Computational Systems Biology
Deep Learning in the Life Sciences

6.802 6.874 20.390 20.490 HST.506

David Gifford

Lecture 11

March 14, 2019

Characterizing Uncertainty
Experiment Planning

http://mit6874.github.io

What's on tap today!

- The prediction of uncertainty an its importance
  - Aleatoric – inherent observational noise
  - Epistemic – model uncertainty
- How to predict uncertainty
  - Gaussian Processes
  - Ensembles
- Using uncertainty
  - Bayesian optimization
  - Experiment Design

What you should know

- Aleatoric and epistemic uncertainty and their prediction
- Gaussian processes
- Bayesian optimization and acquisition functions
  - Upper/Lower confidence bound
  - Expected improvement

Why uncertainty?
(Why not?)

Why quantify the uncertainty of model predictions

- In self-driving cars, if model is uncertain about predictions from visual data, other sensors may be used to improve situational awareness
- In healthcare, if an AI system is uncertain about a decision, one may want to transfer control to a human doctor
- If a model is very sure about a particular drug helping with a condition and less sure about others, you want to go with the first drug

An example of experiment design

- We a model f of the binding of a transcription factor to 8-mer DNA sequences.
  - Binding = f(8-mer sequence)
  - Assume given training data { (s_1, b_1), (s_2, b_2) … (s_n, b_n) } to train f
  - Goal is to discover s_{best} = argmax f(s)
  - Need best model f; what the next next s_x we should ask to observe?
- What is a principled way to choose s_x ?

Experiment design explores the space where a model is uncertain

- *Explore* the space more to *improve* your model as well (in addition to exploiting existing guesses)
- You want to explore the space where your model is not confident about being right – hence uncertainty quantification!
- We can quantify uncertainty with probability for discrete outputs or a standard deviation for continuous outputs
  - P(label | features) Classification
  - (μ, σ^2) = f(input) Regression – Normal distribution parameters

One metric of uncertainty for a given input is entropy for categorical labels

- Suppose we have a multiclass classification problem

- Intuitively, the more uniformly distributed the predicted probability over the different classes, the more uncertain the prediction

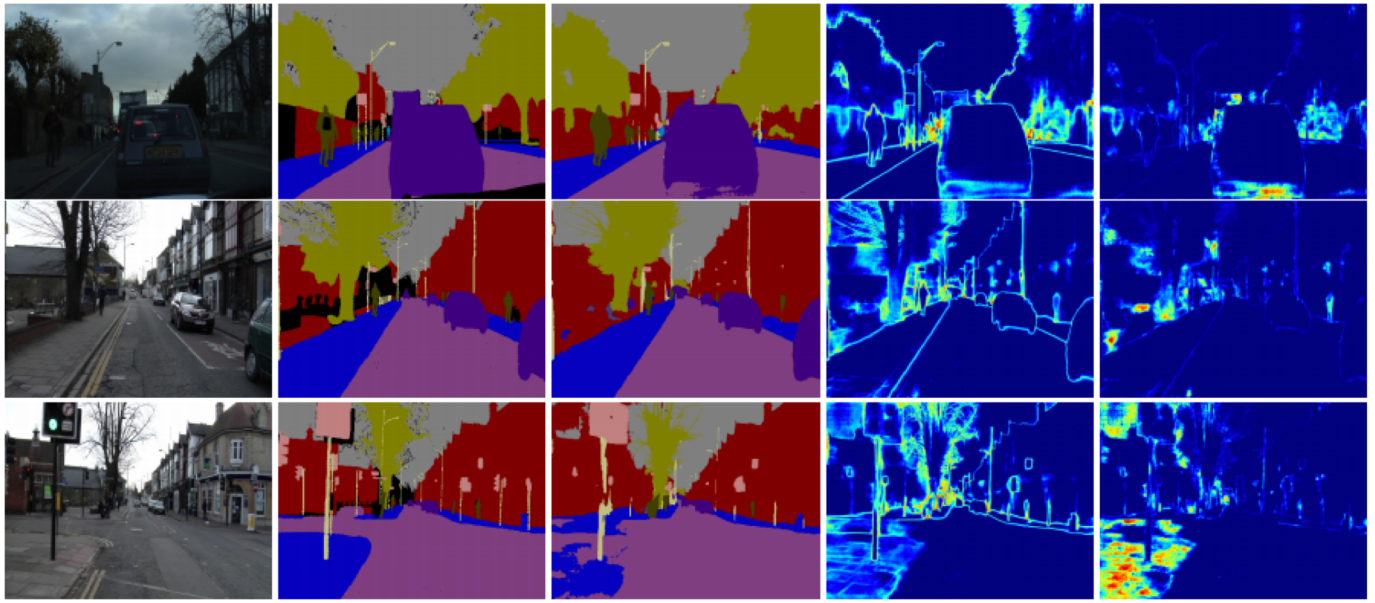
- Formally we can use *information entropy* to quantify uncertainty

$$S = -\sum_i P_i \log P_i$$

There are two types of uncertainty

- Aleatoric (experimental) uncertainty
  - Examples
    - Human error in labeling image categories
    - Noise in biological systems – TF binding to DNA is stochastic
  - Source is the *unmeasured* unknowns that can change every time we repeat an experiment
- Epistemic (model) uncertainty
  - Examples
    - Different hypothesis for why sun moves in the sky (geocentric vs heliocentric)
    - Uncertainty about which factor played a role in determining an election result
  - Represents the different models that fit the training data equally well but generalize differently inducing uncertainty on the test data
  - More training data can reduce epistemic uncertainty

In vision aleatoric uncertainty is seen at edges; epistemic in objects

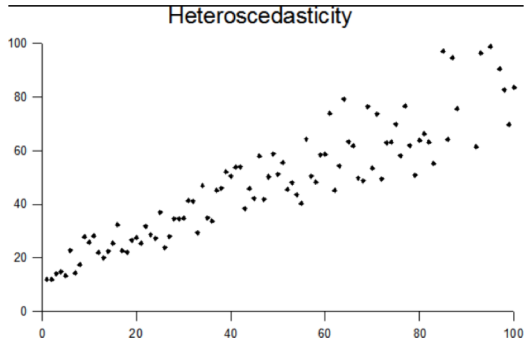


(a) Input Image | (b) Ground Truth | (c) Semantic Segmentation | (d) Aleatoric Uncertainty | (e) Epistemic Uncertainty

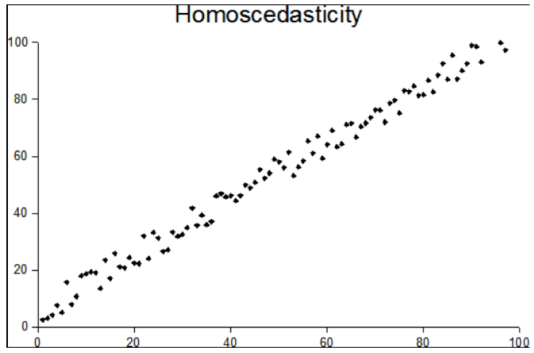
For (d), (e), Dark blue is lower uncertainty, lighter blue is higher uncertainty, and yellow -> red is the highest uncertainty

Aleatoric uncertainty can be constant or change with the output

- Heteroscedastic noise
  - Changes with the label



- Homoscedastic noise
  - Does not change with label



Modeling aleatoric uncertainty

$$y = f(x) + \epsilon$$

- Homoscedastic noise

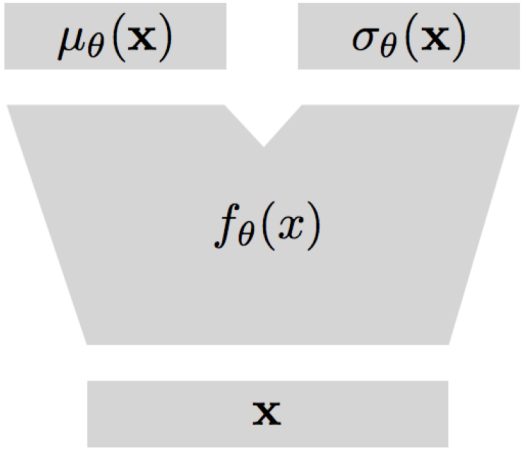
$$\epsilon \sim N(0, 1)$$

- Heteroscedastic noise

$$\epsilon \sim N(0, g(x))$$

- Other popular noise distributions – Poisson, Laplace, Negative Binomial, Gamma, etc.

A “two headed” network can predict aleatoric uncertainty



$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N} \log \mathcal{N}(y_i; \mu_\theta(x_i), \sigma_\theta^2(x_i))$$

$$\mathcal{L}_{BNN}(\theta) = \frac{1}{D}\sum_i \frac{1}{2}\exp(-s_i)||\mathbf{y}_i - \hat{\mathbf{y}}_i||^2 + \frac{1}{2}s_i.$$

Predict $s_i = \log(\sigma^2)$ to avoid divide by zero issues

Confidence intervals

- Intuitively, an interval around the prediction that could contain the true label.

- An X% confidence interval means that for independent and identically distributed (IID) data, X% of the future samples will fall within the interval.

Visualizing uncertainty quantification

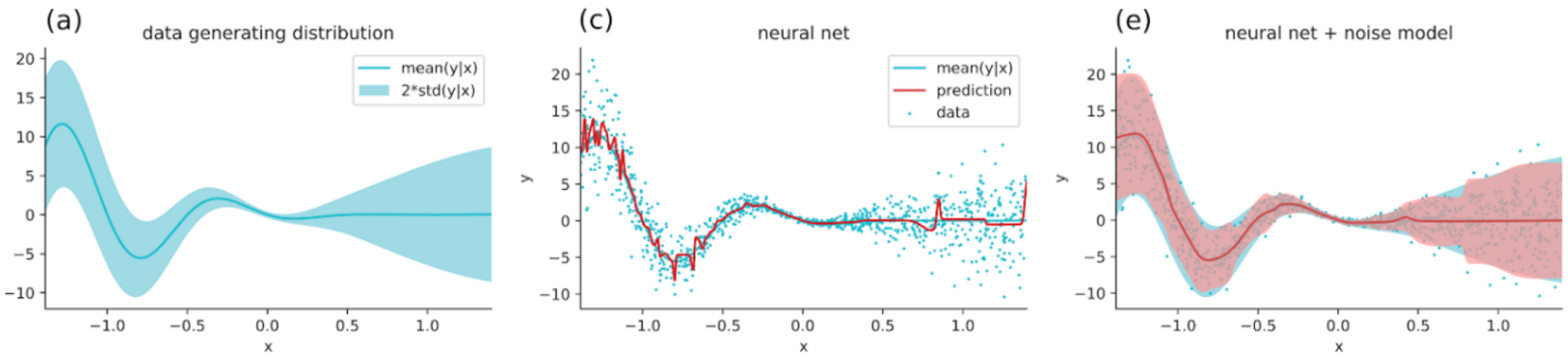
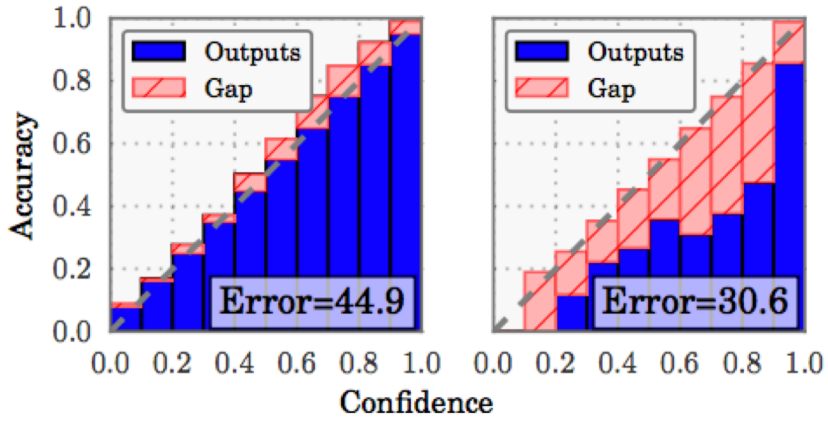


Figure taken from https://medium.com/capital-one-tech/reasonable-doubt-get-onto-the-top-35-mnist-leaderboard-by-quantifying-aleatoric-uncertainty-a8503f134497

How do we tell if uncertainties are right?

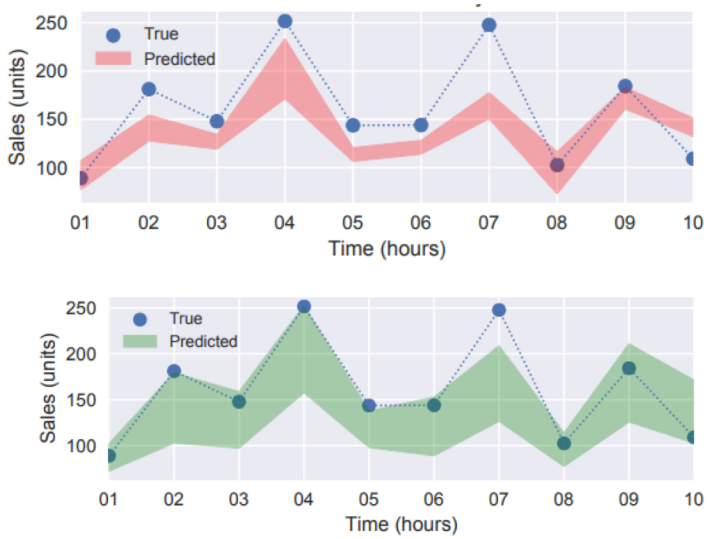
- A *well-calibrated* model produces uncertainty predictions that match held out data
- Classification
  - If we only look at predictions where the probability of a class is 0.3, they should be correct 30% of the time



Error indicates the overall network accuracy

How do we tell if uncertainties are right?

- Regression
  - Compute *confidence intervals* for each prediction
  - 90% of the observations should fall within the 90% intervals, 80% of observations in the 80% interval and so on



Proper scoring rules

- Suppose an algorithm outputs a probability distribution over targets, and gets a loss based on this distribution and the true target.
- A proper scoring rule is a scoring rule where the algorithm's best strategy is to output the true distribution.
- The canonical example is negative log-likelihood (NLL). If k is the category label, $\mathbf{t}$ is the indicator vector for the label, and $\mathbf{y}$ are the predicted probabilities,

$$L(\mathbf{y}, \mathbf{t}) = -\log y_k = -\mathbf{t}^\top (\log \mathbf{y})$$

Slide from https://csc2541-f17.github.io/slides/lec01.pdf

Reasons for miscalibration

- NNs are high capacity. Once they get close to 100% accuracy on the training data, they are incentivized to get the uncertainty down as well (note that the NLL/cross-entropy loss includes both accuracy and uncertainty terms).

Recalibration

- Guo et al. explored 7 different calibration methods, but the one that worked the best was also the simplest: temperature scaling.
- A classification network typically predicts $\sigma(\mathbf{z})$, where σ is the softmax function

$$\sigma(\mathbf{z})_k = \frac{\exp(z_k)}{\sum_{k'} \exp(z_{k'})}$$

  and $\mathbf{z}$ are called the logits.
- They replace this with

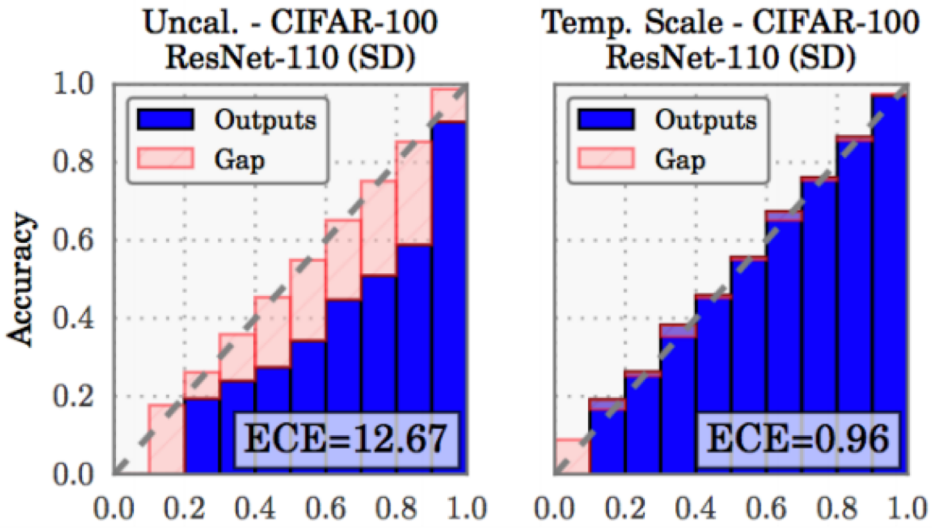
$$\sigma(\mathbf{z}/T),$$

  where T is a scalar called the temperature.
- T is tuned to minimize the NLL on a validation set.
- Intuitively, because NLL is a proper scoring rule, the algorithm is incentivized to match the true probabilities as closely as possible.

Recalibration

- ECE – Expected calibration error – area between calibration curve (line formed by blue histograms) and diagonal

- Before and after temperature scaling:

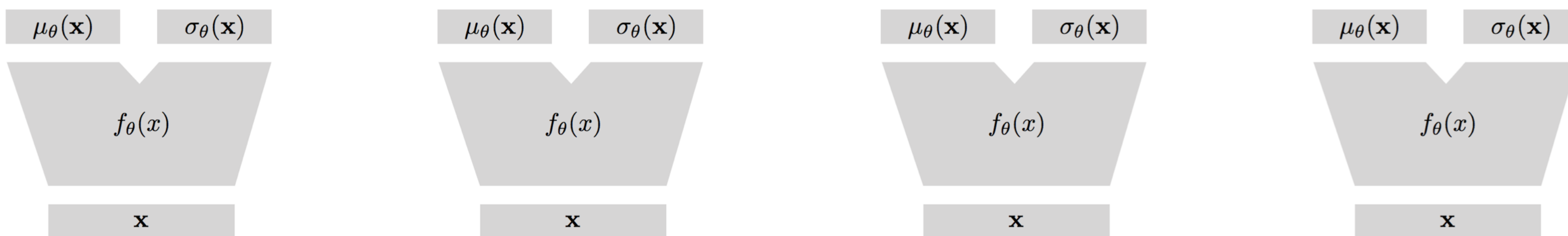


Modeling epistemic uncertainty

- Somehow need to define a space of models – called a *hypothesis* space and assign probabilities to each model

- *Bayesian* modeling is a principled way to assign probabilities to models in a hypothesis space

Epistemic uncertainty quantification with an ensemble of different networks or networks trained on different data



$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N} \log \mathcal{N}(y_i; \mu_\theta(x_i), \sigma_\theta^2(x_i))$$

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N} \log \mathcal{N}(y_i; \mu_\theta(x_i), \sigma_\theta^2(x_i))$$

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N} \log \mathcal{N}(y_i; \mu_\theta(x_i), \sigma_\theta^2(x_i))$$

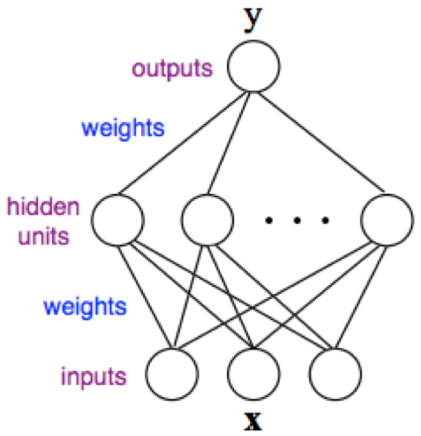
$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N} \log \mathcal{N}(y_i; \mu_\theta(x_i), \sigma_\theta^2(x_i))$$

Epistemic uncertainty - $Var(\mu_{\theta_i}((x)))$

Uncertainly predictions using dropout

- Idea is to randomly drop some fraction of the neurons at prediction time
- Gives an empirical distribution over predictions
- The empirical standard deviation (proportional to entropy in case of Gaussian distribution) is a measure of uncertainty
- Tends to be extremely overconfident

Uncertainty can be produced in a single network by making parameters uncertain – Bayesian Neural Nets



Bayesian neural network

Data: $\mathcal{D} = \{(\mathbf{x}^{(n)}, y^{(n)})\}_{n=1}^{N} = (X, \mathbf{y})$

Parameters $\boldsymbol{\theta}$ are weights of neural net

prior $\quad p(\boldsymbol{\theta}|\boldsymbol{\alpha})$

posterior $\quad p(\boldsymbol{\theta}|\boldsymbol{\alpha}, \mathcal{D}) \propto p(\mathbf{y}|X, \boldsymbol{\theta})p(\boldsymbol{\theta}|\boldsymbol{\alpha})$

prediction $\quad p(y'|\mathcal{D}, \mathbf{x}', \boldsymbol{\alpha}) = \int p(y'|\mathbf{x}', \boldsymbol{\theta})p(\boldsymbol{\theta}|\mathcal{D}, \boldsymbol{\alpha})\, d\boldsymbol{\theta}$

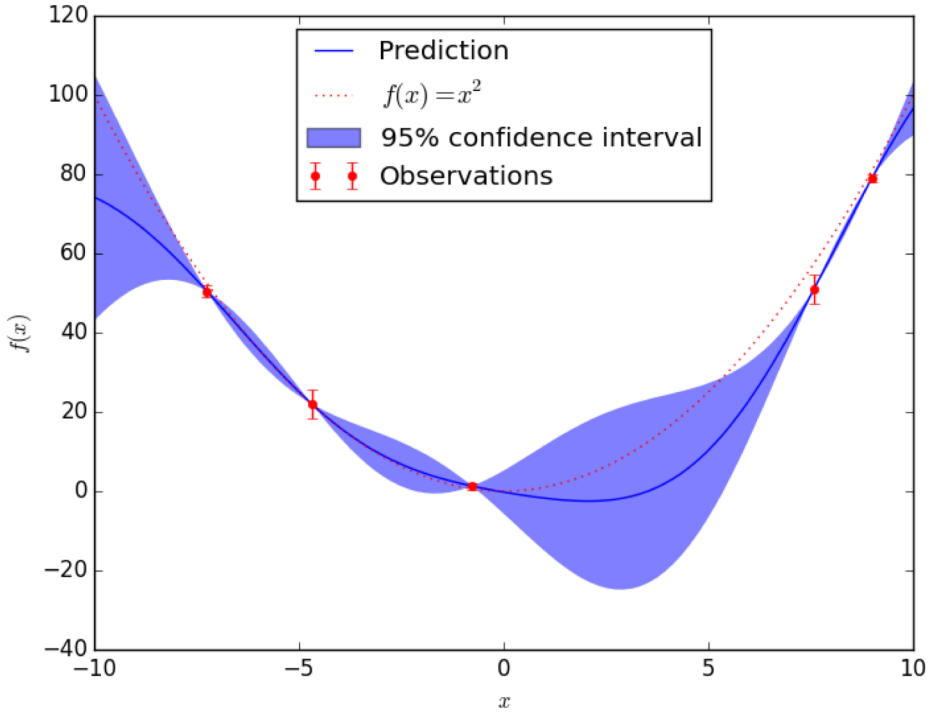
Bayesian NN Advantages/Disadvantages

- Advantages
  - Principled Bayesian approach for deep neural networks

- Disadvantages
  - Tend to be overconfident
  - Common approaches to do inference are expensive
  - While in principle, arbitrary aleatoric noise distributions can be used, in practice, that makes inference even more expensive

Gaussian processes are predictive models that represent uncertainty with a closed form solution

Prediction f* is represented by a multivariate normal



Gaussian process

A Gaussian process defines a distribution over functions, $p(f)$, where f is a function mapping some input space $\mathcal{X}$ to $\Re$.

Notice that f can be an infinite-dimensional quantity (e.g. if $\mathcal{X} = \Re$)

Let $\mathbf{f} = (f(x_1), \ldots, f(x_n))$ be an n-dimensional vector of function values evaluated at n points $x_i \in \mathcal{X}$. Note $\mathbf{f}$ is a random variable.

Definition: $p(f)$ is a Gaussian process if for *any* finite subset $\{x_1, \ldots, x_n\} \subset \mathcal{X}$, the marginal distribution over that finite subset $p(\mathbf{f})$ has a multivariate Gaussian distribution.

Slide from http://mlss2011.comp.nus.edu.sg/uploads/Site/lect1gp.pdf

We can compute the covariance matrix for our unobserved test points

To generate functions we generate random Gaussian vectors with a covariance matrix defined by your input points:

$$\mathbf{f}_* \sim \mathcal{N}(\mathbf{0}, K(X_*, X_*))$$

Slide from http://courses.media.mit.edu/2010fall/mas622j/ProblemSets/slidesGP.pdf

Gaussian process solution can be placed in matrix form

Since we are interested in making predictions more than just generating functions we want to incorporate the knowledge of training data. For a GP with zero mean and covariance function, $K(X, X) + \sigma_n{}^2 I$ the join distribution of training outputs and test outputs is:

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{f}_* \end{bmatrix} \sim \mathrm{N}(\mathbf{0}, \begin{bmatrix} K(X,X) + \sigma_n^2 I & K(X,X_*) \\ K(X_*,X) & K(X_*,X_*) \end{bmatrix})$$

$\sigma_n{}^2 I$ is the aleatoric noise component

Gaussian process exact solution is $O(n^3)$ in number of training points (note matrix inversion)

The predictive equations for GP regression:

$$\mathbf{f}_*|X, \mathbf{y}, X_* \sim \mathcal{N}(m(\mathbf{x}), covf)$$

Where:

$$m(\mathbf{x}) = K(X_*, X)[K(X, X) + \sigma_n^2 I]^{-1}\mathbf{y}$$

$$covf = K(X_*, X_*) - K(X_*, X)[K(X, X) + \sigma_n^2 I]^{-1}K(X, X_*)$$

At the core of a Gaussian Process is the Covariance matrix (Similarity matrix)

- Think of it as the *similarity* matrix between two points

$$K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$$

$$\mathrm{K}_{xx} = \begin{bmatrix} k(x_1, x_1) & k(x_1, x_2) & \dots & k(x_1, x_n) \\ k(x_2, x_1) & k(x_1, x_1) & \dots & k(x_2, x_n) \\ \dots & \dots & \dots & \dots \\ k(x_n, x_1) & k(x_n, x_2) & \dots & k(x_n, x_n) \end{bmatrix}$$

The squared exponential is a common covariance function

Commonly used covariance function:

$$K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j) = \alpha exp\{-\frac{||\mathbf{x}_i - \mathbf{x}_j||^2}{2\sigma_1^2}\}$$

Hyperparameters:

- σ_1 = Characteristic lengthscale
- α = Signal variance

Large for inputs that are close to one another. Decreases are distances in the input space increases.

Gaussian Processes - Advantages/Disadvantages

- Advantages
  - Closed form for the posterior distribution
  - Can easily adapt to new training data
  - Covariance function can be flexibly parameterized by ex. neural networks
  - Uncertainties are usually well calibrated
- Disadvantages
  - Scale cubically with the number of training points (though a lot of recent work trying to that down to a linear scaling)
  - Closed form limited to gaussian output noise
  - Can be adapted for classification but not easy to train as there is no closed form either for that case
  - Need a lot of data to cover the entire input space well

How can we use uncertainty to do experiment design?

Experiment Design - Bayesian optimization

- Suppose we have a black box function we want to optimize. Example -
  - Find a sequence that best binds to a TF
  - Find an airplane wing design that gives the most lift
  - How to tune hyperparameters of a neural network automatically
  - Optimize web design to maximize purchases
  - Find an antibody that best binds to a target

What to do?

Option 1: Use previous knowledge

To select the parameters at hand. Perhaps not very scientific but still in use...

Slide from http://gpss.cc/gpmc17/slides/LancasterMasterclass_1.pdf

What to do?

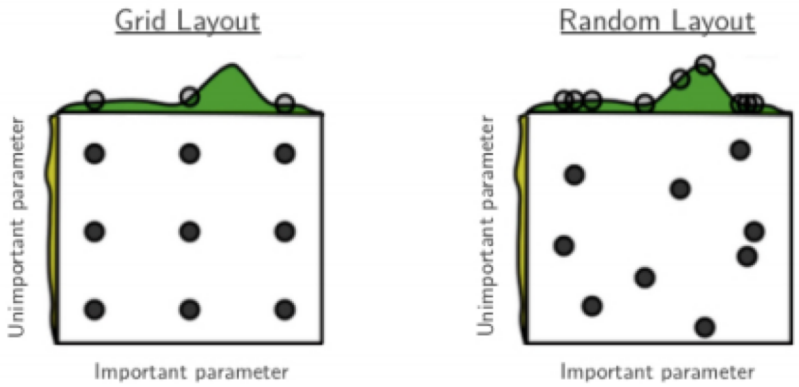
- Grid search?

- Extremely expensive!

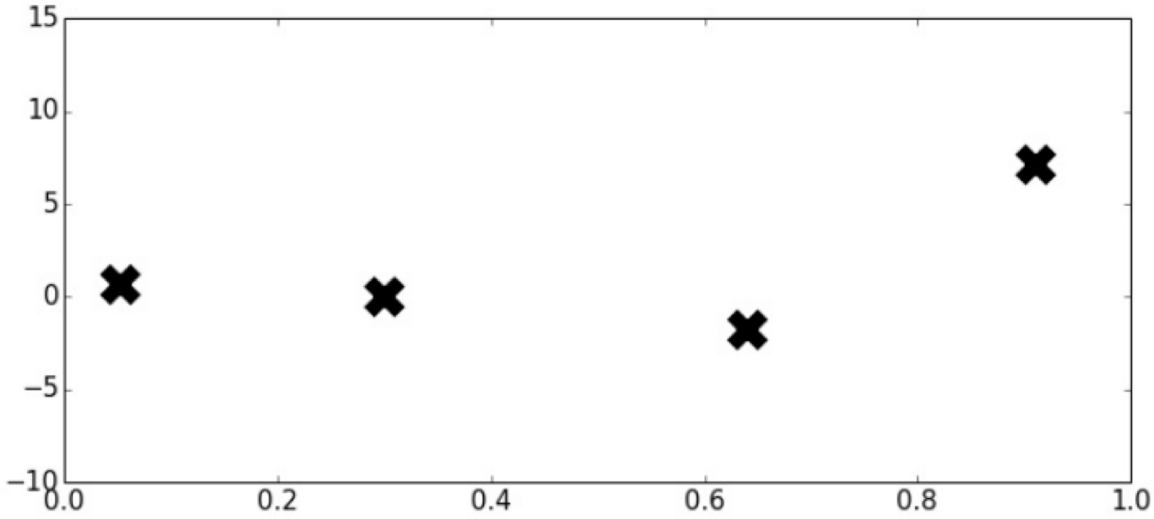
- But still used sometimes when # of parameters is small – example to tune NN hyperparameters

Randomized grid search has advantages over uniform grid search

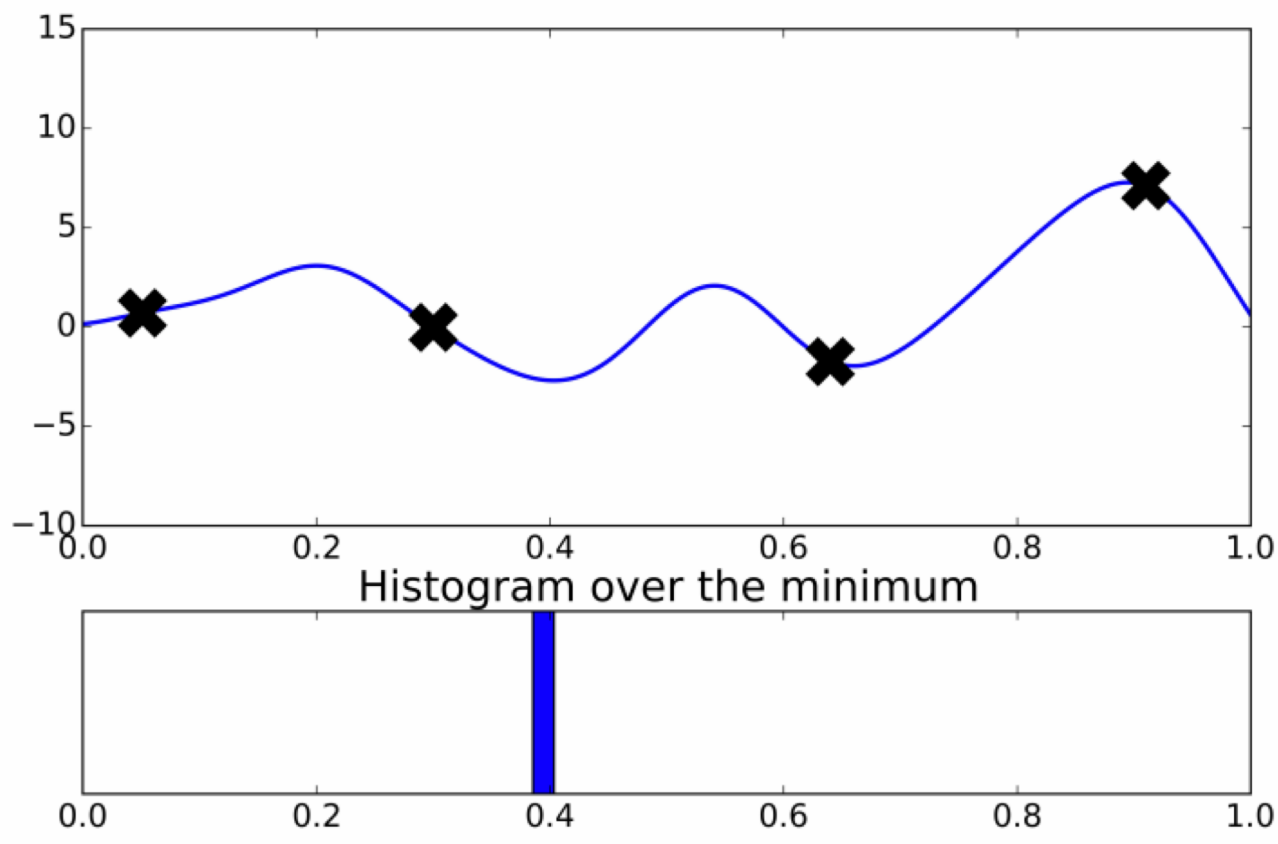
We can sample the space uniformly [Bergstra and Bengio 2012]



Better than grid search in various senses but still expensive to guarantee good coverage.

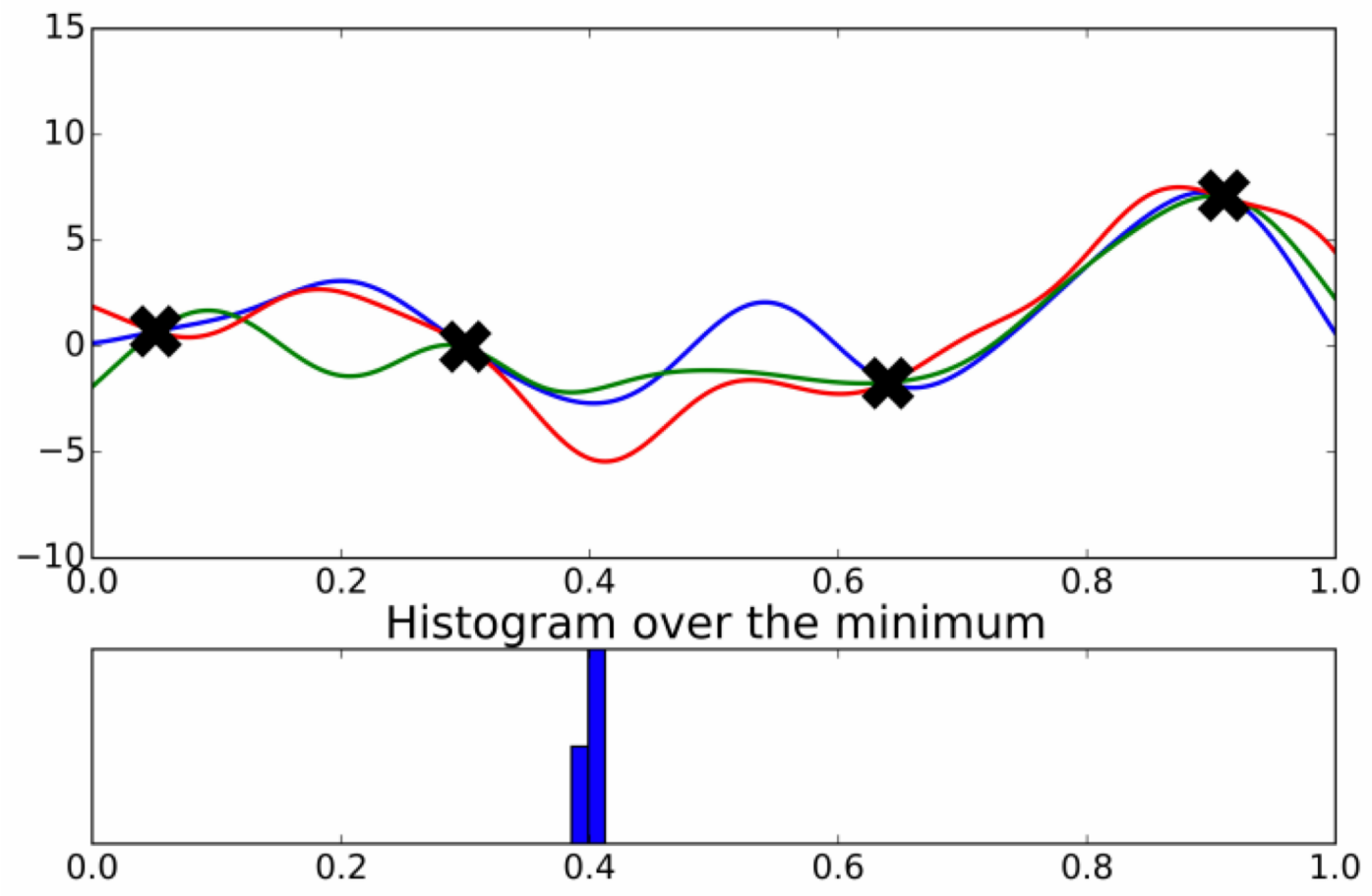


Where is the minimum of f?
Where should the take the next evaluation?


15
10
5
0
−5
−10
0.0
0.2
0.4
0.6
0.8
1.0
Histogram over the minimum
0.0
0.2
0.4
0.6
0.8
1.0

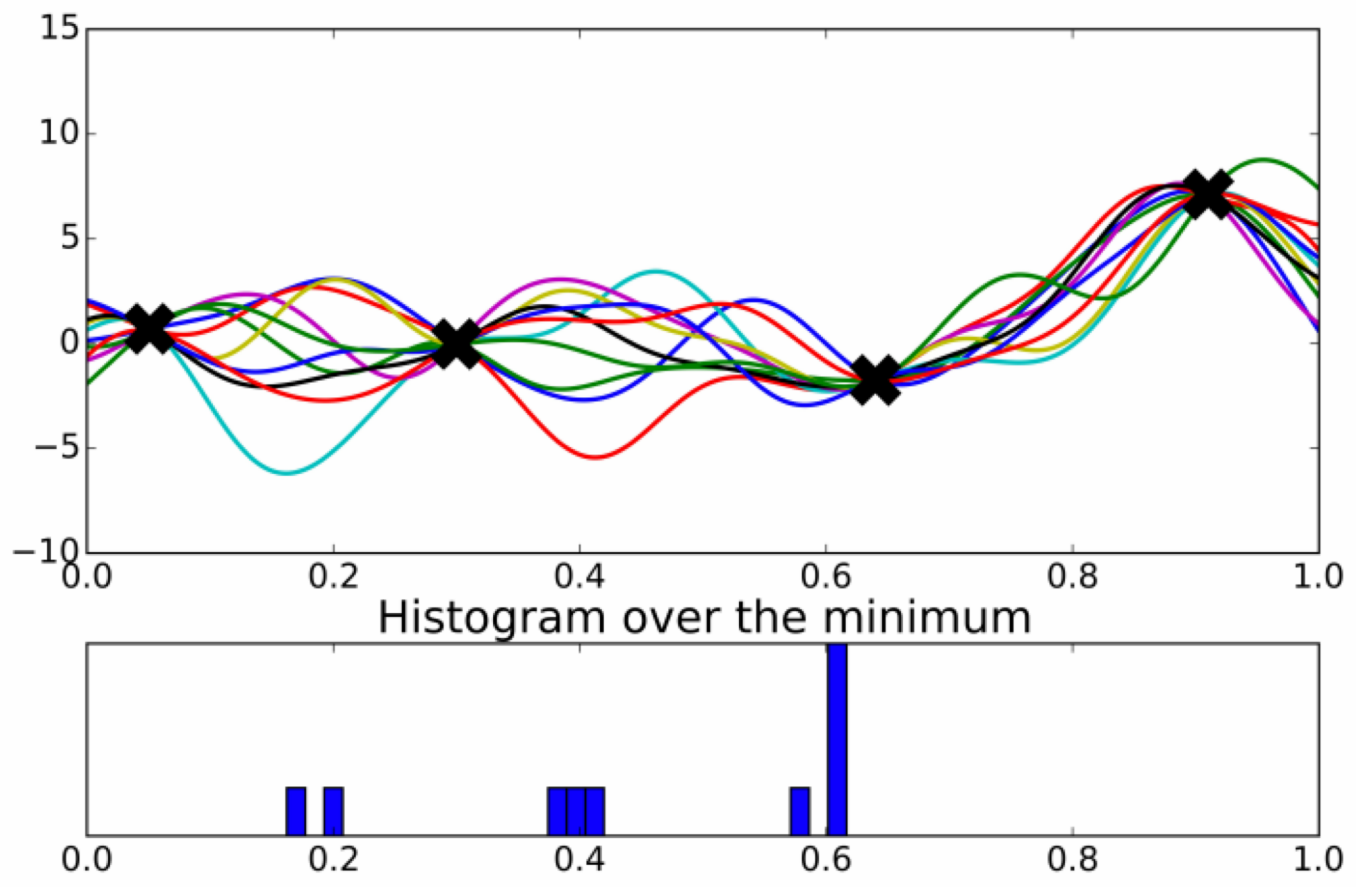
Intuitive solution

Three curves



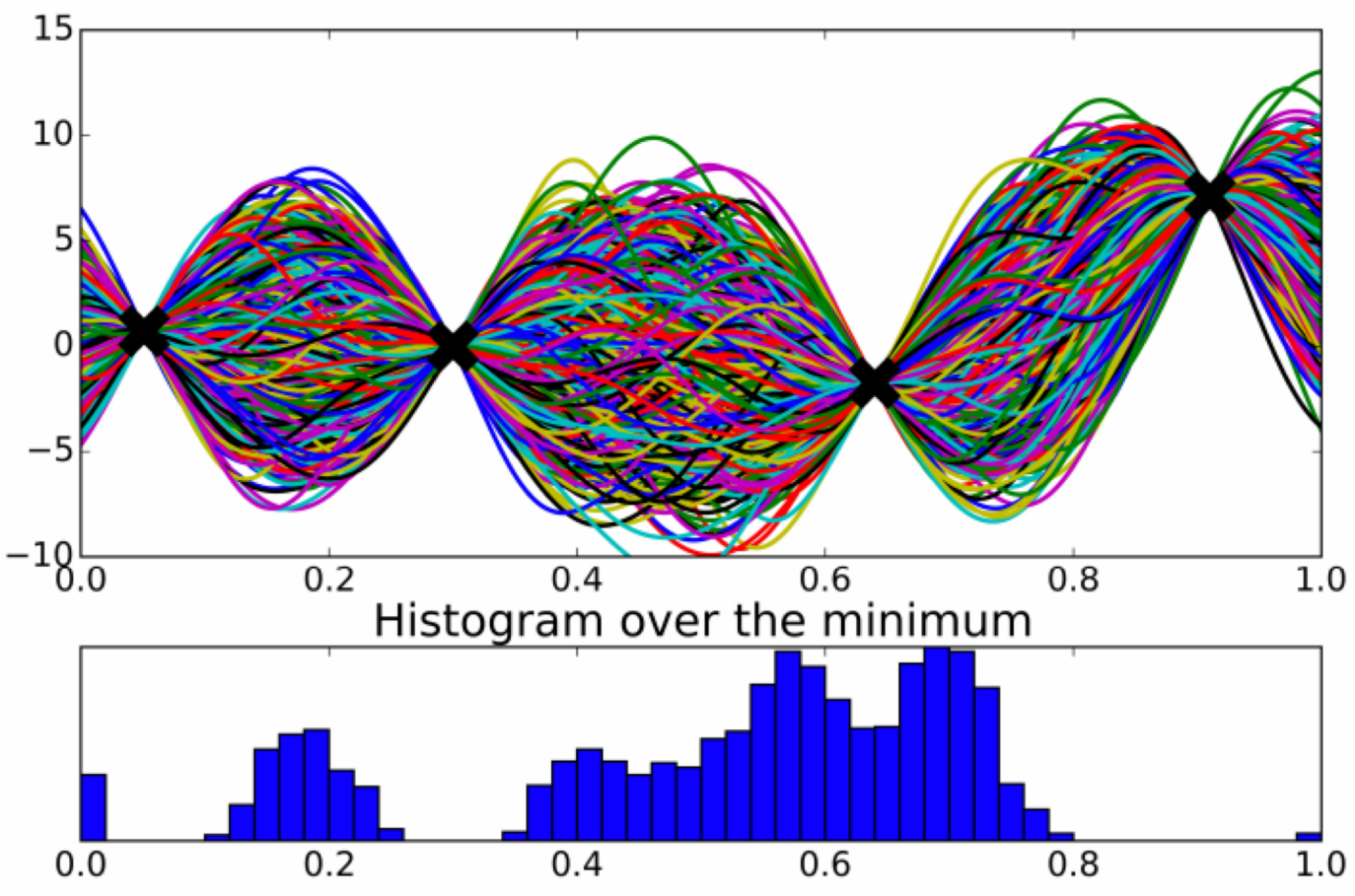
Intuitive solution

Ten curves



Intuitive solution

Many curves



General idea: surrogate modelling

1. Use a surrogate model of f to carry out the optimization.

2. Define an utility function to collect new data points satisfying some optimality criterion: *optimization* as *decision*.

3. Study *decision* problems as *inference* using the surrogate model: use a probabilistic model able to calibrate both, epistemic and aleatoric uncertainty.

Uncertainty Quantification

Utility functions

The utility should represent our design goal:.

1. Active Learning and experimental design: Maximize the differential entropy of the posterior distribution $p(f|X, y)$ (D-optimality in experimental design).

2. Minimize the loss in a sequence $x_1, \ldots, x_n$

$$r_N = \sum_{n=1}^{N} f(x_n) - N f(x_M)$$

(1) does to a lot exploration whereas (2) encourages exploitation about the minimum of f.

- x_M is the best point seen so far

Bayesian Optimisation

[Mockus, 1978]

Methodology to perform global optimisation of multimodal black-box functions.

1. Choose some *prior measure* over the space of possible objectives f.

2. Combine prior and the likelihood to get a *posterior measure* over the objective given some observations.

3. Use the posterior to decide where to take the next evaluation according to some *acquisition/loss function*.

4. Augment the data.

Iterate between 2 and 4 until the evaluation budget is over.

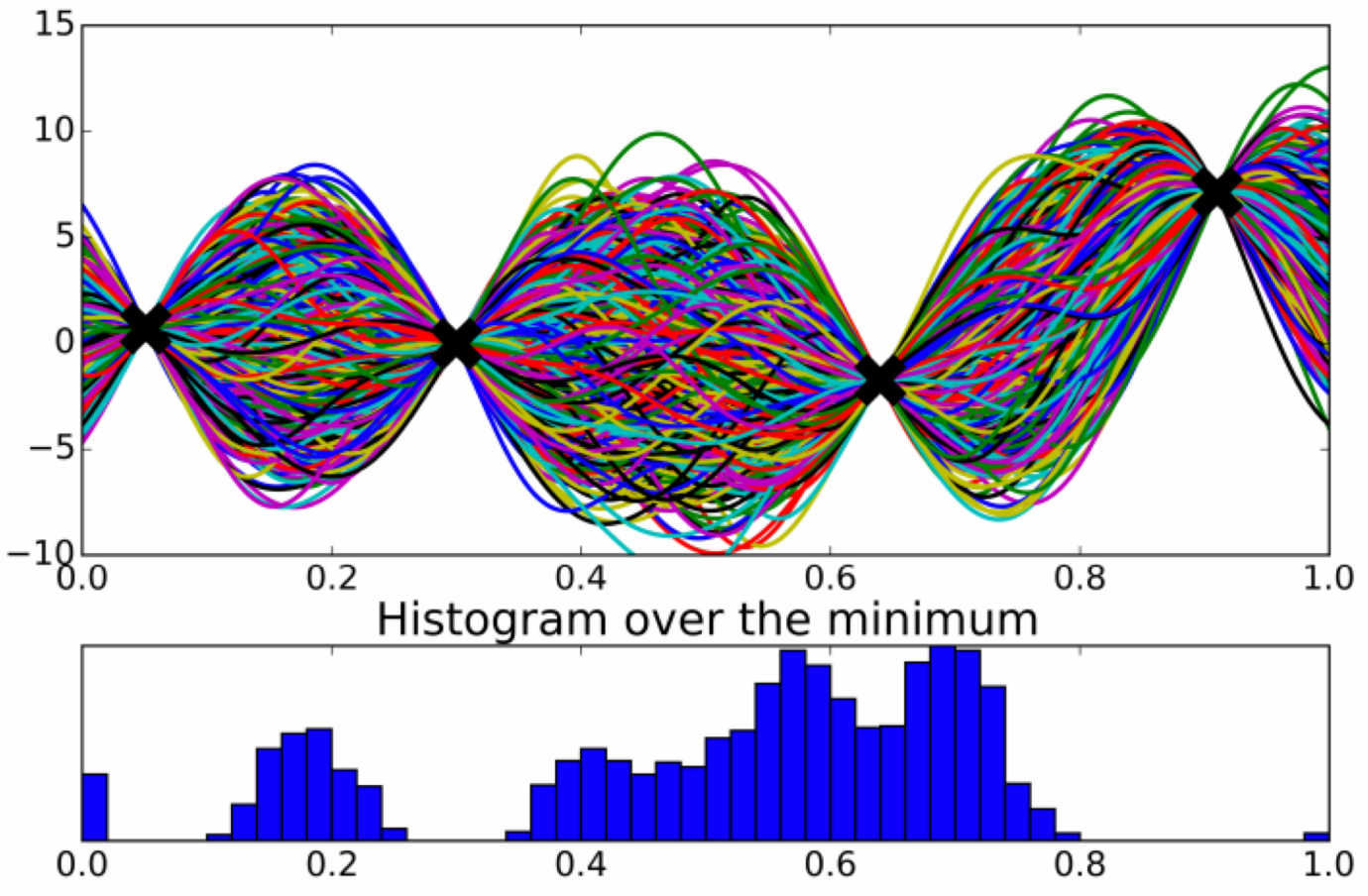
Possible models to represent uncertainty that we can optimize

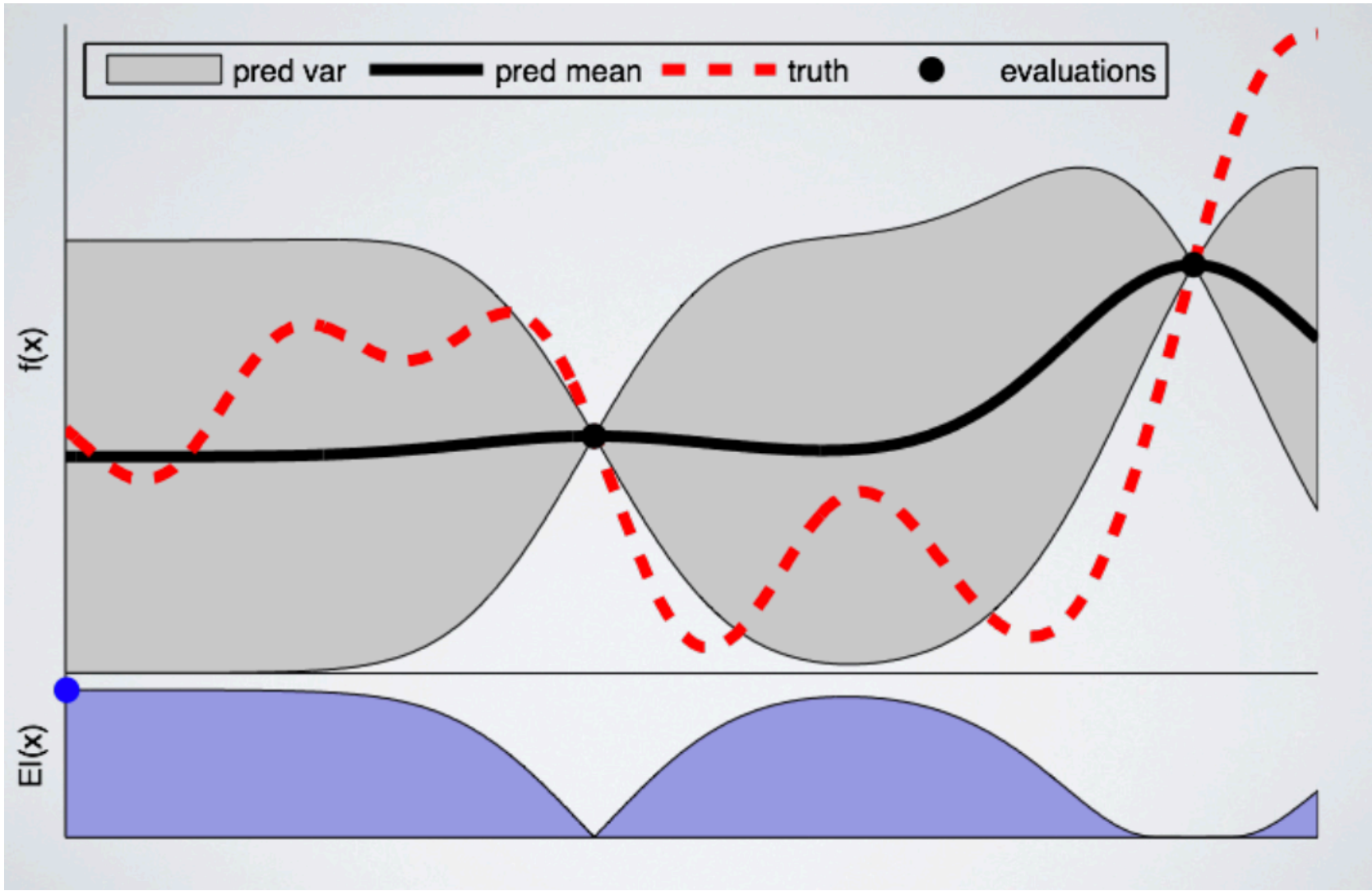
- Gaussian processes
  - Most popular
- Bayesian neural networks
- Random forests
- Deep ensembles
- etc

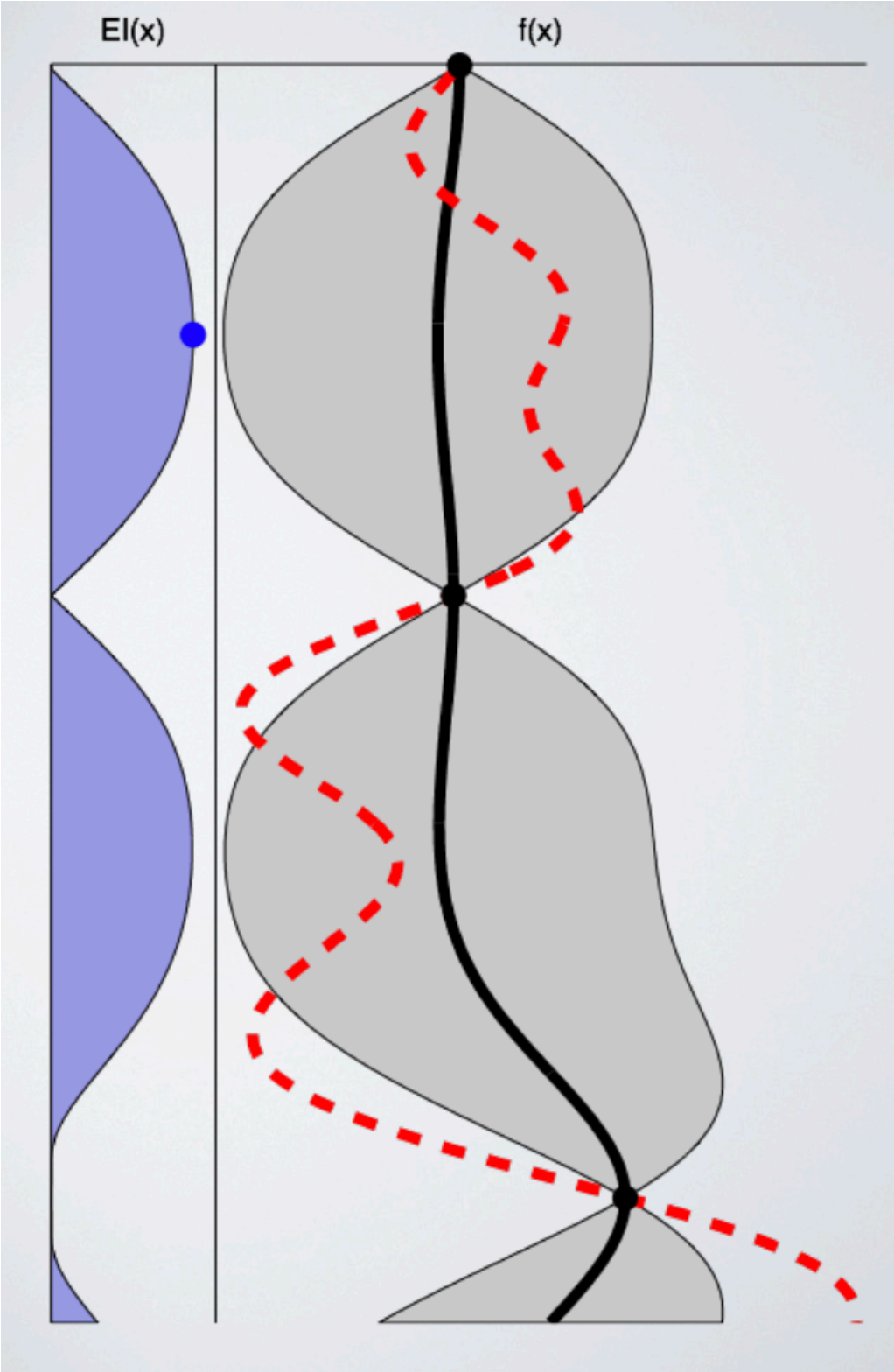
Exploration vs. exploitation

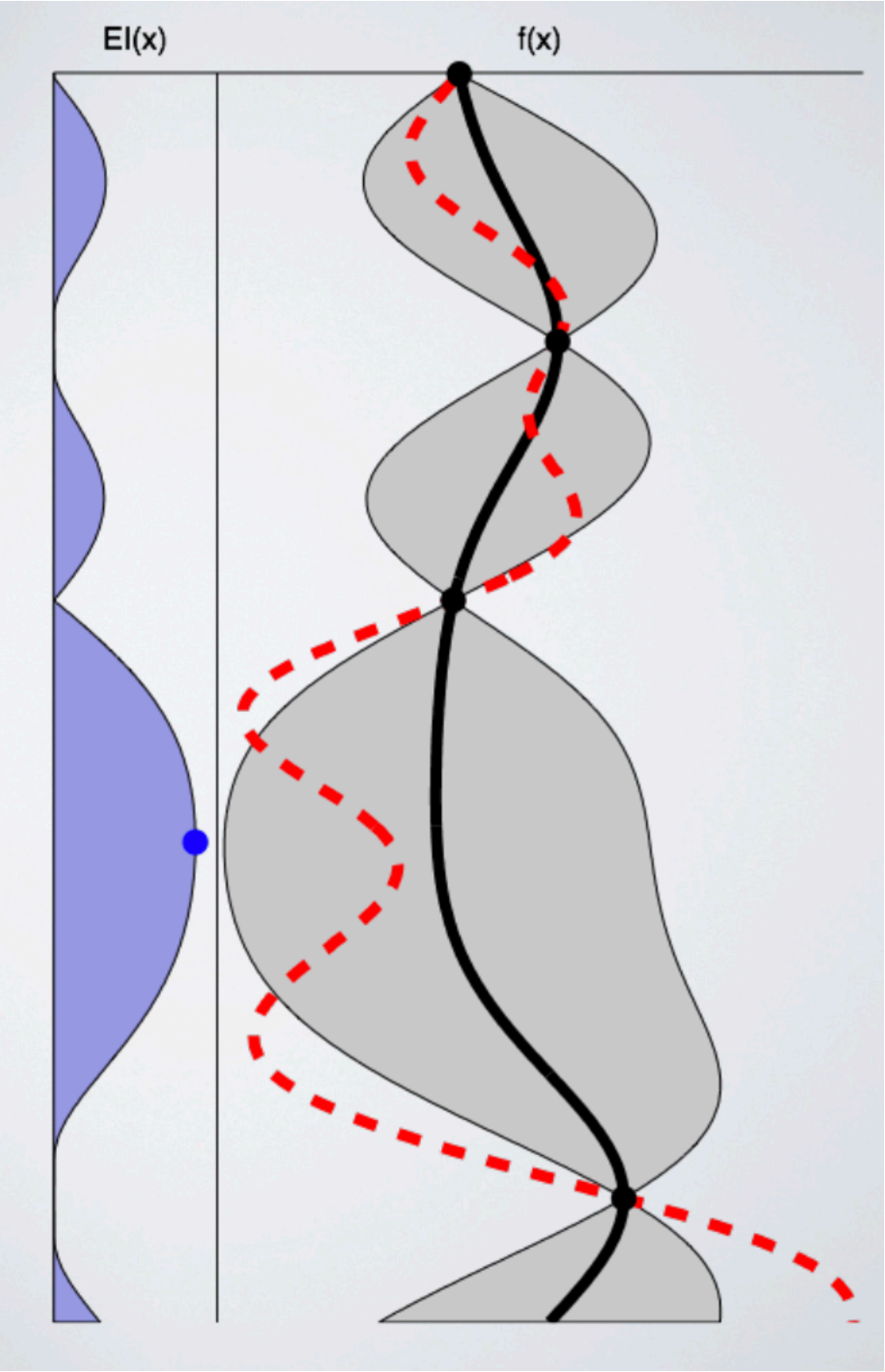
Goal – minimize function

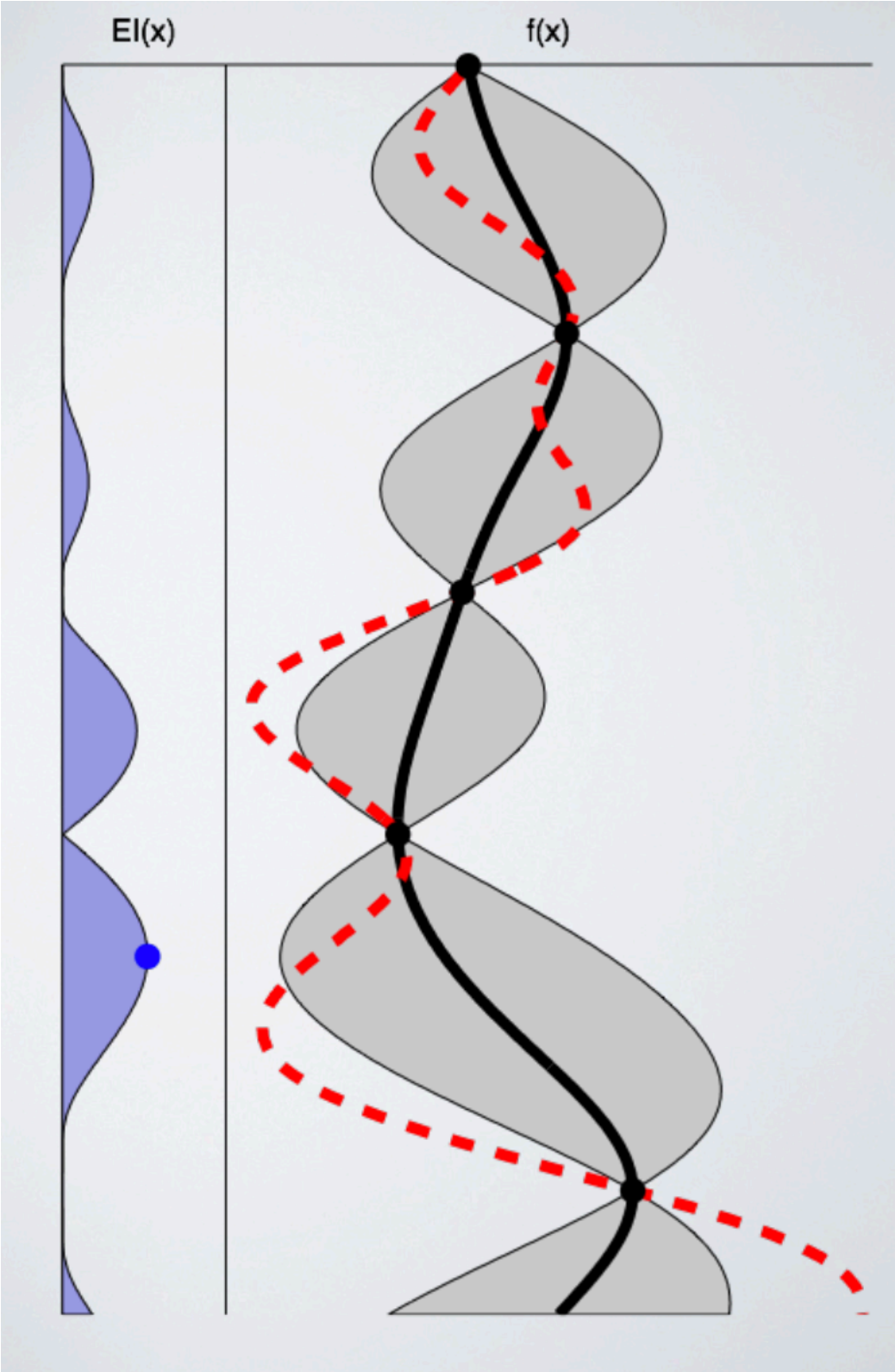
Recall family of functions to define uncertainty

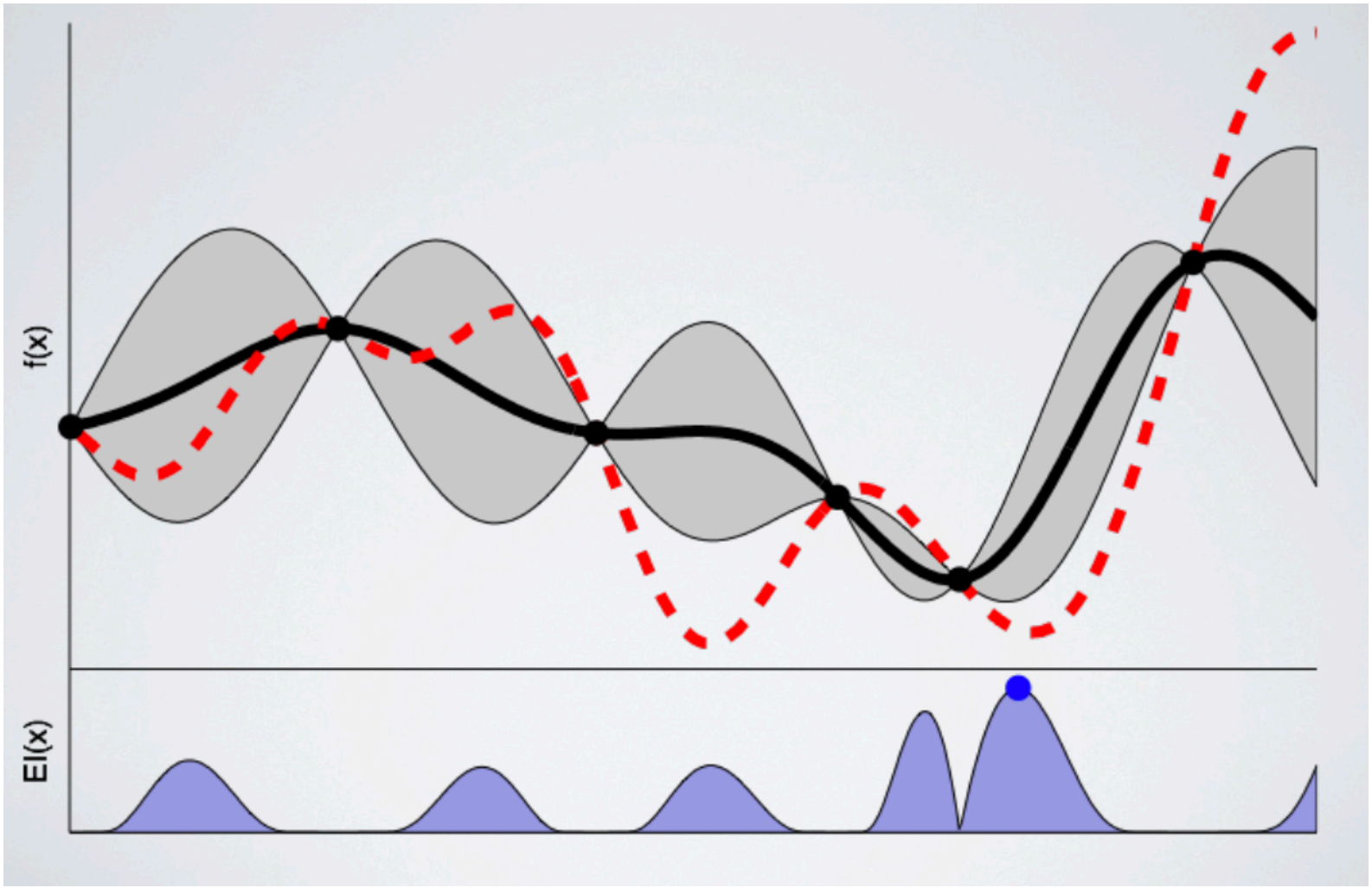


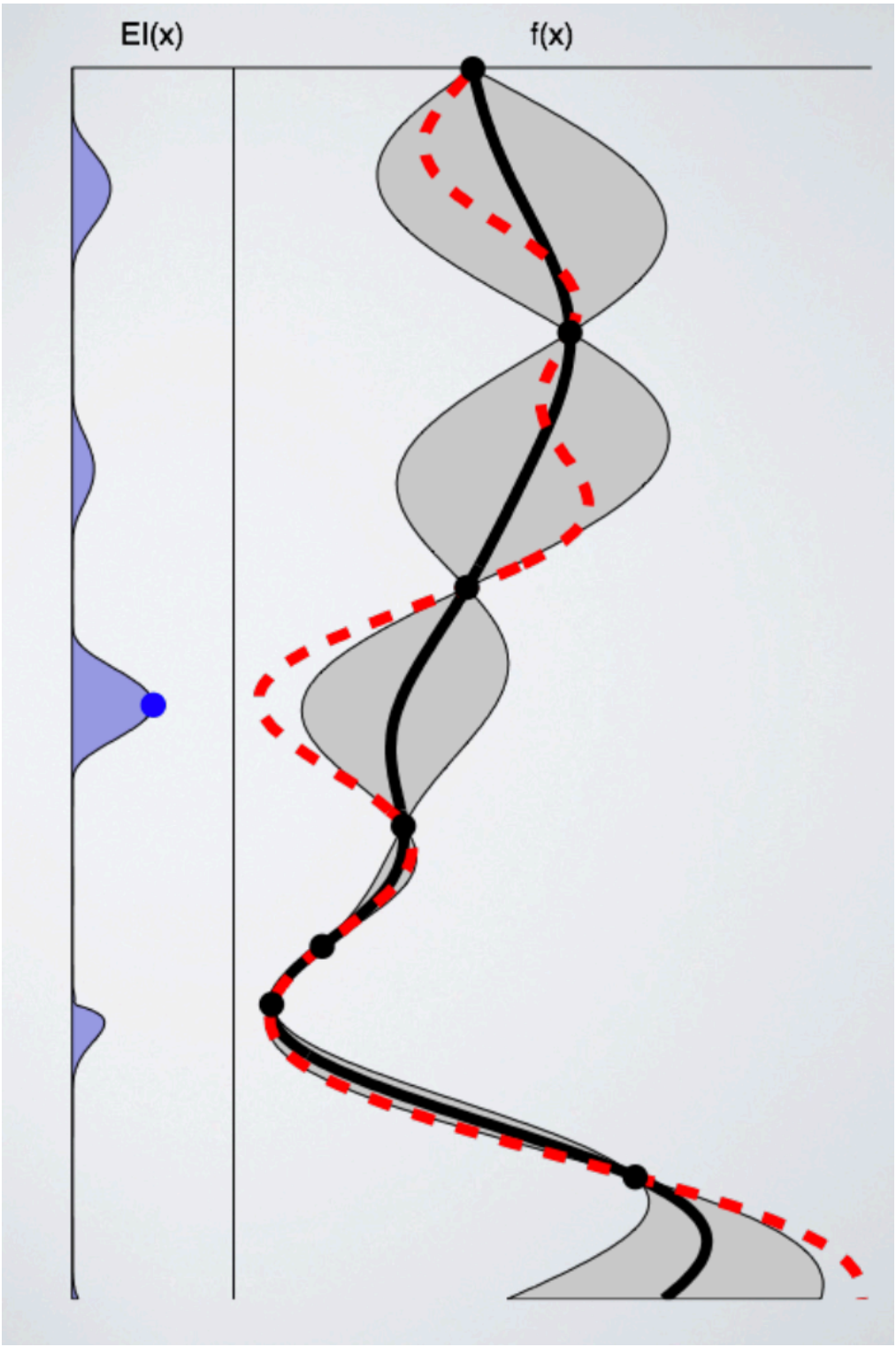

pred var
pred mean
truth
evaluations
f(x)
EI(x)

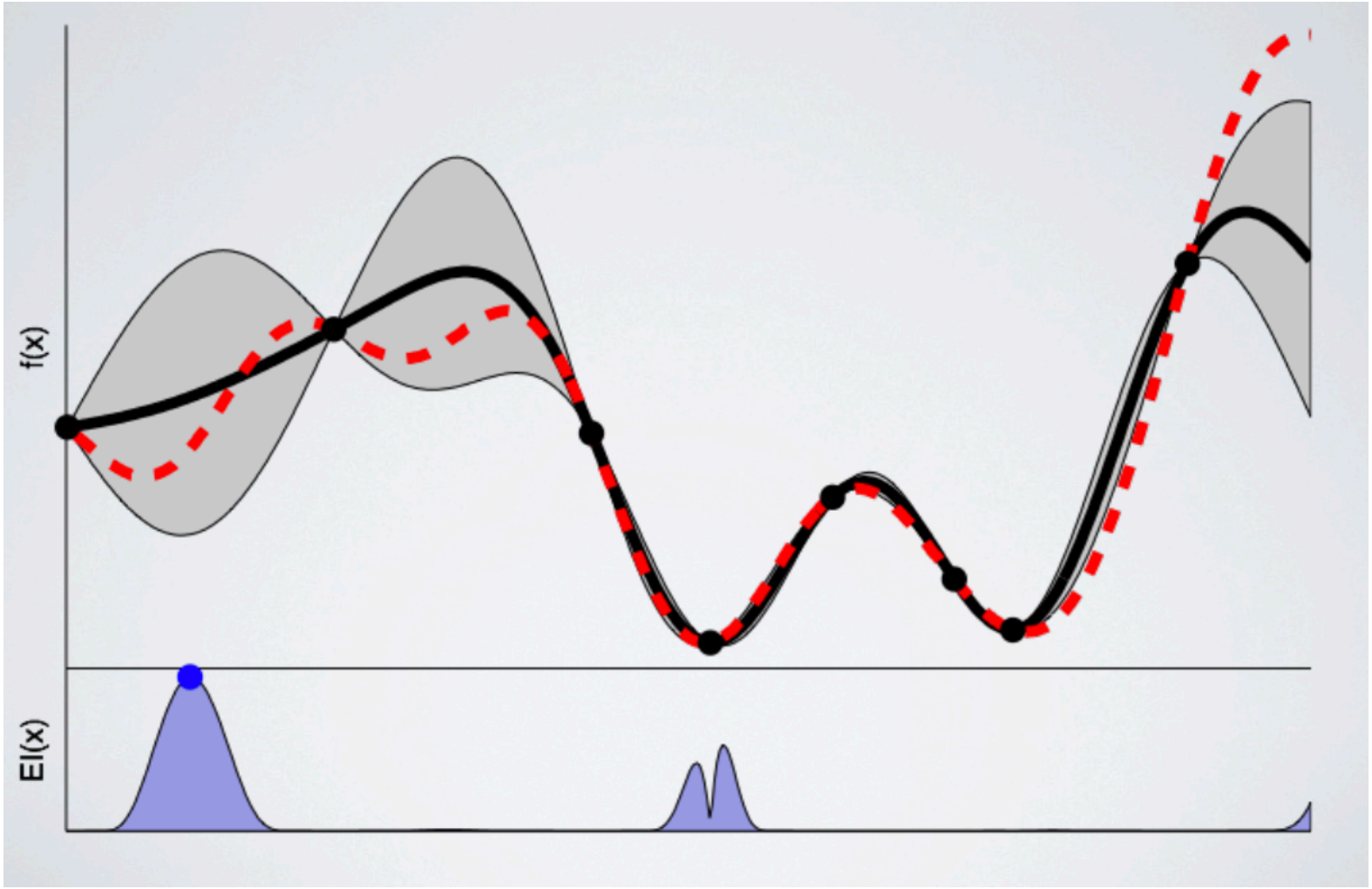

EI(x)
f(x)


EI(x)
f(x)


EI(x)
f(x)


f(x)
EI(x)


EI(x)
f(x)


f(x)
EI(x)

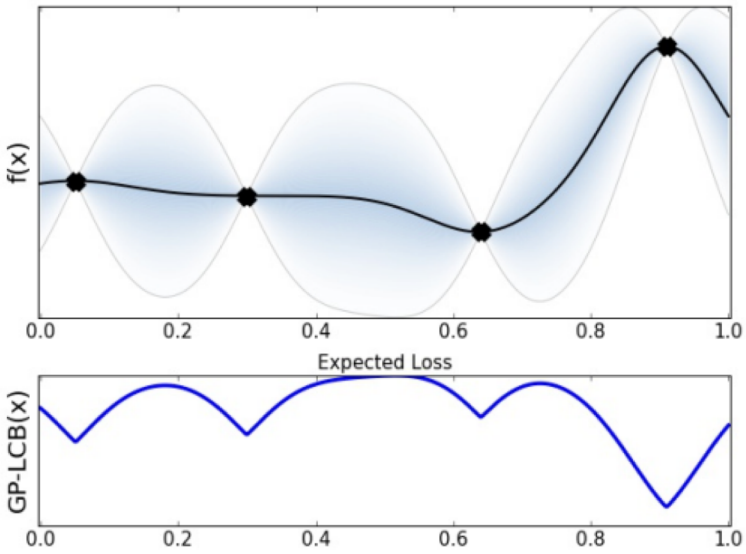
Goal – minimize function
Acquisition function – Upper/Lower confidence bound

$$\alpha_{UCB}(\mathbf{x};\theta,\mathcal{D}) = \mu(\mathbf{x};\theta,\mathcal{D}) + \beta_t\sigma(\mathbf{x};\theta,\mathcal{D})$$

(for maximizing function)

Direct balance between exploration and exploitation:

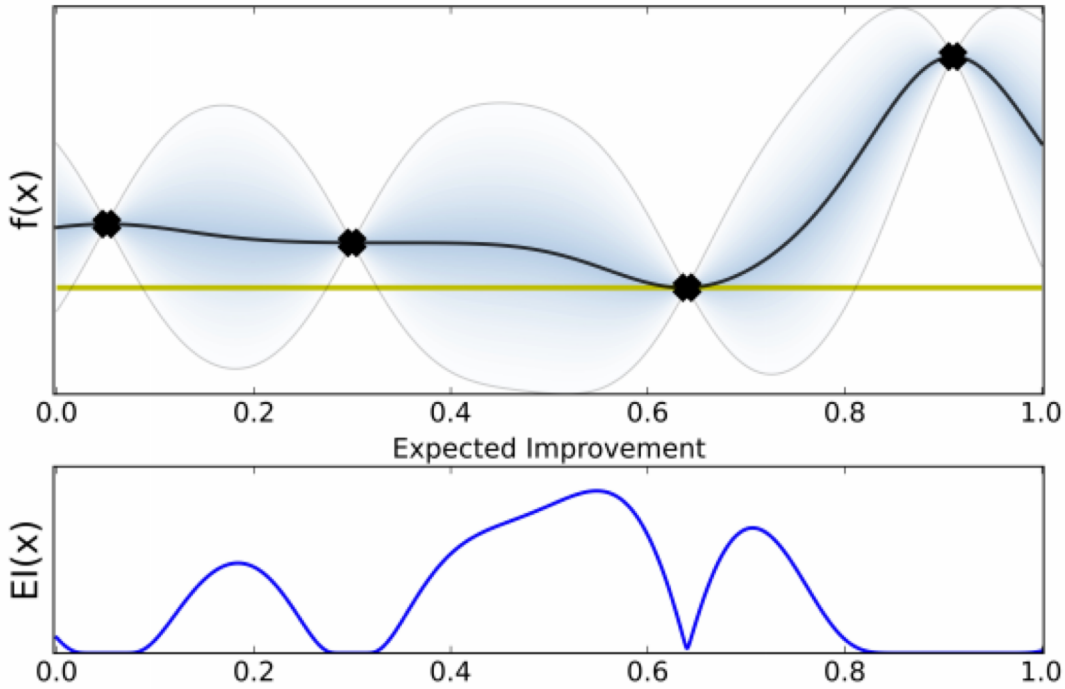
$$\alpha_{LCB}(\mathbf{x};\theta,\mathcal{D}) = -\mu(\mathbf{x};\theta,\mathcal{D}) + \beta_t\sigma(\mathbf{x};\theta,\mathcal{D})$$



Goal – minimize function
Acquisition function – Expected Improvement

$$\alpha_{EI}(\mathbf{x};\theta,\mathcal{D}) = \int_y \max(0, y_{best} - y) p(y|\mathbf{x};\theta,\mathcal{D}) dy$$

- Possibly most used acquisition function
- Explicit form available for Gaussian processes



Why is Bayesian Optimization not widely used?

- Fragility and poor default choices.
  - Getting the function model wrong can be catastrophic.
- There is not standard software available.
  - Tricky to build from scratch
- Experiments are run sequentially
  - Want to use parallel computing
- Gaussian Processes have limited ability to scale
  - Need alternative models of uncertainty

(In part, Bryan Adams)

Experiment design using deep ensembles

How can we use Bayesian optimization to design our next k-mer experiment?

- We a model f of the binding of a transcription factor to 8-mer DNA sequences.
  - Binding = f(8-mer sequence)
  - Assume given training data { (s_1, b_1), (s_2, b_2) … (s_n, b_n) } to train f
  - Goal is to discover s_{best} = argmax f(s)
  - Need best model f; what the next next s_x we should ask to observe?
- What is a principled way to choose s_x ?

How can we use Bayesian optimization to design our next k-mer experiment?

- Let's use Deep ensembles for Bayesian optimization
  - (Though note that ensembles are not "Bayesian")

- But ensembles can be uncalibrated

- One way to improve calibration…

MOD (Maximizing Overall Diversity) can improve uncertainty calibration

- In a nutshell – maximize variance on the uniform distribution over all possible inputs as part of the loss
- Equivalent to maximizing entropy for Gaussian distributions

$$\min_{\theta_1,\ldots,\theta_M} \; L_{in} - \gamma L_{out}$$

$$L_{in} = \frac{1}{M} \sum_{m=1}^{M} E_{P_{in}}[L(\theta_m, x, y)]$$

$$L_{out} = \frac{1}{Z} \int_{\mathcal{X}} \sigma^2_{mod}(x)\, \mathrm{d}x$$

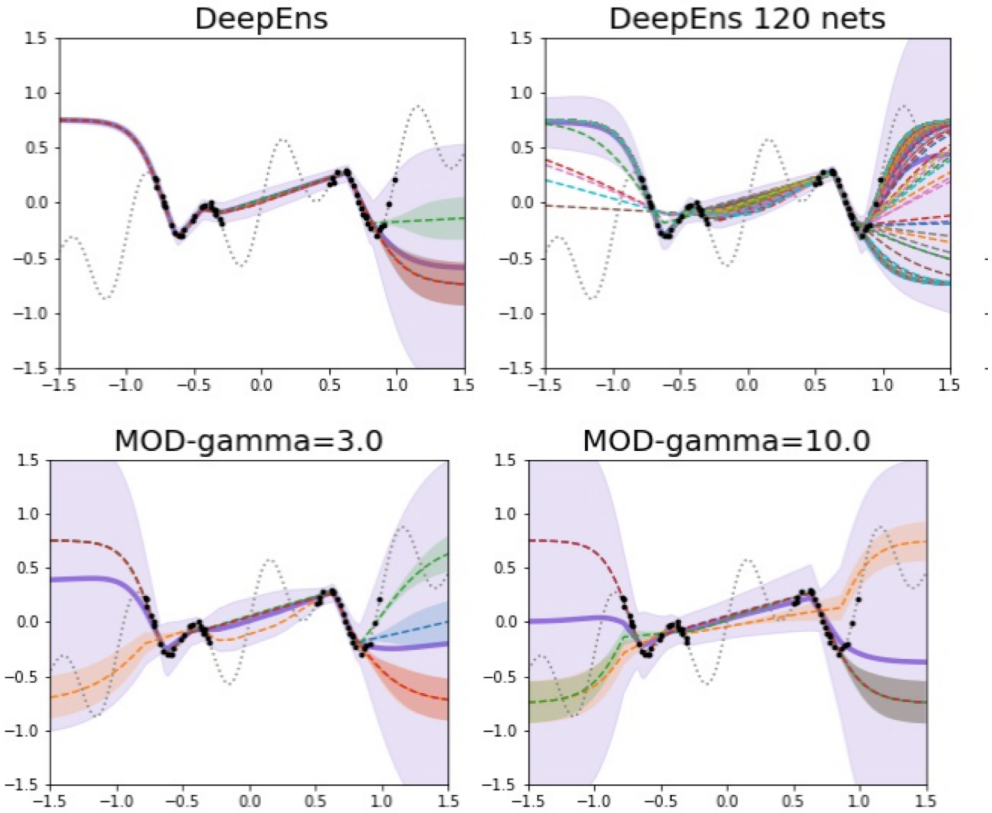
MOD-R (MOD-reweighting) improvement

- Increase weight for randomly sampled points further away from the training points

$$\tilde{w}_b = \frac{\sum_{i=1}^{k} ||\tilde{x}_b - x_i^b||^2}{\max_b \sum_{i=1}^{k} ||\tilde{x}_b - x_i^b||^2}$$

Example MOD application to prediction uncertainty

$$y = 0.3x + 0.3\sin(2\pi x) + 0.3\sin(4\pi x) + \epsilon$$



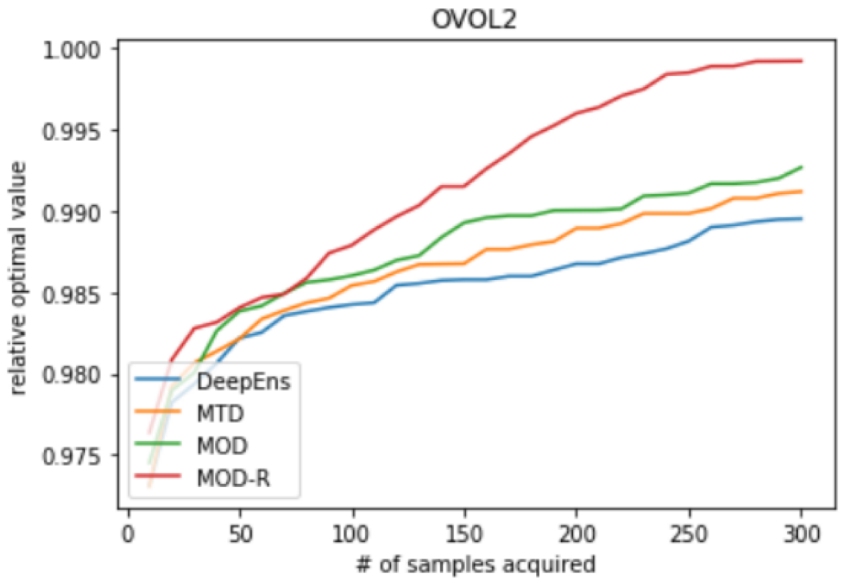
Application of MOD to choose next k-mer

- Protein-DNA binding
  - 38 different TFs, 8-mer binding data derived from PBMs
  - NN architecture used – single hidden layer

- Ensemble size 4 throughout

Better uncertainty metrics converge on best value with fewer new experiments (samples)

- Acquisition function – UCB
- 30 rounds of acquisition of size 10 each
- 10% held out data used for hyperparameter selection
- MTD is a control where you maximize variance on *training* inputs



Better uncertainty metrics converge on best value with fewer new experiments (samples)

- Table indicates # of times row method beat column method across 38 TFs in retrieving fraction of top 1% sequences (p-values in parens)

X	DeepEns	MTD	MOD	MOD-R
DeepEns	X	0	1 (0.051)	0
MTD	0	X	1 (0.037)	1 (0.094)
MOD	**4 (0.005)**	1 **(0.02)**	X	0
MOD-R	**10 <1E-5**	**10 <1E-5**	**6 <1E-5**	X

FIN - Thank You

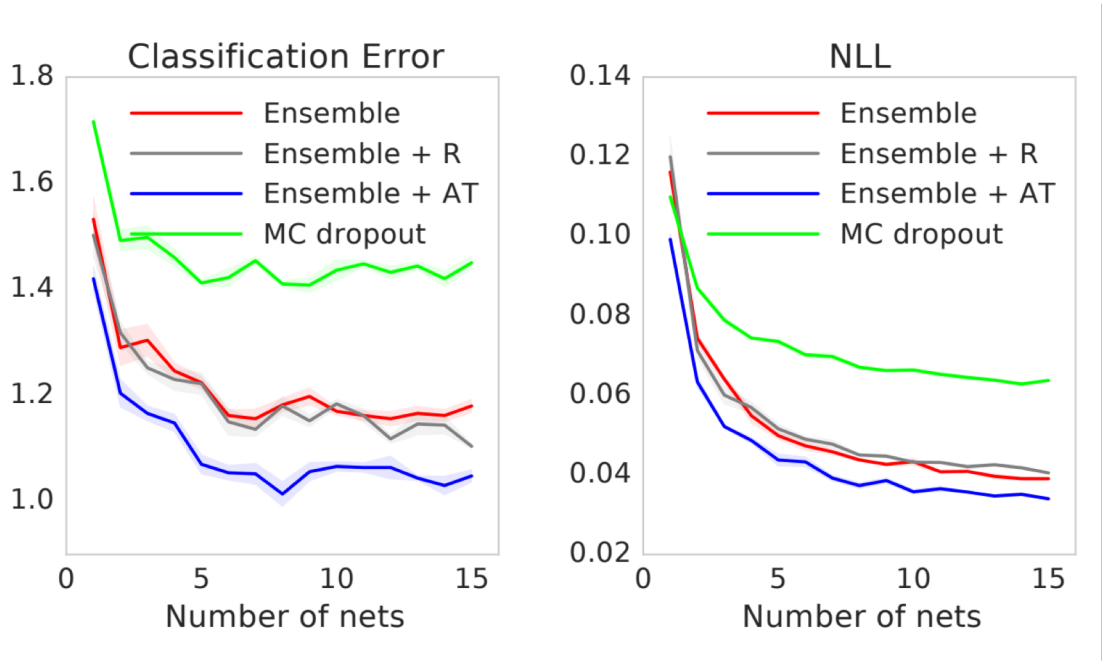
Adversarial training to improve network performance

- Assume points close to training point have same label
- Find point near training point with low likelihood
- Use that point to augment dataset and increase their likelihood

$$\mathbf{x}' = \mathbf{x} + \epsilon \operatorname{sign}\Big(\nabla_{\mathbf{x}}\, \ell(\theta, \mathbf{x}, y)\Big)$$

Deep Ensemble performance

- AT - adversarial training
- R – likelihood maximized for random point near training rather that one with low likelihood
- MC Dropout - # of samples = # of nets



Other covariance functions

- Matern - $\sigma^2 \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\sqrt{2\nu} \frac{d}{\rho} \right)^{\nu} K_\nu \left(\sqrt{(2\nu)} \frac{d}{\rho} \right)$ where K_ν is the modified Bessel function of the second kind and $d = ||x_i - x_j||$
- Periodic Kernel - $\sigma^2 e^{-\frac{2 \sin^2(\pi d/p)}{l^2}}$
- Rational quadratic - $\sigma^2 \left(1 + \frac{d^2}{2al^2} \right)^{-a}$
- Neural network based kernels (Deep Kernel Learning by Wilson et al. 2015)