in as a quick fix for "premature convergence". This naturally leads us to consider statistical models, where we have a full predictive distribution rather than a single prediction at each search point. Gaussian process models [16] are attractive from this standpoint, for 3 reasons: (i) the predictive distribution is easily obtained, (ii) a Bayesian treatment of hyperparameters allows the model to learn general properties of the surface (smoothness etc.) from previous samples, and (iii) a sensible criterion for drawing the next sample is easily obtained from them.

2 Gaussian processes

Given training data $\mathcal{D}$ consisting of N "input" vectors $\mathbf{x}_i$ paired with scalar "outputs" y_i for $i \in \{1, 2, ..., N\}$, Gaussian process regression is a machine learning technique for infering likely values of y for a novel input $\mathbf{x}$. The study of Gaussian processes for prediction began in geostatistics with kriging [3], [4] and O'Hagan's [5] application to one-dimensional curve fitting. Buntine [6], MacKay [7], and Neal [8] introduced a Bayesian interpretation that provided a consistent method for handling network complexity (see [9, 10] for reviews), followed by regression in a machine learning context [11–13]. See [14–16] for good introductions. Interesting machine learning applications include reinforcement learning [17], incorporation of derivative observations [18], speeding up the evaluation of Bayesian integrals [19, 20], and as models of dynamical systems [21].

The key assumption is that the posterior distribution $p(y|\mathbf{x}, \mathcal{D})$ is Gaussian. To compute its mean and variance, one specifies a valid covariance function $\text{cov}(\mathbf{x}, \mathbf{x}')$, and defines vector $\mathbf{k}$ where $k_i = \text{cov}(\mathbf{x}, \mathbf{x}_i)$, and matrix $\mathbf{C}$ where $C_{ij} = \text{cov}(\mathbf{x}_i, \mathbf{x}_j)$. A common choice of covariance function is the squared exponential, with a length scale r_d associated with each axis:

$$\text{cov}(\mathbf{x}_i, \mathbf{x}_j) \;=\; \alpha \exp\left[-\frac{1}{2}\sum_{d=1}^{D}\frac{(x_i^{(d)} - x_j^{(d)})^2}{2r_d^2}\right] \;+\; \beta\delta_{ij} \tag{1}$$

Here $\boldsymbol{\theta} = \{\alpha, r_1, \ldots, r_D, \beta\}$ are hyperparameters, for which Maximum a posteriori (MAP) values can be inferred from the data [16] by maximising the posterior density $p(\boldsymbol{\theta}|\mathbf{X}, \mathbf{y})$, which is the product of the likelihood function and a prior density over hyperparameters. In the experiments here we used the following log normal priors for $p(\boldsymbol{\theta})$: $\log\alpha \sim \mathcal{N}(-\log 0.5, 0.5)$, $\log r_d \sim \mathcal{N}(-\log 0.5, 0.5)$, and $\log\beta \sim \mathcal{N}(-\log 0.05, 0.5)$.

At any test point $\mathbf{x}$ we then have a predictive distribution that can be shown to be Gaussian with mean $y(\mathbf{x}) = \mathbf{k}^{\mathrm{T}}\mathbf{C}^{-1}\mathbf{y}$, and variance $s^2(\mathbf{x}) = \kappa - \mathbf{k}^{\mathrm{T}}\mathbf{C}^{-1}\mathbf{k}$ where $\kappa = \text{cov}(\mathbf{x}, \mathbf{x})$ (*eg.* for the above covariance function $\kappa = \alpha + \beta$).

3 Expected Improvement

If the model we build provides a predictive distribution at any test point, we can use it to ask what improvement, over our current best sample, do we expect to get from sampling at any test point. Such a measure is known as the expected improvement (e.g. see [2]). The expected improvement ("EI") is particularly straightforward to calculate in the case of a Gaussian process.

For a maximisation problem, the predicted improvement at $\mathbf{x}$ is $I(\mathbf{x}) = \hat{y}(\mathbf{x}) - f_{\text{best}}$, where f_{best} is the current best score and $\hat{y}(\mathbf{x})$ is the model's prediction at $\mathbf{x}$. The prediction is Gaussian distributed as $\hat{y}(\mathbf{x}) \sim \mathcal{N}(y(\mathbf{x}), s^2(\mathbf{x}))$, and so is the improvement: $I \sim \mathcal{N}(y(\mathbf{x}) - f_{\text{best}}, s^2(\mathbf{x}))$. The expected improvement at $\mathbf{x}$ for models with Gaussian predictive distributions is therefore

$$EI(\mathbf{x}) \;= E[\max\{0, I(\mathbf{x})\}] \;=\; \int_{I=0}^{I=\infty} Ip(I)dI \;=\; s(\mathbf{x})\,[u\,\Phi(u) + \phi(u)]$$

where $u = \frac{y(\mathbf{x}) - f_{\text{best}}}{s(\mathbf{x})}$. The functions $\Phi(\cdot)$ and $\phi(\cdot)$ are the normal cumulative distribution and normal density function respectively:

$$\Phi(u) = \frac{1}{2}\,\text{erf}\left(\frac{u}{\sqrt{2}}\right) + \frac{1}{2} \qquad\qquad \phi(u) = \frac{1}{\sqrt{2\pi}}\exp\left(-\frac{u^2}{2}\right)$$

Figure 1 illustrates the concept of expected improvement for a GP model in a maximisation context.

To find a new search point that maximises the expected improvement, we can also make use of gradient information. The gradient of EI with respect to $\mathbf{x}$ is:

$$\frac{\partial EI(\mathbf{x})}{\partial \mathbf{x}} \;=\; \left[u\Phi(u) + \phi(u)\right]\frac{\partial s(\mathbf{x})}{\partial \mathbf{x}} \;+\; s(\mathbf{x})\Phi(u)\frac{\partial u}{\partial \mathbf{x}}$$

(the other two terms cancel) where

$$\frac{\partial s(\mathbf{x})}{\partial \mathbf{x}} = -\left(\frac{\partial \mathbf{k}^{\mathrm{T}}}{\partial \mathbf{x}}\mathbf{C}^{-1}\mathbf{k}\right)\Big/ s(\mathbf{x}), \quad \text{and} \quad \frac{\partial u}{\partial \mathbf{x}} = \left(\frac{\partial \mathbf{k}^{\mathrm{T}}}{\partial \mathbf{x}}\mathbf{C}^{-1}\mathbf{y} - u\frac{\partial s(\mathbf{x})}{\partial \mathbf{x}}\right)\Big/ s(\mathbf{x})$$

The Jacobian $\frac{\partial \mathbf{k}^{\mathrm{T}}}{\partial \mathbf{x}}$ is dependent on the form of the covariance function: it is $D \times N$ matrix whose $(i,j)^{\text{th}}$ element is $\frac{\partial \text{cov}(\mathbf{x}, \mathbf{x}_j)}{\partial x_i}$ where $\mathbf{x} = [x_1 \ldots x_D]^{\mathrm{T}}$.

4 GPO

In this section, the Gaussian Processes for Optimisation (GPO) algorithm is described. Our goal is find the position $\mathbf{x}_{\text{opt}}$ of the optimum of a surface $f^*(\cdot)$, using as few samples as possible, so we update *a*ll parameters in the model as each new data point arrives. GPO begins by assuming a start point $\mathbf{x}_0$, which in general is sampled from some distribution reflecting our prior beliefs about $f^*(\cdot)$.

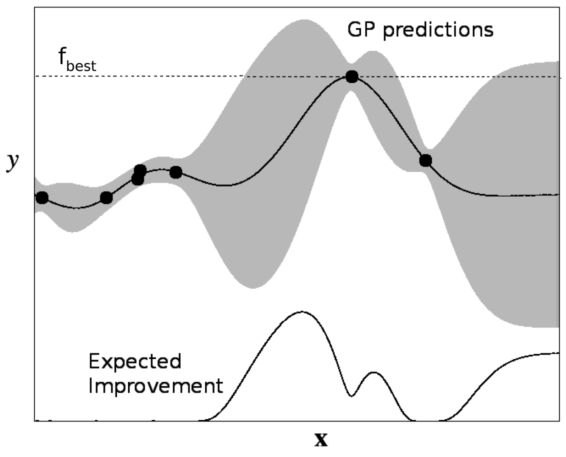

Fig. 1. Expected Improvement for a GP model in a maximisation context. Data points are shown as black dots. The GP model's predictive distribution has a mean shown by the black line that fits the data, and a standard deviation shown by the shaded region surrounding the mean. The black line at the bottom shows the expected improvement given this particular GP model. Notice that expected improvement is maximum when the model's prediction is better than or close to f_{best} *and* the predictive variance is high. Expected improvement is quite high when when the model prediction is low *and* the predictive variance is high. But when the model prediction is low *and* the predictive variance is low, the expected improvement is almost zero.

At each iteration, the algorithm builds a GP model of the current data by finding $\boldsymbol{\theta}_{MAP}$. If multimodal posterior distributions are considered a problem, then the GP model can be built by restarting the log posterior density maximisation multiple times from samples drawn from $p(\boldsymbol{\theta})$.

The resulting GP model is used to select the next $\mathbf{x}_{\text{new}}$ to evaluate by finding a point that maximises the expected improvement. This can be achieved by using the gradient of the expected improvement as input to (*eg.*) the conjugate gradient algorithm [23]. To overcome the problem of suboptimal local maxima, multiple restarts are made starting from randomly selected points in the current data set $\mathbf{X}$. The new observation y_{new} is found from $f^*(\mathbf{x}_{\text{new}})$ and the results are added to the current data set. Iterations continue until some stopping criterion is met.

Figure 2 shows the results of running standard GPO on a simple $1D$ toy problem. Notice how the search initially focuses on the suboptimal maximum on the left. However, once the algorithm has sampled here a few times, the expected improvement of sampling in this suboptimal region diminishes quickly. At this point, the search expands into new regions. The algorithm will exploit local knowledge to make improvement, but will also explore when the expected returns of this exploitation decrease.

Notice that the squared exponential form of the covariance function is "axis-aligned" in the sense that it assigns a separate length scale to each direction in

Algorithm 1: GPO

```
   Input: optimisation problem $f^*(\cdot)$, and starting point $\mathbf{x}_0$
1  $\mathbf{x}_{\text{best}} \leftarrow \mathbf{x}_0,\ y_{\text{best}} \leftarrow f^*(\mathbf{x}_0)$;
2  $\mathbf{y} \leftarrow [y_{\text{best}}],\ \mathbf{X} \leftarrow \mathbf{x}_{\text{best}}$;
3  repeat
4      $\boldsymbol{\theta}_{MAP} \longleftarrow \arg\max_{\theta} p(\boldsymbol{\theta}|\mathbf{X}, \mathbf{y})$;
5      $\mathbf{x}_{\text{new}} \longleftarrow \arg\max_{x} EI(\mathbf{x}|\mathbf{X}, \mathbf{y}, \boldsymbol{\theta}_{MAP})$;
6      $y_{\text{new}} \leftarrow f^*(\mathbf{x}_{new})$;
7      if $y_{new} \geq y_{best}$ then
8          $y_{\text{best}} \leftarrow y_{\text{new}},\ \mathbf{x}_{\text{best}} \leftarrow \mathbf{x}_{\text{new}}$;
9      $\mathbf{X} \leftarrow [\mathbf{X}\,|\,\mathbf{x}_{\text{new}}],\ \mathbf{y} \leftarrow \left[\mathbf{y}^{\text{T}}\,|\,y_{\text{new}}\right]^{\text{T}}$;
10     $\mathbf{V}_{rot} \leftarrow \arg\max_{\mathbf{V}} p(\boldsymbol{\theta}^V_{MAP}|\mathbf{V}\mathbf{X}, \mathbf{y})$ where $\boldsymbol{\theta}^V_{MAP} = \arg\max_{\theta} p(\boldsymbol{\theta}|\mathbf{V}\mathbf{X}, \mathbf{y})$;
11     $\mathbf{X} \leftarrow \mathbf{V}_{rot}\mathbf{X}$
12 until (stopping criteria satisfied);
13 return $\mathbf{x}_{best}$
```

input space. However in optimization problems in more than one dimension this is a poor assumption, since there will often be interactions between input variables which induce covariance structure that is "off-axis". To allow the algorithm to discover such structure, at each step we test out a number of random rotations $\mathbf{X}' = \mathbf{V}_{rot}\mathbf{X}$ of the current input data $\mathbf{X}$ using randomly generated orthonormal matrices $\mathbf{V}_{rot}$, and choose the rotated data set for which $p(\boldsymbol{\theta}_{MAP}|\mathbf{X}', \mathbf{y})$ is greatest. This preprocessing of the data is equivalent to learning a rotated covariance function and allows GPO deal with off-axis structure in the properties of the search surface. Further details are given in [20].

Jones [2] first introduced kriging for optimisation using expected improvement to select the next iterate. Büche, Schraudolph and Koumoutsakos [22] explicitly used Gaussian processes for optimisation, and demonstrated the algorithm's effectiveness on a number of benchmark problems. This work did not make use of expected improvement, did not place prior distributions over the hyperparameters, and did not consider the deficiencies of using an axis-aligned covariance function to optimise objective functions with correlated output (dependent) variables. The algorithm presented here takes all these factors into account. Recently [28] have used similar ideas to those presented here to optimize the gait of a mobile robot, although they use a different criterion (probability of *any* improvement) and don't deal with correlated variables.

5 Double Pole Balancing with GPO

The double pole balancing task consists of two upright poles (or inverted pendulums), attached by hinges to a cart. The goal is to keep the two poles balanced by applying a $[-10, 10]N$ force to the cart. Balanced poles are defined as within $\pm 36^\circ$ from vertical, and the cart is limited to a track which is $4.8m$ long. The

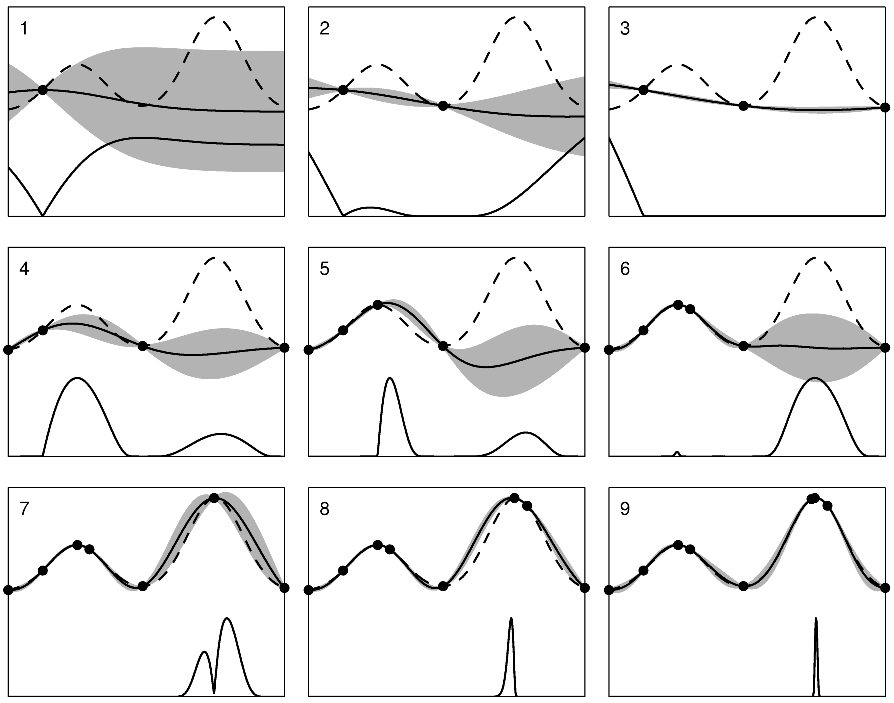

Fig. 2. GPO applied to a $1D$ function for 9 iterations. The dashed line shows the underlying function to be optimised. The sample points are shown as black dots, along with the model's predictive distribution (the line surrounded by the shaded area shows the mean and standard deviation). The expected improvement is rescaled and shown by the solid line at the bottom of each window. Note the hill-climbing behaviour (*eg.* iterations 4-5) exploiting regularity of the surface, and exploratory behaviour at other times (*eg.* iteration 6).

controller is supplied with inputs from sensors measuring the cart's position and velocity $x, \dot{x}$ and the angle and angular velocity of each pole with respect to the cart $\theta_1, \dot{\theta}_1, \theta_2, \dot{\theta}_2$. The poles have different lengths and masses (pole 1 is $0.1m$ and $0.01kg$; pole 2 is $1.0m$ and $0.1kg$) and the system is noiseless with initial state vector $\mathbf{s} = [x\ \dot{x}\ \theta_1\ \dot{\theta}_1\ \theta_2\ \dot{\theta}_2]^{\mathrm{T}} = [0\ 0\ 0\ 0\ \frac{\pi}{180}\ 0]^{\mathrm{T}}$, where angles are measured in rad from vertical, and angular velocities are measured in radians / sec. The centre of the track is defined as $x = 0$, and is the position of the cart at the beginning of the task. Note that this task is Markovian as the full system state vector $\mathbf{s}$ is available to the controller and is the same as the "double pole balancing with velocity information" problem as presented by Stanley and Miikkulainen [24–26].

If the goal is to keep the poles balanced for as long as possible, one solution is to wiggle the poles back and forth about a central position. To prevent this, Gruau [27] defined a fitness function that penalises such solutions, $f_{\text{gruau}} = 0.1 f_1 + 0.9 f_2$, [24, 26]. The two components are defined over 1000 time steps (10 seconds simulated time):

$$f_1 = t/1000 \tag{2}$$

$$f_2 = \begin{cases} 0 & \text{if } t < 100, \\ \frac{0.75}{\sum_{i=t-100}^{t} (|x^i| + |\dot{x}^i| + |\theta_1| + |\dot{\theta_1}|)} & \text{otherwise.} \end{cases} \tag{3}$$

where t is the number of time steps both poles remained balanced during a trial lasting 10 seconds. f_{gruau} can be maximised by keeping both poles balanced, and by maintaining the cart steady at the centre of the track during the final second of the trial. Effectively, to maximise f_{gruau} the controller must balance the poles without 'wiggling'. As the denominator of (3) approaches zero f_2 approaches infinity, so f_{gruau} was non-linearly rescaled into the range $[0, 1]$ giving $f_{\text{gpo}} = \tanh(f_{\text{gruau}}/2)$. Controllers were considered successful solutions when $f_{\text{gpo}} \geq \tanh\left(\frac{5}{2}\right)$.

5.1 Feedforward Neural Network Controllers

The double pole balancing task described above is a non-linear, unstable control problem. However, because the poles have different lengths and masses, the system is controllable. In addition, the task is Markovian. Overall, full knowledge of the system state is sufficient to balance the poles, and this can be achieved with a mapping from $\mathbf{s}$ to u, our control force. In other words, there exists at least one mapping $\mathbf{s} \mapsto u$ that is capable of balancing the poles. A successful controller must functionally approximate such a mapping.

The control force is implemented by a feedforward neural network with a single hidden layer having H units, and output limited to $[-10, 10]N$:

$$u = 10 \tanh\left(\mathbf{w}_o^{\mathrm{T}} \tanh\left(\mathbf{W}_i^{\mathrm{T}} \mathbf{s} + \mathbf{b}\right)\right)$$

where $\mathbf{w}_o$ is an $H \times 1$ vector of output weights, $\mathbf{b}$ is an $H \times 1$ vector of biases, $\mathbf{W}_i$ is a $6 \times H$ matrix of input weights, $\mathbf{s}$ is the 6×1 state vector.

5.2 Optimisation and Incremental Network Growth

We optimized $f^{\star} = f_{\text{gruau}}(\mathbf{w}_o, \mathbf{W}, \mathbf{b})$. The optimisation started with a single unit in the network, $H = 1$. Initially, therefore, there were 8 parameters that need optimising. GPO, with the axis-aligned covariance function (Eq. 1) and data rotation prior to training, was used optimise these weights until either there had been no improvement in the best fitness for 64 consecutive samples, or 250 samples had been taken. When either of these conditions were met, the current optimised parameters were frozen, and a new unit with zeroed weights was added to the network. The cycle repeated until a solution was found, or 5 units and their weights had been optimised. Note that the initial weights for the first iteration were zero (the algorithm started from the same place every time it was run), and 8 parameters were being optimised at every stage.

Figure 3 shows 100 runs of GPO on this task. 96 of these runs found a successful controller solution with $f_{\text{gruau}} \geq \tanh\left(\frac{5}{2}\right)$ in < 1000 evaluations (samples). The median number of evaluations required to find a successful controller was 151, and the mean was 194. The majority (78%) of successful controllers used only 1 unit in their solution (i.e. 6 input weights, 1 bias and 1 output weight). 12 runs used 2 units, while 9 needed from 3 to 5 units.

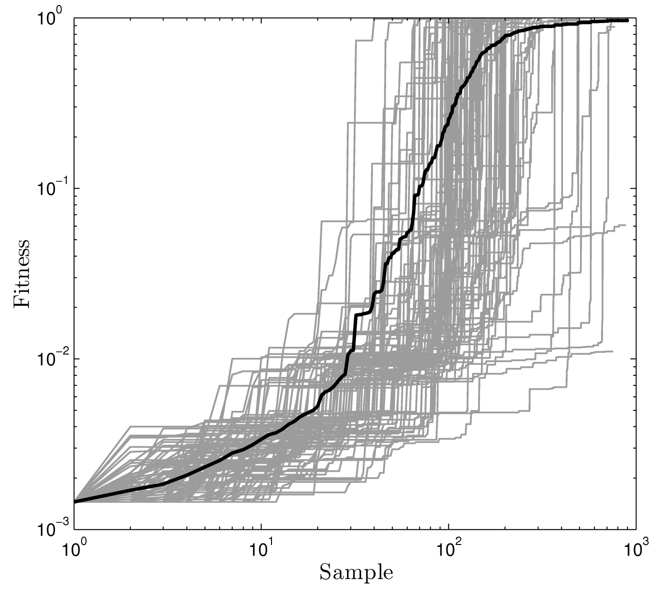

Fig. 3. Double Pole Balancing with Gruau fitness, optimised using GPO. The figure shows 100 separate optimisations (grey) and the average (black).

Stanley and Miikkulainen [24–26] introduced “Neuroevolution of Augmenting Topologies” (NEAT), and applied it to a number of pole balancing tasks, including the double pole balancing problem presented above. The NEAT method is a genetic algorithm with mutation and crossover operations specially crafted to enhance the evolution of neural network controllers. The details of NEAT are not important here, other than that it produced impressive results in solving the double pole balancing task with velocity information. NEAT required an average 3578 network evaluations to find a controller solution, which compared favourably with other results from literature.

GPO produced successful controllers in 96 out of 100 trials, and did so with a mean of 194 evaluations. This is a significant improvement over NEAT.

6 Summary

We have presented an optimization algorithm that uses Gaussian process regression to suggest where to take samples, with the goal of finding a good solution in a small number of function evaluations. GPO uses data rotation to allow for interactions between input variables, and learns about the search space as it proceeds, by finding maximum a posteriori values for hyperparameters at every step. In this way search is carried out as much as possible on the model instead of the real world, using the conjugate gradient method to find points having the highest expected improvement. Sequences of samples taken in this way exhibit a variety of intuitively sensible yet emergent properties, such as hill-climbing behaviour, and avoidance of regions the model considers to be well characterised already. Despite being a deterministic procedure, GPO also shows non-trivial exploratory behaviour, in testing out regions that seem promising under the current model.

As a demonstration of this algorithm we applied it to the task of finding optimal parameters for the double pole-balancing problem, with that result that it learns successful controllers using about 200 evaluations of possible controllers, compared to over 3500 reported for other algorithms.

References

1. Myers, R. H., and Montgomery, D. C. *Response Surface Methodology: Process and Product Optimization Using Designed Experiments*, 2 ed. Wiley-Interscience, 2002.
2. Jones, D. R. A taxonomy of global optimization methods based on response surfaces. *Journal of Global Optimization 21* (2001), 345–383.
3. Matheron, G. Principles of geostatistics. *Economic Geology 58* (1963), 1246–1266.
4. Cressie, N. *Statistics for Spatial Data.* Wiley, 1993.
5. O'Hagan, A. Curve fitting and optimal design for prediction (with discussion). *J. Roy. Statist. Soc. Ser. B 40* (1978), 1–42.
6. Buntine, W., and Weigend, A. Bayesian backpropagation. *Complex Systems 5* (1991), 603–643.
7. MacKay, D. J. C. *Bayesian Methods for Adaptive Models.* PhD thesis, California Institute of Technology, 1992.
8. Neal, R. M. Bayesian training of backpropagation networks by the hybrid Monte Carlo method. Tech. Rep. CRG-TR-92-1, Dept. of Computer Science, Univ. of Toronto, 1992.
9. MacKay, D. J. C. Probable networks and plausible predictions — a review of practical Bayesian methods for supervised neural networks. *Network: Computation in Neural Systems 6* (1995), 469–505.
10. Bishop, C. *Neural Networks for Pattern Recognition.* Oxford University Press, 1995.
11. Williams, C. K., and Rasmussen, C. E. Gaussian processes for regression. In *Advances in Neural Information Processing Systems* (1996), D. Touretzsky, M. Mozer, and M. Hasselmo, Eds., vol. 8.

12. Rasmussen, C. E. *Evaluation of Gaussian Processes and other methods for Non-Linear Regression.* PhD thesis, Graduate Department of Computer Science, University of Toronto, 1996.
13. Neal, R. Monte carlo implementation of Gaussian process models for Bayesian regression and classification. Tech. Rep. CRG-TR-97-2, Dept. of Computer Science, Univ. of Toronto, 1997.
14. Gibbs, M. *Bayesian Gaussian Processes for Classification and Regression.* PhD thesis, University of Cambridge, Cambridge, U.K., 1997.
15. MacKay, D. J. *Information theory, inference, and learning algorithms.* Cambridge University Press, 2003.
16. Rasmussen, C. E., and Williams, C. K. I. *Gaussian Processes for Machine Learning.* The MIT Press, 2006.
17. Rasmussen, C. E., and Kuss, M. Gaussian processes in reinforcement learning. In *Advances in Neural Information Processing Systems, NIPS* (2002), S. Thrun, L. Saul, and B. Schlkopf, Eds., vol. 16, The MIT Press.
18. Solak, E., Murray-Smith, R., Leithead, W. E., Leith, D., and Rasmussen, C. E. Derivative observations in Gaussian process models of dynamic systems. In *Advances in Neural Information Processing Systems, NIPS* (2003), vol. 15, The MIT Press, pp. 1033–1040.
19. Rasmussen, C. E. Gaussian processes to speed up hybrid monte carlo for expensive bayesian integrals. In *Bayesian Statistics* (2003), J. M. Bernardo, M. J. Bayarri, J. O. Berger, A. P. Dawid, D. Heckerman, A. F. M. Smith, and M. West, Eds., vol. 7, Oxford University Press, pp. 651–659.
20. Boyle, P. Gaussian processes for regression and optimisation. PhD thesis, Victoria University of Wellington, New Zealand, 2007.
21. Wang, J., Fleet, D., and Hertzmann, A. Gaussian process dynamical models. In *Advances in Neural Information Processing Systems, NIPS* (2006), Y. Weiss, B. Schlkopf, and J. Platt, Eds., vol. 18, The MIT Press, pp. 1443–1450.
22. Buche, D., Schraudolph, N., and Koumoutsakos, P. Accelerating evolutionary algorithms with gaussian process fitness function models. *IEEE Transactions on Systems, Man, and Cybernetics 35*, 2 (2005), 183–194.
23. Press, W., Flannery, B., Teukolsky, S., and Vetterling, W. *Numerical recipes in C: The art of scientific computing.* Cambridge University Press., 1989.
24. Stanley, K. O., and Miikkulainen, R. Evolving neural networks through augmenting topologies. *Evolutionary Computation 10*, 2 (2002), 99–127.
25. Stanley, K. O., and Miikkulainen, R. Efficient reinforcement learning through evolving neural network topologies. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002)* (2002), Morgan Kaufmann.
26. Stanley, K. O. *Efficient Evolution of Neural Networks Through Complexification.* PhD thesis, Department of Computer Sciences, The University of Texas at Austin, 2004.
27. Gruau, F., Whitley, D., and Pyeatt, L. A comparison between cellular encoding and direct encoding for genetic neural networks. In *Genetic Programming 1996: Proceedings of the First Annual Conference* (Stanford University, CA, USA, 28–31 1996), J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo, Eds., MIT Press, pp. 81–89.
28. Lizotte,D., Wang,T., Bowling,M. and Schuurmans,D. Automatic gait optimization with gaussian process regression In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence*, pages 944–949, 2007.