Multi-Task Bayesian Optimization

K Karthikeyan
Roll No.150311
Adviser :Prof. Piyush Rai

Todays Agenda

- Bayesian Optimization
- Gaussian Process
- Multi-Task Bayesian Optimization
- Previous work
- Our Work
- Future works

Bayesian Optimization(BO)

- Bayesian Optimization is method for global optimization of Black-box function
- Black-Box function
  - We know neither the form of function nor gradient nor Higher order Moments
  - We can query the function values at certain points
  - Query is expensive
  - Querying at some point can be more expensive than other
  - Lets assume cost of the query at any point is same. We will remove this assumption in Multi-task Bayesian Optimization (MTBO)
- We are allowed to query sequentially, our aim is to find the optimal value of black-box function with minimum cost.
- Bayesian Optimization chooses the next point using previous observed function values.
  - Function evaluation can be noisy

Bayesian Optimization(BO)

- Two step Process:
  - Bayesian Regression Model, to approximate black-box function to optimize.
    - Gaussian Process, Bayesian Neural networks, etc..
  - An Acquisition function, that tells us utility of a point.
    - Expected Improvement(EI), Probability of Improvement(PI), Upper confidence Bound(UCB) or Lower confidence bound (LCB), Expected Improvement(EI) per unit cost etc..

Bayesian Optimization(BO)

- Acquisition Function:
  - It is a tradeoff between exploration(high variance) and exploitation(high function value).
  - We want to explore more uncertain regions so that we don't end up at local optima (we may get better optimal solutions there).
  - We want to get more information in the region where our approximated function is near optimal.
  - To calculate acquisition function we need function values along with their uncertainty, so we need Bayesian regression models for approximating black-box function.

- Applications of BO:
  - Hyper-Parameter tuning for Machine learning models
    - In this presentation most of our examples are based on Hyper-Parameter tuning.
  - Design of expensive experiments
    - Drug design

Bayesian Regression Model:

- Two Popular Bayesian Regression Models
  - Gaussian Process
  - Bayesian Neural networks

- Bayesian Neural Networks:
  - Fully Bayesian inference on all the weights of neural network is difficult
  - So fully Bayesian estimate only from last hidden layer to output layer, others are point estimates
  - Fast to train and test
  - We will revisit Bayesian Neural network during Multi-Task BO.

Gaussian Process

- Probabilistic non-linear model
- More natural and Inspired from human intuitions
- Gaussian process (GP) defines prior over functions
  - Denoted by $GP(\mu,\ K)$, where μ is mean function and K is kernel/co-variance function
  - $f \sim GP(\mu,\ K)$ then $\mu(x) = E[f(x)]$, Mostly we will assume mean to be 0
  - K is kernel function
- GP's are very flexible model
- It's not just the hyper-parameters of kernels, we can learn the kernels itself.
  - Through combination of Multiple kernels

Gaussian Process

- GP Regression Predictions:
  - $P(y_* \,|\boldsymbol{y}) = N(y_* | \,\mu_*,\ \ \sigma_*^2)$
  - $\mu_* = \boldsymbol{k_*^T\ C_N^{-1}\ y}$
  - $\sigma_*^2 = k(x_*,x_*) + \sigma^2 - \boldsymbol{k_*^T\ \ C_N^{-1}\ k_*}$
  - $k_* = [k(x_*,x_1), k(x_*,x_2)\ \ldots\ldots\ldots\ldots\ .\quad k(x_*,x_N)\ ]$
  - $C_N \in R^{NXN}\quad where\ C_N[i,j] = k(x_i,x_j) + I[i=j]\ \sigma^2$

- Kernels functions (k) :
  - Says similarity between two data points
  - Popular Kernels
    - RBF kernels, ARD kernels, Matern Kernels, Linear kernels
- We will revisit Gaussian Process during Multi-Task Gaussian Process

Problems with Bayesian Optimization

- Lets say we want to find optimal hyper-parameters of a costly to evaluate machine learning model
- Can we use Bayesian Optimization ?
  - BO requires us to approximate the Black-Box function, but for any non-linear regression model to work nicely we need more data points (not-too high but at least few hundred data points for slightly)
  - It's not a good idea to evaluate ML model hundreds of time.
  - Then how can we use BO in that case.
    - Multi-Task Bayesian Optimization comes to rescue.

Multi-Task Bayesian Optimization

- We can view Multi-Task Bayesian Optimization (MTBO) in many ways
  - We want to optimize several Black-Box function simultaneously, and want to use their potential relationships.
  - We may be still interested in optimizing only function, but can query some other co-related function which is cheaper to evaluate.
  - We want to optimize only one function, but we want to transfer the knowledge of similar/co-related functions

- How MTBO solves the problem of less number of data ?
  - Let's say the model for which we want to tune hyper-parameter trains using 100000 data points (Task of interest).
  - We can create some co-related multiple tasks (with 100, 1000, 10000, data points), which are cheaper.
  - Now we can query on these cheaper tasks and use their results to optimize hyper-parameter of our task of interest
  - We might be able to find optimal hyper-parameter even without querying our true model once. (we can learn approximate co-relation function itself)

MTBO

- MTBO again has two components:
  - MT Bayesian regression
  - Acquisition function for MTBO
- MT Bayesian regression
  - Given the data points from multiple tasks $\{(x_i, y_i^t)\}$ we want the approximated function value at new data point for new task $(x_*, y_*^{t'})$.
  - Multi-task Gaussian Process, Multi-task Bayesian neural networks are popular choices.
- Acquisition functions
  - EI per unit cost, PI per unit cost, Entropy search etc…..

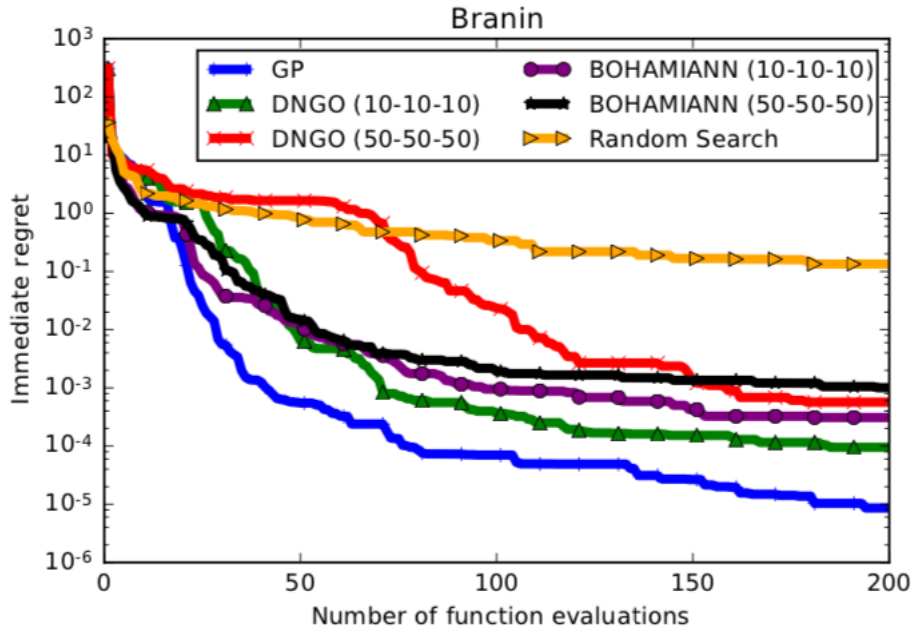
MTBO Acquisition functions

- Acquisition functions
  - Let's say our aim is to optimize only one function
  - We want to choose a data point from any task such that it is most useful in inferring optimal values of our task of interest at a minimum cost.
  - Entropy search per unit cost:
    - Choose the data point and the task, such that knowing it's value decreases the entropy most, for our primary task.
    - If we just want to pick such a candidate without considering cost, we always end up picking our primary task, which is not what we want.
    - So, change it slightly, choose the data point and the task, such that knowing it's value decreases the entropy most **per unit cost**, for our primary task
    - How do we know cost ?
      - Learn it ?

MT Bayesian Regression

- MT Bayesian neural networks (MTBNN):
  - Many ways of implementing MTBNN
  - We can model the network in such that every task shares same weight parameters till the last hidden layer, and each task has it's own weights from last hidden layer to output layer.
    - All the task shares the information through last hidden layer, which is common
  - We can give concatenate the task information (task embedding/one hot representation) in the input and learn the usual Bayesian neural network.
    - All the tasks share information as they have same weight parameters, and differ from each other through task embedding.
    - Two similar tasks gets similar embedding and hence similar results.
- Although BNN is nicely scalable and easy to implement, it is observed that in general they are not as accurate as Gaussian Process.

MTBNN vs MTGP



Ref : Bayesian Optimization with Robust Bayesian Neural Networks

Multi-Task Gaussian Process(MTGP)

- Similar to usual Gaussian Process but the kernel functions are task dependent kernel functions.
  - Lets assume that task t is modelled by function f^t
  - $y^{(1)} = f^{(1)}(x)$, $y^{(2)} = f^{(2)}(x), \ldots .$ $y^{(t)} = f^{(t)}(x)$
  - We know $cov\left(y^{(t)}, y'^{(t')}\right) = cov\left(f^{(t)}(x)\, f^{t'}(x')\right)$
  - Since our kernel function is same as the above co-variance function, we need to model our kernel function as a function of two tasks along with two inputs
  - $kernel\ function = k^{tt'}(x,x'), where\ t\ and\ t^1\ are\ corresponding\ task\ of\ output\ y\ and\ y'$
  - By above formulation we need to fix $O(T^2)$ number of kernels, which is a harder task
  - A commonly used approximation to the above kernel is
    - $k^{tt'}(x,x') = k^a(t,t')k^b(x,x')$

MTGP

- $k^{tt'}(x,x') = k^a(t,t')k^b(x,x')$
  - We can write it as covariance between tasks and covariance between the data points
  - Since we don't know any representation of tasks t, t', we can not calculate $k^a(t,t')$ like usual kernel function, we need to find $O(T^2)$ number of co-variances one for each pair of tasks.
  - $k^{tt'}(x,x') = cov(t,t')k^b(x,x')$
  - In our model we over come this problem, we get meaningful representation (implicitly) for each task.
  - We also suggest another method which weakens the above assumption of splitting the task dependent kernel function into two components (covariance of task and co-variance of data point), but it is more expensive.

Predictions with MTGP:

- Similar to usual Gaussian Process but use task dependent kernels instead of usual kernels.
- MTGP Regression Predictions:
  - $P(y_*^t \,|\boldsymbol{y}) = N(y_* | \, \mu_*^t, \;\; \sigma_*^{2(t)})$
  - $\mu_*^t \; = \boldsymbol{k}_*^{(\boldsymbol{t})\boldsymbol{T}} \; \boldsymbol{C_N^{-1}} \; \boldsymbol{y}$
  - $\sigma_*^{2(t)} \; = k^{t,t}(x_*, x_*) + \sigma^2 \; - \boldsymbol{k}_*^{(\boldsymbol{t})\boldsymbol{T}} \; \boldsymbol{C_N^{-1}} \; \boldsymbol{k}_*^{(\boldsymbol{t})}$
  - $\boldsymbol{k_*^t} = \left[k^{t,t_1}(x_* \,, x_1), k^{(t,t_2)}(x_*, x_2) \; \ldots\ldots\ldots\ldots \quad k^{(t,t_N)}(x_* \,, x_N)\right]^T$
  - $C_N \; \in R^{NXN} \quad where \; C_N[i,j] = k^{t_i,t_j}\left(x_i, x_j\right) + I[i=j] \; \sigma^2$
- Same as usual GP except everything now is task dependent. It is very much non scalable $O(N^3)$, time complexity which is not desired.

Our Method

- Let's view the predictions of usual GP in a slightly different perspective.
- SVM interpretation of mean of GP
  - $P(y_* \,|\boldsymbol{y}) = N(y_* | \,\mu_*,\ \ \sigma_*^2)$
  - $\mu_* = \boldsymbol{k}_*^{\boldsymbol{T}}\, \boldsymbol{C}_{\boldsymbol{N}}^{-\boldsymbol{1}}\, \boldsymbol{y}$
  - $k_* = [k(x_*, x_1), k(x_*, x_2)\ \ldots\ldots\ldots\ldots\ .\quad k(x_*, x_N)\ ]$
  - $\mu_* = \boldsymbol{k}_*^{\boldsymbol{T}}\, \boldsymbol{C}_{\boldsymbol{N}}^{-\boldsymbol{1}}\, \boldsymbol{y}\ = \boldsymbol{k}_*^{\boldsymbol{T}} \boldsymbol{\alpha} = \sum_{i=1}^{n} k(x_*, x_n)\alpha_n$
- The above interpretation of GP is useful as we can use the optimization techniques of kernel SVM for GP also.
- Our method is inspired from the above idea.

Our Method

- Lets try to write MTGP mean in a SVM form, without using task dependent kernels.
  - $\mu_*^t = \boldsymbol{k}_*^{(t)T} \boldsymbol{C}_N^{-1} \boldsymbol{y}$
  - $\mu_*^t = \boldsymbol{k}_*^T \boldsymbol{\alpha^t} = \sum_{i=1}^{n} k(x_*, x_n)\alpha_n^t$
  - $\boldsymbol{\alpha^t} = \sum_{i=1}^{K} z_{t,i} \boldsymbol{B}[:, \boldsymbol{i}]$ $\boldsymbol{\alpha^t}$ is a task dependent linear combination of some basis vectors, and $B \in R^{NXK}$ matrix $\mathbf{B}[:, \mathbf{i}]$ is the i^{th} column of the matrix, and $z \in R^{TXK}, z_{t,i}$ is the t^{th} row and i^{th} $column$ of z
  - We will see the proof of above equations in the next slide.
- Note that the above mean does not use task dependent kernels, instead task dependency part is taken care by α^t, which again taken by z_t
  - From above equations we know how to find mean what about variance.
    - It follows from the proof/equivalence of above equations. Lets see it.

Our Method

- Proof of equivalence of previous equations to Multi-task GP's
  - $k^{t_i t_j}(x_i, x_j) = k^a(x_i\ x_j)k^b(t_i\ t_j)$
  - Let us make the following assumption on k^b, which is a slightly stronger assumption than mercer kernels.
    - $k^b(t_i\ t_j) = z(t_i)^T z(t_j)$, where $z(t) \in R^K$
    - Where $z(t)$ is some feature vector (embedding/representation) of task t (we don't need to specify this our model can learn it)
  - $C_N[i,j] = k^{t_i,t_j}(x_i,x_j) + I[i=j]\sigma^2$
  - $C_N[i,j] = k^a(x_i\ x_j)k^b(t_i\ ,t_j) + I[i=j]\sigma^2$
  - $C_N[i,j] = k^a(x_i\ x_j)z(t_i)^T z(t_j) + I[i=j]\sigma^2$

Our Method

- Proof contd.
  - $\mu_*^{(t)} = \quad \boldsymbol{k}_*^{(t)T} C_N^{-1} \boldsymbol{y} \quad = \quad \boldsymbol{k}_*^{(t)T} \boldsymbol{\beta}$ where $\boldsymbol{\beta} = C_N^{-1} \boldsymbol{y} \in R^N$
  - $\mu_*^{(t)} = \sum_{i=1}^{N} k^{t,t_i}(x_*, x_i)\, \beta_n$
  - $\mu_*^{(t)} = \sum_{n=1}^{N} k^a\,(x_*, x_n)\, k^b(t, t_n)\, \beta_n$
  - $\mu_*^{(t)} = \sum_{n=1}^{N} k^a\,(x_*, x_n) z(t)^T z(t_n)\, \beta_n$
  - Lets call $z(t_n)\beta_n = B_n \in R^K$
  - $\mu_*^{(t)} = \sum_{n=1}^{N} k^a\,(x_*, x_n) z(t)^T B_n$
  - Lets call $z(t)^T B_n = \sum_{i=1}^{K} z(t)_i B_{ni} = \quad \alpha_n^t \quad \in \quad R$
  - $\mu_*^{(t)} = \sum_{n=1}^{N} k^a\,(x_*, x_n) \alpha_n^t$
  - The exact equation what we saw in the previous slide, where B_n is the n^{th} row of matrix B, and $z(t)$ is the t^{th} row of matrix z.
  - We need to find B and z

Our Method

- Inference for $B\ and\ z$
  - $\mu_*^{(t)} = \sum_{n=1}^{N} k^a(x_*, x_n) \sum_{i=1}^{K} z(t)_i B_{ni}$
  - To calculate μ we need to find B and z, others are known
  - Model should be able to explain our data points properly.
  - $y_n^{(t)} = \sum_{n=1}^{N} k^a(x_n, x_n) \sum_{i=1}^{K} z(t)_i B_{ni} + \epsilon_n$
  - $\epsilon_n \sim Likelihood\ Model\ (mostly\ Gaussian\ )$
  - We need to find $B\ and\ z$ such that it minimizes overall error or maximizes likelihood of the data.
  - How to find B and z ?
    - Alternating optimization

Our Method

- Alternating Optimization:
  - Assume we know matrix z and matrix $B\ except\ one\ column\ of\ B\ (say\ k^{th}\ column)$, we can find k^{th} column of B easily.
    - Same as linear regression model with $z_{tk}\boldsymbol{k}_*^{\boldsymbol{a(t)T}}(\boldsymbol{x_n})$ as feature vector and B_k unknown column as weight vector.
    - $O(N)$ time per SGD iteration.
  - Assume we know B and find each row of z
    - Again same as linear regression with $\boldsymbol{k}_*^{\boldsymbol{a(t)T}}(\boldsymbol{x_n})\mathrm{B}$ as feature vector and unknown row of z as weight.
    - $O(KN)$ time per SGD iteration. Where K is constant we fix for our model.

Our Method

- Prediction complexity
  - Once we know B and z, then we can find $\alpha^t \ \forall t,$ once we have α^t then it is same as kernel SVM with α^t as support
  - Prediction complexity for Kernel SVM is $O(1)$ (approximation method), so we can also predict in $O(1)$ time.

- Variance:
  - To calculate variance we need $C_N^{-1}\ and\ k = k^a k^b\ we\ know\ k^a\ we\ need\ k^b$
    - $k^b(t,t') = z(t)^T z(t')\ we\ know\ z, hence\ we\ know\ k^b$
  - $\boldsymbol{\beta} = C_N^{-1}\boldsymbol{y},$ we know both $\boldsymbol{\beta}\ and\ \boldsymbol{y},$ hence we can find C_N^{-1}.

Our Method

- Summary:
  - We have shown our method is theoretically equivalent to current existing method of MTGP.
  - Gradient calculations in our method is very simple (same as linear regression)
  - Easy to implement
  - $O(NK)$ time per SGD iteration during training.
  - Nicely scalable.
  - $O(1)$ time prediction complexity

Our Method 2

- Sometimes we may be more interested in accuracy not scalability.
- In such cases assuming task dependent kernels can be decomposed into two parts co-variance between tasks and co-variance between inputs in a very strong assumption to make.
  - $k^{t_i t_j}(x_i, x_j) = k^a(x_i\ x_j)k^b(t_i\ t_j)$
- In this section we provide a method which makes weaker assumption than above equation and show that above equation is a special case of our method.
- Basic idea is same as we saw in the previous slides, i.e. write task dependent quantities as task dependent linear combination of some basis quantities.
- Idea:
  - We will write our task dependent kernel function as task dependent linear combination of task independent basis kernels.

Our Method 2

- Task dependent kernel
  - $k^{t_i t_j}(x_i, x_j)$
  - We can write above as the following linear combination.
  - $k^{t_i t_j}(x_i, x_j) = \sum_{m=1}^{K} \sum_{n=1}^{K} U_m^{t_i} V_n^{t_j} K_{mn}(x_i, x_j)$
  - Where $U \text{ and } V \text{ are some embedding of tasks}$, we can learn them.
  - K_{mn} is a usual kernel functions, we totally need only K^2 kernels instead of T^2 kernels without approximation.
  - Previous assumption of decomposing kernel is a special case of above method.
    - When all $K_{mn}(x_i, x_j) = k^a(x_i, x_j)$ are same
    - $k^{t_i t_j}(x_i, x_j) = \sum_{m=1}^{K} \sum_{n=1}^{K} U_m^{t_i} V_n^{t_j} K_{mn}(x_i, x_j)$
    - $k^{t_i t_j}(x_i, x_j) = \sum_{m=1}^{K} \sum_{n=1}^{K} U_m^{t_i} V_n^{t_j} k^a(x_i\ x_j)$
    - $k^{t_i t_j}(x_i, x_j) = k^a(x_i\ x_j)[\sum_{m=1}^{K} \sum_{n=1}^{K} U_m^{t_i} V_n^{t_j}]$
    - $k^{t_i t_j}(x_i, x_j) = k^a(x_i\ x_j) k^b(t_i\ t_j)$

Our Method 2

- Advantages of method2:
  - We can learn kernels implicitly.
    - In general we will choose the kernel by writing kernel as linear combination of some popular kernels and optimize the linear combination.
    - Here it is implicit, we can choose basis kernels from some popular kernels, and linear combination is optimized taking tasks into account.
  - Weaker assumption to make than decomposing task dependent kernel into two components, task and input components respectively.

Some interesting Future Works

- Zero-shot Bayesian Optimization:
  - We need to optimize a new black-box function even without querying once.
  - We can find some features of task (learn task embedding $z(t)$ using these features)
  - Do MTBO, along with learning the co-variance using those features of tasks.

- MTGP with tasks from different domain
  - In this work we have assumed that all the task are from same domain (at least same dimensions), but kernels can be calculated even if two we have data from two different spaces (Mahalanobis distance).
  - We can also do this another way, first find a non-linear transformation such that reconstruction error is minimum and co-relation between tasks is more.

THANK YOU FOR
YOUR LİSTENİNG

DO YOU HAVE
ANY QUESTİONS?