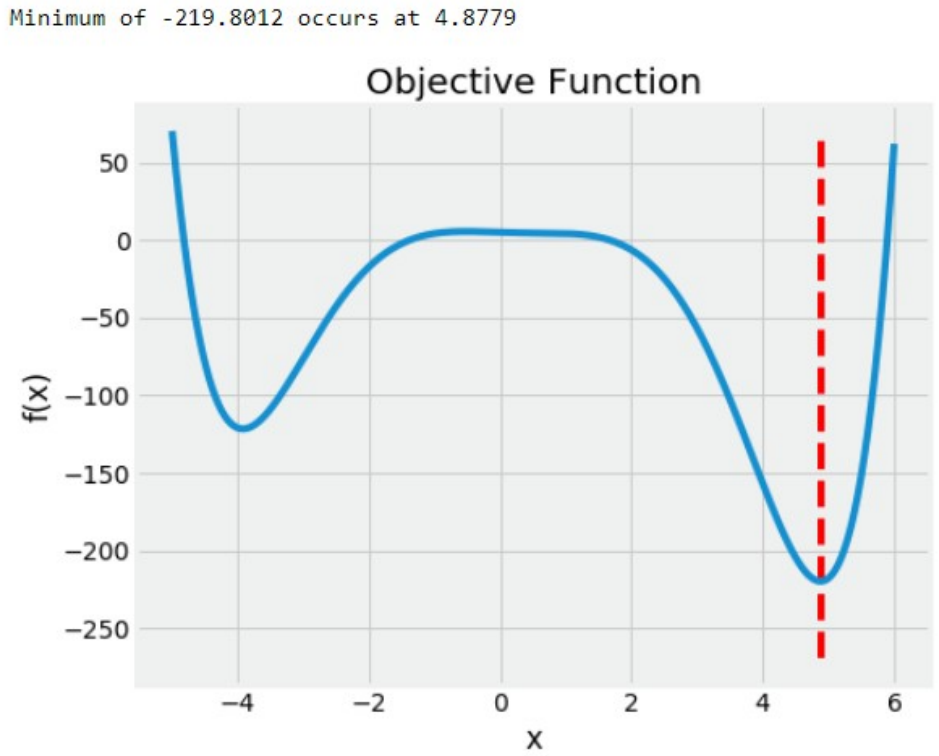
Machine learning
Lecture 7



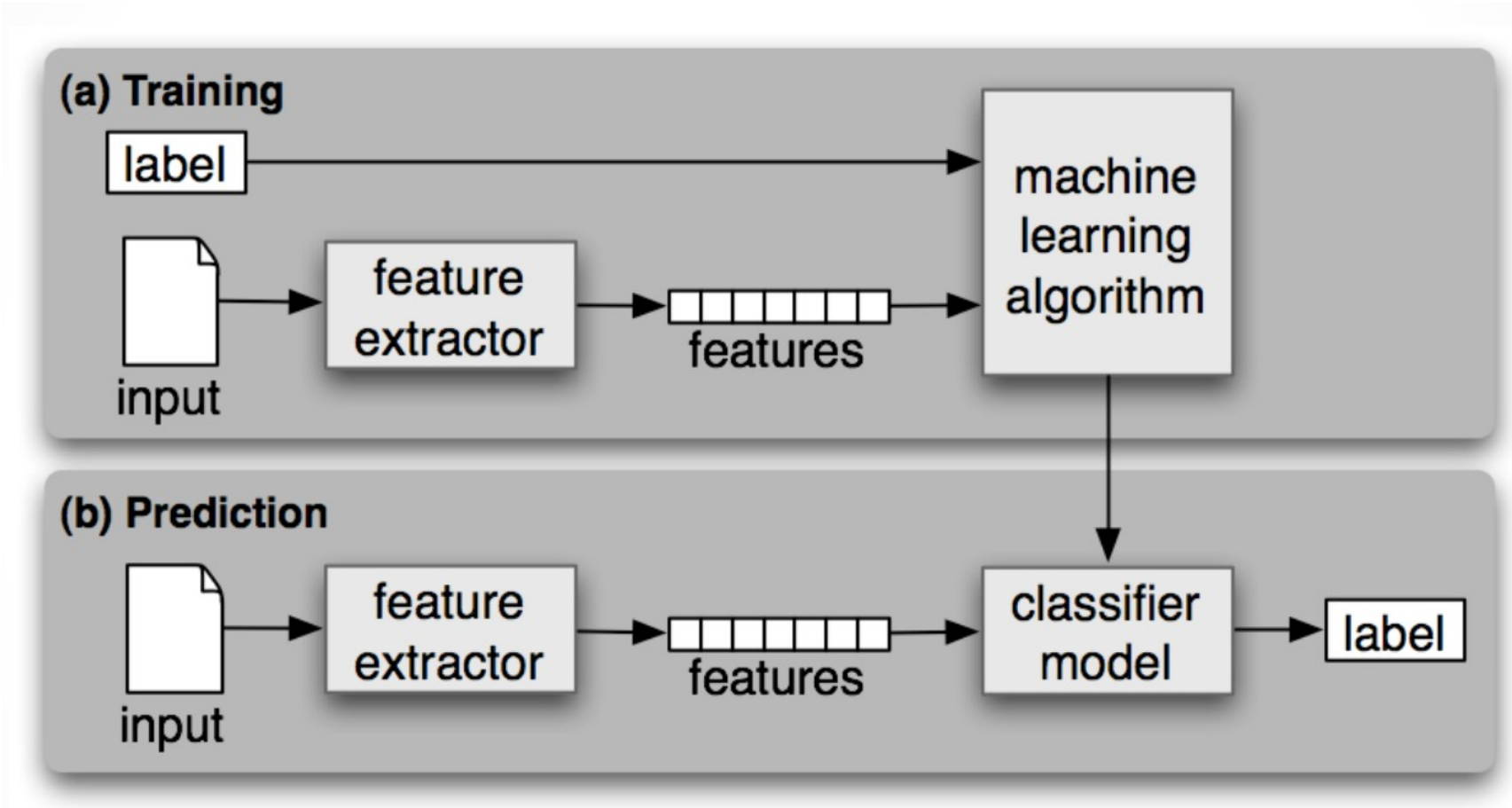
- Training - cross-validation.
- Optimization of hyperparameters.
- Cluster analysis

Marcin Wolter

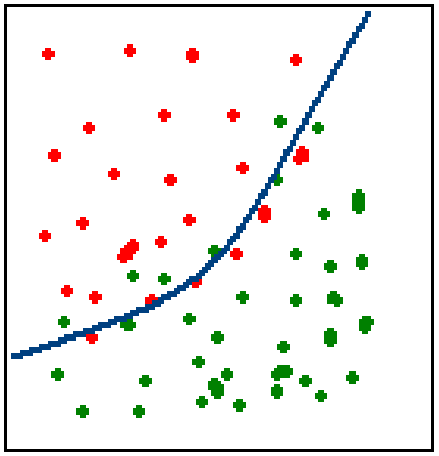
IFJ PAN

24 January 2020

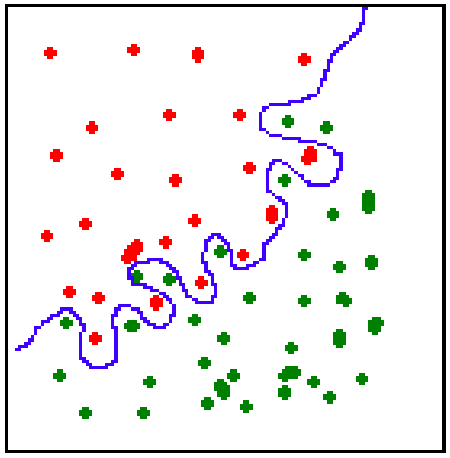
Supervised training



Overtraining

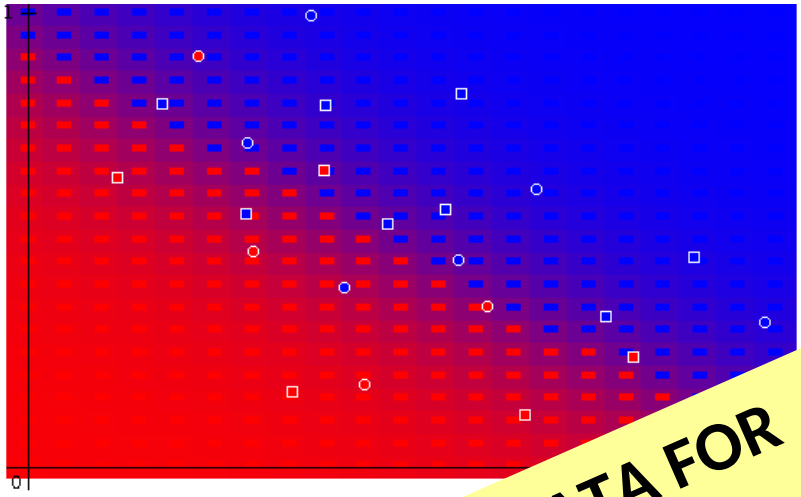


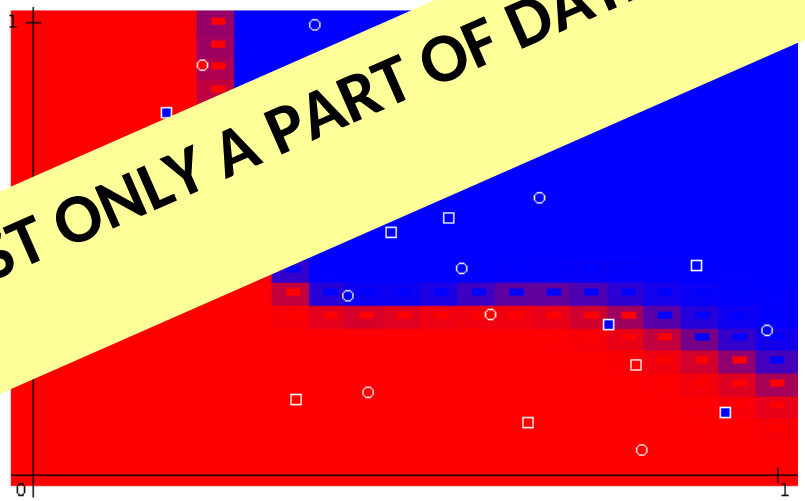
Correct

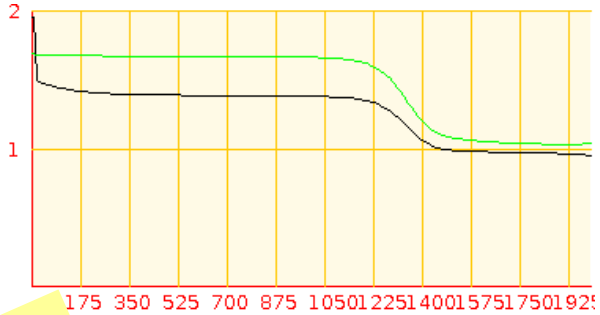


Overtraining

- **Overtraining** – algorithm “learns” the particular events, not the rules.
- This effect appears for all ML algorithms.
- Remedy – checking with another, independent dataset.

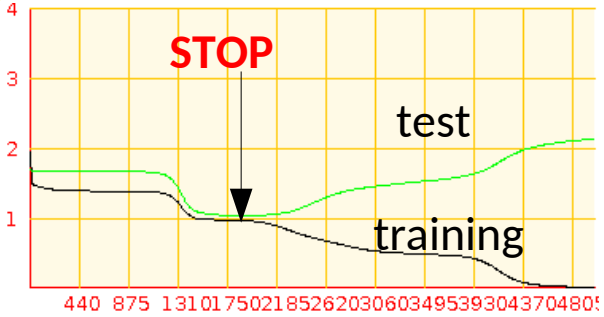






Training sample

Test sample



Example of using Neural Network.

BUT WE USE JUST ONLY A PART OF DATA FOR TRAINING!!!

How to train a ML algorithm?

- How to avoid **overtraining** while learning?
- Should we use one sample for **training** and another for **validating**?
- Then we increase the error – we use just a part of data for training.

- **Second remark:** to avoid ovetraining and find the performance of the trained algorithm we should use one more, **third data sample** to measure the final performance of the ML algorithm.

- How to **optimize the hyperparameters** of the ML algorithm (number of trees and their depth for BDT, number of hidden layers, nodes for Neural Network)?

Validation

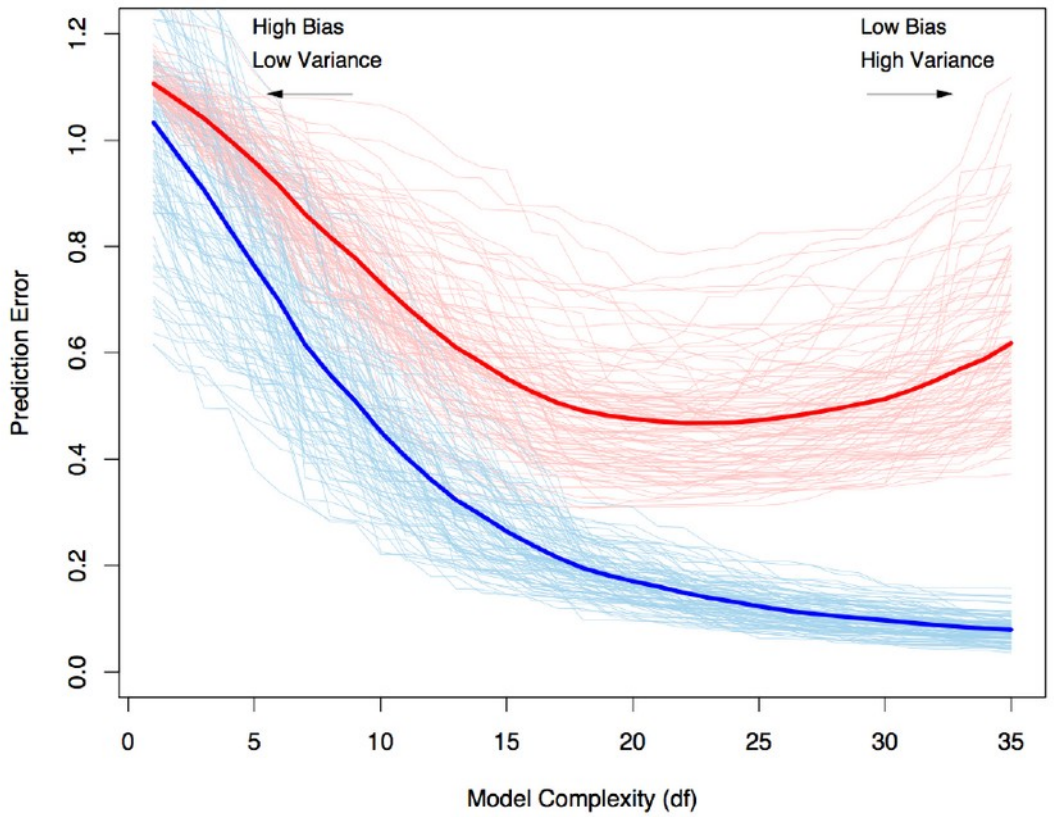


FIGURE 7.1. *Behavior of test sample and training sample error as the model complexity is varied. The light blue curves show the training error* $\overline{\text{err}}$*, while the light red curves show the conditional test error* $\text{Err}_{\mathcal{T}}$ *for* 100 *training sets of size* 50 *each, as the model complexity is increased. The solid curves show the expected test error* Err *and the expected training error* $\text{E}[\overline{\text{err}}]$.

Source: Elements of Statistical Learning

Cross-validation

- We have independant training sample L_n and a test sample T_m.
- Error level of the classifier $\hat{d}(\boldsymbol{x}) = \hat{d}(\boldsymbol{x};\ \mathcal{L}_n)$ built on the training sample L_n

$$\hat{e}_T = \frac{1}{m}\sum_{j=1}^{m} I\left(\hat{d}(\boldsymbol{X}_j^t; \mathcal{L}_n) \neq Y_j^t\right)$$

- Estimator using "recycled data" (the same data for training and for error calculation) is biased.
- Reduction of bias: division of data into two parts (training & validation). But than we use just a half of information only.
- **Cross-validation** – out of sample L_n we remove just one event j, train classifier, validate on single event j. We repeat n times and get the estimator (an average of all n estimators):

$$\hat{e}_{CV} = \frac{1}{n}\sum_{j=1}^{n} I\left(\hat{d}(\boldsymbol{X}_j; \mathcal{L}_n^{(-j)}) \neq Y_j\right)$$

- We get an estimator, which is unbiased (in limit of huge n), but training is CPU demanding.

Cross-validation

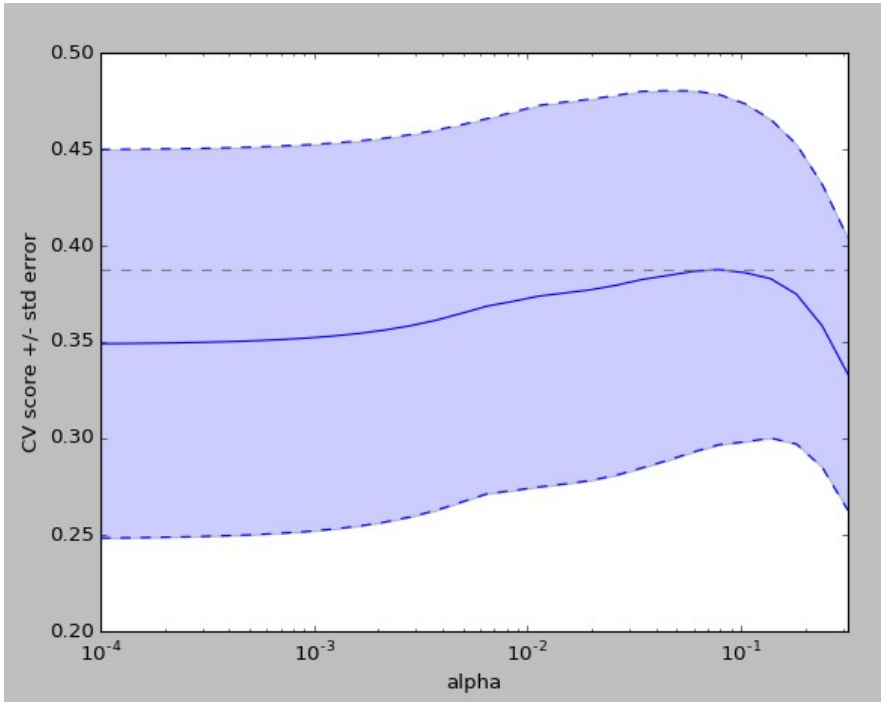
- Intermediate solution – ***k-fold cross-validation***
- The sample is divided into k subsamples, k-1 of them we use for training, the one for validation. Then the procedure is repeated with other subsamples and the procedure is repeated k tir

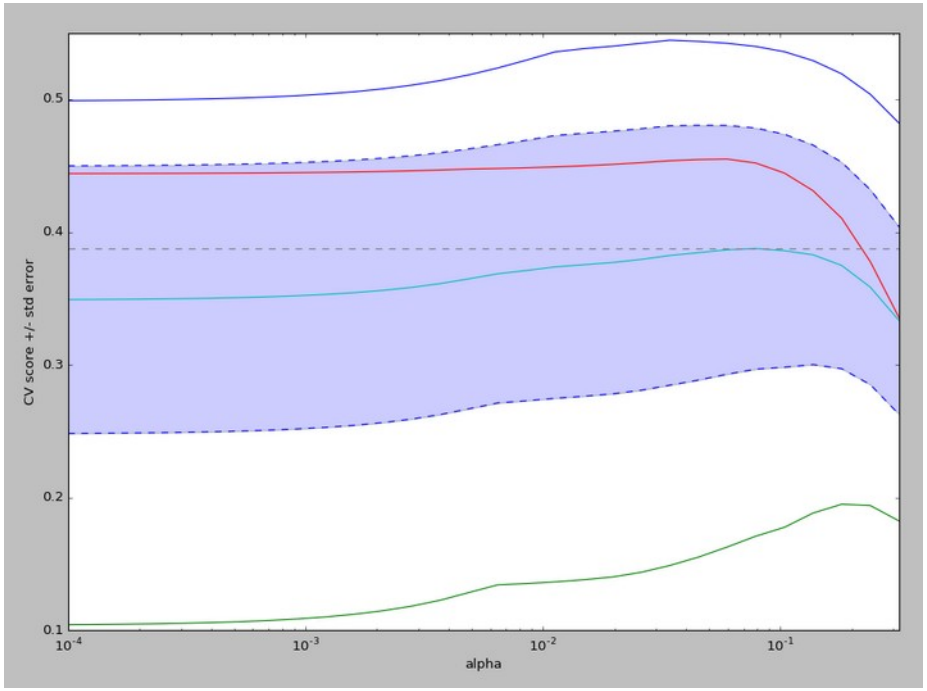
$$\hat{e}_{vCV} = \frac{1}{n}\sum_{i=1}^{v}\sum_{j=1}^{n} I(\boldsymbol{Z}_j \in \tilde{\mathcal{L}}_n^{(i)}) I\left(\hat{d}(\boldsymbol{X}_j; \tilde{\mathcal{L}}_n^{(-i)}) \neq Y_j\right)$$

- Smaller CPU usage comparing to cross-validation.
- Recommended k ~ 10 .
- Resulting classifier might be in the simplest case an average of all k classifiers (or they might be joined together in another way) .

Cross-validation

- 4-times folding
- Finding a dependence of CV from alpha (medical data)
- We can draw the mean and a standard deviation. In the next plot we draw the dependence for each folding.
- As a result we can estimate an error of the cross-validated classifier.





Model performance

Test Error

- Partition the original data (randomly) into a training set and a test set. (e.g. 70/30)
- Train a model using the training set and evaluate performance (a single time) on the test set.

K-fold Cross-validation

- Train & test K models as shown.
- Average the model performance over the K test sets.
- Report cross-validated metrics.

Performance Metrics

- Regression: R^2, MSE, RMSE
- Classification: Accuracy, F1, H-measure, Log-loss
- Ranking (Binary Outcome): AUC, Partial AUC

https://www.slideshare.net/0xdata/top-10-data-science-practitioner-pitfalls

Train vs Test vs Valid

Training Set vs. Validation Set vs. Test Set

- If you have “enough” data and plan to do some model tuning, you should really partition your data into three parts — Training, Validation and Test sets.
- There is **no general rule** for how you should partition the data and it will depend on how strong the signal in your data is, but an example could be: 50% Train, 25% Validation and 25% Test

Validation is for Model Tuning

- The validation set is used **strictly for model tuning** (via validation of models with different parameters) and the test set is used to make a final estimate of the generalization error.

Cross-validation example

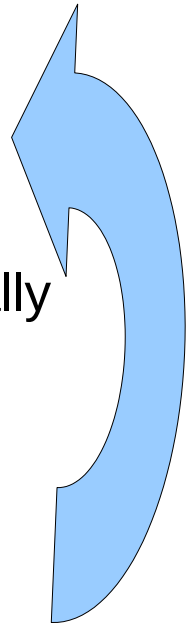
https://github.com/marcinwolter/MachineLearnin2019/blob/master/cross_validation.ipynb

Hyperparameter optimization

- Nearly each ML method has few hyperparameters (structure of the Neural Net, number of trees and their depth for BDT etc).
- They should be optimized for a given problem.
- **Task: for a given data sample find a set of hyperparameters, that the estimated error of the given method is minimized.**
- Looks like a typical minimization problem (fitting like), but:
  - Getting each measurement is costly
  - High noise
  - We can get the value of the minimized function (so our error) in the pont **x** of the hyperparameter space, but we can't get the differential easily.

Optimization of hyperparameters

- How to optimize:
  - „Grid search” - scan over all possible values of parameters.
  - „Random search”
  - Some type of fitting…
- Popular method is the **„bayesian optimization”**
  - Build the probability model
  - Take „a priori” distributions of parameters
  - Find, for which point in the hyperparameter space you can maximally improve your model
  - Find the value of error
  - Find the „a posteriori” probability distribution
  - Repeat



How does it work in practice?

- Straight line fitting

  $y(x, w) = w_0 + w_1 x$ fit to the data.

1) Gaussian prior, no data used

2) First data point. We find the likelihood based on this point (left plot) and multiply: priori*likelihood. We get the posterior distribution (right plot).

3) We add the second point and repeat the procedure.

4) Adding all the points one by one.

Remark: data are noisy.

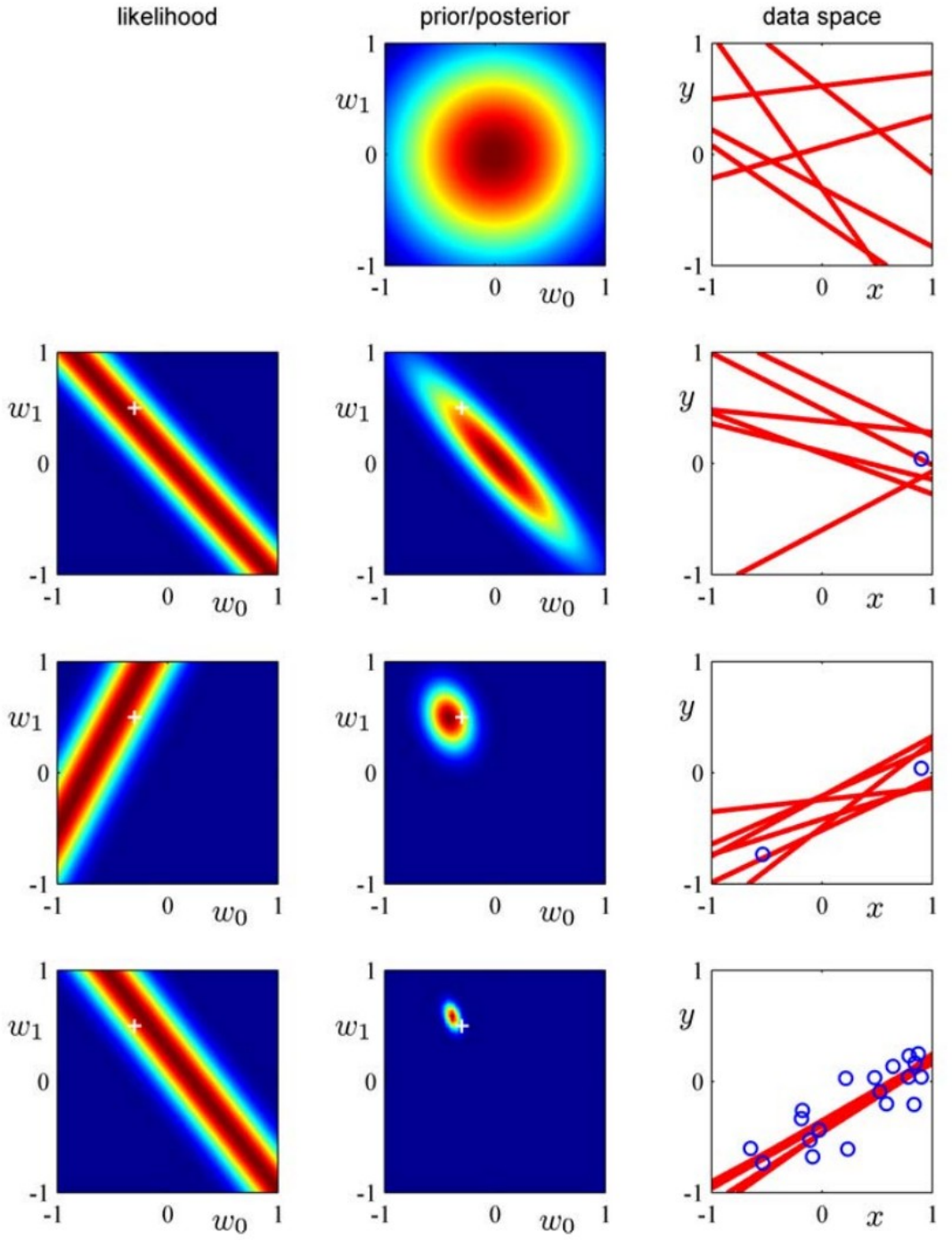
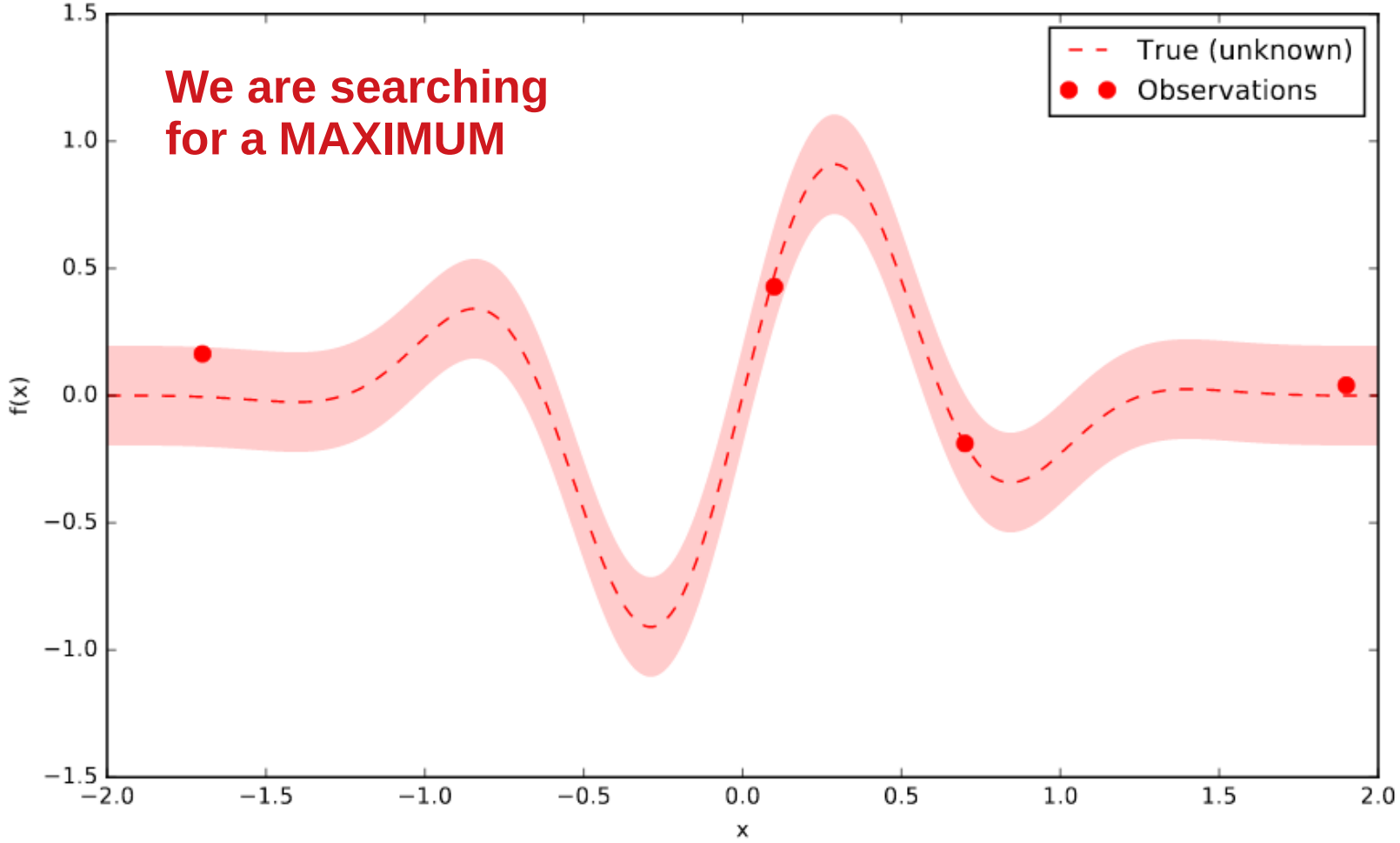


Illustration of sequential Bayesian learning for a simple linear model of the form $y(x, \mathbf{w}) = w_0 + w_1 x$. A detailed description of this figure is given in the text.

Starting point

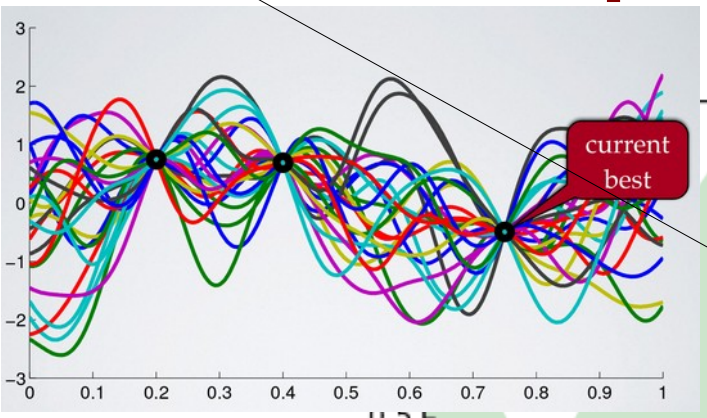


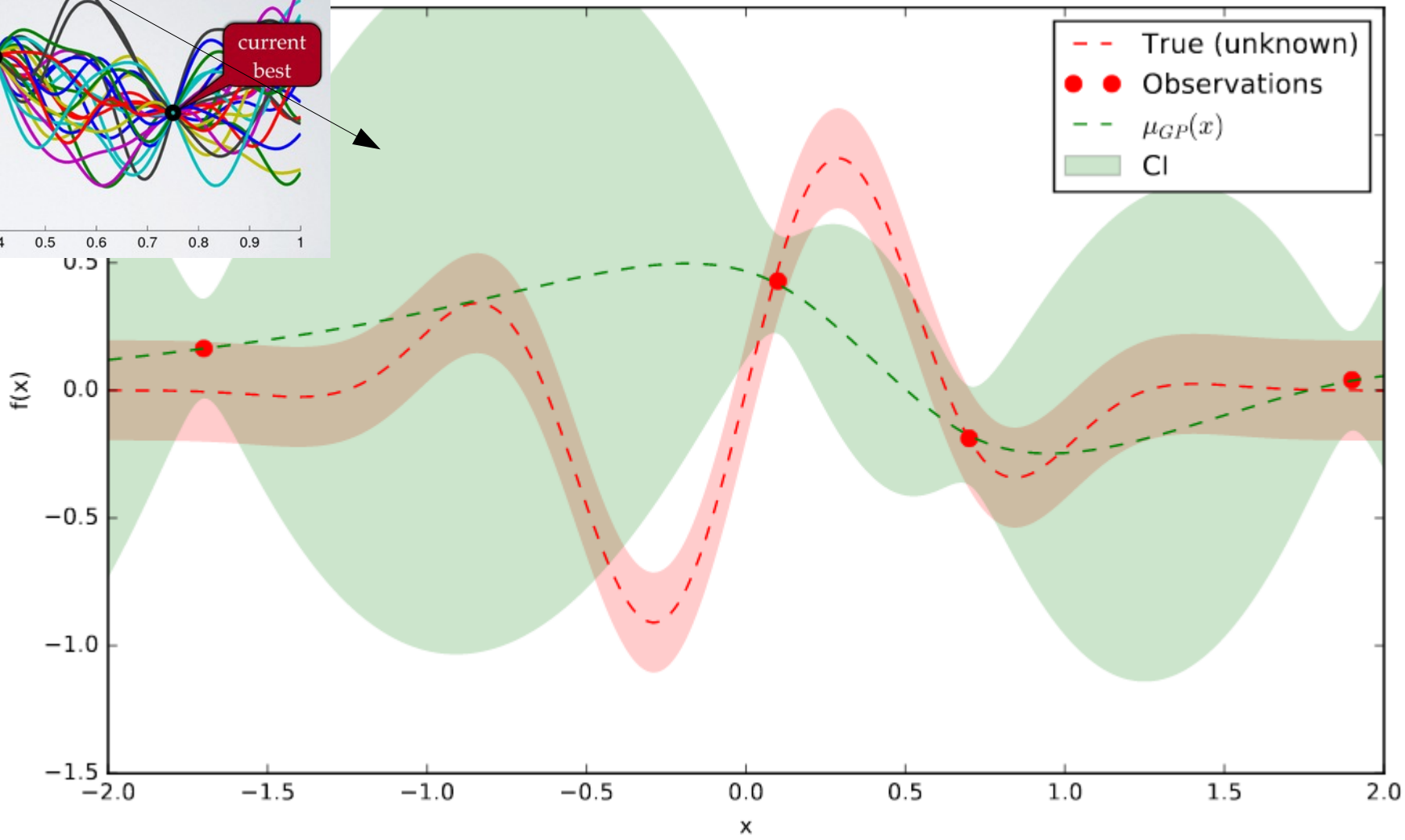
Unknown function (with noise), four observations.
Where should we do the next costly probing?

https://www.iro.umontreal.ca/~bengioy/cifar/NCAP2014-summerschool/slides/Ryan_adams_140814_bayesopt_ncap.pdf

A posteriori distribution

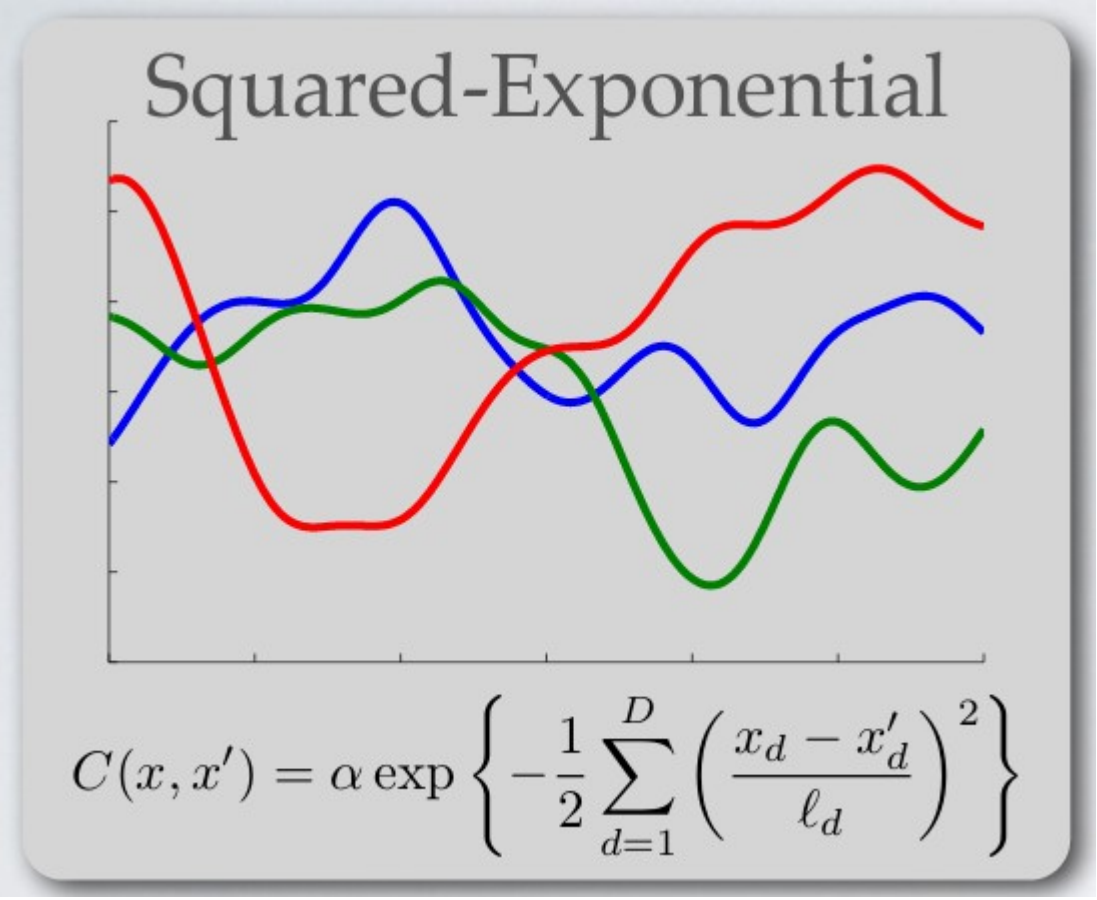
A set of functions





The a posteriori distribution of possible functions, those functions could generate the observed data points.

A posteriori functions – Gaussian Processes (GP)



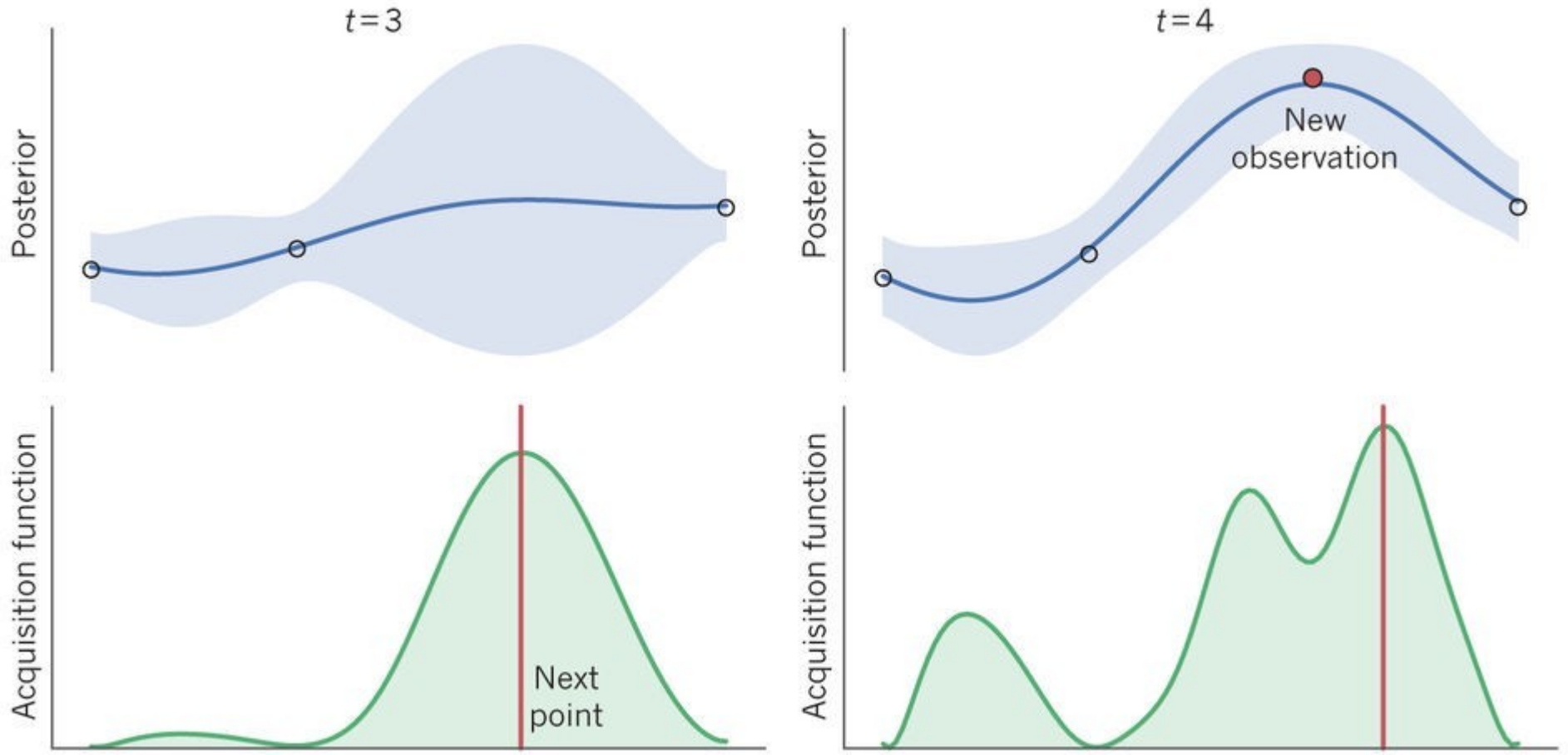
These functions should be somehow parametrized, for example they could be Gaussian functions.

Acquisition function

- Posterior GP (Gaussian Processes) give us the mean of GP functions $\mu(x)$ and their expected variation $\sigma^2(x)$.
  - **Exploration** – searching for huge variation
  - **Exploitation** – searching for a smallest/greatest (depends on sign and convention) value of mean $\mu(x)$
- The acquisition policy has to balance these two approaches

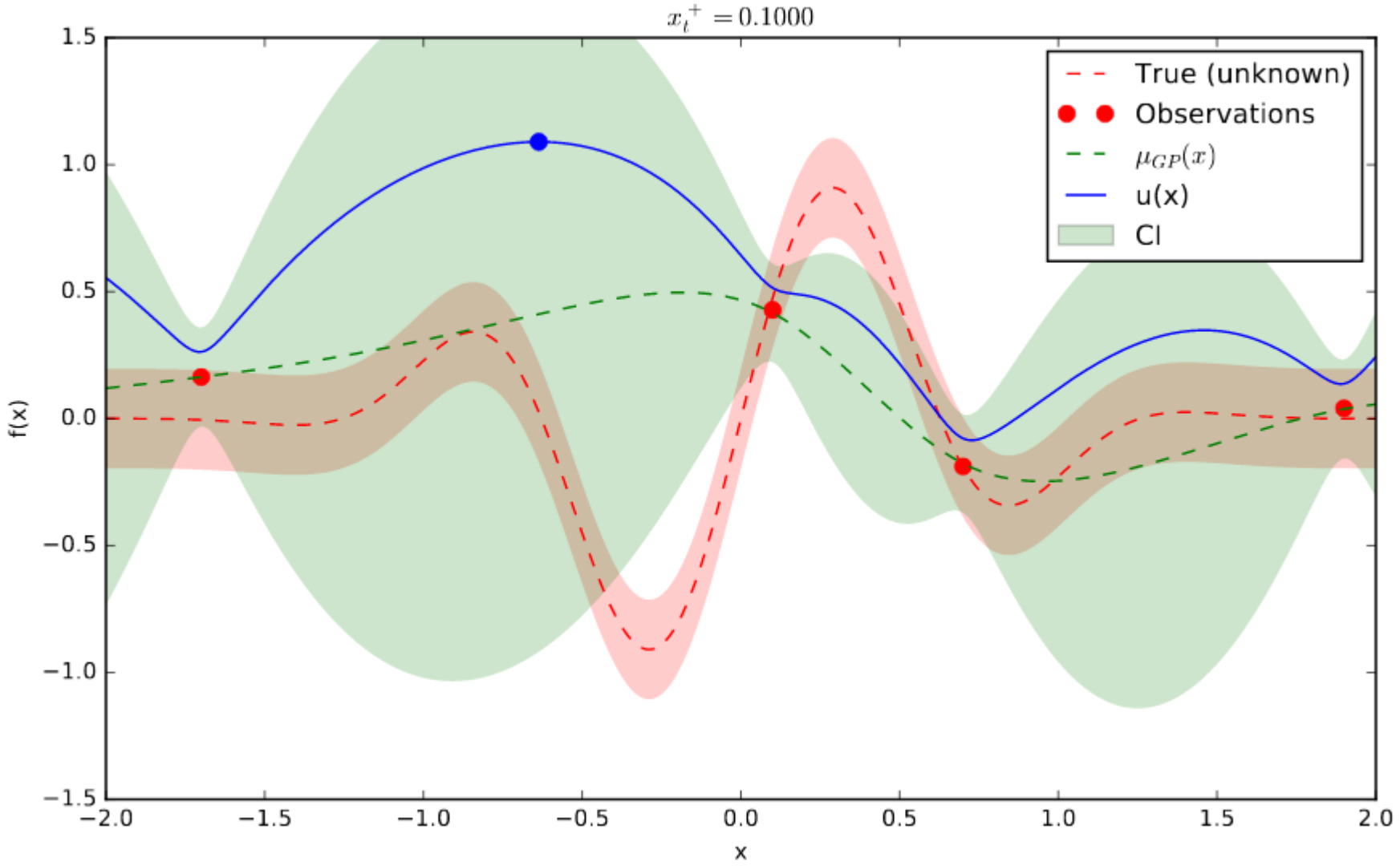
Where to put the next point?

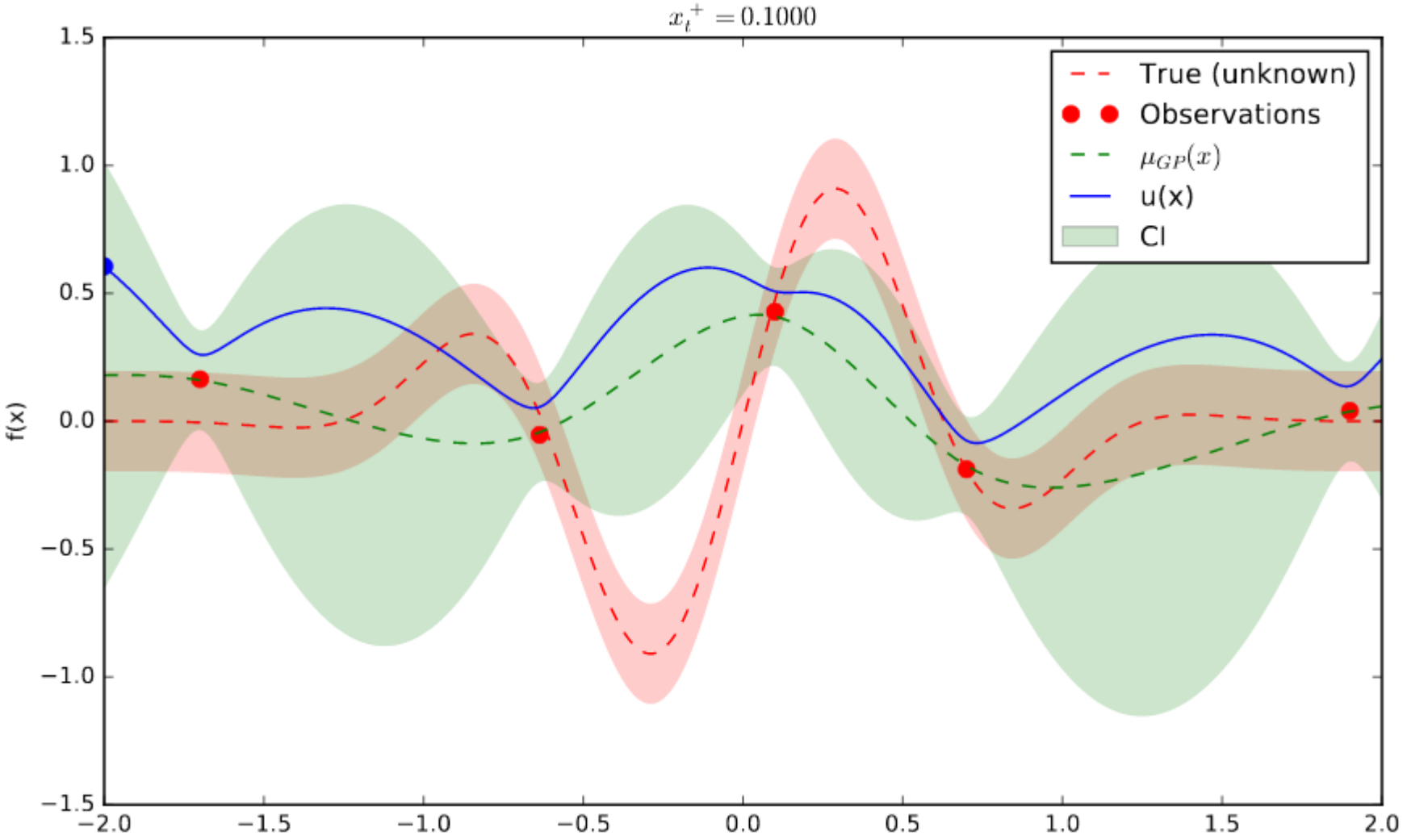
- Our next chosen point(x) should has high mean (exploitation) & high variance (exploration).



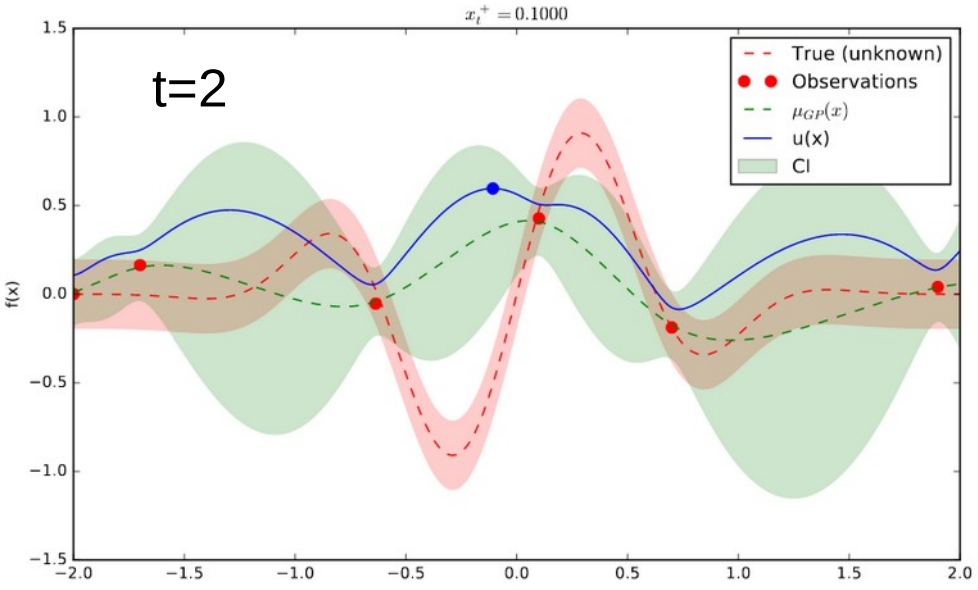
We choose next x

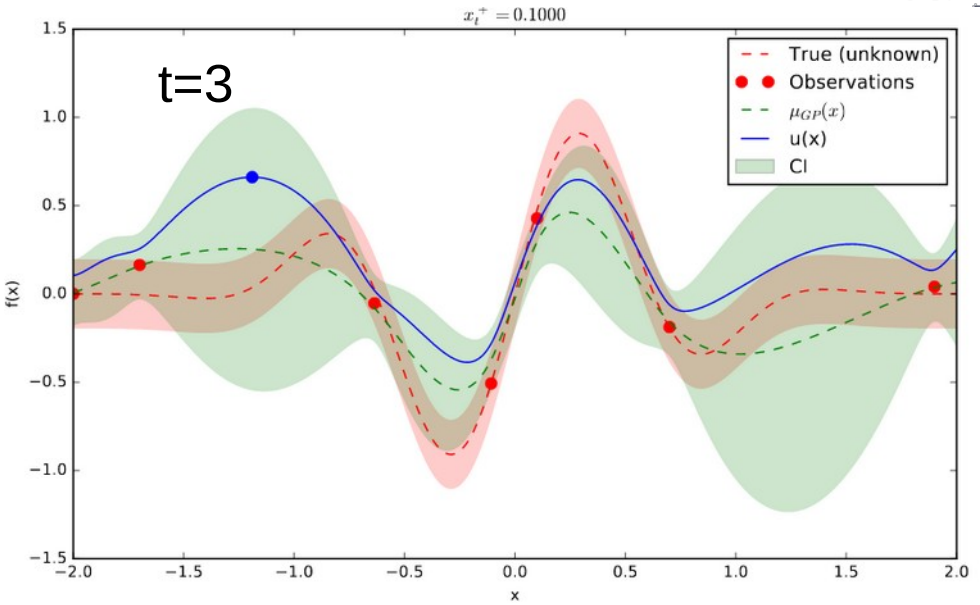
u(x) – acqusition function
(finding maximum)

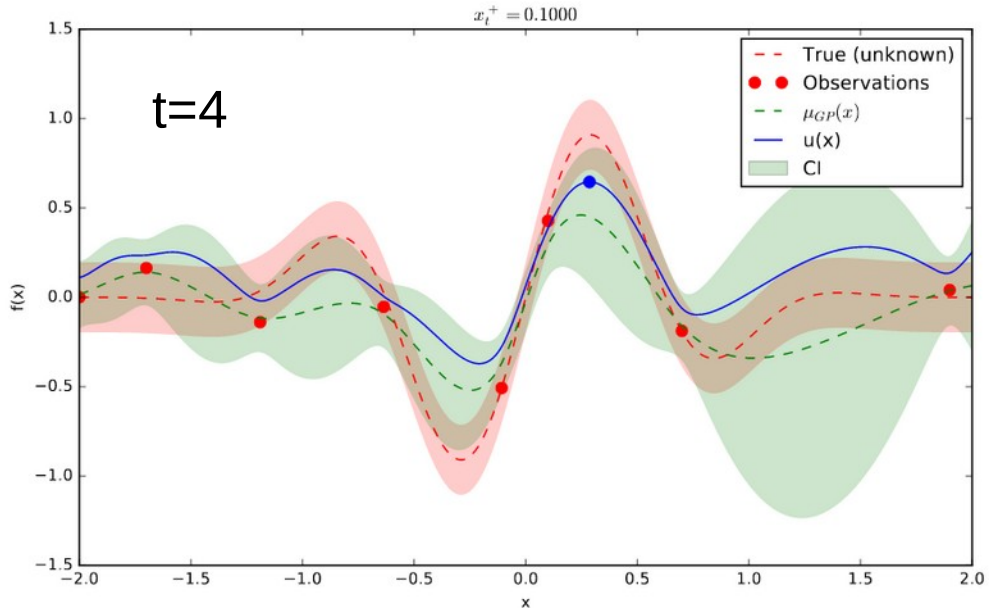


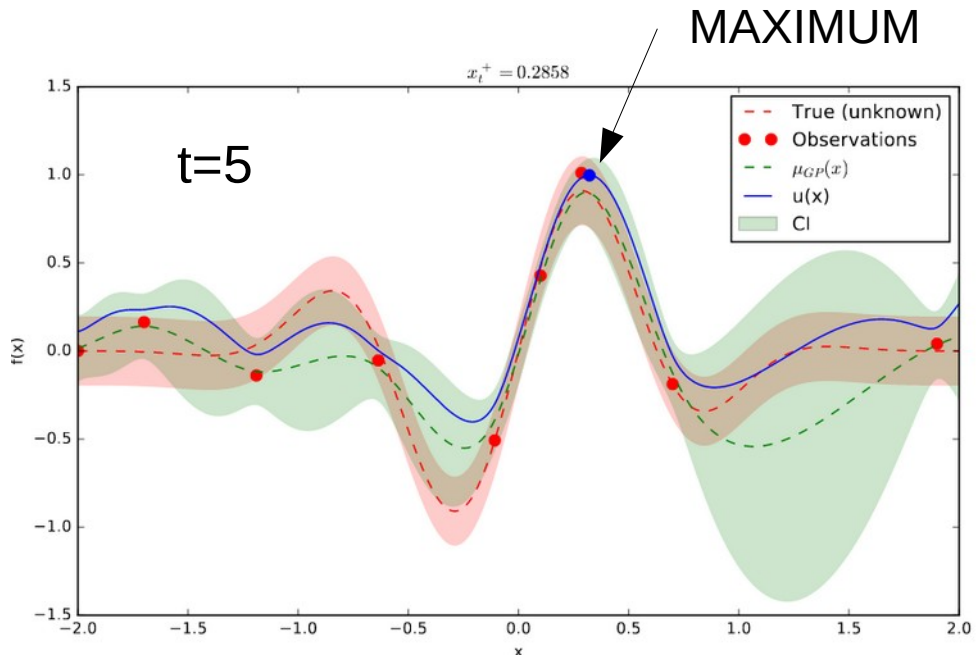


Dokonujemy próbkowania i powtarzamy procedurę...


$x_t^+ = 0.1000$
t=2
True (unknown)
Observations
$\mu_{GP}(x)$
u(x)
CI
f(x)


$x_t^+ = 0.1000$
t=3
True (unknown)
Observations
$\mu_{GP}(x)$
u(x)
CI
f(x)
x


$x_t^+ = 0.1000$
t=4
True (unknown)
Observations
$\mu_{GP}(x)$
u(x)
CI
f(x)
x


MAXIMUM
$x_t^+ = 0.2858$
t=5
True (unknown)
Observations
$\mu_{GP}(x)$
u(x)
CI
f(x)
x

Limitations

- Bayesian optimization depends on the parameters chosen
- On the acquisition function
- On the prior selected....
- It’s **sequential.**
- There are alternative methods, which can be done in parallel (like Random Search or Tree of Parzen Estimators (TPE) used by the HyperOpt package https://github.com/hyperopt/hyperopt).

Hyperparameter optimization - examples

- https://github.com/marcinwolter/MachineLearnin2019/blob/master/cross_validation.ipynb

- Simple HYPEROPT example:
- https://github.com/marcinwolter/MachineLearnin2019/blob/master/hyperopt_demo.ipynb

- Optimization of MNIST hand-written letters recognition using HYPEROPT:
- https://github.com/marcinwolter/MachineLearnin2019/blob/master/mnist_mlp_minimal_hyperopt.ipynb

Articles

- Brochu, E., Cora, V. M., and De Freitas, N. (2010). A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. arXiv preprint arXiv:1012.2599.
- Shahriari, B., Swersky, K., Wang, Z., Adams, R. P., and de Freitas, N. (2016). Taking the human out of the loop: A review of bayesian optimization. Proceedings of the IEEE, 104(1):148–175.
- Nice tutorial:

  https://www.iro.umontreal.ca/~bengioy/cifar/NCAP2014-summerschool/slides/Ryan_adams_140814_bayesopt_ncap.pdf