Bayesian Optimization

CSC2541 - Topics in Machine Learning
Scalable and Flexible Models of Uncertainty
University of Toronto - Fall 2017

Overview

1. Bayesian Optimization of Machine Learning Algorithms
2. Gaussian Process Optimization in the Bandit Setting
3. Exploiting Structure for Bayesian Optimization

Bayesian Optimization of Machine Learning Algorithms

J. Snoek, A. Krause, H. Larochelle, and R.P. Adams (2012)
Practical Bayesian Optimization of Machine Learning Algorithms

J. Snoek et al. (2015)
Scalable Bayesian Optimization Using Deep Neural Nets

Presentation by: Franco Lin, Tahmid Mehdi, Jason Li

Motivation

Performance of Machine Learning algorithms are usually dependent on the choice of hyperparameters

Picking the optimal hyperparameter values are hard

- Ex. grid search, random search, etc.
- Instead could we use a model to select which hyperparameters will be good next?

Bayes Opt. of Machine Learning Algorithms

- Bayesian Optimization uses all of the information from previous evaluations and performs some computation to determine the next point to try
- If our model takes days to train, it would be beneficial to have a well structured way of selecting the next combination of hyperparameters to try
- Bayesian Optimization is much better than a person finding a good combination of hyperparameters

Bayesian Optimization

Intuition:

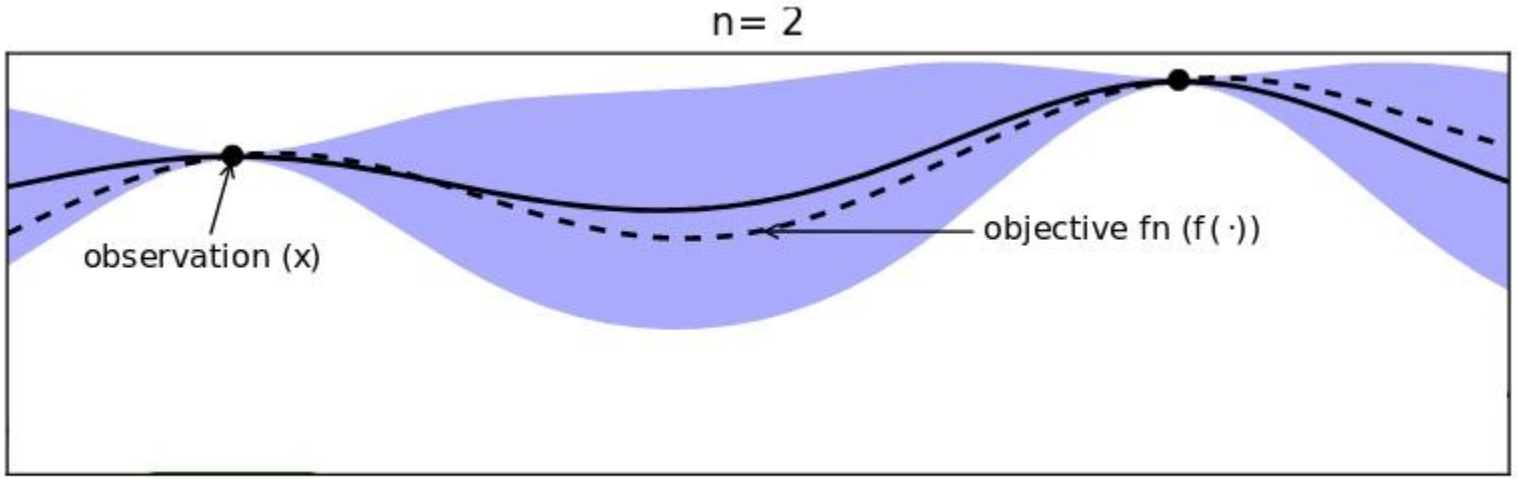
We want to find the peak of our true function (eg. accuracy as a function of hyperparameters)

To find this peak, we will fit a Gaussian Process to our observed points and pick our next best point where we believe the maximum will be.

This next point is determined by an acquisition function - that trades of exploration and exploitation

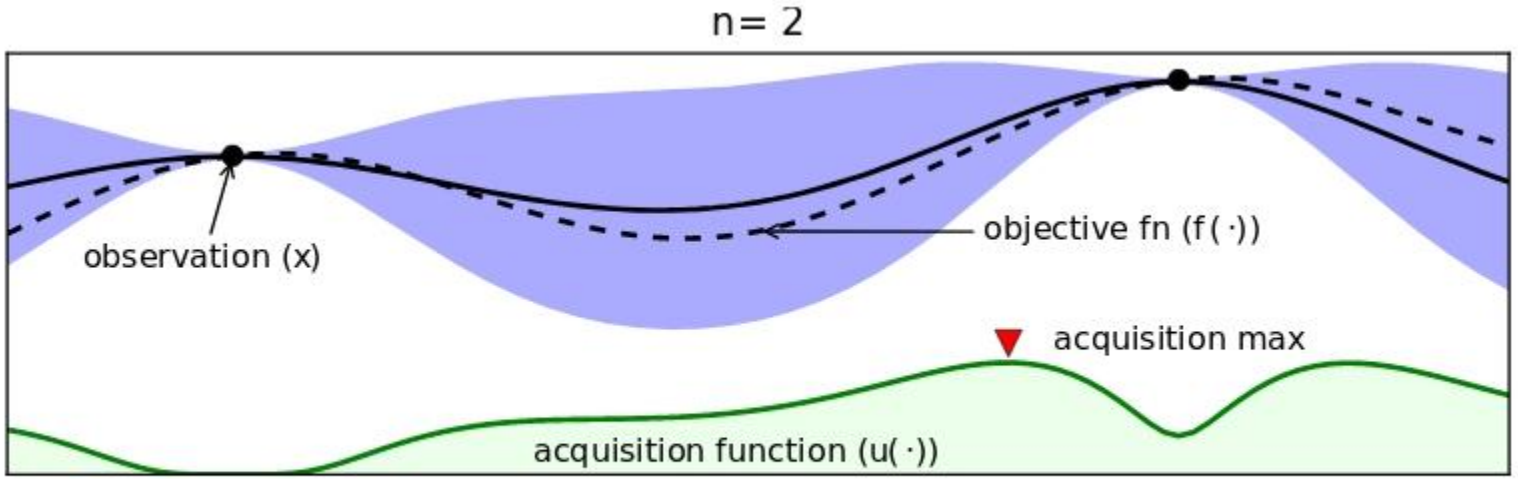
Lecture by Nando de Freitas and a Tutorial paper by Brochu et al.

Bayesian Optimization Tutorial



Brochu et al., 2010, A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning

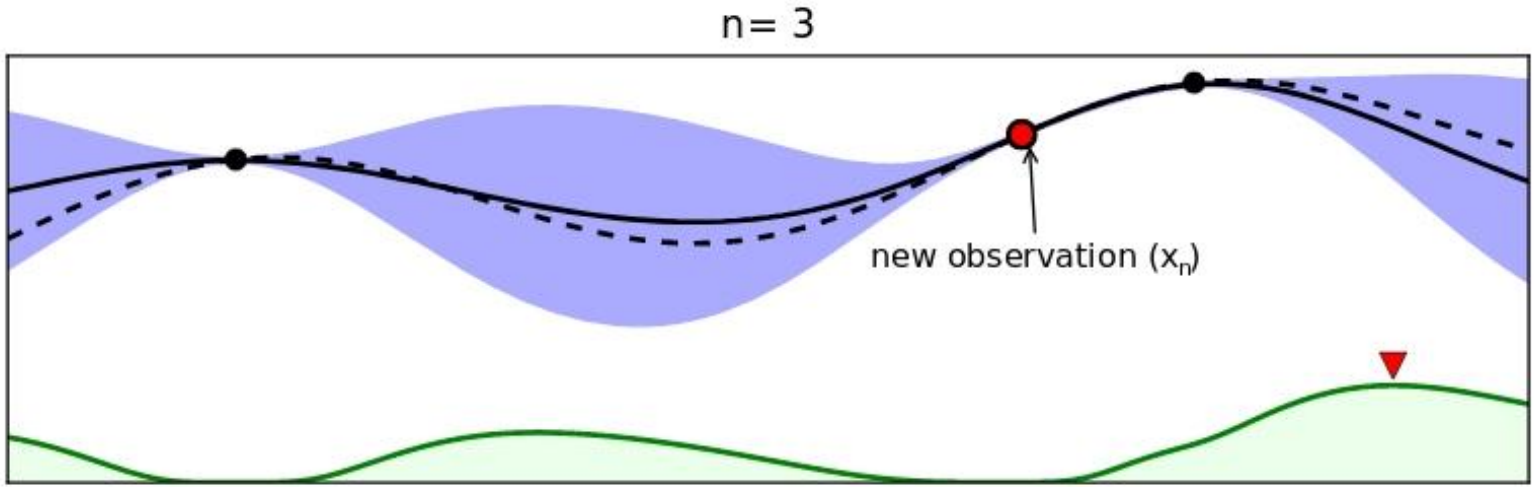
Bayesian Optimization Tutorial



Find the next best point x_n that maximizes acquisition function

Brochu et al., 2010, A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning

Bayesian Optimization Tutorial



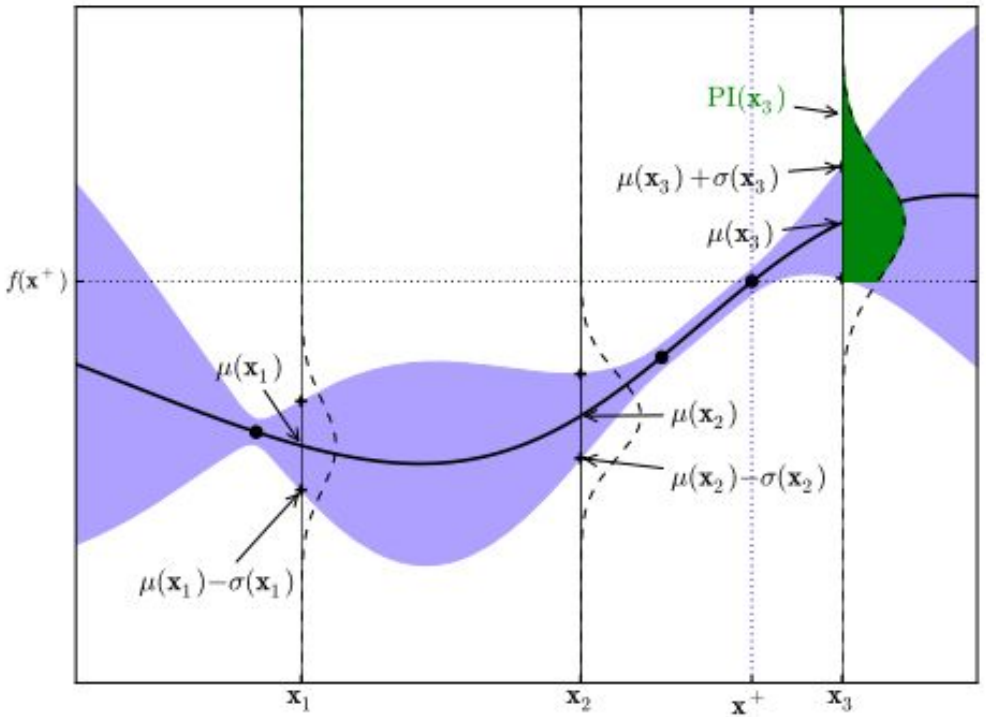
Evaluate f at the new observation x_n and update posterior

Update acquisition function from new posterior and find the next best point

Brochu et al., 2010, A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning

Acquisition Function Intuition

- We will use the acquisition function Probability of Improvement (PI) as an example.
- We want to find the point with the largest area above our best value
- This corresponds to the maximum of our acquisition function



Brochu et al., 2010, A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning

Acquisition Functions

- Guides the optimization by determining which point to observe next and is easier to optimize to find the next sample point

Probability of Improvement (PI)

$$a_{\mathsf{PI}}(\mathbf{x}\,;\,\{\mathbf{x}_n, y_n\}, \theta) = \Phi(\gamma(\mathbf{x})) \qquad \gamma(\mathbf{x}) = \frac{f(\mathbf{x}_{\mathsf{best}}) - \mu(\mathbf{x}\,;\,\{\mathbf{x}_n, y_n\}, \theta)}{\sigma(\mathbf{x}\,;\,\{\mathbf{x}_n, y_n\}, \theta)}$$

Expected Improvement (EI)

$$a_{\mathsf{EI}}(\mathbf{x}\,;\,\{\mathbf{x}_n, y_n\}, \theta) = \sigma(\mathbf{x}\,;\,\{\mathbf{x}_n, y_n\}, \theta)\,(\gamma(\mathbf{x})\,\Phi(\gamma(\mathbf{x})) + \mathcal{N}(\gamma(\mathbf{x})\,;\,0, 1))$$

GP-Upper/Lower Confidence Bound (GP-UCB/LCB)

$$a_{\mathsf{LCB}}(\mathbf{x}\,;\,\{\mathbf{x}_n, y_n\}, \theta) = \mu(\mathbf{x}\,;\,\{\mathbf{x}_n, y_n\}, \theta) - \kappa\,\sigma(\mathbf{x}\,;\,\{\mathbf{x}_n, y_n\}, \theta)$$

The Prior

- Power of Gaussian Process depends on covariance function
- For optimization, we don't want kernels that produce unrealistically smooth sample functions
- Automatic Relevance Determination (ARD) Matern 5/2 kernel is a good choice

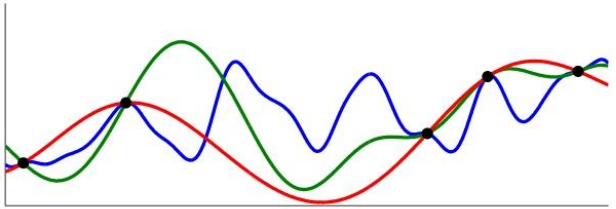
$$K_{\text{M52}}(\mathbf{x}, \mathbf{x}') = \theta_0 \left(1 + \sqrt{5r^2(\mathbf{x}, \mathbf{x}')} + \frac{5}{3}r^2(\mathbf{x}, \mathbf{x}')\right) \exp\left\{-\sqrt{5r^2(\mathbf{x}, \mathbf{x}')}\right\}$$

Kernel Hyperparameters

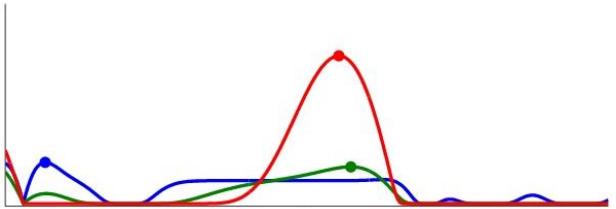
Marginalize over hyperparameters and compute integrated acquisition function

$$\hat{a}(\mathbf{x}\,;\,\{\mathbf{x}_n, y_n\}) = \int a(\mathbf{x}\,;\,\{\mathbf{x}_n, y_n\}, \theta)\, p(\theta \mid \{\mathbf{x}_n, y_n\}_{n=1}^{N})\, \mathrm{d}\theta$$

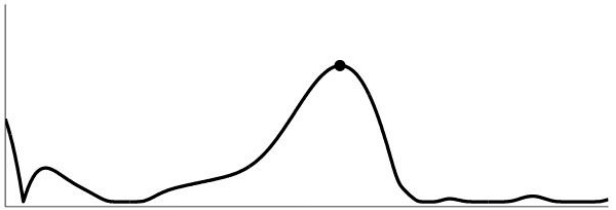
Approximate integral with Monte Carlo methods



(a) Posterior samples under varying hyperparameters



(b) Expected improvement under varying hyperparameters



(c) Integrated expected improvement

Considerations for Bayes Opt

- Evaluating f may be time-consuming

- Modern optimization methods should take advantage of multi-core/parallel programming

Expected Improvement per Second

- Evaluating f will take longer in some regions of the parameter space
- We want to pick points that are likely to be good and evaluated quickly
- Let c(x) be the duration time to evaluate f(x)
- Use GP to model ln[c(x)]
- we can compute predicted expected inverse duration which allows us to obtain the EI per Second as a function of x

Parallelizing Bayes Opt

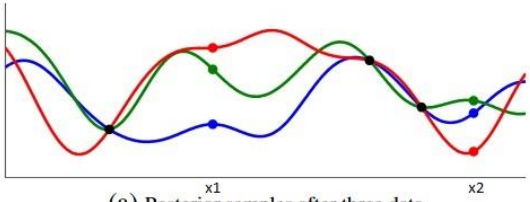
- Can we determine which x to evaluate next, while other points are being evaluated?
- Idea: Utilize tractable properties of GP to get Monte Carlo estimates of acquisition function under different results from pending function evaluations

Consider the case where N evaluations have completed, with data $\{x_n,y_n\}_{n=1}^{N}$, and J evaluations are pending $\{x_j\}_{j=1}^{J}$

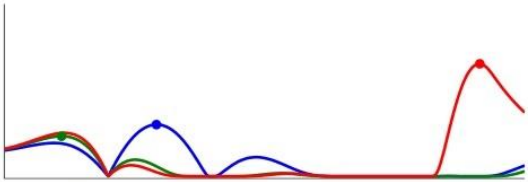
$$\hat{a}(\mathbf{x}\,;\,\{\mathbf{x}_n, y_n\}, \theta, \{\mathbf{x}_j\}) = \int_{\mathbb{R}^J} a(\mathbf{x}\,;\,\{\mathbf{x}_n, y_n\}, \theta, \{\mathbf{x}_j, y_j\})\, p(\{y_j\}_{j=1}^{J} \mid \{\mathbf{x}_j\}_{j=1}^{J}, \{\mathbf{x}_n, y_n\}_{n=1}^{N})\, \mathrm{d}y_1 \cdots \mathrm{d}y_J$$

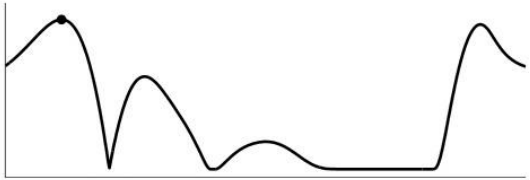
Parallelization Example

- We've evaluated 3 observations and 2 are pending $\{x_1, x_2\}$
- Fit a model for each possible realization of $\{f(x_1), f(x_2)\}$
- Calculate acquisition for each model
- Integrate all acquisitions over x



(a) Posterior samples after three data

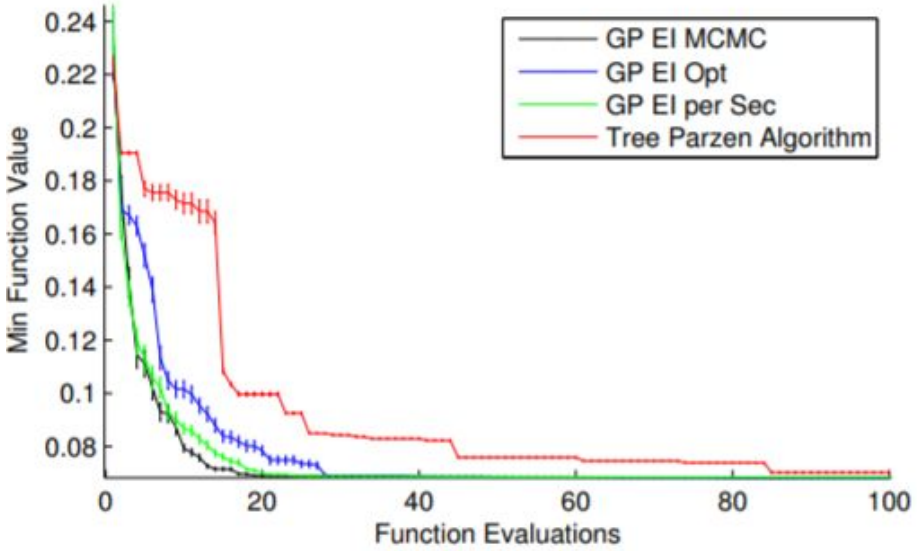

(b) Expected improvement under three fantasies

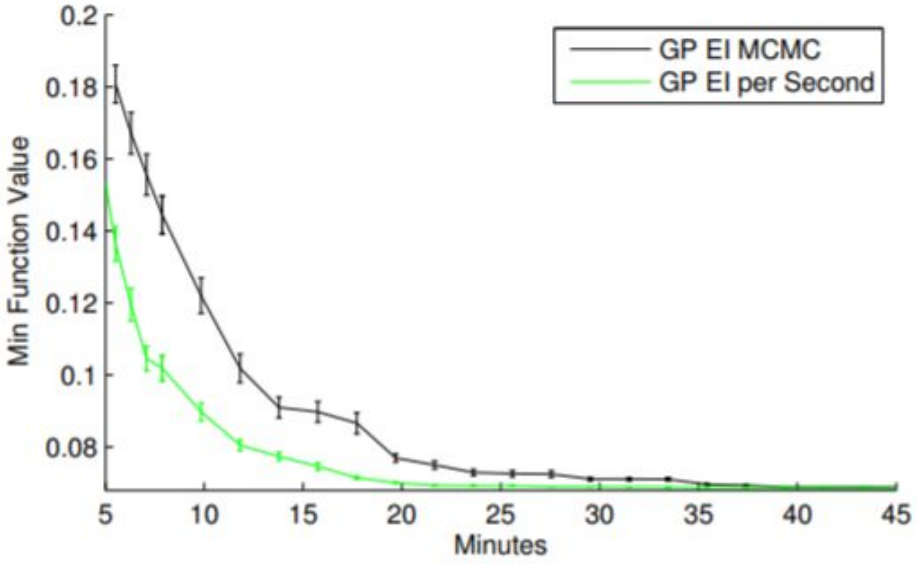

(c) Expected improvement across fantasies

Results

- Branin-Hoo
- Logistic Regression MNIST
- Online LDA
- M3E
- CNN CIFAR-10

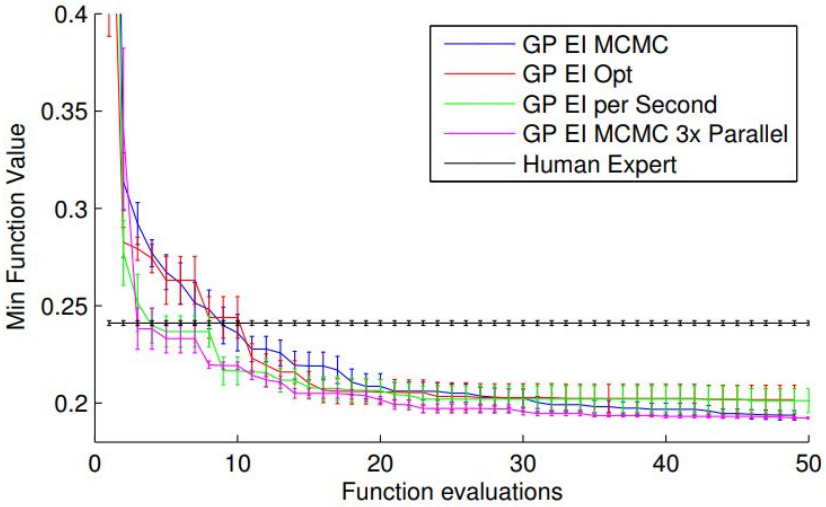
Logistic Regression - MNIST





CIFAR-10

- 3-layer conv-net
- Optimized over
  - Number of epochs
  - Learning rate
  - L2-norm constants
- Achieved state of the art
  - 9.5% test error



GP Bayesian Optimization - Pros and Cons

- Advantages
  - Computes the mean and variance

- Disadvantages
  - Function evaluation is cubic on the number of inputs

Scalable Bayesian Optimization Using Deep Neural Networks

- Replace a Gaussian Process with a Bayesian Neural Network
- Use a deterministic neural network with Bayesian linear regression on the last hidden layer
- More accurately, use Bayesian linear regression with basis functions
  - DNN: R^k -> R^d
  - Bayesian linear regression: R^d -> R
  - k is the dimensionality of the input, and d is the number of hidden units in the last layer

Bayesian Linear Regression

- Still requires an inversion
- Linear in the number of observations
- Cubic in the basis function dimension or number of hidden units, D

$$\mu(\mathbf{x}; \mathcal{D}, \boldsymbol{\Theta}) = \mathbf{m}^{\mathsf{T}} \boldsymbol{\phi}(\mathbf{x}) + \eta(\mathbf{x}) \,, \tag{4}$$

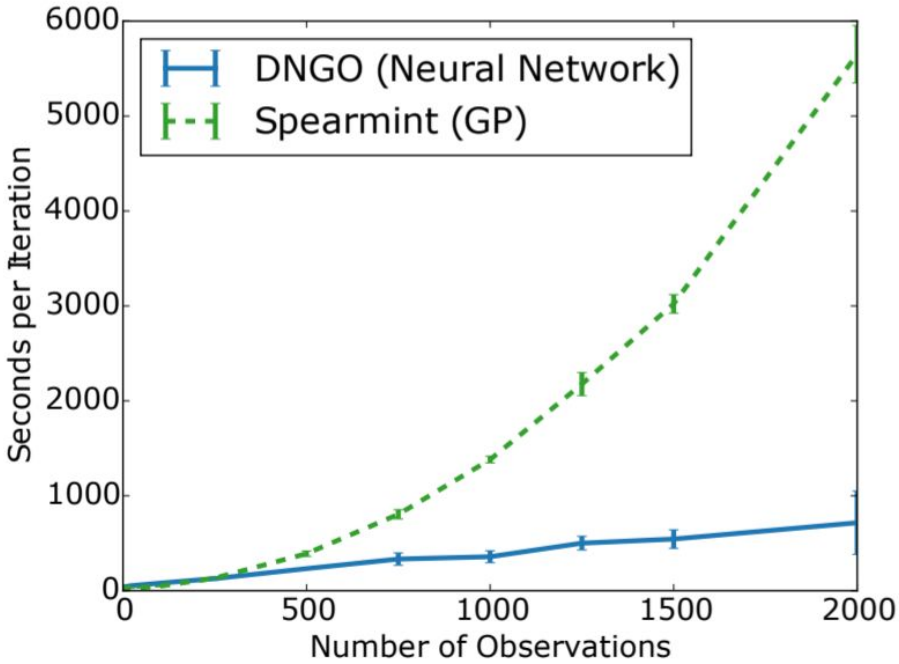
$$\sigma^2(\mathbf{x}; \mathcal{D}, \boldsymbol{\Theta}) = \boldsymbol{\phi}(\mathbf{x})^{\mathsf{T}} \mathbf{K}^{-1} \boldsymbol{\phi}(\mathbf{x}) + \frac{1}{\beta} \tag{5}$$

where

$$\mathbf{m} = \beta \mathbf{K}^{-1} \boldsymbol{\Phi}^T \tilde{\mathbf{y}} \in \mathbb{R}^D \tag{6}$$

$$\mathbf{K} = \beta \boldsymbol{\Phi}^T \boldsymbol{\Phi} + \mathbf{I}\alpha \in \mathbb{R}^{D \times D}. \tag{7}$$

Results


DNGO (Neural Network)
Spearmint (GP)
Seconds per Iteration
Number of Observations
0
1000
2000
3000
4000
5000
6000
0
500
1000
1500
2000

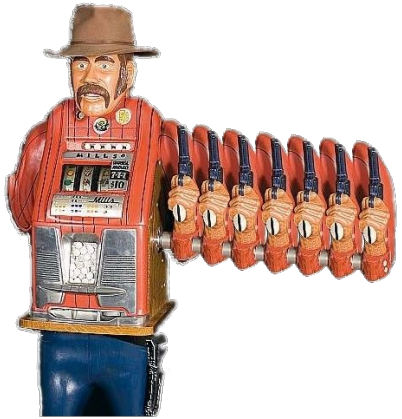
Gaussian Process Optimization in the Bandit Setting

N. Srinivas, A. Krause, S. Kakade, and M. Seeger (2010)
Gaussian process optimization in the bandit setting: No regret and experimental design

Presentation by: Shadi Zabad, Wei Zhen Teoh, Shuja Khalid

The Bandits are Back!

- We just learned about some exciting **new techniques for optimizing black box functions**. Can we apply them to the classic multi-armed bandit problem?
- In this case, we'd like to optimize the **unknown reward function**.



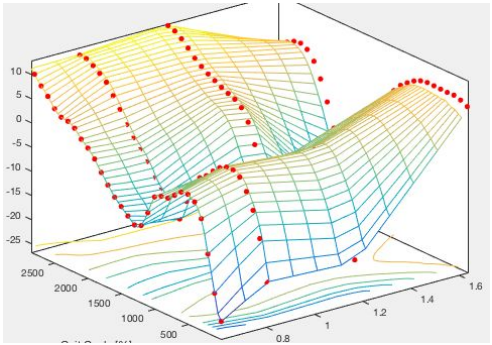
Credit: D. Tolpin at ECAI 2012

Cost-bounded Optimization

- In the bandit setting, the optimization procedure is **cost-sensitive**: There's a cost incurred each time we evaluate the function.
- The cost is **proportional** to how far the point is from the point of maximum reward.
- Therefore, we have to optimize the reward function while minimizing the cost incurred along the way.

An Infinite Number of Arms

- The multi-armed bandit algorithms and analyses we've seen so far assumed a **discrete decision space** (e.g. a decision space where we have K slot machines).
- However, in Gaussian Process optimization, we'd like to consider **continuous decision spaces**.
- And in this domain, some of the theoretical analyses that we derived for discrete decision spaces **can't be extended in a straightforward manner**.

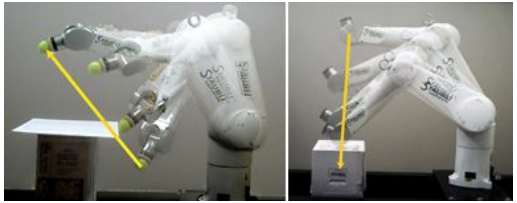


Credit: @Astrid, CrossValidated

Multi-armed Bandit Problem: Recap

- **The basic setting**: We have a decision space that's associated with an **unknown reward function**.
  - **Discrete examples**: Slot machines at a casino, drug trials.
  - **Continuous examples**: Digging for oil or minerals, robot motion planning.
- In this setting, a "**policy**" is a procedure for **exploring** the decision space. An optimal policy is defined as a procedure which minimizes a cost measure. The most common cost measure is the "**regret**".

Credit: Gatis Gribusts



Credit: Intelligent Motion Lab (Duke U)

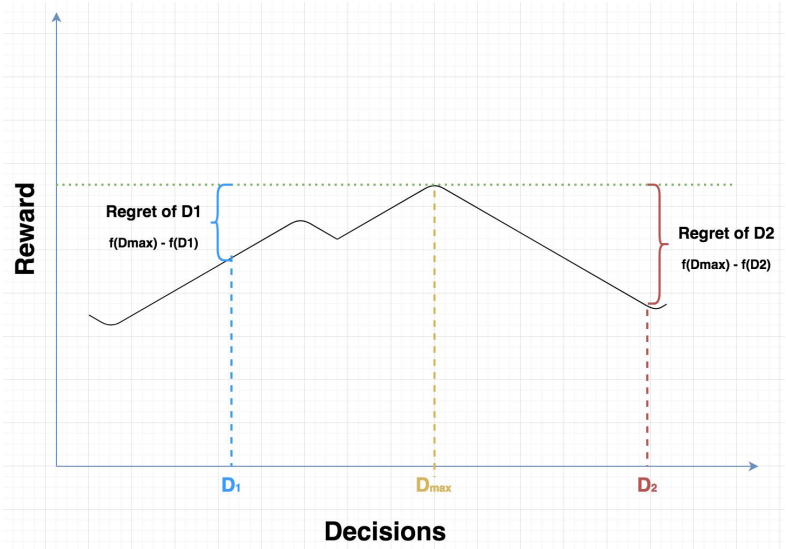
A Measure of Regret

- In general terms, **regret** is defined as “the loss in reward due to not knowing” the maximum points beforehand.
- We can formalize this notion with 2 concepts:
  - **Instantaneous regret (r_t)**: the loss in reward at step t:

$$r_t = f(D_{max}) - f(D_t)$$

  - **Cumulative regret (R_T):** the total loss in reward after T steps:

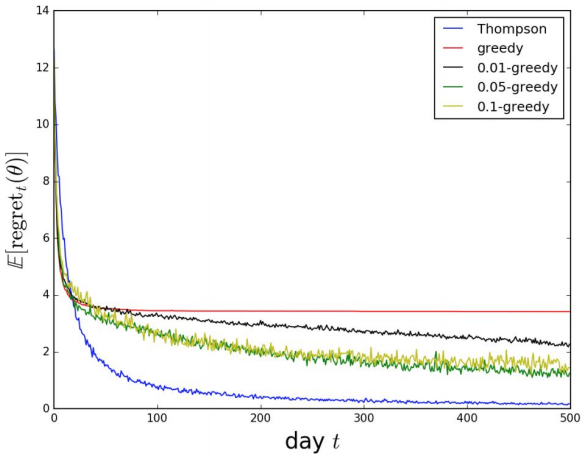
$$R_T = \sum r_t$$



Minimizing Regret: A Tradeoff

- As we have seen before, we can define policies that **balance exploration and exploitation**. Some of the policies we've looked at are:
  - Epsilon-greedy
  - Thompson sampling
  - Upper Confidence Bound (UCB)
- Some of these policies perform better than others in minimizing the **average regret** over time.

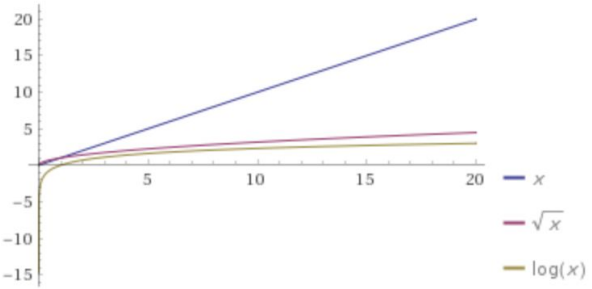
$$\text{Average Regret} = R_T / T$$



Credit: Russo et al., 2017

Asymptotic Regret

- We can also look at the cumulative or average regret measure as the number of iterations goes to infinity.
- An algorithm is said to be **no-regret** if its asymptotic cumulative regret rate is **sublinear** with respect to T (i.e. the number of iterations)

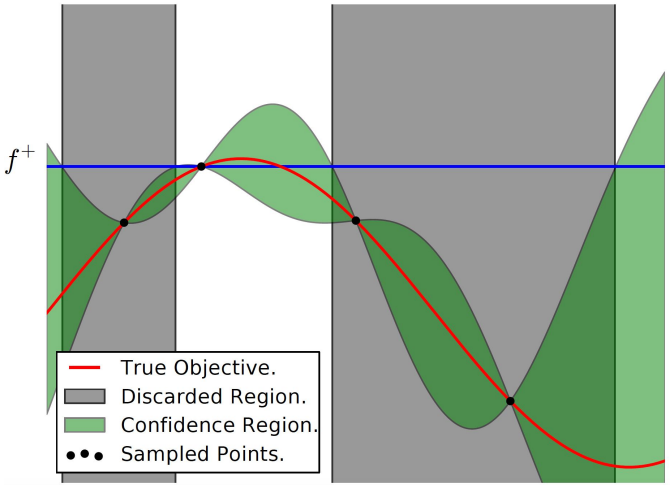


sqrt(T) and **log(T)** are examples of sublinear regret rates w.r.t. T.

$$\lim_{T \to \infty} R_T / T = 0$$

Why is Asymptotic Regret Important?

- In real world applications, we know neither instantaneous nor average regret. So, why are we concerned with characterizing their asymptotic behavior?
- Answer: Bounds on the average regret tell us about the **convergence rate** (i.e. how fast we approach the maximum point) of the optimization algorithm.



Credit: N. de Freitas et al., 2012

Regret Bounds in Discrete Decision Spaces

- In the previous lecture, we discussed asymptotic regret in **discrete decision spaces** where we have K slot machines or drug trials.
- We also looked at theorems by Auer et al. that derive an **upper bound** on the regret rate for the UCB algorithm in discrete settings.

"In the traditional K-arm bandit literature, the **regret is often characterized for a particular problem in terms of T, K, and problem dependent constants**. In the K-arm bandit results of Auer et al. [2002], this problem dependent constant is the 'gap' between the loss of the best arm and the second best arm."

Dani et al. 2008

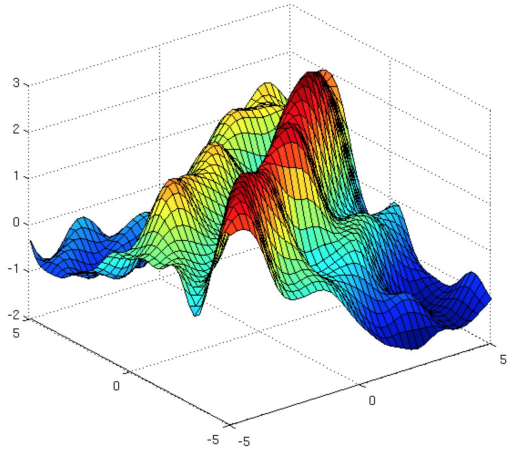
Regret Bounds in Continuous Decision Spaces

- Dani et al.** extended Auer et al.'s theoretical results to continuous decision spaces and proved **upper and lower regret bounds** for the UCB algorithm.
- However, their method places **restrictions on the types of reward functions considered**, primarily: The functions are defined over **finite-dimensional linear spaces**.

** Dani, V., Hayes, T. P., and Kakade, S. M. **Stochastic linear optimization under bandit feedback**. In COLT, 2008.

Infinite-dimensional Functions

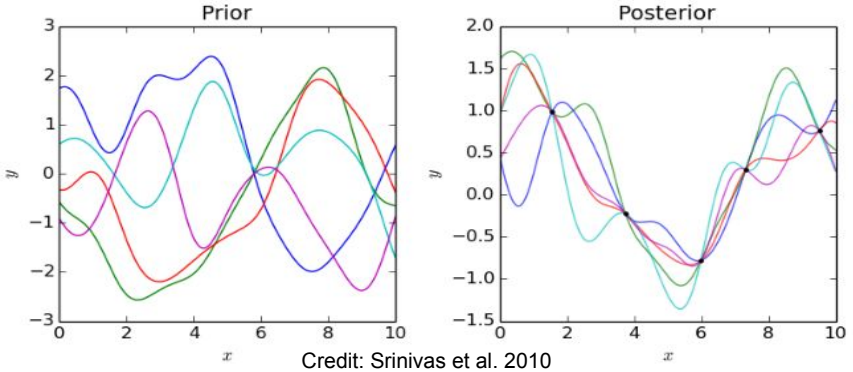
- Srinivas et al. propose to relax some of the restrictions of Dani et al.'s analysis and extend the results to random, infinite-dimensional functions.
- Earlier in the semester, we learned about a method for generating such classes of functions: **Gaussian Processes**.
- **Idea**: Assuming the target reward function is sampled from a Gaussian Process, try to optimize it using **GP-UCB**.
- How to derive **regret bounds** for those classes of functions?



Credit: Duvenaud, The Kernel Cookbook

Using Information Gain To Derive Regret Bounds

Information Gain



Credit: Srinivas et al. 2010

Recall Mackay (1992) paper: Information gain can be quantified as change in entropy

In this context:

Information gain = entropy in prior - entropy in posterior after y_A sampled

= $H(f) - H(f \mid y_A)$

= $I(f; y_A)$, mutual information between f and observed y_A

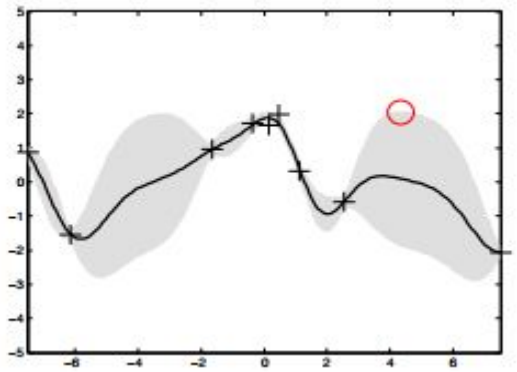
= $I(y_A; f) = H(y_A) - H(y_A|f) = (\log|\sigma^2\mathbf{I} + \mathbf{K_A}|)/2 - (\log|\sigma^2\mathbf{I}|)/2 = (\log|\mathbf{I} + \sigma^{-2}\mathbf{K_A}|)/2$

Note:
information gain depends on kernel of GP prior and input space

If our goal is just exploration ...

Greedy Experimental Design Algorithm:

Sequentially, find $x_t = argmax_{x \in D} \; I(y_{A \cup \{x\}}; f) = argmax_{x \in D} \; \sigma_{t-1}(x) n$ = point with highest variance



Credit: Srinivas et al. 2010

However the worse point we select, the more penalty we get

GP - UCB to the rescue

Algorithm 1 The GP-UCB algorithm.

Input: Input space D; GP Prior $\mu_0 = 0,\ \sigma_0,\ k$

for $t = 1, 2, \ldots$ **do**

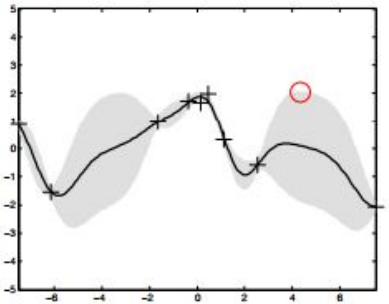
Choose $\boldsymbol{x}_t = \underset{\boldsymbol{x} \in D}{\operatorname{argmax}}\, \mu_{t-1}(\boldsymbol{x}) + \sqrt{\beta_t}\sigma_{t-1}(\boldsymbol{x})$

Sample $y_t = f(\boldsymbol{x}_t) + \epsilon_t$

Perform Bayesian update to obtain μ_t and σ_t

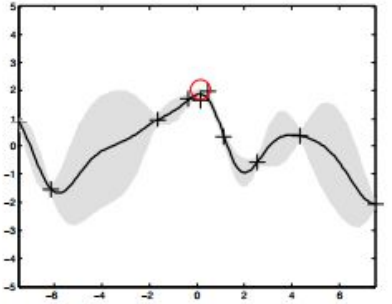
end for

Explore



(b) *Iteration t*

Exploit



(c) *Iteration t + 1*

Credit: Srinivas et al. 2010

Maximum information Gain

Definition:

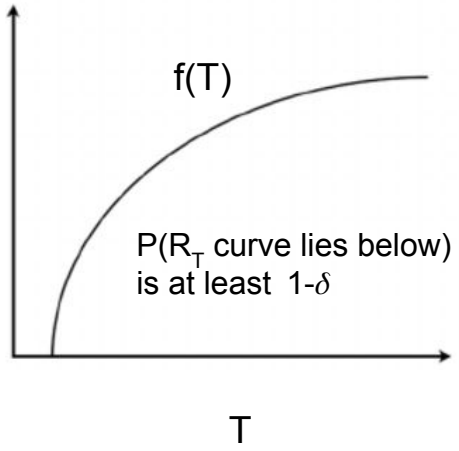
Maximum information gain after T data points sampled,

$$\gamma_T := \max_{A \subset D : |A| = T} \mathrm{I}(\boldsymbol{y}_A; \boldsymbol{f}_A)$$

This term will be used to quantify the regret bound for the algorithm

Regret Bounds - Finite Domain

Assuming some strictly sublinear γ_T... (we will verify later that this is achievable by choice of kernels), We can find some sublinear function f(T) bounding above $\sqrt{C_1 T \beta_T \gamma_T}$



Theorem 1:

Assumptions:

- Finite D
- f sample of a GP with mean 0,
- k(x, x') of GP s.t. k(x,x) (variance) not greater than 1

Then, by running GP-UCB for f with

$$\beta_t = 2\log(|D|t^2\pi^2/6\delta)$$
$$C_1 = 8/\log(1+\sigma^{-2})$$

We obtain:

$$\Pr\left\{R_T \leq \sqrt{C_1 T \beta_T \gamma_T} \quad \forall T \geq 1\right\} \geq 1-\delta$$

Regret Bounds II - General Compact+Convex Space

Theorem 2:

Assumptions:

- D compact and convex in $[0,r]^d$,
- f sample of a GP with mean 0,
- k(x, x') of GP s.t. k(x,x) (variance) not greater than 1
- k(x,x') s.t. f fulfills smoothness condition -- discussed next

Then, by running GP-UCB for f with $\beta_t = 2\log(t^2 2\pi^2/(3\delta)) + 2d\log\left(t^2 dbr\sqrt{\log(4da/\delta)}\right)$

$$C_1 = 8/\log(1+\sigma^{-2})$$

We obtain:

$$\Pr\left\{R_T \leq \sqrt{C_1 T \beta_T \gamma_T} + 2 \quad \forall T \geq 1\right\} \geq 1-\delta$$

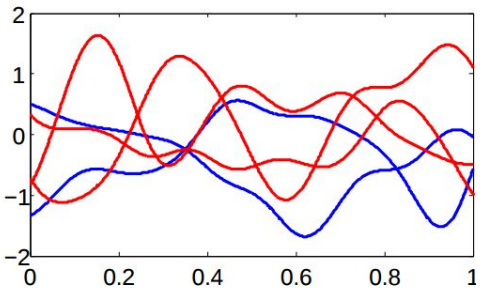
Regret Bounds II Continued

Theorem 2 requires f to fulfill:

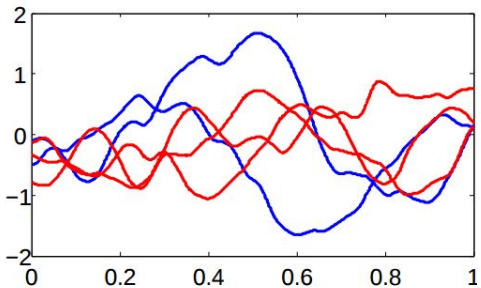
$$\Pr\{\sup_{\boldsymbol{x}\in D} |\partial f/\partial x_j| > L\} \leq a e^{-(L/b)^2}, \quad j = 1, \ldots, d.$$

This holds for stationary kernels k(x,x') = k(x-x') which are 4-times differentiable:

Squared Exponential Kernel



Matern Kernels with v>2



Credit: Srinivas et al. 2010

Bounding Information Gain

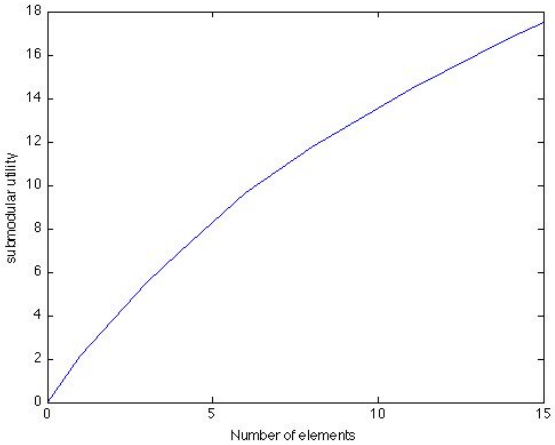
$F(A) = \mathrm{I}(\boldsymbol{y}_A; f)$ -- F is submodular function

$\Rightarrow \; F(A_T) \geq (1 - 1/e) \underbrace{\max_{|A| \leq T} F(A)}_{\gamma_T}$

This holds if A constructed by
Greedy Experiment Design Rule

$\Rightarrow \quad (1 - 1/e)^{-1} \mathrm{I}(\boldsymbol{y}_{A_T}; f) \geq \gamma_T$

Submodularity



Credit: Krause, https://las.inf.ethz.ch/sfo/

Bounding Information Gain Continued

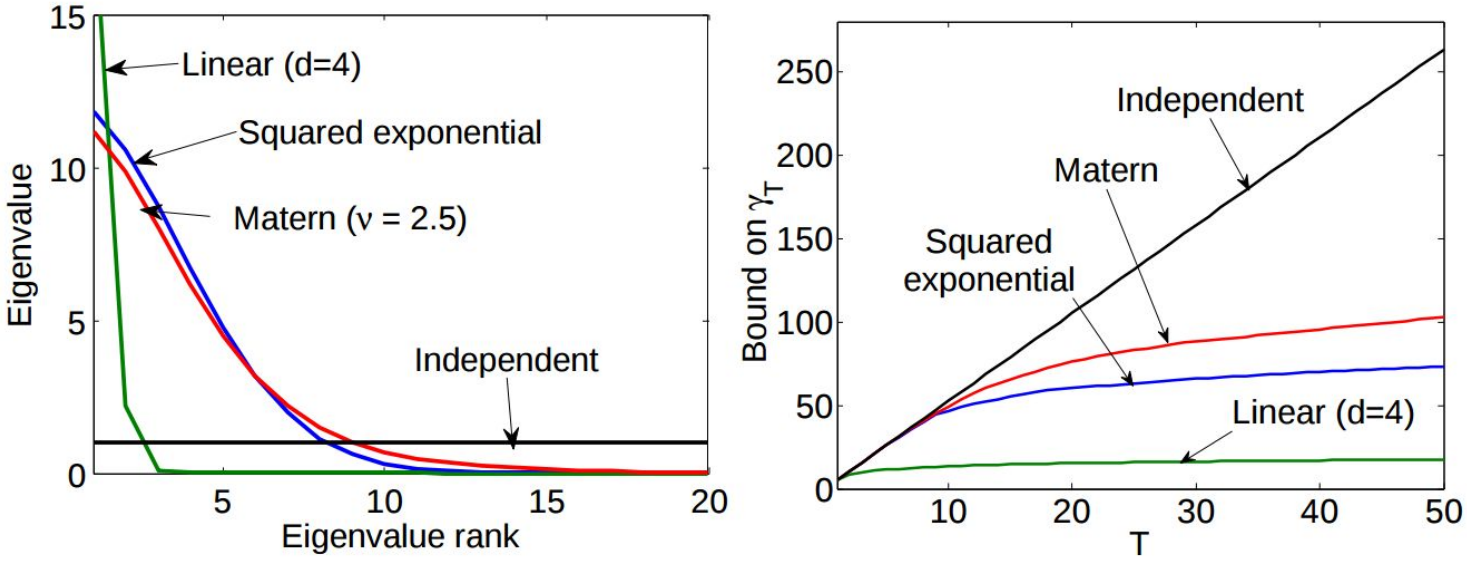
We can bound the term by considering the worst allocation of the T samples under some relaxed greedy procedure (see appendix section C).

In finite space D, this eventually gives us a bound in terms of the eigenvalues of the covariance matrix for all |D| points:

$$Spec\,(K_D) = \{\lambda_1 \geq \lambda_2 \geq \dots\}$$

The faster the spectrum decays, the slower the growth of the bound

Bounding Information Gain Continued



Credit: Srinivas et al. 2010

Bounding Information Gain Continued

Theorem 5: Assume general compact and convex set D in R^d, kernel $k(x,x') \leq 1$:

1. d- dimensional bayesian linear regression: $\gamma_T = \mathcal{O}(d \log T)$
2. Squared exponential kernel: $\gamma_T = \mathcal{O}((\log T)^{d+1})$
3. Matern kernel (v>1) : $\mathcal{O}(T^{d(d+1)/(2\nu+d(d+1))}(\log T))$

Now recall the bound obtained for GP-UCB in theorem 2: $\sqrt{C_1 T \beta_T \gamma_T}$ With $\beta_T = \mathcal{O}(d \log T) + \mathcal{O}(d \log d)$

Combining the two theorems we obtain the following **(1-δ) upper confidence bound** for the total regret, R_T (up to polylog factors):

Kernel	Linear	RBF	Matérn
Regret R_T	$d\sqrt{T}$	$\sqrt{T(\log T)^{d+1}}$	$T^{\frac{\nu+d(d+1)}{2\nu+d(d+1)}}$

Credit: Srinivas et al. 2010

Results and Discussion

Experimental Setup

- Synthetic and real sensor network data (traffic and temperature) used to illustrate the differences
- Gaussian Processes - Upper Confidence Bound (GP-UCB) is compared with various heuristics:
  - Expected Improvement (EI)
  - Most Probable Improvement (MPI)
  - Naive Methods (only mean or only variance)

Experimental Setup

- Synthetic Data Breakdown:
  - Functions were sampled from a GP with a squared exponential kernel
  - Sample parameters: $\sigma^2 = 0.025$, T = 1000 iterations, $\delta = 0.1$

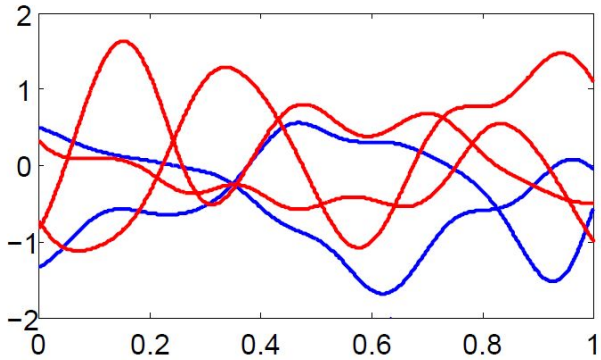


Figure: Functions drawn from a GP with squared exponential kernel (lengthscale=0.2) Credit: Srinivas et al. 2010

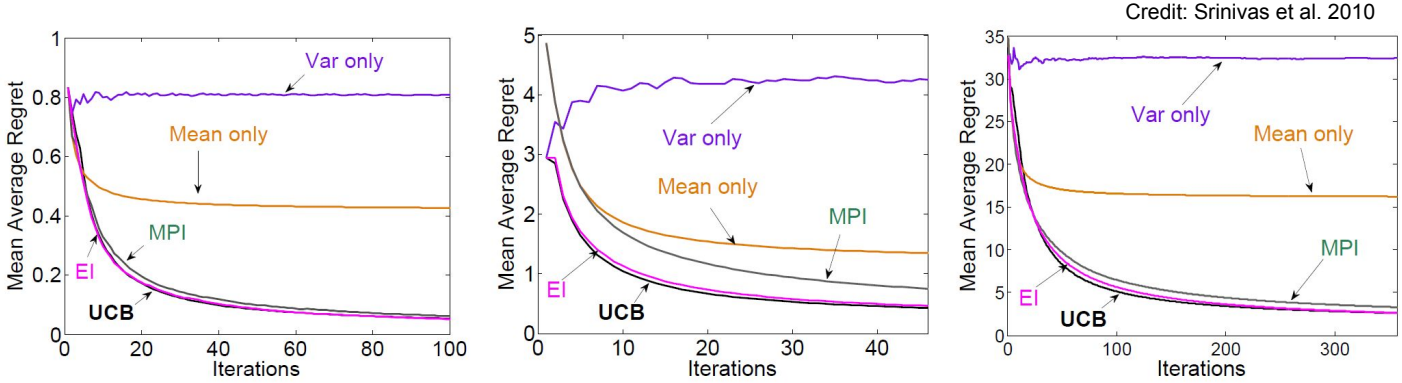
Experimental Setup

- Real Sensor Temperature Data Breakdown:
  - 46 sensors deployed at Intel Research Center (Berkeley) that acquired data over 5 days at 1 minute intervals
  - ⅔ of the data was used to create the kernel matrix
  - Sample parameters: $\sigma^2 = 0.5$, T = 46 iterations, $\delta = 0.1$
  - Remaining ⅓ of data was used for testing
  - Results averaged over 2000 runs

Experimental Setup

- Real Sensor Traffic Data Breakdown:
  - Data from 357 traffic sensors along highway I-880 in South California captured data for one month from 6am - 11am
  - Purpose was to find the minimum speed to identify the most congested part of the highway during rush-hour
  - ⅔ of the data was used to create the kernel matrix
  - Sample parameters: $\sigma^2 = 4.78$, T = 357 iterations, $\delta = 0.1$
  - Remaining ⅓ of data was used for testing
  - Results averaged over 900 runs

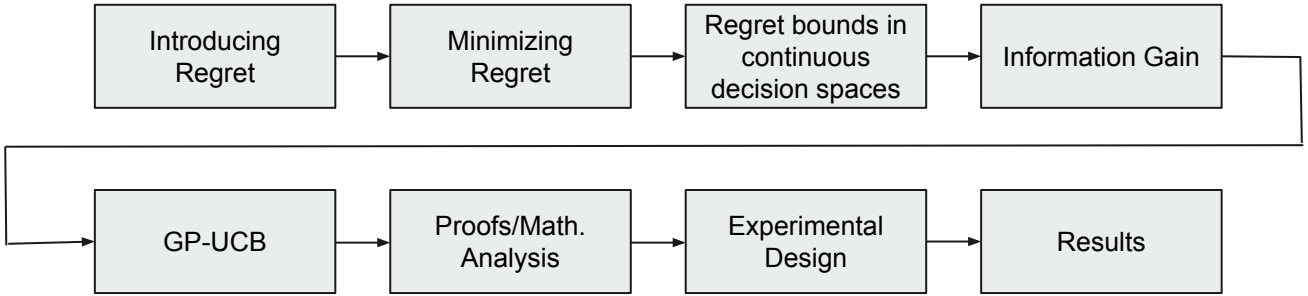
Results



(a) *Squared exponential* (b) *Temperature data* (c) *Traffic data*

- For temperature data (b), GP-UCB and EI perform best
- For synthetic data, GP-UCB and EI perform best and MPI is a close comparable
- **GP-UCB performs at least on par with existing approaches which do not include regret bounds**

Conclusion



- GP-UCB sample rule is discussed in detail and an algorithm is presented

Algorithm 1 The GP-UCB algorithm.

Input: Input space D; GP Prior $\mu_0 = 0,\ \sigma_0,\ k$

for $t = 1, 2, \ldots$ **do**

Choose $\boldsymbol{x}_t = \underset{\boldsymbol{x} \in D}{\operatorname{argmax}}\, \mu_{t-1}(\boldsymbol{x}) + \sqrt{\beta_t}\sigma_{t-1}(\boldsymbol{x})$

Sample $y_t = f(\boldsymbol{x}_t) + \epsilon_t$

Perform Bayesian update to obtain μ_t and σ_t

end for

Credit: Srinivas et al. 2010

Conclusion

- The concepts of Information Gain and Regret Bounds are analyzed and their relations represented in the following theorems:
  - Regret Bounds for Finite Domain
  - Regret Bounds for General Compact + Convex Space
  - Bounding Information Gain
- Synthetic and Real experimental data used to test the algorithm
- GP-UCB is found to perform at least on par with existing approaches which do not include regret bounds
- Their results are encouraging as they illustrate exploration and exploitation trade-offs for complex functions
- The paper uses tools (concept of regret and information gain) to come up with a convergence rate for the GP-UCB algorithm

Exploiting Structure for Bayesian Optimization

K. Swersky, J. Snoek, R.P. Adams (2014)
Freeze-Thaw Bayesian Optimization

K. Swersky, J. Snoek, R.P. Adams (2013)
Multi-Task Bayesian Optimization

Presentation by: Shu Jian (Eddie) Du, Romina Abachi, William Saunders

Freeze-Thaw Bayesian Optimization

K. Swersky, J. Snoek, R.P. Adams (2014)

Presentation by: Shu Jian (Eddie) Du, Romina Abachi

Intuition

- Human experts tend to stop model training halfway if the loss curve looks bad.
- Like Snoek 2012 alluded to, we'd like to leverage partial information (before a model finishes training) to determine what points to evaluate next.

Big Idea

- To use partial information, we model training loss curves with a GP.
- Let's assume loss curves look Exponential; roughly $e^{-\lambda t}$.
- Derive a GP kernel $k(t, t')$ to model this between 2 time steps.
- Input x: A set of hyperparameters
- Output y: The model's loss at a particular time

Exponential Decay Kernel

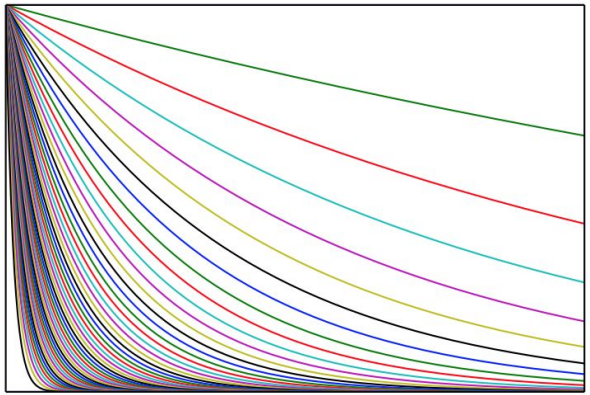
$$k(t,t') = \int_0^\infty e^{-\lambda t} e^{-\lambda t'} \psi(d\lambda)$$

$$\psi(\lambda) = \frac{\beta^\alpha}{\Gamma(\alpha)} \lambda^{\alpha-1} e^{-\lambda\beta}$$

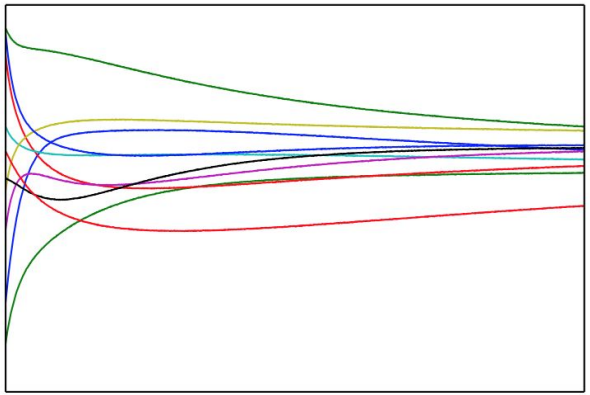
$$k(t,t') = \int_0^\infty e^{-\lambda(t+t')} \frac{\beta^\alpha}{\Gamma(\alpha)} \lambda^{\alpha-1} e^{-\lambda\beta}\, d\lambda$$

$$= \frac{\beta^\alpha}{\Gamma(\alpha)} \int_0^\infty e^{-\lambda(t+t'+\beta)} \lambda^{\alpha-1}\, d\lambda = \boxed{\frac{\beta^\alpha}{(t+t'+\beta)^\alpha}}$$

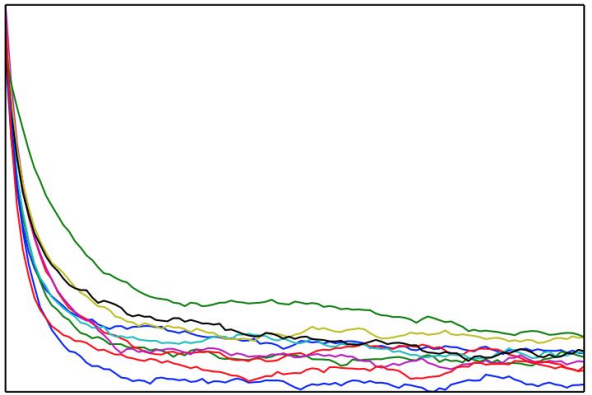
Exponential Decay Kernel



(a) Exponential Decay Basis



(b) Samples



(c) Training Curve Samples

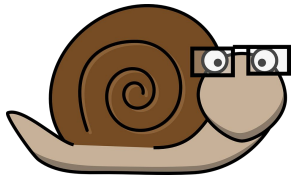
Demo

Demo:
https://github.com/esdu/misc/raw/master/csc2541/demo1.pdf

Code:
https://github.com/esdu/misc/blob/master/csc2541/csc2541_ftbo_pres_demo.ipynb

Are we done?

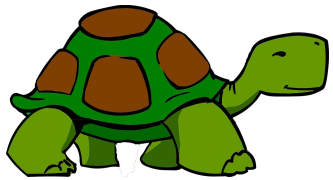
- We could model all **N** training curves over all **T** timesteps jointly using a single GP using the **Exp Decay Kernel**.
- However, since GP takes cubic time to fit, it would run in $O(N^3T^3)$ time. (We have N*T data points) this is way too slow!
- Paper proposes a generative model to speed this up.

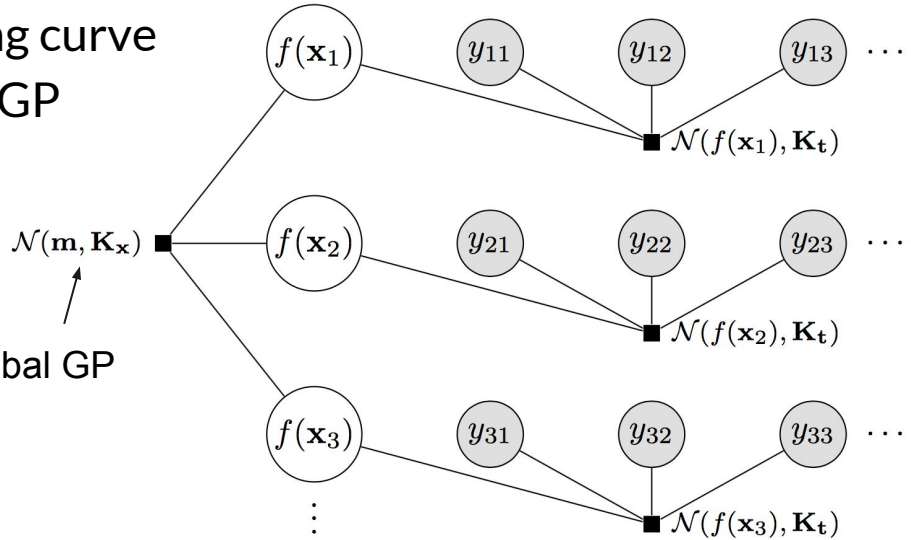


A more efficient way

- Use a global GP to model the asymptotes of each training curve.
- Each training curve's GP sample their prior mean function (the best-guess asymptote) from the global GP
- This assumes each training curve is drawn from a separate GP

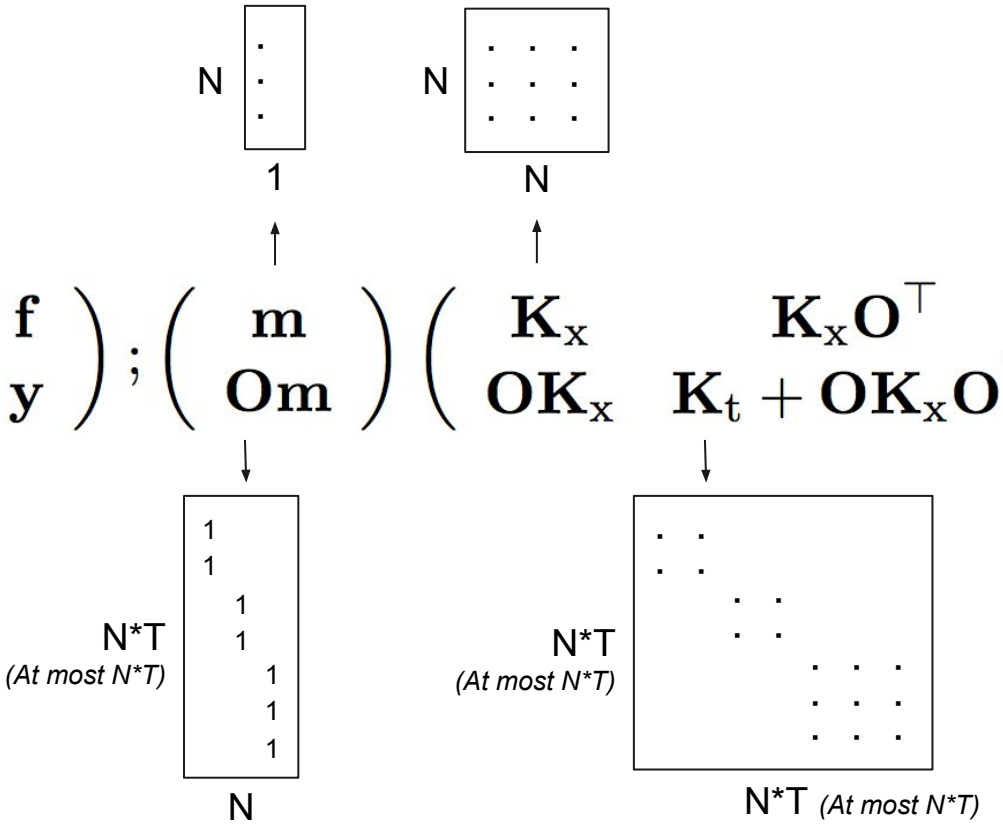
$$O(N^3 + T^3 + NT^2)$$





Joint distribution

$$P(\mathbf{y}, \mathbf{f} \mid \{\mathbf{x}_n\}_{n=1}^{N}) = \mathcal{N}\left(\begin{pmatrix} \mathbf{f} \\ \mathbf{y} \end{pmatrix}; \begin{pmatrix} \mathbf{m} \\ \mathbf{O}\mathbf{m} \end{pmatrix} \begin{pmatrix} \mathbf{K}_x & \mathbf{K}_x\mathbf{O}^\top \\ \mathbf{O}\mathbf{K}_x & \mathbf{K}_t + \mathbf{O}\mathbf{K}_x\mathbf{O}^\top \end{pmatrix}\right)$$



Marginal likelihood

$$P(\mathbf{y}, \mathbf{f} \mid \{\mathbf{x}_n\}_{n=1}^N) = \mathcal{N}\left(\begin{pmatrix} \mathbf{f} \\ \mathbf{y} \end{pmatrix}; \begin{pmatrix} \mathbf{m} \\ \mathbf{O}\mathbf{m} \end{pmatrix} \begin{pmatrix} \mathbf{K}_\mathrm{x} & \mathbf{K}_\mathrm{x}\mathbf{O}^\top \\ \mathbf{O}\mathbf{K}_\mathrm{x} & \mathbf{K}_\mathrm{t} + \mathbf{O}\mathbf{K}_\mathrm{x}\mathbf{O}^\top \end{pmatrix}\right)$$

$$P(\mathbf{y} \mid \{\mathbf{x}_n\}_{n=1}^N) = \mathcal{N}\left(\mathbf{y}; \mathbf{O}\mathbf{m}, \mathbf{K}_\mathrm{t} + \mathbf{O}\mathbf{K}_\mathrm{x}\mathbf{O}^\top\right)$$

Posterior distribution

$$P(\mathbf{y}, \mathbf{f} \mid \{\mathbf{x}_n\}_{n=1}^N) = \mathcal{N}\left(\begin{pmatrix} \mathbf{f} \\ \mathbf{y} \end{pmatrix}; \begin{pmatrix} \mathbf{m} \\ \mathbf{O}\mathbf{m} \end{pmatrix} \begin{pmatrix} \mathbf{K}_\mathrm{x} & \mathbf{K}_\mathrm{x}\mathbf{O}^\top \\ \mathbf{O}\mathbf{K}_\mathrm{x} & \mathbf{K}_\mathrm{t} + \mathbf{O}\mathbf{K}_\mathrm{x}\mathbf{O}^\top \end{pmatrix}\right)$$

$$P(\mathbf{f} \mid \mathbf{y}, \{\mathbf{x}_n\}_{n=1}^N) = \mathcal{N}(\mathbf{f}; \boldsymbol{\mu}, \mathbf{C}),$$

$$\boldsymbol{\mu} = \mathbf{m} + \mathbf{C}\boldsymbol{\gamma},$$

$$\mathbf{C} = \mathbf{K}_\mathrm{x} - \mathbf{K}_\mathrm{x}(\mathbf{K}_\mathrm{x} + \boldsymbol{\Lambda}^{-1})^{-1}\mathbf{K}_\mathrm{x}.$$

$$\boldsymbol{\gamma} = \mathbf{O}^\top \mathbf{K}_\mathrm{t}^{-1}(\mathbf{y} - \mathbf{O}\mathbf{m})$$

$$\boldsymbol{\Lambda} = \mathbf{O}^\top \mathbf{K}_\mathrm{t}^{-1}\mathbf{O}$$

Posterior predictive distribution

$$P(\mathbf{y}, \mathbf{f} \mid \{\mathbf{x}_n\}_{n=1}^N) = \mathcal{N}\left(\begin{pmatrix} \mathbf{f} \\ \mathbf{y} \end{pmatrix}; \begin{pmatrix} \mathbf{m} \\ \mathbf{O}\mathbf{m} \end{pmatrix} \begin{pmatrix} \mathbf{K}_\mathrm{x} & \mathbf{K}_\mathrm{x}\mathbf{O}^\top \\ \mathbf{O}\mathbf{K}_\mathrm{x} & \mathbf{K}_\mathrm{t} + \mathbf{O}\mathbf{K}_\mathrm{x}\mathbf{O}^\top \end{pmatrix}\right)$$

$$P(f_* \mid \mathbf{y}, \{\mathbf{x}_n\}_{n=1}^N, \mathbf{x}_*) = \mathcal{N}(f_*; m + \mathbf{K}_{\mathrm{x}*}^\top \mathbf{K}_\mathrm{x}^{-1}(\boldsymbol{\mu} - \mathbf{m}), \mathbf{K}_{\mathrm{x}**} - \mathbf{K}_{\mathrm{x}*}^\top (\mathbf{K}_\mathrm{x} + \boldsymbol{\Lambda}^{-1})^{-1} \mathbf{K}_{\mathrm{x}*}).$$

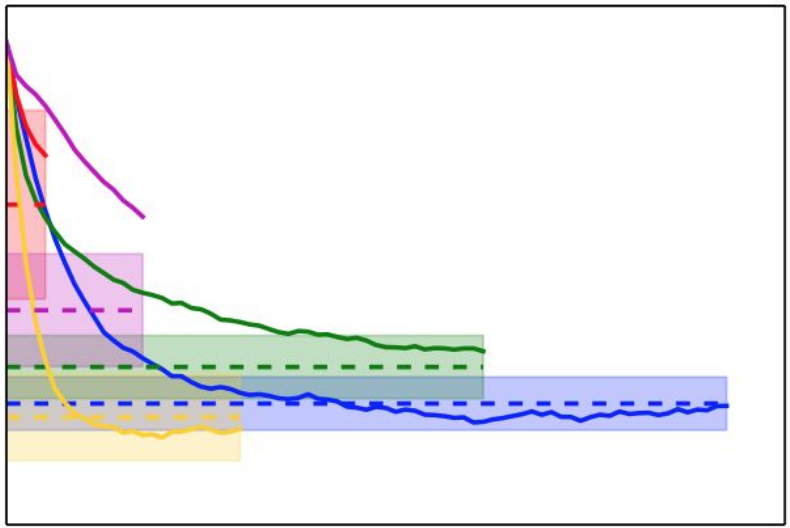
$$P(y_{n*} \mid \{\mathbf{x}_n\}_{n=1}^N, \mathbf{y}) = \mathcal{N}(y_{n*}; \mathbf{K}_{\mathrm{t}n*}^\top \mathbf{K}_{\mathrm{t}n}^{-1} \mathbf{y}_n + \Omega \boldsymbol{\mu}_n, \mathbf{K}_{\mathrm{t}n**} - \mathbf{K}_{\mathrm{t}n*}^\top \mathbf{K}_{\mathrm{t}n}^{-1} \mathbf{K}_{\mathrm{t}n*} + \Omega \mathbf{C}_{nn} \Omega^\top),$$
$$\Omega = \mathbf{1}_* - \mathbf{K}_{\mathrm{t}n*}^\top \mathbf{K}_{\mathrm{t}n}^{-1} \mathbf{1}_n.$$

Aside: To Derive These...

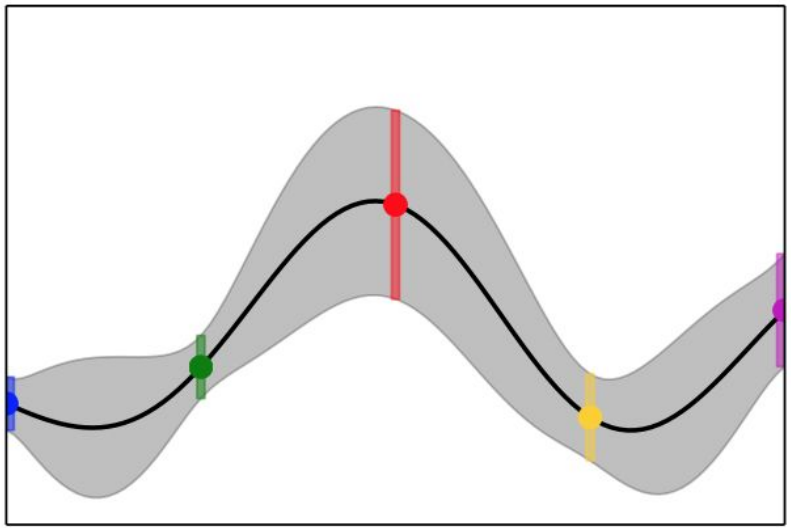
- Authors repeated used:
  - Basic Multivariate Gaussian identities
  - Woodbury Matrix Identity (See Wikipedia)

$$(A+UCV)^{-1} = A^{-1} - A^{-1}U\left(C^{-1} + VA^{-1}U\right)^{-1}VA^{-1}$$

What does it look like?



(b) Training curve predictions



(c) Asymptotic GP

Demo

Demo:
https://github.com/esdu/misc/raw/master/csc2541/demo2.pdf

Code:
https://github.com/esdu/misc/blob/master/csc2541/csc2541_ftbo_pres_demo.ipynb

Which acquisition function to use?

Expected Improvement:

$$a_{\mathrm{EI}}(\mathbf{x}) = \sqrt{v(\mathbf{x})}(\gamma(\mathbf{x})\Phi(\gamma(\mathbf{x})) + \phi(\gamma(\mathbf{x})), \quad \gamma(\mathbf{x}) = \frac{f(\mathbf{x}_{\mathrm{best}} - \mu(\mathbf{x}))}{\sqrt{v(\mathbf{x})}},$$

$\mathbf{x}_{\mathrm{best}}$ -- input corresponding to minimum output observed so far

μ(x) and v(x) -- posterior mean and variance of the probabilistic model evaluated at x

EI used to determine which hyperparameters to try next (baskets)

Acquisition Function: Entropy Search

Idea: How much information does evaluating a new point give us about the location of the minimum?

- While EI focuses on finding the minimum value of the function, ES tries to reduce the uncertainty over the location of the minimum.
- Unlike Expected Improvement, takes into account the possibility that some point other than the best known will be the best.

Acquisition Function: Entropy Search

Given C points $\tilde{\mathbf{X}} \subset X$, probability of x $\subset \tilde{\mathbf{X}}$ having the minimum value is:

$$\Pr(\text{min at } \mathbf{x} \mid \theta, \tilde{\mathbf{X}}, \{\mathbf{x}_n, y_n\}_{n=1}^N) = \int_{\mathbb{R}^C} p(\mathbf{f} \mid \mathbf{x}, \theta, \{\mathbf{x}_n, y_n\}_{n=1}^N) \prod_{\tilde{\mathbf{x}} \in \tilde{\mathbf{X}} \setminus \mathbf{x}} h\left(f(\tilde{\mathbf{x}}) - f(\mathbf{x})\right) \mathrm{d}\mathbf{f}, \quad (6)$$

where $\mathbf{f}$ is the vector of function values at the points $\tilde{\mathbf{X}}$ and h is the Heaviside step function.

Probability of function values at all candidate points

1 if x is minimum, 0 otherwise

Goal: reduce uncertainty over this if we observe y at x.

Acquisition Function: Entropy Search

$$a_{\mathrm{KL}}(\mathbf{x}) = \int \int \left[H(\mathrm{P}_{\mathrm{min}}) - H(\mathrm{P}^{y}_{\mathrm{min}})\right] p(y \mid \mathrm{f})\, p(\mathrm{f} \mid \mathbf{x})\, \mathrm{d}y\, \mathrm{df},$$

$$a_{\mathrm{ES}}(\mathbf{x}) = \int (H(P^{y}_{\mathrm{min}}) - H(P_{\mathrm{min}}))P(y \mid \{(\mathbf{x}_n, y_n)\}_{n=1}^{N})\, dy.$$

P_{min} -- current estimated distribution over the minimum

P^{y}_{min} is the updated distribution over the location of the minimum with the added observation y.

In practice, no simple form, so we use Monte Carlo sampling to estimate P_{min}

Which acquisition function to use?

Why not choose the model to run based on EI?

-EI looks at value of function
→ would need more trials to find minimum
-ES maximizes information gain from each trial
→ can make better decisions with fewer trials

Algorithm

Algorithm 1 Entropy Search Freeze-Thaw Bayesian Optimization

1: Given a basket $\{(\mathbf{x}, \mathbf{y})\}_{B_{\text{old}}} \cup \{(\mathbf{x})\}_{B_{\text{new}}}$
2: $a = (0, 0, \ldots, 0)$
3: Compute $P_{\min}$ over the basket using Monte Carlo simulation and Equation 19. → $P(f_* \mid \mathbf{y}, \{\mathbf{x}_n\}_{n=1}^N, \mathbf{x}_*)$
4: **for** each point $\mathbf{x}_k$ in the basket **do**
5: // n_{fant} is some specified number, e.g., 5.
6: **for** $i = 1 \ldots n_{\text{fant}}$ **do**
7: **if** the point is old **then**
8: Fantasize an observation y_{t+1} using Equation 20. $P(y_{n*} \mid \{\mathbf{x}_n\}_{n=1}^N, \mathbf{y})$
9: **end if**
10: **if** the point is new **then**
11: Fantasize an observation y_1 using Equation 21. $P(y_* \mid \{\mathbf{x}_n\}_{n=1}^N, \mathbf{y}, \mathbf{x}_*)$
12: **end if**
13: Conditioned on this observation, compute $P_{\min}^y$ over the basket using Monte Carlo simulation and Equation 19.
14: $a(k) \leftarrow a(k) + \frac{H(P_{\min}^y) - H(P_{\min})}{n_{\text{fant}}}$ // information gain.
15: **end for**
16: **end for**
17: Select $\mathbf{x}_k$, where $k = \text{argmax}_k\, a(k)$ as the next model to run.

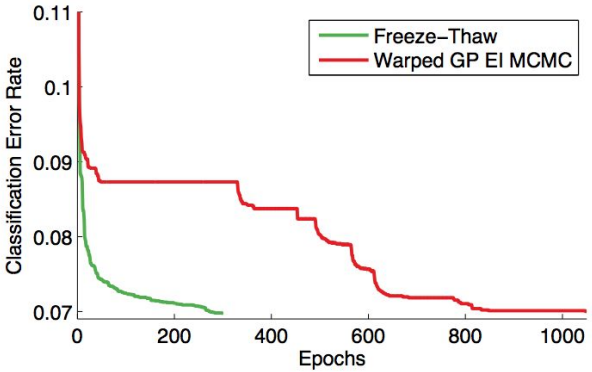
Experiments

-Logistic Regression: trained using SGD on MNIST. Hyperparameters: norm constraint on weights, l_2 regularization penalty, minibatch size, dropout regularization, learning rate

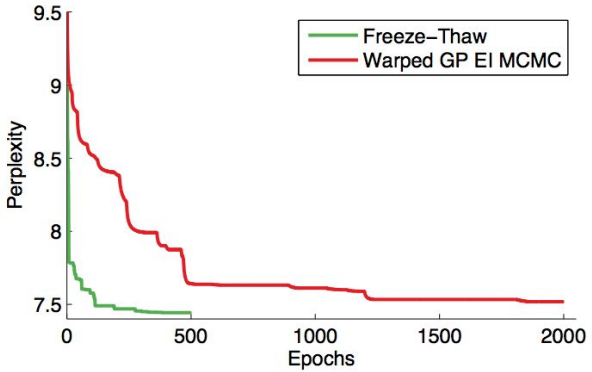
-Online Latent Dirichlet Allocation (LDA): Trained on 250,000 Wikipedia docs. Hyperparams: number of topics, 2x Dirichlet distribution prior base measures, learning rate, decay.

-Probabilistic Matrix Factorization (PMF): Trained on 100,000 MovieLens ratings. Hyperparameters: rank, learning rate, l_2 regularization penalty

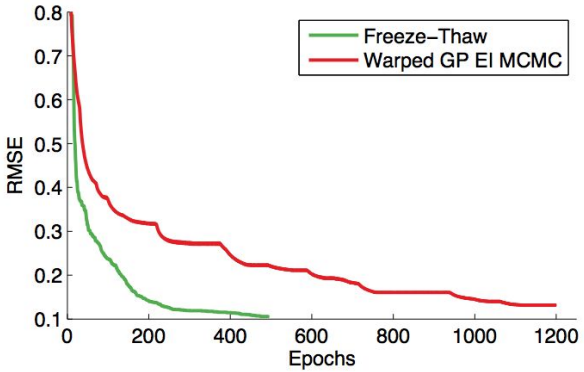
Results



(a) Logistic Regression

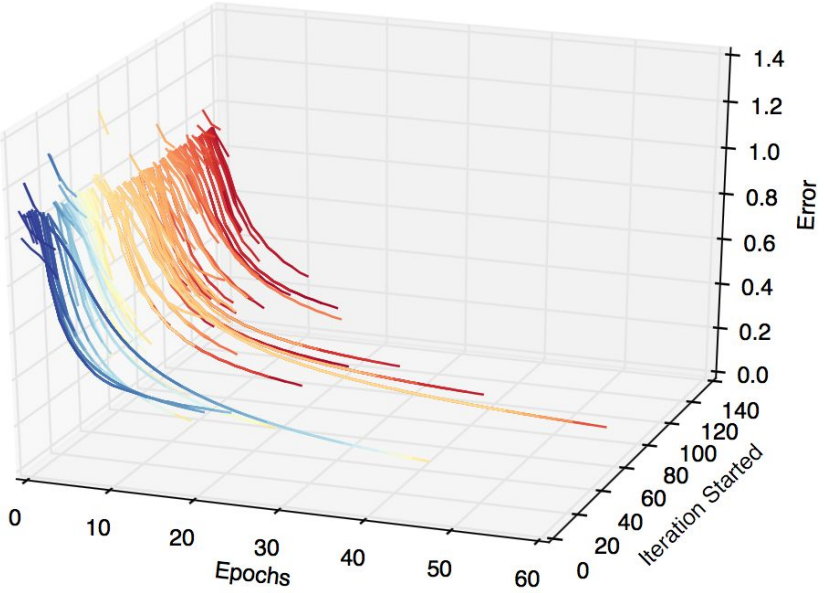


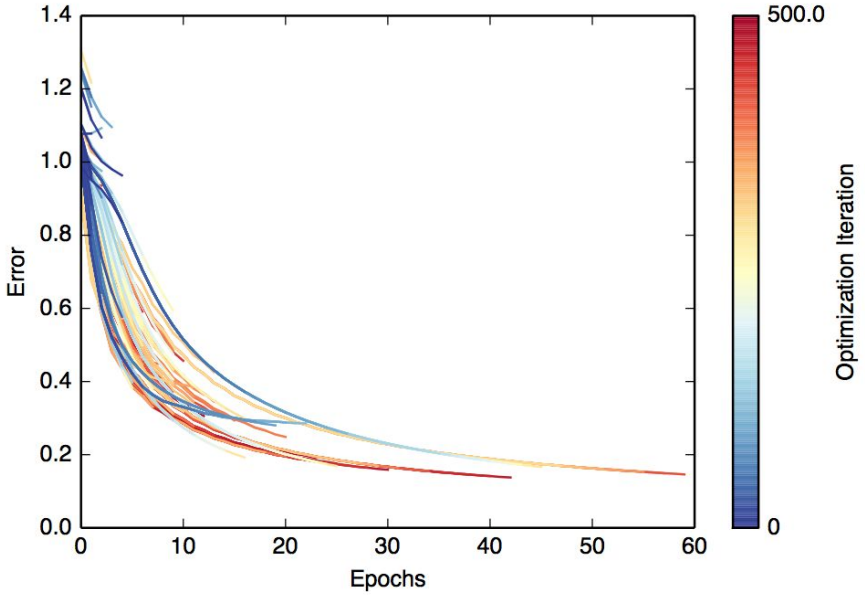
(b) Online LDA



(c) PMF

Results


Error
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Epochs
0
10
20
30
40
50
60
Iteration Started
0
20
40
60
80
100
120
140


Error
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Epochs
0
10
20
30
40
50
60
Optimization Iteration
500.0
0

Conclusion & Future Work

- Exploit partial information as training is happening.
  - Stop, resume, start new runs dynamically
  - Can be extended to other problems where partial observations reduce uncertainty.
- Relies on the key assumption that training curves follow exponential decay.
  - It would be interesting to use more flexible priors for other problems

Multi-Task Bayesian Optimization

K. Swersky, J. Snoek, R.P. Adams (2013)

Presentation by: William Saunders

Goal

In Bayesian Optimization, it would be useful to be able to re-use information from related tasks to reduce sample complexity

- Have data from running bayesian optimization on other similar problems
- Use a computationally cheaper task to find information about a more expensive task (ie. small subset of the training data)
- Optimize average performance of a set of related tasks

Multi-Task Gaussian Process Kernel

$$K_{\text{multi}}((\mathbf{x}, t), (\mathbf{x}', t')) = K_{\text{t}}(t, t') \otimes K_{\text{x}}(\mathbf{x}, \mathbf{x}')$$

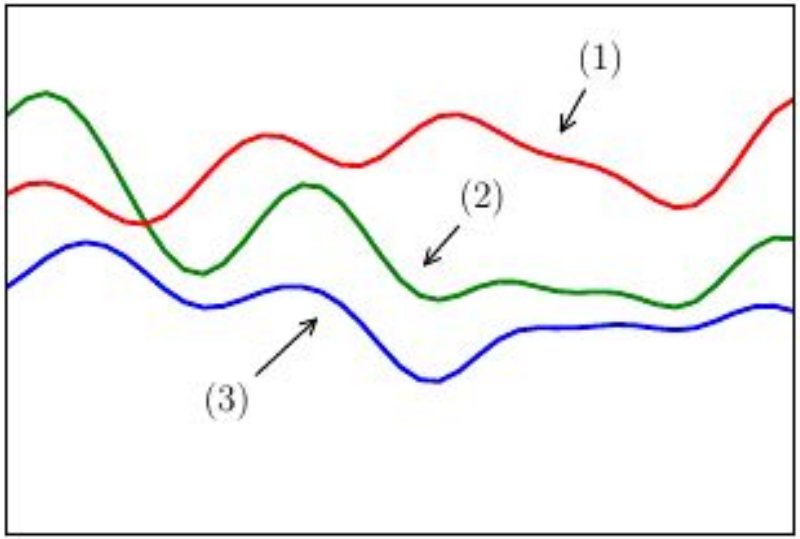
K_x is a kernel indicating the covariance between inputs

K_t is a matrix indicating the covariance between tasks
K_t is marginalized over using a Monte-Carlo sampling method (slice sampling), as are other kernels parameter (ie. length scale)
K_t is parameterized by its cholesky decomposition R^T*R, where R is upper diagonal with positive diagonal elements

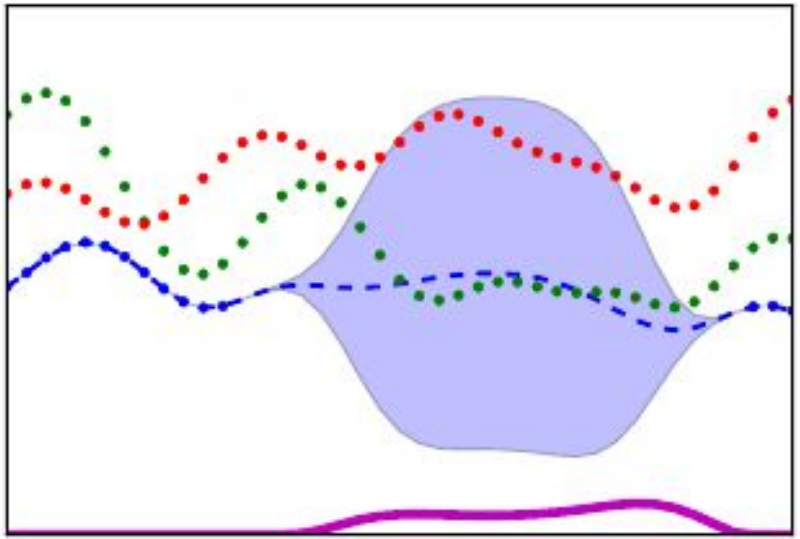
$\otimes$ is the Kronecker Product

Multi-Task Gaussian Process

Blue = target task, Red and Green are related tasks



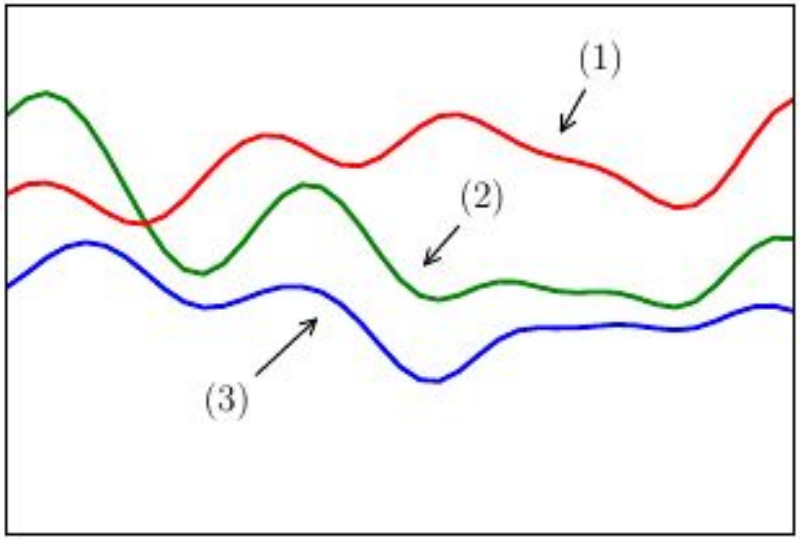
(a) Multi-task GP sample functions



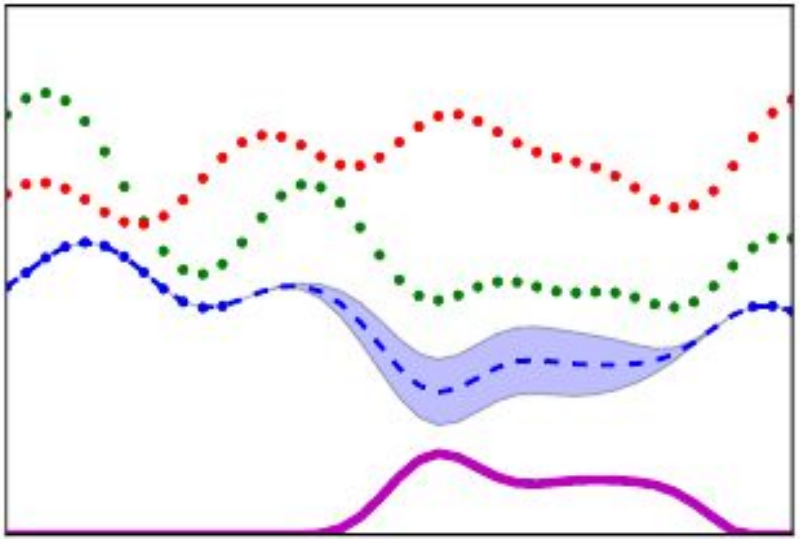
(b) Independent GP predictions

Multi-Task Gaussian Process

Blue = target task, Red and Green are related tasks



(a) Multi-task GP sample functions



(c) Multi-task GP predictions

Acquisition Function: Expected Improvement

Choose the point that, in expectation, will have the greatest improvement over the best known point

Assumes that after querying, either the best known point or the queried point will be the maximum

$$a_{\mathrm{EI}}(\mathbf{x}\,;\,\{\mathbf{x}_n, y_n\}, \theta) = \sqrt{\Sigma(\mathbf{x}, \mathbf{x}\,;\,\{\mathbf{x}_n, y_n\}, \theta)}\,(\gamma(\mathbf{x})\,\Phi(\gamma(\mathbf{x})) + \mathcal{N}(\gamma(\mathbf{x})\,;\,0, 1))\,,$$

$$\gamma(\mathbf{x}) = \frac{y_{\mathrm{best}} - \mu(\mathbf{x}\,;\,\{\mathbf{x}_n, y_n\}, \theta)}{\sqrt{\Sigma(\mathbf{x}, \mathbf{x}\,;\,\{\mathbf{x}_n, y_n\}, \theta)}}.$$

Acquisition Function: Entropy Search

$$a_{\mathrm{KL}}(\mathbf{x}) = \int \int \left[H(\mathrm{P}_{\mathrm{min}}) - H(\mathrm{P}^{y}_{\mathrm{min}}) \right] \, p(y \mid \mathrm{f}) \, p(\mathrm{f} \mid \mathbf{x}) \, \mathrm{d}y \, \mathrm{df},$$

- Select set of candidate points based on Expected Improvment
- f is assignment of values to all candidate points
- Evaluate using monte-carlo sampling
  - P_{min} = the current estimated distribution over the minimum
  - P^{y}_{min} = the new distribution over the minimum, given an observation
  - Both these distributions can be approximated by repeatedly sampling f and determining the minimum of the sample
  - p(y|f), p(f|x) calculated from gaussian process

Acquisition Function: Information Gain/Cost

Observing a point on a related task can never reveal more information than sampling the same point on the target task

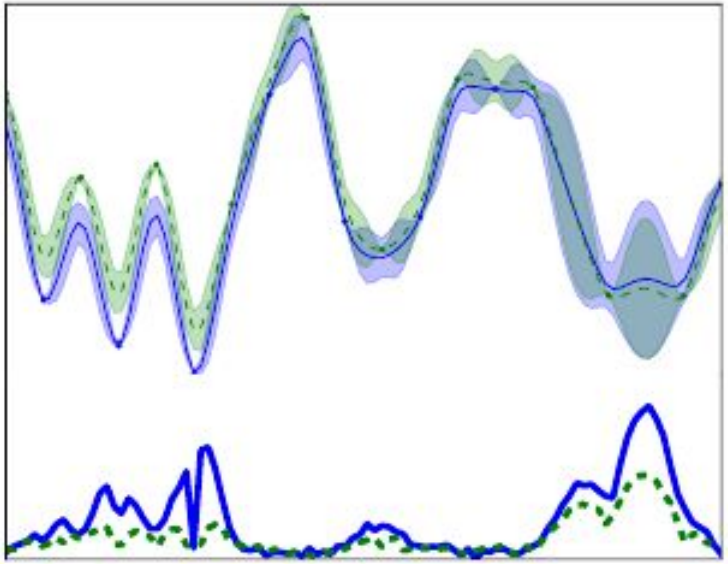
But, it can be better when information per unit cost is taken into account

$$a_{\mathsf{IG}}(\mathbf{x}^t) = \int\int \left(\frac{H[\mathrm{P}_{\mathsf{min}}] - H[\mathrm{P}^y_{\mathsf{min}}]}{c_t(\mathbf{x})} \right) p(y \mid \mathrm{f})\, p(\mathrm{f} \mid \mathbf{x}^t)\, \mathrm{d}y\, \mathrm{df},$$

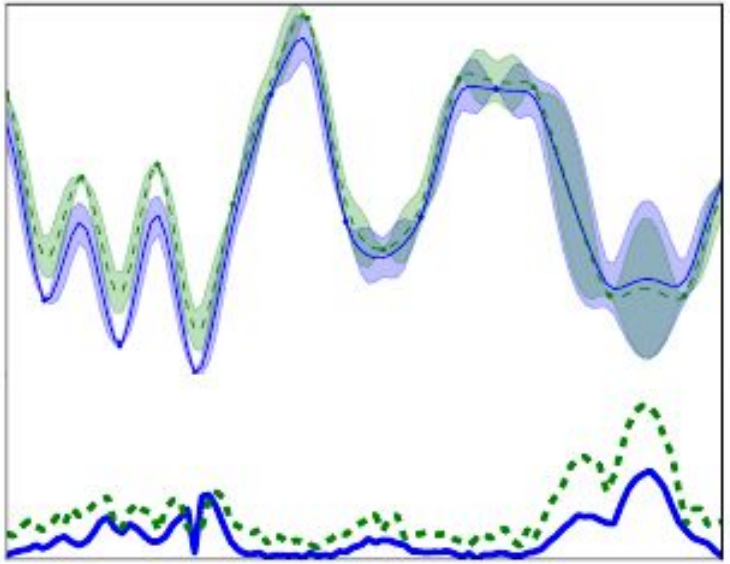
where $c_t(\mathbf{x})$, $c_t : \mathcal{X} \to \mathbb{R}^+$, is the real valued cost of evaluating task t at $\mathbf{x}$.

Acquisition Function: Taking Cost Into Account

Blue = target task, expensive; Green = related task, cheap

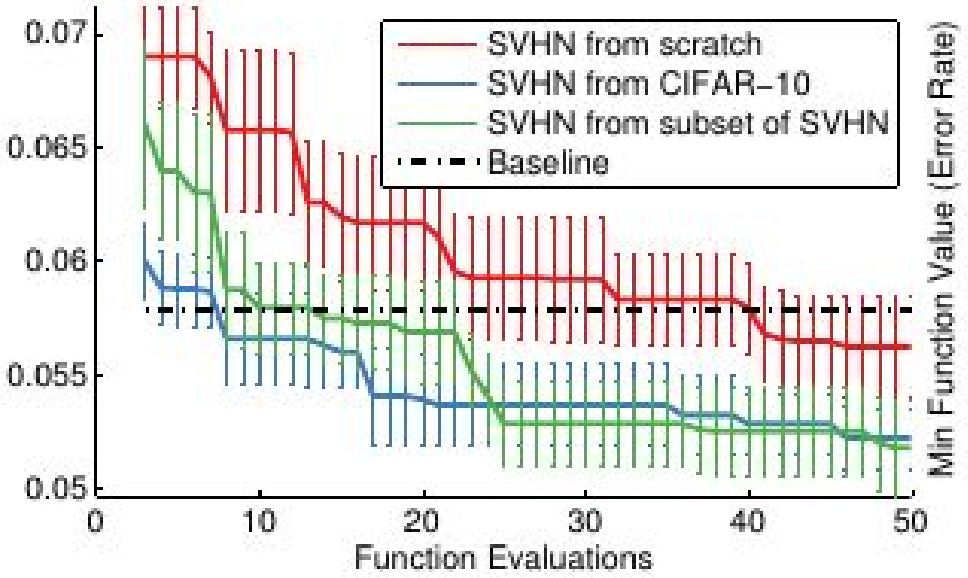


(b) Correlated functions



(c) Correlated functions scaled by cost

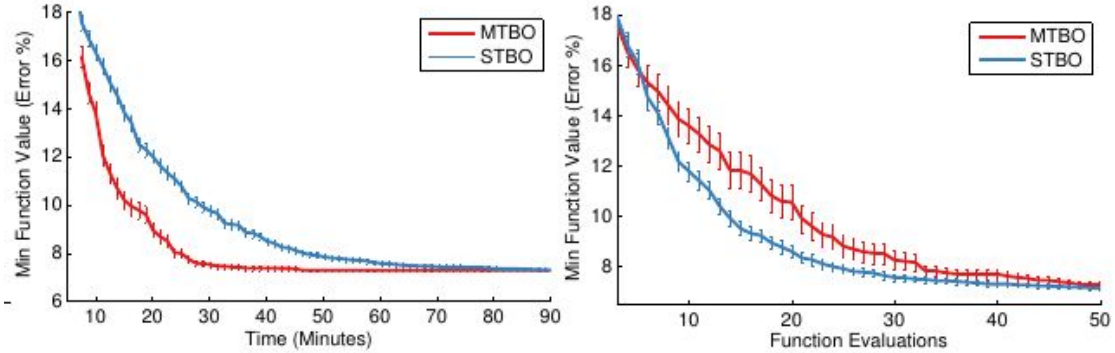
Results - Information from Related Task



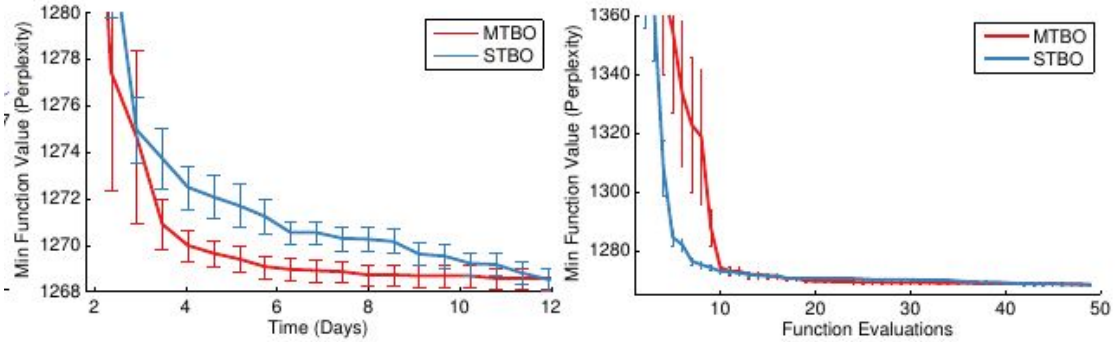
Results - Faster Task

Red = Multi Task

Blue = Single Task



(b) LR Time (c) LR Fn Evaluations



(e) Online LDA Time (f) Online LDA Fn Evaluations

Conclusion - Multi-Task Bayesian Optimization

- Information from other tasks can be used to speed up bayesian optimization
- Entropy search can help to find points which are useful for providing information about where the minimum is, but are not themselves the minimum