Two Use-cases of Uncertainty:
Active Learning and Bayesian Optimization

CS772A: Probabilistic Machine Learning

Piyush Rai

Announcement and Plan today

- Announcement
  - Quiz 3 on March 18 at 6pm (that day's class will start after the quiz is over)
  - Syllabus for quiz 3 includes VI, sampling, and whatever else we covered by March 11
- Plan today:
  - Wrap-up HMC and parallel/distributed methods for MCMC
  - Two use-cases of uncertainty
    - Active Learning
    - Bayesian optimization

Hamiltonian/Hybrid Monte Carlo (HMC)

- HMC (Neal, 1996) is an “auxiliary variable sampler” and incorporates gradient info
- Uses the idea of simulating a Hamiltonian Dynamics of a physical system
- Consider the target posterior $\boldsymbol{p(\theta|\mathcal{D}) \propto \exp(-U(\theta))}$
- Think of $\boldsymbol{\theta}$ as “position” then $\boldsymbol{U(\theta) = -\log p(\mathcal{D}|\theta)p(\theta)}$ is like “potential energy”
- Let’s introduce an <u>auxiliary variable</u> - the momentum $\boldsymbol{r}$ of the system
- The total energy (potential + kinetic) or the Hamiltonian of the system

Constant w.r.t. time

$$H(\theta,\boldsymbol{r}) = U(\theta) + \frac{1}{2}\boldsymbol{r}^\top M^{-1}\boldsymbol{r} = U(\theta) + K(\boldsymbol{r})$$

- Can now define a joint distribution over the position and momentum as

$$p(\theta,\boldsymbol{r}|\mathcal{D}) \propto \exp\left(-H(\theta,\boldsymbol{r})\right) = \exp\left(-U(\theta) - \frac{1}{2}\boldsymbol{r}^\top M^{-1}\boldsymbol{r}\right) \propto p(\theta|\mathcal{D})p(\boldsymbol{r})$$

- Given a sample $\boldsymbol{(\theta,r)}$ from $\boldsymbol{p(\theta,r|\mathcal{D})}$, ignoring $\boldsymbol{r}$, $\boldsymbol{\theta}$ will be a sample from $\boldsymbol{p(\theta|\mathcal{D})}$

Generating Samples in HMC

$$H(\theta, \mathbf{r}) = U(\theta) + \tfrac{1}{2}\mathbf{r}^\top M^{-1}\mathbf{r} = U(\theta) + K(\mathbf{r})$$

- Given an initial $(\boldsymbol{\theta}, \boldsymbol{r})$, Hamiltonian Dynamics defines how $(\boldsymbol{\theta}, \boldsymbol{r})$ changes w.r.t. time $\boldsymbol{t}$

$$\frac{\partial\theta}{\partial t} = \frac{\partial H}{\partial \mathbf{r}} = \frac{\partial K}{\partial \mathbf{r}} \qquad \frac{\partial \mathbf{r}}{\partial t} = -\frac{\partial H}{\partial \theta} = -\frac{\partial U}{\partial \theta}$$

Hamiltonian Dynamics ensures conservation of total energy

➡ $$\frac{dH}{dt} = \sum_{i=1}^{D}\left[\frac{\partial H}{\partial \theta_i}\frac{\partial \theta_i}{\partial t} + \frac{\partial H}{\partial r_i}\frac{\partial r_i}{\partial t}\right] = \sum_{i=1}^{D}\left[\frac{\partial H}{\partial \theta_i}\frac{\partial H}{\partial r_i} - \frac{\partial H}{\partial r_i}\frac{\partial H}{\partial \theta_i}\right] = 0$$

- We can use these equations to update $(\boldsymbol{\theta}, \boldsymbol{r}) \rightarrow (\boldsymbol{\theta}^*, \boldsymbol{r}^*)$ by <u>discretizing time</u>

Reason: Getting analytical solutions for the above requires integrals which is in general intractable

- For $s = 1{:}S$, sample as follows
  - Initialize $\theta_0 = \theta^{(s-1)}$, $\mathbf{r}_* \sim \mathcal{N}(0, \mathbf{I})$ and $\mathbf{r}_0 = \mathbf{r}_* - \frac{\rho}{2}\frac{\partial U}{\partial \theta}|_{\theta_0}$
  - Do L "leapfrog" steps with learning rates $\rho_\ell = \rho$ for $\ell < L$ and $\rho_L = \rho/2$
    - For $\ell = 1{:}L$

$$\theta_\ell = \theta_{\ell-1} + \rho\frac{\partial K}{\partial \mathbf{r}}|_{\mathbf{r}_{\ell-1}}$$

$$\mathbf{r}_\ell = \mathbf{r}_{\ell-1} - \rho_\ell\frac{\partial U}{\partial \theta}|_{\theta_\ell}$$

L usually set to 5 and learning rate tuned to make acceptance rate around 90%

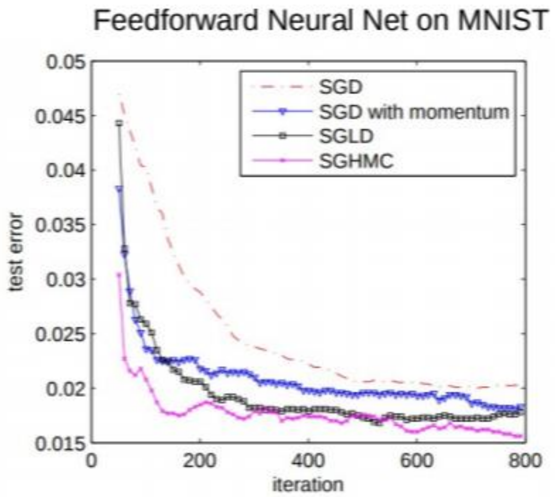
  - Perform MH accept/reject test on $(\boldsymbol{\theta}_L, \boldsymbol{r}_L)$. If accepted $\boldsymbol{\theta}^{(s)} = \boldsymbol{\theta}_L$

A single sample generated by taking L steps

- Use of momentum helps exploring the distribution more efficiently

HMC in Practice

- HMC typically has very low rejection rate (that too, primarily due to discretization error)
- Performance can be sensitive to L (no. of leapfrog steps) and step-sizes, tuning hard
- A lot of renewed interest in HMC (you may check out NUTS - No U-turn Sampler – doesn't require setting L)
  - Prob. Prog. packages e.g., Tensorflow Prob., Stan, etc, contain implementations of HMC
- Can also do HMC on minibatches (Stochastic Gradient HMC - Chen et al, 2014)
- An illustration: SGHMC vs other methods on MNIST classification (Bayesian neural net)



(Figure: Stochastic Gradient Hamiltonian Monte Carlo (Chen et al, 2014))

Parallel/Distributed MCMC

- Suppose our goal is to compute the posterior of $\theta \in \mathbb{R}^D$ (assuming N is very large)

$$p(\theta|\mathbf{X}) \propto p(\theta)p(\mathbf{X}|\theta) = p(\theta)\prod_{n=1}^{N} p(\boldsymbol{x}_n|\theta)$$

- Suppose we have J machines with data partitioned as $\mathbf{X} = \{\mathbf{X}^{(j)}\}_{j=1}^{J}$
- Let's assume that the posterior $p(\theta|\mathbf{X})$ factorizes as

$$p(\theta|\mathbf{X}) = \prod_{i=1}^{J} p^{(j)}(\theta|\mathbf{X}^{(j)})$$

- Here $p^{(j)}(\theta|\mathbf{X}^{(j)}) \propto p(\theta)^{1/J}\prod_{\boldsymbol{x}_n \in \mathbf{X}^{(j)}} p(\boldsymbol{x}_n|\theta)$ is known as the "subset posterior"
- Assume the j^{th} machine generates T MCMC samples $\{\theta_{j,t}\}_{t=1}^{T}$
- We need a way to combine these subset posteriors using a "consensus"

$$\hat{\theta}_1, \ldots, \hat{\theta}_T = \texttt{CONSENSUSSAMPLES}(\{\theta_{j,1}, \ldots, \theta_{j,T}\}_{j=1}^{J})$$

Parallel/Distributed MCMC

- Many ways to compute the consensus samples. Let's look at two of them
- Approach 1: Weighted Average: $\hat{\theta}_t = \sum_{j=1}^{J} W_j \theta_{j,t}$ where W_j can be learned as follows
  - Assuming Gaussian likelihood and Gaussian prior on θ

$$
\begin{aligned}
\bar{\Sigma}_j &= \text{sample covariance of } \{\theta_{j,1}, \ldots, \theta_{j,T}\} \\
\Sigma &= (\Sigma_0^{-1} + \sum_{j=1}^{J} \bar{\Sigma}_j^{-1})^{-1} \quad (\Sigma_0 \text{ is the prior's covariance}) \\
W_j &= \Sigma(\Sigma_0^{-1}/J + \bar{\Sigma}_j^{-1})
\end{aligned}
$$

These approaches can also be used to make VI parallel/distributed

- Approach 2: Fit J Gaussians, one for each $\{\theta_{j,t}\}_{t=1}^{T}$and take their product

$$
\begin{aligned}
\bar{\mu}_j &= \text{sample mean of } \{\theta_{j,1}, \ldots, \theta_{j,T}\}, \quad \bar{\Sigma}_j = \text{sample covariance of } \{\theta_{j,1}, \ldots, \theta_{j,T}\} \\
\hat{\Sigma}_J &= (\sum_{j=1}^{J} \bar{\Sigma}_j^{-1})^{-1}, \quad \hat{\mu}_J = \hat{\Sigma}_J(\sum_{j=1}^{J} \bar{\Sigma}_j^{-1} \bar{\mu}_j) \quad (\text{cov and mean of prod. of Gaussians}) \\
\hat{\theta}_t &\sim \mathcal{N}(\hat{\mu}_J, \hat{\Sigma}_J), t = 1, \ldots, T \quad (\text{the final consensus samples})
\end{aligned}
$$

- For detailed proofs and other approaches, may refer to the reference below

Patterns of Scalable Bayesian Inference (Angelino et al, 2016)

Approximate Inference: VI vs Sampling

- VI approximates a posterior distribution $\boldsymbol{p(Z|X)}$ by another distribution $\boldsymbol{q(Z|\phi)}$
- Sampling uses $\boldsymbol{S}$ samples $\boldsymbol{Z^{(1)}, Z^{(2)}, \dots, Z^{(S)}}$ to approximate $\boldsymbol{p(Z|X)}$
- Sampling can be used within VI (ELBO approx using Monte-Carlo)
- In terms of “comparison” between VI and sampling, a few things to be noted
  - Convergence: VI only has local convergence, sampling (in theory) can give exact posterior
  - Storage: Sampling based approx needs to storage all samples, VI only needs var. params $\boldsymbol{\phi}$
  - Prediction Cost: Sampling always requires Monte-Carlo avging for posterior predictive; with VI, sometimes we can get closed form posterior predictive

PPD if using sampling:

$$p(x_*|X) = \int p(x_*|Z)p(Z|X)dZ \approx \frac{1}{S}\sum_{s=1}^{S} p(x_*|Z^{(s)})$$

PPD if using VI:

$$p(x_*|X) = \int p(x_*|Z)p(Z|X)dZ \approx \int p(x_*|Z)q(Z|\phi)dZ$$

Closed form if integral is tractable (otherwise Monte Carlo avg still needed for PPD)

Compressing the $\boldsymbol{S}$ samples into something more compact

- There is some work on “compressing” sampling-based approximations*

*”Compact approximations to Bayesian predictive distributions” by Snelson and Ghaharamani, 2005; and “Bayesian Dark Knowledge” by Korattikara et al, 2015

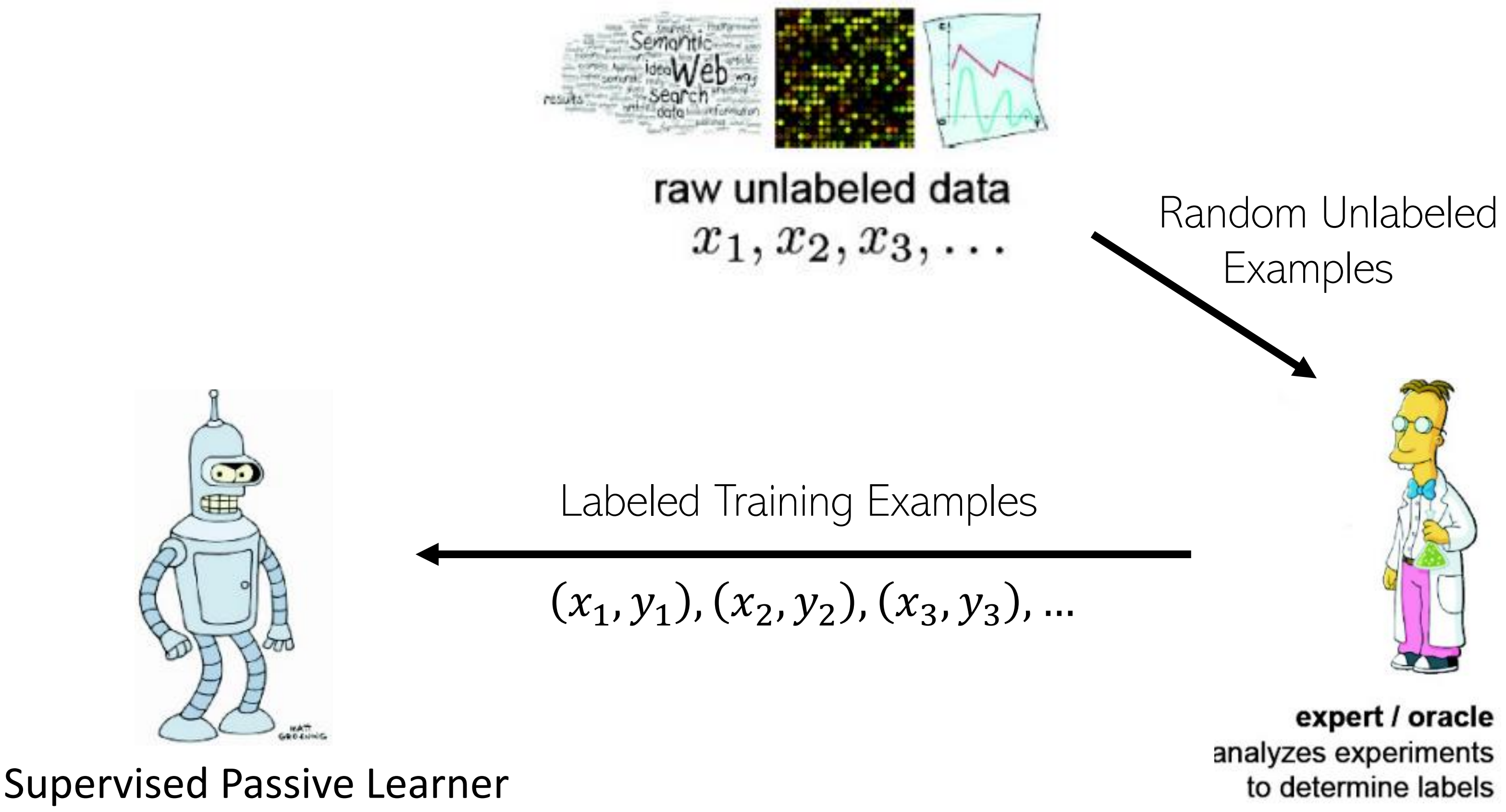
Inference Methods: Summary

- MLE/MAP: Straightforward for differentiable models (can even use automatic diff.)
- Conjugate models with one “main” parameter: Straightforward posterior updates
- MLE-II/MAP-II: Often useful for estimating the hyperparameters
- EM: If we want to do MLE/MAP for models with latent variables
  - Very general algorithm, can also be made online
  - Used when we want point estimates for some unknowns and posterior over others
  - Can use it for hyperparameter estimation as well
  - Often better than using direct gradient methods
- VI and sampling methods can be used to get full posterior for complex models
  - Quite easy if we have local conjugacy (VI has closed form updates, Gibbs sampler is easy to derive)
  - In other cases, we have general VI with Monte-Carlo gradients, MH sampling
  - MCMC can also make use of gradient info (LD/SGLD)
- For large-scale problems, online/distributed VI/MCMC, or SGD based posterior approx

Active Learning

Passive Learning

- Standard supervised learning is passive
  - Learner has no control over what labelled training examples it gets to learn from



raw unlabeled data

$x_1, x_2, x_3, \ldots$

Random Unlabeled Examples

Labeled Training Examples

$(x_1, y_1), (x_2, y_2), (x_3, y_3), \ldots$

Supervised Passive Learner

expert / oracle
analyzes experiments
to determine labels

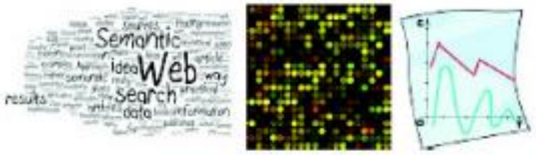
Active Learning

It is therefore also a sequential learning strategy (training data is not given all at once)

- In Active Learning, the learner can request specific labelled examples as it trains
  - In particular, examples that the learner thinks will be most useful to learn the underlying function

Will soon see what are some common notions of usefulness

Using the current model, identify the most useful example(s) from the unlabeled data pool



raw unlabeled data

$x_1, x_2, x_3, \ldots$

Although one example in each iteration is more common, labels of one or more than one examples can be queried ("batch" active learning")

Assumes some small amount of initial labelled training examples $\mathcal{D} = \{(x_n, y_n)\}_{n=1}^N$ are available to learn an initial model

Query the expert for the true label of the selected unlabeled example, say x_1

$\langle x_1, ? \rangle$

$\langle x_1, y_1 \rangle$

$\langle x_2, ? \rangle$

$\langle x_2, y_2 \rangle$

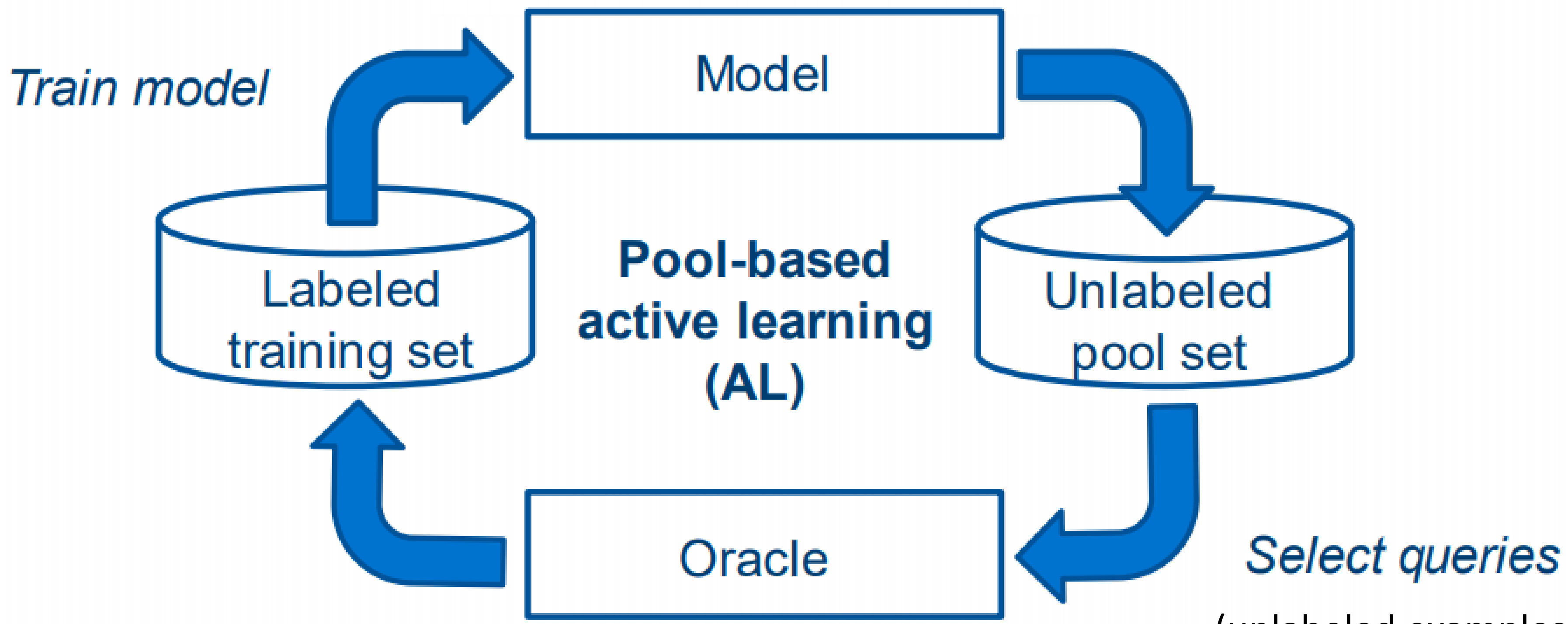
Supervised Active Learner

expert / oracle
analyzes experiments
to determine labels

The new labelled examples acquired "actively" are used to improve the initial model by retraining it using the updated training data $\mathcal{D} = \{\mathcal{D} \cup (x_*, y_*)\}$ and repeating the process until we get the desired accuracy or our budget exhausts

Active Learning

- The figure below is another illustration of AL



(provides true labels for the unlabeled examples)

(unlabeled examples)

Measuring Usefulness in AL

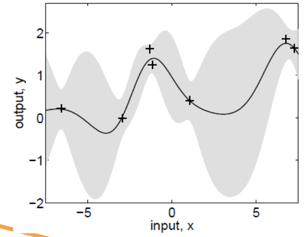
Given the acquisition function, the most useful example can be selected from the pool as

$$\hat{\boldsymbol{x}}_* = \text{argmax}_{\boldsymbol{x}_* \in \mathcal{X}_{pool}} A(\boldsymbol{x}_*|p(\theta|\mathcal{D}))$$

- Various ways to measure the usefulness of an unlabeled example $\boldsymbol{x}_*$
  - Defined by an "acquisition function" $A(\boldsymbol{x}_*)$ (high value for most useful unlabeled examples)

Note: We will use shorthand $A(\boldsymbol{x}_*)$

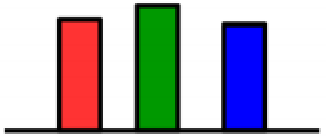
- Approach 1: For $\boldsymbol{x}_*$, look at uncertainty in output $\boldsymbol{y}_*$ predicted by the current model
  - Can use variance in the posterior predictive distribution: $A(\boldsymbol{x}_*) = \text{var}(\boldsymbol{y}_*)$
  - More generally, can use entropy of the PPD: $A(\boldsymbol{x}_*) = \mathbb{H}[p(\boldsymbol{y}_*|\boldsymbol{x}_*, \mathcal{D})]$



Note that this is the "marginal entropy" of the output distribution since the posterior predictive of the output is obtained by marginalizing over the posterior

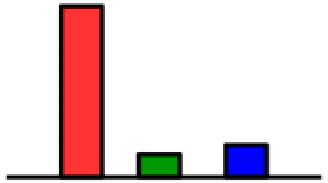
High entropy

Assume 3 classes



Learner uncertain about label of x (getting its label would be useful)

Low entropy



Learner already certain about label of x (getting its label won't be much useful)

- Approach 2: Look at how much our model will improve if we add this unlabeled example with its true label, to our training set, and retrain the model

$$A(x_*) = \mathbb{H}[p(\theta|\mathcal{D})] - \mathbb{E}_{p(y_*|x_*,\mathcal{D})}\mathbb{H}[p(\theta|\mathcal{D} \cup (\boldsymbol{x}_*, \boldsymbol{y}_*))] = \mathbb{I}[\theta;\ \boldsymbol{y}_*|\mathcal{D}, \boldsymbol{x}_*]$$

Mutual Information

Entropy of the current posterior

Need to use expectation here since $\boldsymbol{y}_*$ is not known

Entropy of the new posterior after including the new example in our training set

Batch Active Learning

- Approaches we saw work by querying and adding one example at a time
- Expensive in practice since we have to retrain every time after including a new example
  - Especially true for deep learning models which are computationally expensive to train
- In practice, we want to use AL to jointly query the labels of $B > 1$ examples

$$(\hat{\boldsymbol{x}}_1, \hat{\boldsymbol{x}}_2, \dots, \hat{\boldsymbol{x}}_B) = \text{argmax}_{(\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_B) \in \mathcal{X}_{pool}} A\big(\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_B | p(\theta|\mathcal{D})\big)$$

- Difficult to construct such joint acquisition function and maximize them
- A greedy scheme is to simply select the B highest scoring points

$$A\big(\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_B | p(\theta|\mathcal{D})\big) = \sum_{b=1}^{B} \mathbb{I}[\theta;\ y_b|\mathcal{D}, \boldsymbol{x}_b]$$

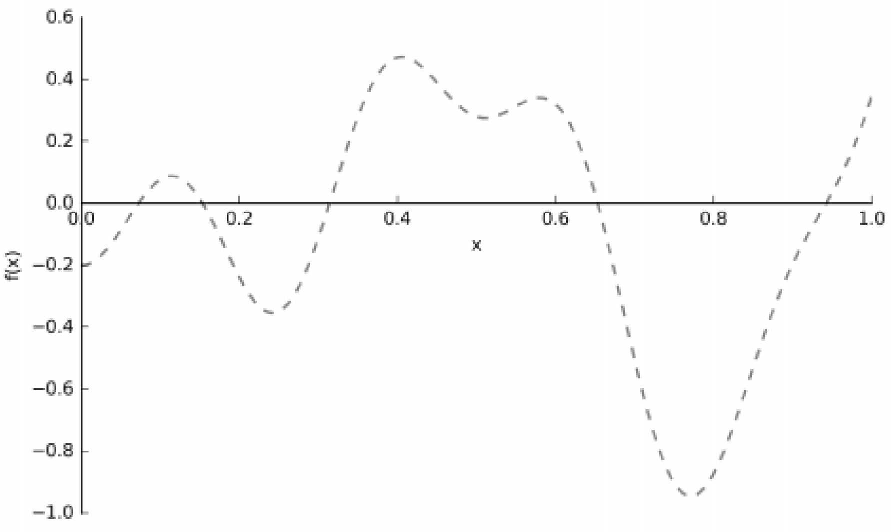
- The above however is myopic and ignores correlations among the selected points
  - Some recent works have addresses this issue[1]

[1] BatchBALD: Efficient and Diverse Batch Acquisition for Deep Bayesian Active Learning (Kirsch et al, NeurIPS 2019)

Bayesian Optimization

Bayesian Optimization: The Basic Formulation

- Consider finding the optima x_* (say minima) of a function $f(x)$



- Caveat: We don’t know the form of the function; can’t get its gradient, Hessian, etc
- Can only query the function’s values at certain points (i.e., only “black-box” access)
  - The values may or may not be noisy (i.e., we may be given $f(x)$ or $f(x) + \epsilon$)

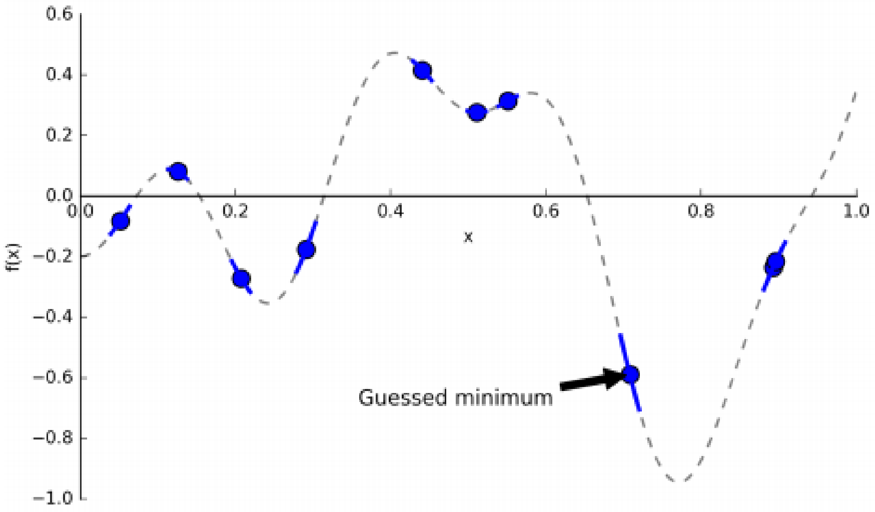
Bayesian Optimization: Some Applications

- Drug Design: Want to find the optimal chemical composition for a drug
  - Optimal composition will be the one that has the best efficacy
  - But we don't know the efficacy function
  - Can only know the efficacy via doing clinical trials
  - Each trial is expensive; can't do too many trials
- Hyperparameter Optimization: Want to find the optimal hyperparameters for a model
  - Optimal hyperparam values will be those that give the lowest test error
  - Don't know the true "test error" function
  - Need to train the model each time with different h.p. values and compute test error
  - Training every time will be expensive (e.g., for deep nets)
  - Note: Hyperparams here can even refer to the structure of a deep net (depth, width, etc)
- Many other applications: Website design via A/B testing, material design, optimizing physics based models (e.g., aircraft design), etc

Bayesian Optimization

- Can use BO to find maxima or minima

- Would like to locate the optima by querying the function's values (say, from an oracle)



Assuming the goal is to find the minima

- We would like to do so using as few queries as possible

- Reason: The function's evaluation may be time-consuming or costly

Bayesian Optimization

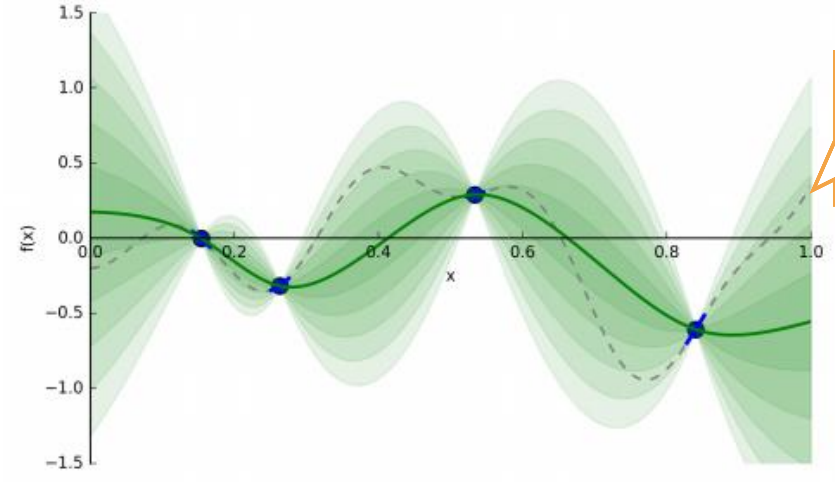
Note: Function values can be noisy too, e.g., $f(x_n) + \epsilon_n$

- Suppose we are allowed to make the queries sequentially
- This information will be available to us in form of query-function value pairs

$\{(x_n, f(x_n))\}_{n=1}^N$

- Queries so far can help us estimate the function

By solving a regression problem



Dotted curve: True function
Green curve: Current estimate ("surrogate") of the function
Shaded region: Uncertainty in the function's estimate

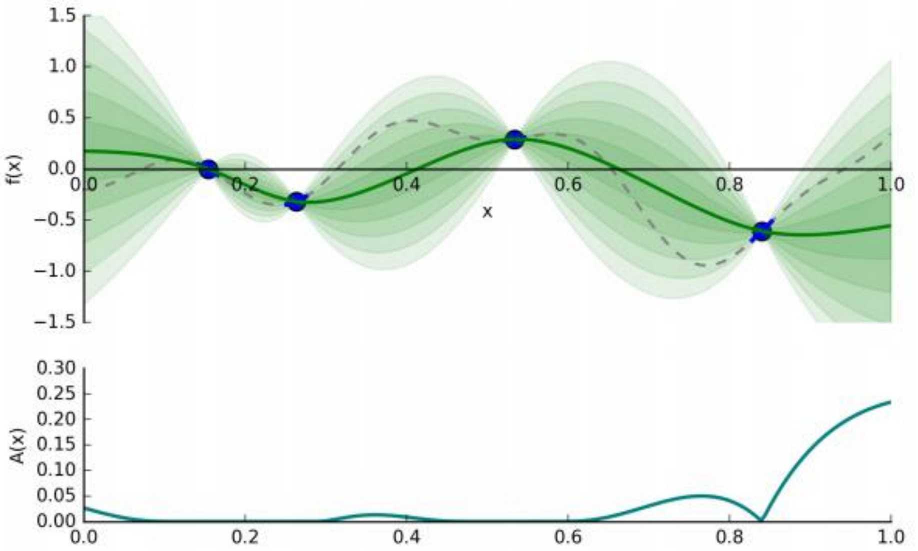
- BO uses past queries + function's estimate+uncertainty to decide where to query next
- Similar to Active Learning but the goal is to learn f as well as finds its optima

Bayesian Optimization

Note: Function values can be noisy too, e.g., $f(x_n) + \epsilon_n$

- BO requires two ingredients
  - A regression model to learn a surrogate of $f(x)$ given previous queries $\{(x_n, f(x_n))\}_{n=1}^N$
  - An acquisition function $A(x)$ to tell us where to query next

Assumption: $A(x)$ should be easier to optimize than $f(x)$



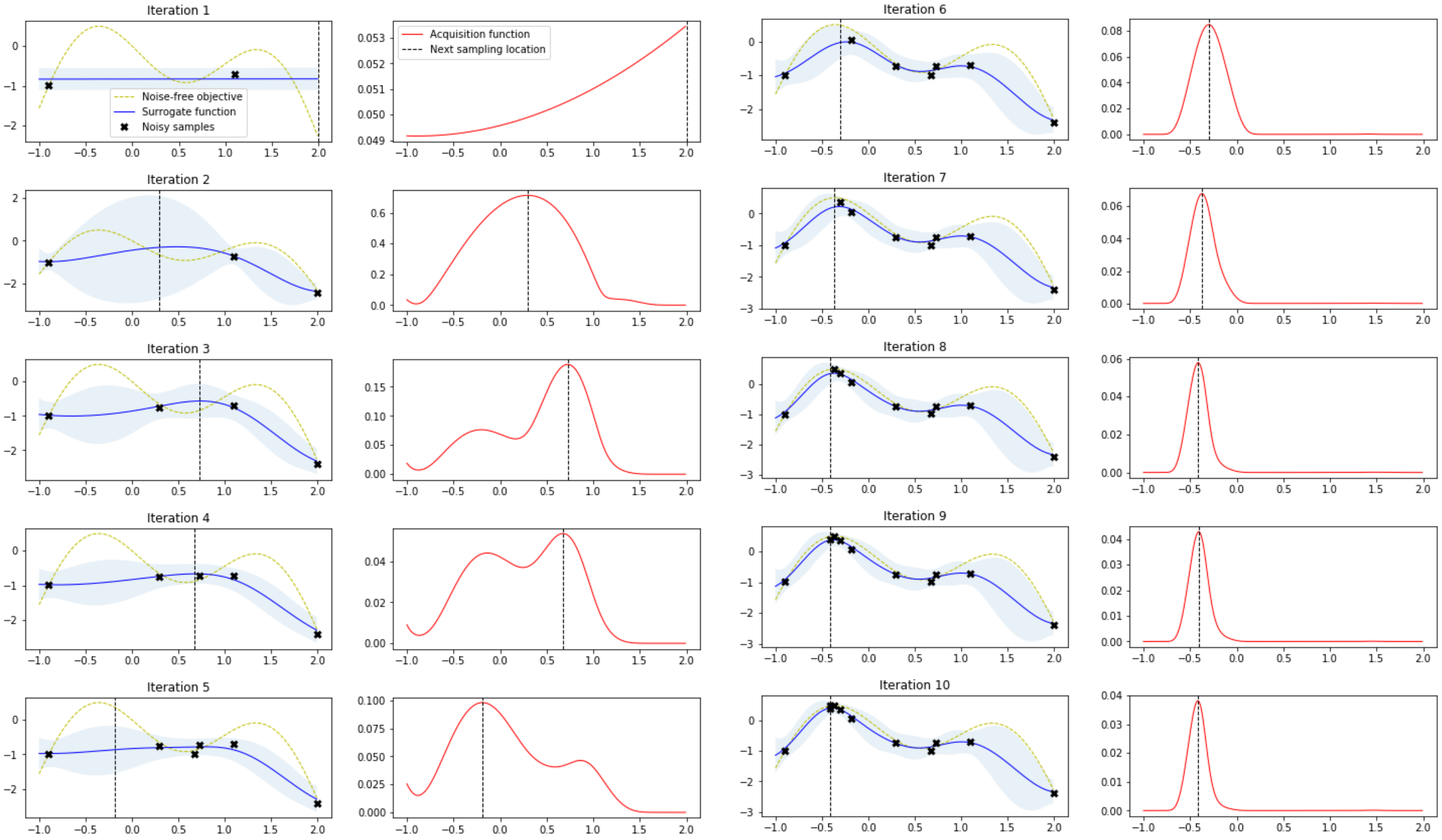
Dotted curve: True function
Green curve: Current surrogate of the function
Shaded region: Uncertainty in the function's estimate

A typical example of what $A(x)$ might look like, assuming that the goal is to find the maxima of $f(x)$

- Note: The regression model must also have estimate of function's uncertainty
  - Bayesian nonlinear regression, such as GP, Bayesian Neural network, etc would be ideal

Bayesian Optimization: An Illustration

- Suppose our goal is to find the <u>maxima</u> of $f(x)$ using BO



Pic source: http://krasserm.github.io/2018/03/21/bayesian-optimization/

Some Basic Acquisition Functions for BO

(assuming we are finding the minima)

Acquisition Functions: Probability of Improvement

- Assume past queries $\mathcal{D}_N = (\boldsymbol{X}, \boldsymbol{f}) = \left(x_n, f(x_n)\right)_{n=1}^{N}$ and suppose $\boldsymbol{f_{min}} = \min \boldsymbol{f}$
- Suppose f_{new} denotes the function's value at the next query point $\boldsymbol{x_{new}}$
- We have an improvement if $f_{new} < f_{min}$ (recall we are doing minimization)
- Assuming the function is real-valued, suppose the posterior predictive for $\boldsymbol{x_{new}}$ is

$$p(f_{new}|x_{new}, \mathcal{D}_N) = \mathcal{N}(f_{new}|\mu(x_{new}), \sigma^2(x_{new}))$$

- We can define a probability of improvement based acquisition function

Exercise: Verify

$$A_{PI}(x_{new}) = p(f_{new} \le f_{min}) = \int_{-\infty}^{f_{min}} \mathcal{N}(f_{new}|\mu(x_{new}), \sigma^2(x_{new}))df_{new} = \Phi\left(\frac{f_{min} - \mu(x_{new})}{\sigma(x_{new})}\right)$$

$\Phi()$ denotes CDF of $\mathcal{N}(0,1)$

- The optimal query point will be one that maximizes $A_{PI}(x_{new})$

$$x_* = \text{argmax}_{x_{new}} A_{PI}(x_{new})$$

Acquisition Functions: Expected Improvement

- PI doesn't take into account the amount of improvement
- Expected Improvement (EI) takes this into account and is defined as

$$A_{EI}(x_{new}) = \mathbb{E}[f_{min} - f_{new}] = \int_{-\infty}^{f_{min}} (f_{min} - f_{new})\mathcal{N}(f_{new}|\mu(x_{new}), \sigma^2(x_{new}))df_{new}$$

$$= (f_{min} - \mu(x_{new}))\Phi\left(\frac{f_{min} - \mu(x_{new})}{\sigma(x_{new})}\right) + \sigma(x_{new})\mathcal{N}\left(\frac{f_{min} - \mu(x_{new})}{\sigma(x_{new})}; 0,1\right)$$

Exercise: Prove this result

- The optimal query point will be one that maximizes $A_{EI}(x_{new})$

$$x_* = \text{argmax}_{x_{new}} A_{EI}(x_{new})$$

Focus on points where the function has small values (since we are looking for its minima)

Focus on points where the function has high uncertainty (so that including them improves our estimate of the function)

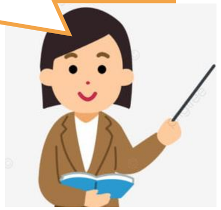
- Note that the above acquisition function trades off exploitation vs exploration
  - Will prefer points with small predictive mean $\mu(x_{new})$: Exploitation
  - Will prefer points with large predictive variance $\sigma(x_{new})$: Exploration

Acquisition Functions: Lower Confidence Bound

- Lower Confidence Bound (LCB) also takes into account exploitation vs exploration
- Used when the regression model is a Gaussian Process (GP)
- Assume the posterior predictive for a new point to be

$$p(f_{new}|x_{new}, \mathcal{D}_N) = \mathcal{N}(f_{new}|\mu(x_{new}), \sigma^2(x_{new}))$$

When using BO for <u>maximization</u>, we use Upper Confidence Bound (UCB) defined as $A_{UCB}(x_{new}) = \mu(x_{new}) + \kappa\,\sigma(x_{new})$ and $x_* = \text{argmax}_{x_{new}} A_{UCB}(x_{new})$



- The LCB based acquisition function is defined as

$$A_{LCB}(x_{new}) = \mu(x_{new}) - \kappa\,\sigma(x_{new})$$

Thus prefer points at which the function has low mean but high variance

- Point with the smallest LCB is selected as the next query point

$$x_* = \text{argmin}_{x_{new}} A_{LCB}(x_{new})$$

- κ is a parameter to trade-off exploitation (low mean) and exploration (high variance)
- Under certain conditions, the iterative application of this acquisition function will converge to the true global optima of f (Srinivas et al. 2010)

Bayesian Optimization: The Overall Algo

- Initialize $\mathcal{D} = \{\}$

- For $n = 1,2,\ldots,N$ (or until the budget doesn't exhaust)

  - Select the next query point x_n by optimizing the acquisition function

$$x_n = \text{argopt}_x\, A(x)$$

  - Get function's value from the black-box oracle: $f_n = f(x_n)$

  - $\mathcal{D} = \{\mathcal{D} \cup (x_n, f_n)\}$

Can get the function's minima from this set of function's values

  - Update the regression model for f using data $\mathcal{D}$

BO: Some Challenges/Open Problems

- Learning the regression model for the function
  - GPs are flexible but can be expensive as N grows
  - Bayesian neural networks can be an more efficient alternative to GPs (Snoek et al, 2015)
  - Hyperparams of the regression model itself (e.g., GP cov. function, Bayesian NN hyperparam)

- High-dimensional Bayesian Optimization (optimizing functions of many variables)
  - Most existing methods work well only for a moderate-dimensional x
  - Number of function evaluations required would be quite large in high dimensions
  - Lot of recent work on this (e.g., based on dimensionality reduction)

- Multitask Bayesian Optimization (joint BO for several related functions)
  - Basic idea: If two functions are similar their optima would also be nearby