Hyper-parameter selection with Bayesian optimization

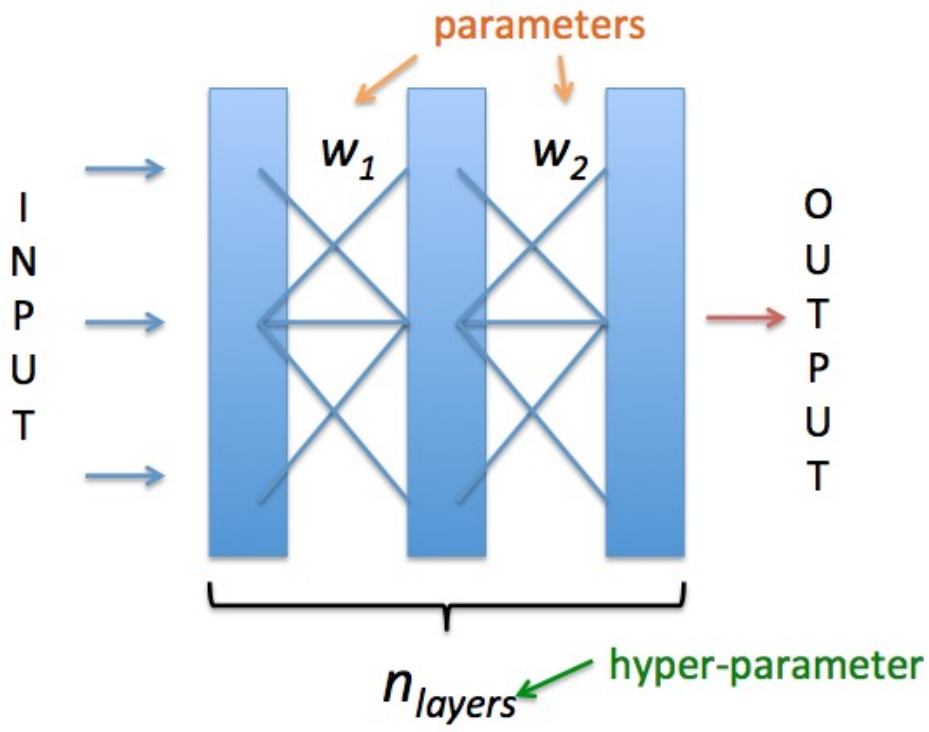
Matthew B. Blaschko & Amal Rannen Triki
Center for Processing Speech and Images
Department of Electrical Engineering

Schedule

- 09:00-10:30 - Session 1 - Introduction, intuition, and Gaussian Processes
- 10:30-11:00 - Coffee break
- 11:00-12:30 - Session 2 - Practical session 1
- 12:30-14:00 - Lunch
- 14:00-15:30 - Session 3 - Bayesian Optimization
- 15:30-16:00 - Coffee break
- 16:00-17:30 - Session 4 - Practical session 2

What is a hyperparameter

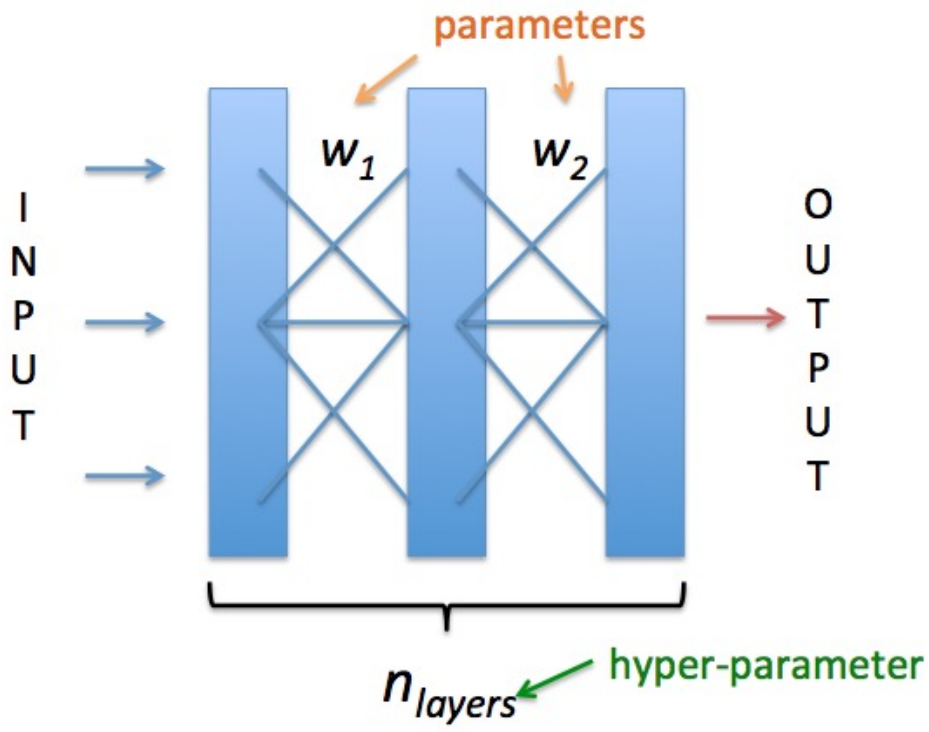
Folk definition: any parameter "left over" that isn't apparent how to optimize during training



Bayesian statistics - a parameter of the prior distribution, but we will use the folk definition for convenience (sometimes they overlap)

What is a hyperparameter

Folk definition: any parameter "left over" that isn't apparent how to optimize during training



Bayesian statistics - a parameter of the prior distribution, but we will use the folk definition for convenience (sometimes they overlap)

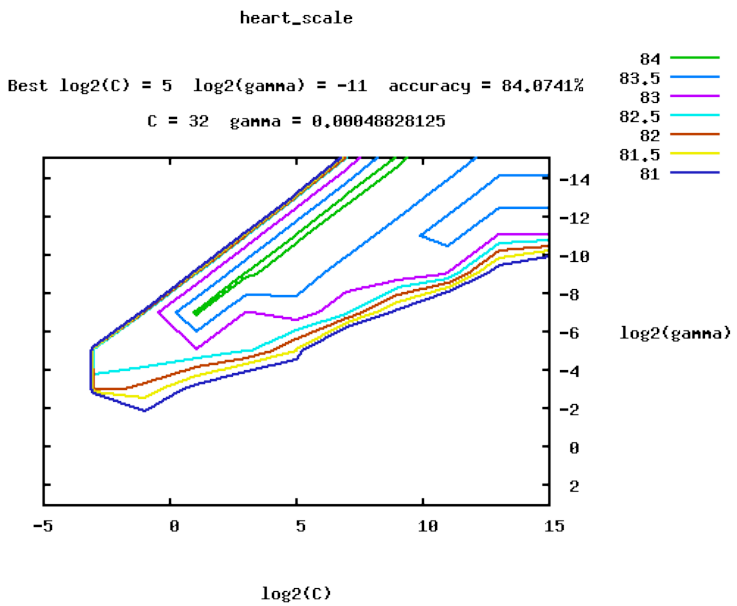
Hyperparameter due to optimization

Support vector machine:

$$\min_{\alpha} \; C\sum_{i=1}^{n}\xi_i + \sum_{i,j=1}^{n}\alpha_i\alpha_j \exp(-\gamma\|x_i - x_j\|^2)$$

$$\text{s.t. } 1-\xi_i \le y_i \sum_{j=1}^{n}\alpha_j \exp(-\gamma\|x_i - x_j\|^2), \quad \xi_i \ge 0$$

Optimization is a QP (convex) once C and γ are fixed



Hyperparameter due to non-differentiability

Discrete parameters:

- Choice of kernel
- Choice of activation function
- Network architecture parameters
- etc.

stage	output	ResNet-50	**ResNeXt-50 (32×4d)**
conv1	112×112	7×7, 64, stride 2	7×7, 64, stride 2
conv2	56×56	3×3 max pool, stride 2	3×3 max pool, stride 2
		$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128, C{=}32 \\ 1\times1, 256 \end{bmatrix} \times 3$
conv3	28×28	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256, C{=}32 \\ 1\times1, 512 \end{bmatrix} \times 4$
conv4	14×14	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512, C{=}32 \\ 1\times1, 1024 \end{bmatrix} \times 6$
conv5	7×7	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 1024 \\ 3\times3, 1024, C{=}32 \\ 1\times1, 2048 \end{bmatrix} \times 3$
	1×1	global average pool 1000-d fc, softmax	global average pool 1000-d fc, softmax
# params.		**25.5**×10^6	**25.0**×10^6
FLOPs		**4.1**×10^9	**4.2**×10^9

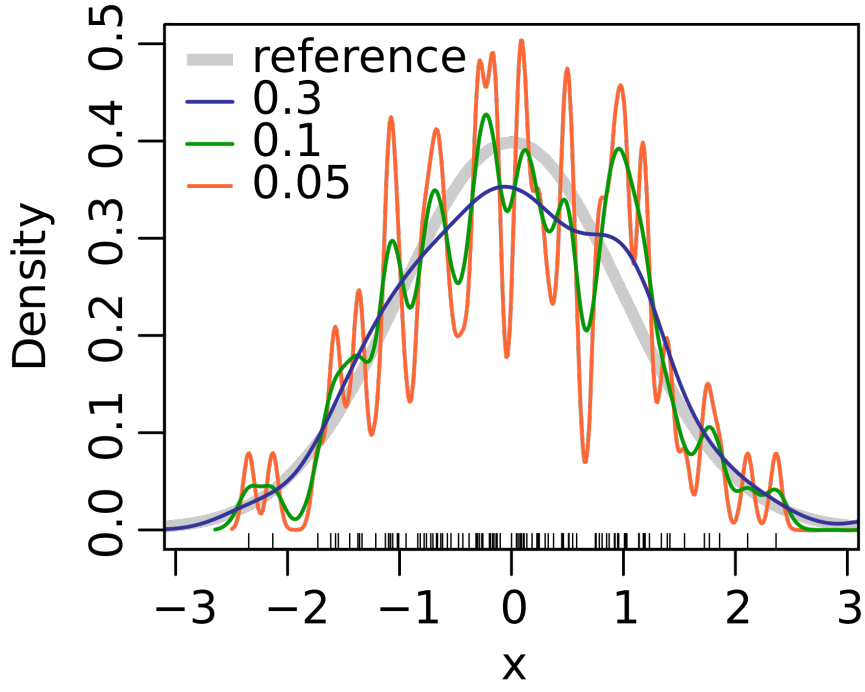
Table 1. (**Left**) ResNet-50. (**Right**) ResNeXt-50 with a 32×4d template (using the reformulation in Fig. 3(c)). Inside the brackets are the shape of a residual block, and outside the brackets is the number of stacked blocks on a stage. "C=32" suggests grouped convolutions [24] with 32 groups. *The numbers of parameters and FLOPs are similar between these two models.*

arXiv:1611.05431

Hyperparameter due to statistical considerations

Maximum likelihood of a kernel density estimate

$$p(x) = \frac{1}{n} \sum_{i=1}^{n} K_h \left(|x - x_i| \right)$$

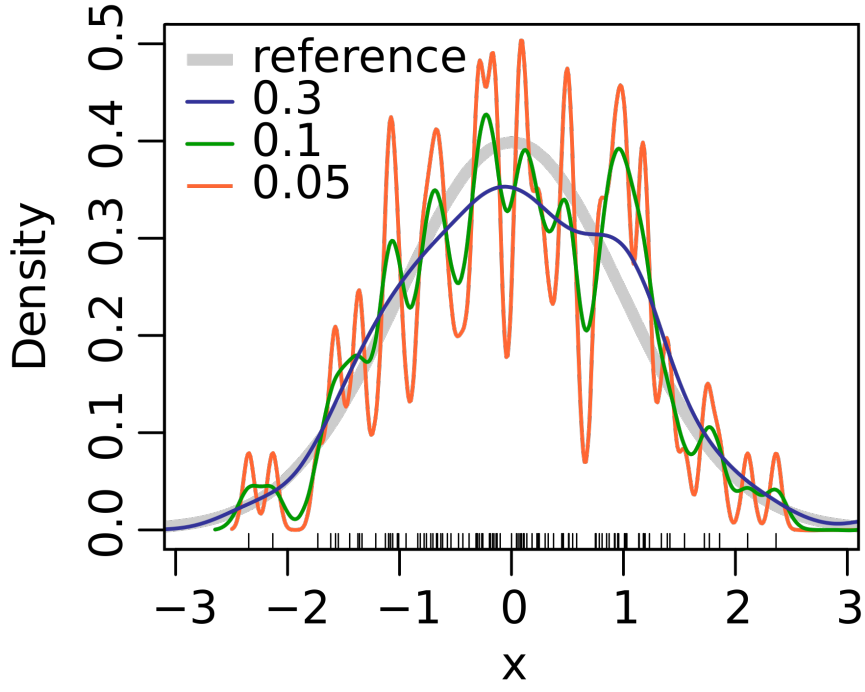


degenerate as $h \to 0$, use a train/val split

Hyperparameter due to statistical considerations

Maximum likelihood of a kernel density estimate

$$p(x) = \frac{1}{n} \sum_{i=1}^{n} K_h \left(|x - x_i| \right)$$



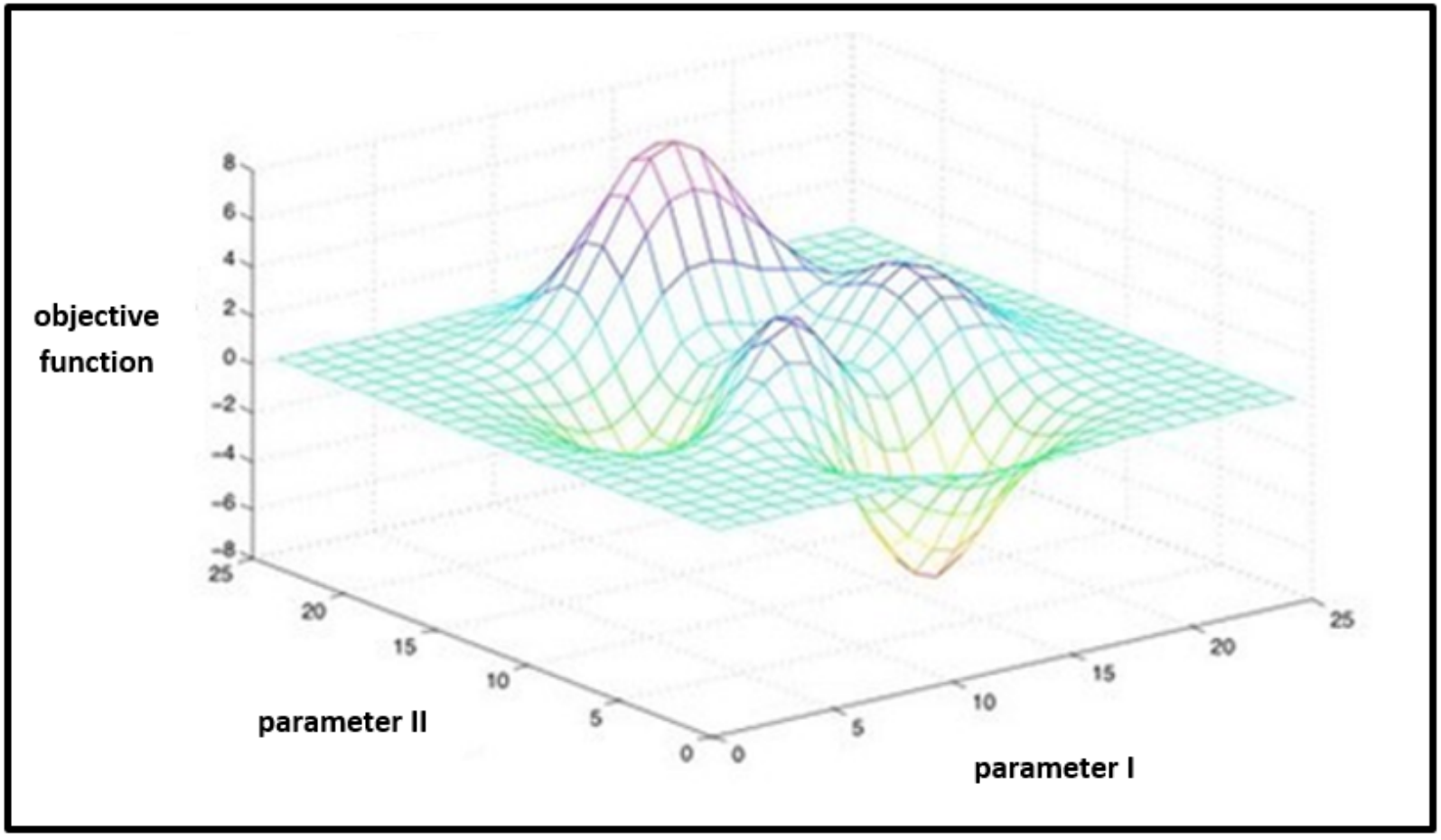
degenerate as $h \to 0$, use a train/val split

Hyper-parameter selection

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
from sklearn.svm import SVC

print(__doc__)

# Loading the Digits dataset
digits = datasets.load_digits()

# To apply an classifier on this data, we need to flatten the image, to
# turn the data in a (samples, feature) matrix:
n_samples = len(digits.images)
X = digits.images.reshape((n_samples, -1))
y = digits.target

# Split the dataset in two equal parts
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.5, random_state=0)

# Set the parameters by cross-validation
tuned_parameters = [{'kernel': ['rbf'], 'gamma': [1e-3, 1e-4],
                     'C': [1, 10, 100, 1000]},
                    {'kernel': ['linear'], 'C': [1, 10, 100, 1000]}]

scores = ['precision', 'recall']
```

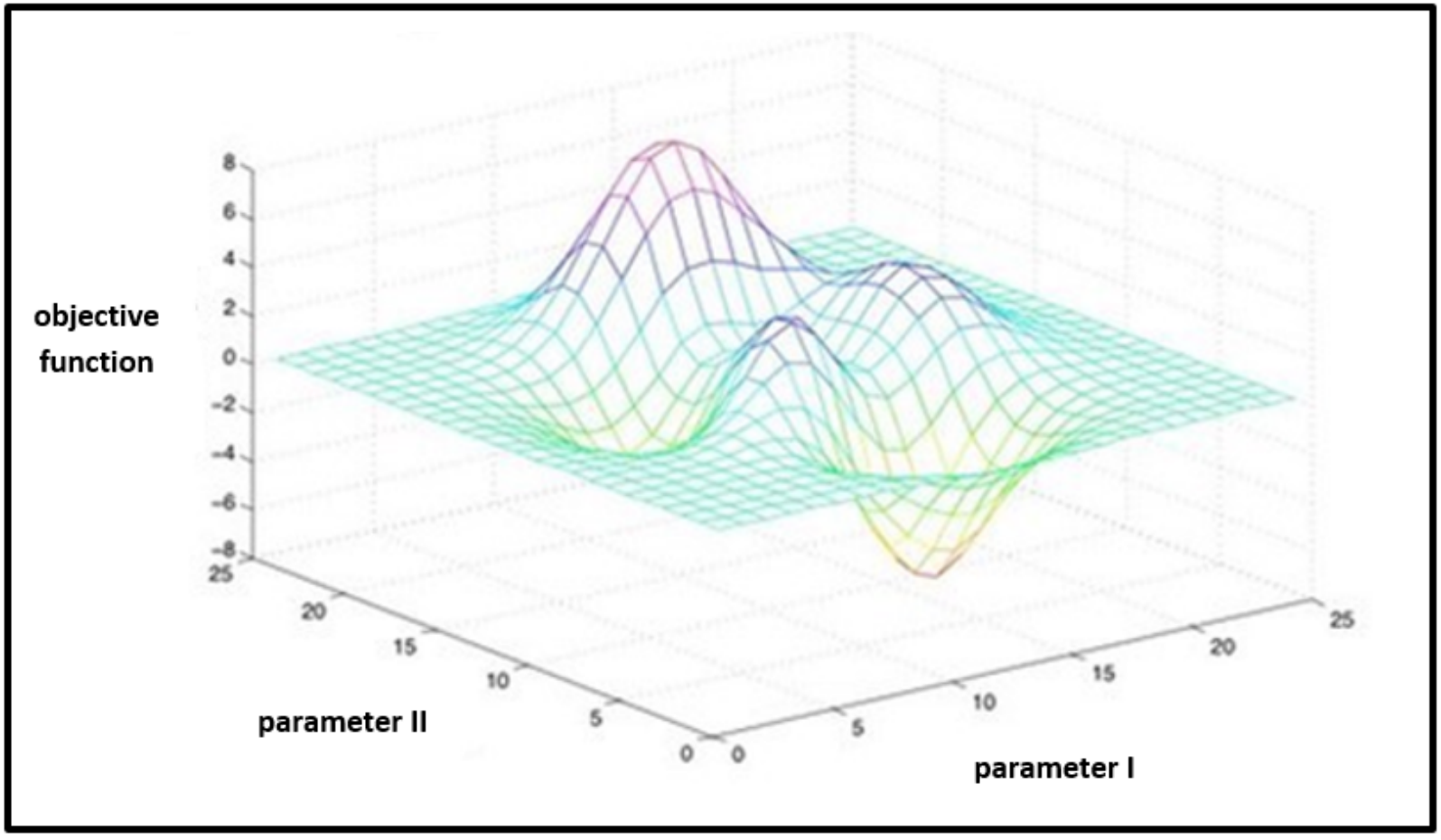
source: https://scikit-learn.org/stable/auto_examples/model_selection/plot_grid_search_digits.html

Grid search



but usually we don't spend this many evaulations, visualize the objective, or even publish the range of parameters searched

Grid search



but usually we don't spend this many evaulations, visualize the objective, or even publish the range of parameters searched

Grid search

If f is K-Lipschitz continuous,

$$|f(x_1) - f(x_2)| \leq K\|x_1 - x_2\|,$$

to guarantee

$$f(x^*) - f(\hat{x}) \leq \varepsilon$$

we need to evaluate f on a d-dimensional unit hypercube $\left(\frac{K}{\varepsilon}\right)^d$ times!

Grid search as it appears in the literature

We implemented a Variational LSTM for both the medium model of [4] (2 layers with 650 units in each layer) as well as their large model (2 layers with 1500 units in each layer). The only changes we've made to [4]'s setting are 1) using our proposed dropout variant instead of naive dropout, and 2) tuning weight decay (which was chosen to be zero in [4]). All other hyper-parameters are kept identical to [4]: learning rate decay was not tuned for our setting and is used following [4]. Dropout parameters were optimised with grid search (tying the dropout probability over the embeddings together with the one over the recurrent layers, and tying the dropout probability for the inputs and outputs together as well). These are chosen to minimise validation perplexity[3]. We further compared

NeurIPS 2016

For the primary experiments reported here, the overall network architecture is sketched in Figure 2, with details as follows: The input encoder is a 3-layer CNN with a FC+`relu` layer on top. The output decoder is a 1-layer LSTM. For the META model, the task encoder uses 1-layer CNN to encode the input and output for a single example, which are concatenated on the feature map dimension and fed through a 6-layer CNN with a FC+`relu` layer on top. Multiple I/O examples were combined with max-pooling on the final vector. All convolutional layers use a 3×3 kernel with a 64-dimensional feature map. The fully-connected and LSTM are 1024-dimensional. Different model sizes are explored later in this section. The dropout, learning rate, and batch size were optimized with grid search for each value of n using a separate set of validation tasks. Training was performed using SGD + momentum and gradient clipping using an in-house toolkit.

NeurIPS 2017

Magic parameters

The RNADE model had 512 rectified-linear hidden units and a mixture of 20 one-dimensional Gaussian components per output. Training was done by minibatch gradient descent, with 25 datapoints per minibatch, for a total of 200 epochs, each comprising 1,000 minibatches. The learning-rate was scheduled to start at 0.001 and linearly decreased to reach 0 after the last epoch. Gradient momentum with momentum factor 0.9 was used, but initiated at the beginning of the second epoch. A weight decay rate of 0.001 was applied to the input-to-hidden weight matrix only. Again, we found that multiplying the gradient of the mean output parameters by the standard deviation improves results. RNADE training was early stopped but didn't show signs of overfitting. We produced a further run

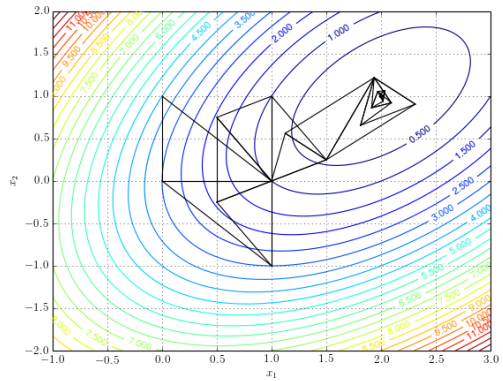
...

The RNADE model has 1024 rectified-linear hidden units and a mixture of 20 one-dimensional Gaussian components per output. Given the larger scale of this dataset hyperparameter choices were again made manually using validation data, and the same minibatch training procedures for RNADE and MoG were used as for natural image patches.

NeurIPS 2013

What optimization strategies are available to us?

- non-differentiability of function - can't just assume we can compute a gradient and use a joint gradient descent optimization
- fminunc, fminsearch (MatLab)
  - fminsearch uses a simplex search method (Nelder & Mead, 1965)

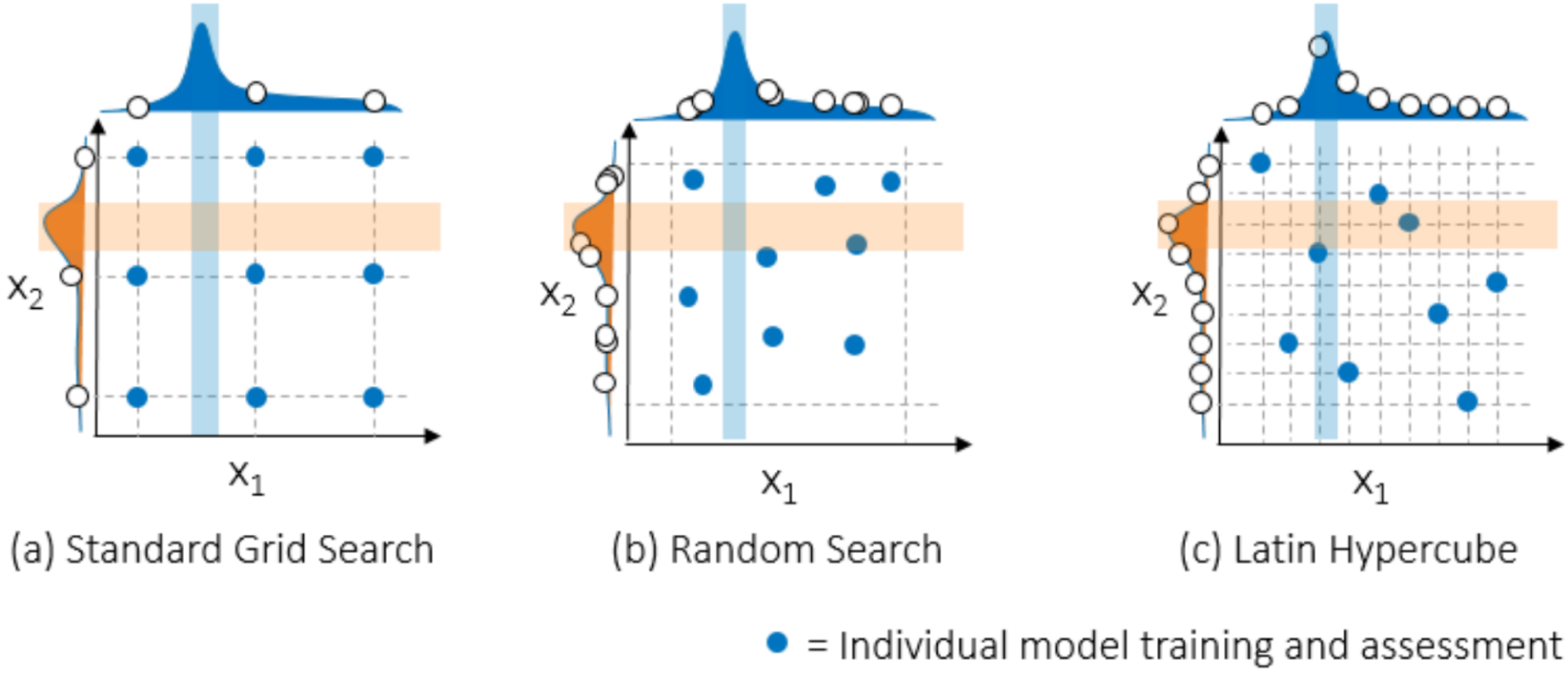


heuristic, can converge to non-stationary points (Powell, 1973)

  - fminunc numerically approximates the gradient and then use a gradient based method - extra $\mathcal{O}(d)$ factor, will not work for discontinuous functions

Random search

Can actually be not so bad sometimes - let's keep it in mind but see if we can do better



(Bergstra & Bengio, 2012; McKay, Beckman & Conover, 1979)

Genetic algorithms

Can be thought of as a form of stochastic search

- Initialize a population of points
- while (budget is not exhausted) {
  - evaluate fitness (function value) of each point in the population
  - with probability depending on fitness (many variations, diversity terms, etc.) subsample the population
  - increase the population by randomizing (many variations) around the remaining samples

  }

Many problems with the family:

- stopping criterion unclear
- can converge to local optimum
- many variations, frequently without proper analysis or theoretical guarantees
- often better, more principled methods available to solve the same optimization problem

Genetic algorithms

Can be thought of as a form of stochastic search

- Initialize a population of points
- while (budget is not exhausted) {
  - evaluate fitness (function value) of each point in the population
  - with probability depending on fitness (many variations, diversity terms, etc.) subsample the population
  - increase the population by randomizing (many variations) around the remaining samples

  }

Many problems with the family:

- stopping criterion unclear
- can converge to local optimum
- many variations, frequently without proper analysis or theoretical guarantees
- often better, more principled methods available to solve the same optimization problem

“Hyper-parameter selection with Bayesian optimization”

What do we mean when we say Bayesian optimization?

Bayesian - encode a prior distribution over functions that explicitly encodes assumptions about e.g. smoothness

Optimization - use (a criterion based on) the posterior distribution conditioned on function observations to determine where to evaluate the function next

Bayesian optimization

- Assume a prior on f
- while (budget is not exhausted) {
  - Find $\hat{x}$ that maximizes acquisition(x, posterior)
  - observe $f(\hat{x})$
  - update the posterior distribution on f

  }
- return the $\hat{x}$ with the highest observed $f(\hat{x})$

Open questions: prior/posterior representation, termination criterion, choice of acquisition function, maximization of acquisition function, ...

Bayesian optimization

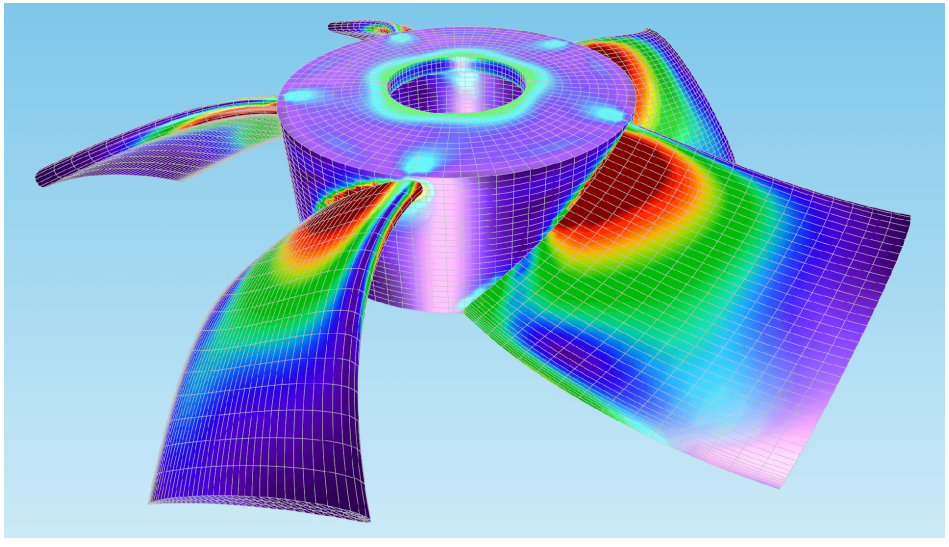
- Assume a prior on f
- while (budget is not exhausted) {
  - Find $\hat{x}$ that maximizes acquisition(x, posterior)
  - observe $f(\hat{x})$
  - update the posterior distribution on f

  }
- return the $\hat{x}$ with the highest observed $f(\hat{x})$

Open questions: prior/posterior representation, termination criterion, choice of acquisition function, maximization of acquisition function, ...

Bayesian optimization

- Frequently used in engineering design
- Prior knowledge of domain can be incorporated into the mean of the function distribution
- Iteratively optimize model parameters to maximize performance objective



When should we use Bayesian optimization?

- Function evaluations are expensive
- Gradients are unknown (though versions that can handle gradients exist)
- Number of dimensions is not too high (about 20 or less)
- State of the art for black box optimization in this setting

Acquisition function optimization

- Assume a prior on f
- while (budget is not exhausted) {
  - Find $\hat{x}$ that maximizes acquisition(x, posterior)
  - observe $f(\hat{x})$
  - update the posterior distribution on f

  }
- return the $\hat{x}$ with the highest observed $f(\hat{x})$

Solve an optimization problem by repeatedly solving optimization problems:

- Evaluation of f is very expensive + no gradients
- Acquisition function & distribution on f should be tractable so that we can maximize acquisition(x, posterior) efficiently

Acquisition function optimization

- Assume a prior on f
- while (budget is not exhausted) {
  - Find $\hat{x}$ that maximizes acquisition(x, posterior)
  - observe $f(\hat{x})$
  - update the posterior distribution on f

  }
- return the $\hat{x}$ with the highest observed $f(\hat{x})$

Solve an optimization problem by repeatedly solving optimization problems:

- Evaluation of f is very expensive + no gradients
- Acquisition function & distribution on f should be tractable so that we can maximize acquisition(x, posterior) efficiently

Gaussian Process definition

Definition (Gaussian Process)

A Gaussian process (GP) is a collection of random variables, any finite number of which have consistent Gaussian Distributions

A GP is fully specified by a mean function $\mu : \mathcal{X} \to \mathbb{R}$ and a positive definite covariance function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$

Gaussian Process marginal distribution

How is it we can compute the GP result for a finite sample?

1. Marginal distribution of a Gaussian is Gaussian

$$p(y_1) = \int p(y_1, y_2) dy_2$$

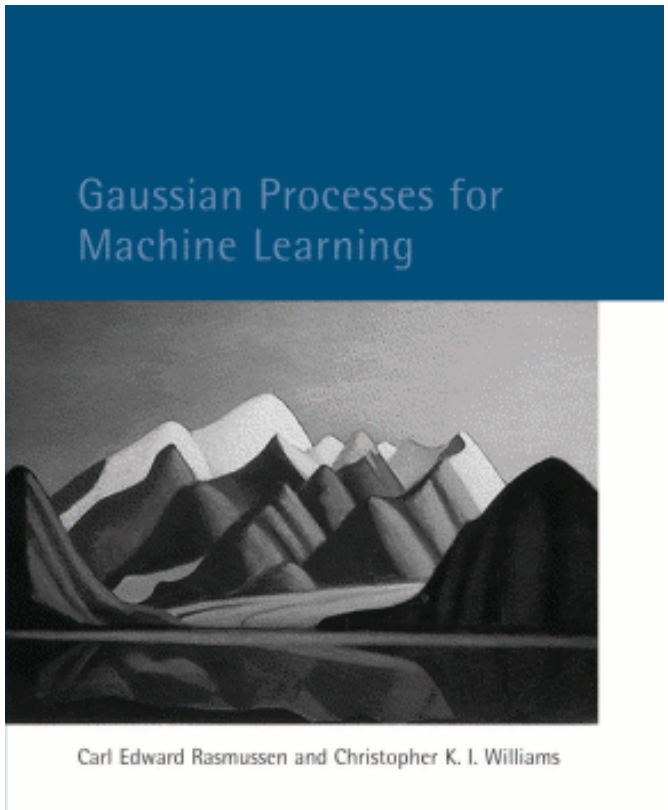
$$p(y_1, y_2) = \mathcal{N}\left(\begin{bmatrix} \mu_1 \\ \mu 2 \end{bmatrix}, \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{12}^T & \Sigma_{22} \end{bmatrix} \right) \implies p(y_1) = \mathcal{N}(\mu_1, \Sigma_{11})$$

2. Posterior distribution is Gaussian as there is a Gaussian prior, so the posterior distribution marginalized over a finite sample is Gaussian

GP key predictive equations

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

$$\begin{aligned} \mathbf{f}_*|X,\mathbf{y},X_* &\sim \mathcal{N}\big(\bar{\mathbf{f}}_*,\ \mathrm{cov}(\mathbf{f}_*)\big), \quad \text{where} \\ \bar{\mathbf{f}}_* &\triangleq \mathbb{E}[\mathbf{f}_*|X,\mathbf{y},X_*] = K(X_*,X)[K(X,X)+\sigma_n^2 I]^{-1}\mathbf{y}, \\ \mathrm{cov}(\mathbf{f}_*) &= K(X_*,X_*) - K(X_*,X)\left[K(X,X)+\sigma_n^2 I\right]^{-1}K(X,X_*) \end{aligned}$$



GP prior and posterior

GP prior and posterior

Conditional marginals of the normal distribution

Let $\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{12} & \Sigma_{22} \end{bmatrix}\right)$

Show that $p(x_2|x_1) = \mathcal{N}\left(\frac{\Sigma_{12}}{\Sigma_{11}} x_1, \Sigma_{22} - \frac{\Sigma_{12}^2}{\Sigma_{11}}\right)$

Predictive equations with non-zero mean

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

$$\begin{aligned} \mathbf{f}_*|X,\mathbf{y},X_* &\sim \mathcal{N}\big(\bar{\mathbf{f}}_*,\ \mathrm{cov}(\mathbf{f}_*)\big), \quad \text{where} \\ \bar{\mathbf{f}}_* &\triangleq \mathbb{E}[\mathbf{f}_*|X,\mathbf{y},X_*] = K(X_*,X)[K(X,X)+\sigma_n^2 I]^{-1}\mathbf{y}, \\ \mathrm{cov}(\mathbf{f}_*) &= K(X_*,X_*) - K(X_*,X)\left[K(X,X)+\sigma_n^2 I\right]^{-1}K(X,X_*) \end{aligned}$$

What are the equations with non-zero mean?

Predictive equations with non-zero mean

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

$$\begin{aligned} \mathbf{f}_*|X,\mathbf{y},X_* &\sim \mathcal{N}\big(\bar{\mathbf{f}}_*,\ \mathrm{cov}(\mathbf{f}_*)\big), \quad \text{where} \\ \bar{\mathbf{f}}_* &\triangleq \mathbb{E}[\mathbf{f}_*|X,\mathbf{y},X_*] = K(X_*,X)[K(X,X)+\sigma_n^2 I]^{-1}\mathbf{y}, \\ \mathrm{cov}(\mathbf{f}_*) &= K(X_*,X_*) - K(X_*,X)\left[K(X,X)+\sigma_n^2 I\right]^{-1}K(X,X_*) \end{aligned}$$

What are the equations with non-zero mean?

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

variance is unchanged.

Marginal log-likelihood of a GP

What is the marginal log-likelihood of a GP: $\log p(\mathbf{y}|X, \theta)$? ($\theta$ are the parameters of the covariance function.)

Hint: The multivariate Gaussian distribution has the form:

$$p(x) = \frac{1}{(2\pi)^{d/2}\sqrt{\det(\Sigma)}} e^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)}$$

Marginal log-likelihood of a GP

What is the marginal log-likelihood of a GP: $\log p(\mathbf{y}|X,\theta)$? ($\theta$ are the parameters of the covariance function.)

$$\begin{aligned}\log p(\mathbf{y}|X,\theta) = &-\frac{1}{2}(\mathbf{y}-\mu(X))^T(K+\sigma_n^2 I)^{-1}(\mathbf{y}-\mu(X)) \\ &-\frac{1}{2}\log\det(K+\sigma_n^2 I) - \frac{n}{2}\log 2\pi\end{aligned}$$

Note that the θ are hidden in K.

How should we optimize μ?

What if we want to use MAP to set θ?

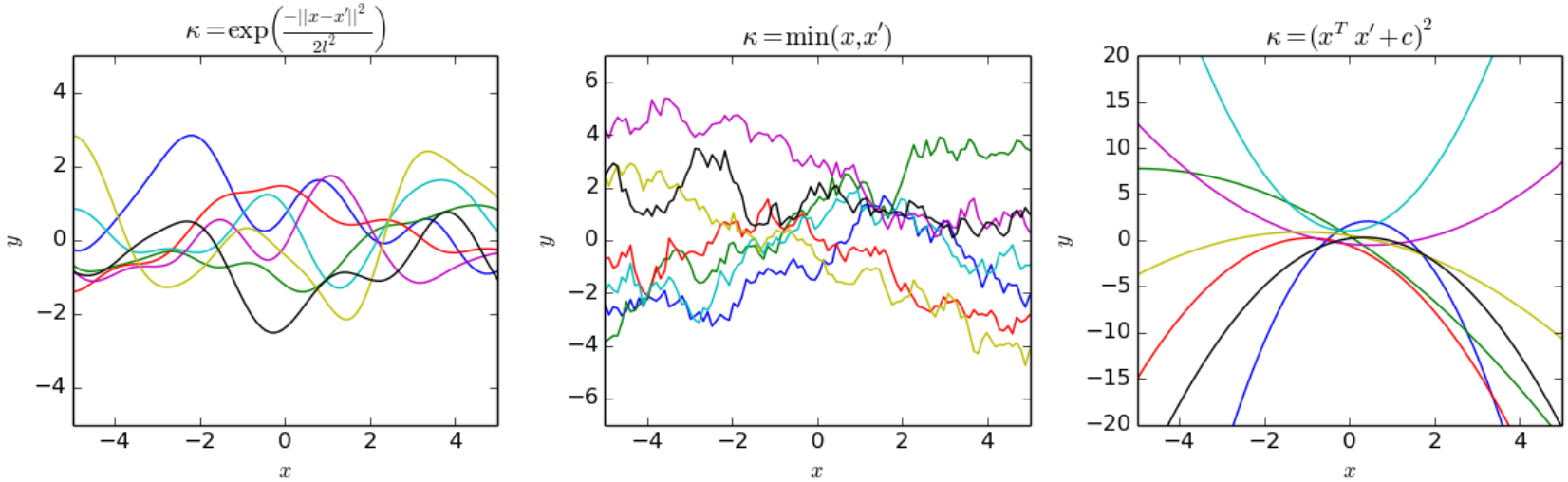
Kernel ridge regression

Kernel ridge regression (KRR) can be thought of as L_2 regularized least squares after projecting your data through a non-linear (possibly infinite dimensional) map $x \mapsto \phi(x) \in \mathcal{H}$, and predicting by an inner product $\langle \phi(x), w \rangle_{\mathcal{H}}$.

What is the solution to KRR? (You will have to make use of the representer theorem for kernels.)

How does it compare to a GP?

More covariance functions



Covariance function needs only be positive semi-definite.

The same rules as kernel construction apply:

- $k = \sum_i \alpha_i k_i$ is a valid covariance function for $\alpha \geq \mathbf{0}$
- $k = k_1 \times k_2$ is a valid covariance function
- $k(x, x') = [x = x']$ is a valid covariance function
- $k(x, x') = \begin{cases} 1 & \text{if } x \in A \wedge x' \in A \\ 0 & \text{otherwise} \end{cases}$ is a valid covariance function for any fixed set A
- $k = C$ is a valid covariance function for any $C \geq 0$

Termination criteria

- $\|x_t - x_{t-1}\| \leq \varepsilon$ (Lorenz et al., 2015)
  missing theoretical justification

- Threshold acquisition function, e.g. (Nguyen et al., ACML 2017)
  regret minimization justification for UCB, EI, ...

Covariance effect on Bayesian optimization

Case I:

- Assume $K \to \lambda I$: What are the effects on Bayesian optimization?

For "probability of improvement" (function maximization version)

$$\alpha(x) = \Phi\left(\frac{\mu(x) - f(\hat{x}) - \psi}{\sigma(x)}\right)$$

$$= \Phi\left(\frac{\overbrace{K(x,X)[K(X,X)+\sigma_n^2 I]^{-1}\mathbf{y}}^{=0 \text{ or prior function}} - f(\hat{x}) - \psi}{\sqrt{\underbrace{K(x,x)}_{=\lambda} - \underbrace{K(x,X)[K(X,X)+\sigma_n^2 I]^{-1}K(X,x)}_{=0}}}\right)$$

$$= \Phi\left((-f(\hat{x}) - \psi)/\sqrt{\lambda}\right)$$

which for a flat prior does not depend on $x \implies$ Bayesian optimization degenerates to exhaustive search!

$K \to I$ is sound, but computationally inefficient
(similar story for other acquisition functions)

Covariance effect on Bayesian optimization

Case I:

- Assume $K \to \lambda I$: What are the effects on Bayesian optimization?

For "probability of improvement" (function maximization version)

$$\begin{aligned}
\alpha(x) =& \Phi\left(\frac{\mu(x) - f(\hat{x}) - \psi}{\sigma(x)}\right) \\
=& \Phi\left(\frac{\overbrace{K(x,X)[K(X,X)+\sigma_n^2 I]^{-1}\mathbf{y}}^{=0 \text{ or prior function}} - f(\hat{x}) - \psi}{\sqrt{\underbrace{K(x,x)}_{=\lambda} - \underbrace{K(x,X)[K(X,X)+\sigma_n^2 I]^{-1}K(X,x)}_{=0}}}\right) \\
=& \Phi\left((-f(\hat{x}) - \psi)/\sqrt{\lambda}\right)
\end{aligned}$$

which for a flat prior does not depend on $x \implies$ Bayesian optimization degenerates to exhaustive search!

$K \to I$ is sound, but computationally inefficient
(similar story for other acquisition functions)

Covariance effect on Bayesian optimization

Case I:

- Assume $K \to \lambda I$: What are the effects on Bayesian optimization?

For "probability of improvement" (function maximization version)

$$\begin{aligned}
\alpha(x) =& \Phi\left(\frac{\mu(x) - f(\hat{x}) - \psi}{\sigma(x)}\right) \\
=& \Phi\left(\frac{\overbrace{K(x,X)[K(X,X)+\sigma_n^2 I]^{-1}\mathbf{y}}^{=0 \text{ or prior function}} - f(\hat{x}) - \psi}{\sqrt{\underbrace{K(x,x)}_{=\lambda} - \underbrace{K(x,X)[K(X,X)+\sigma_n^2 I]^{-1}K(X,x)}_{=0}}}\right) \\
=& \Phi\left((-f(\hat{x}) - \psi)/\sqrt{\lambda}\right)
\end{aligned}$$

which for a flat prior does not depend on $x \implies$ Bayesian optimization degenerates to exhaustive search!

$K \to I$ is sound, but computationally inefficient
(similar story for other acquisition functions)

Covariance effect on Bayesian optimization

Case II:

- Assume $k(x, x') \to \lambda$ for all x, x': Effects?

Probability of improvement (maximization version)

$$\alpha(x) = \Phi\left(\frac{\overbrace{K(x, X)[K(X, X) + \sigma_n^2 I]^{-1}\mathbf{y}}^{\to \text{mean}(y)} - f(\hat{x}) - \psi}{\sqrt{\underbrace{K(x, x)}_{=\lambda} - \underbrace{K(x, X)[K(X, X) + \sigma_n^2 I]^{-1}K(X, x)}_{\to \lambda \text{ from below}}}}\right)$$

$\text{mean}(y) \leq f(\hat{x}) := \max(y)$ so numerator is negative. Denominator $\to \infty \implies \alpha(x) \to 0$ and we terminate immediately.

$k(x, x') \to \lambda$ is unsound, but very fast (I can just as easily return a random x, though)

Covariance effect on Bayesian optimization

Case II:

- Assume $k(x, x') \to \lambda$ for all x, x': Effects?

Probability of improvement (maximization version)

$$\alpha(x) = \Phi\left(\frac{\overbrace{K(x,X)[K(X,X)+\sigma_n^2 I]^{-1}\mathbf{y}}^{\to \text{mean}(y)} - f(\hat{x}) - \psi}{\sqrt{\underbrace{K(x,x)}_{=\lambda} - \underbrace{K(x,X)[K(X,X)+\sigma_n^2 I]^{-1}K(X,x)}_{\to \lambda \text{ from below}}}}\right)$$

$\text{mean}(y) \leq f(\hat{x}) := \max(y)$ so numerator is negative. Denominator $\to \infty \implies \alpha(x) \to 0$ and we terminate immediately.

$k(x, x') \to \lambda$ is unsound, but very fast (I can just as easily return a random x, though)

Covariance effect on Bayesian optimization

Case II:

- Assume $k(x, x') \to \lambda$ for all x, x': Effects?

  Probability of improvement (maximization version)

$$\alpha(x) = \Phi\left(\frac{\overbrace{K(x,X)[K(X,X)+\sigma_n^2 I]^{-1}\mathbf{y}}^{\to \text{mean}(y)} - f(\hat{x}) - \psi}{\sqrt{\underbrace{K(x,x)}_{=\lambda} - \underbrace{K(x,X)[K(X,X)+\sigma_n^2 I]^{-1}K(X,x)}_{\to \lambda \text{ from below}}}}\right)$$

  $\text{mean}(y) \leq f(\hat{x}) := \max(y)$ so numerator is negative. Denominator $\to \infty \implies \alpha(x) \to 0$ and we terminate immediately.

$k(x, x') \to \lambda$ is unsound, but very fast (I can just as easily return a random x, though)

Covariance effect on Bayesian optimization

Case II:

- Assume $k(x, x') \to \lambda$ for all x, x': Effects?

  Probability of improvement (maximization version)

$$\alpha(x) = \Phi\left(\frac{\overbrace{K(x,X)[K(X,X)+\sigma_n^2 I]^{-1}\mathbf{y}}^{\to \text{mean}(y)} - f(\hat{x}) - \psi}{\sqrt{\underbrace{K(x,x)}_{=\lambda} - \underbrace{K(x,X)[K(X,X)+\sigma_n^2 I]^{-1}K(X,x)}_{\to \lambda \text{ from below}}}}\right)$$

  $\text{mean}(y) \leq f(\hat{x}) := \max(y)$ so numerator is negative. Denominator $\to \infty \implies \alpha(x) \to 0$ and we terminate immediately.

$k(x, x') \to \lambda$ is unsound, but very fast (I can just as easily return a random x, though)

Covariance selection in Bayesian optimization

Look for a Goldilocks covariance:

1. Models the prior belief on the function well in order to speed up computation.

2. Does not overestimate its confidence with large covariances.

Impact on parameter settings:

- Most typically set with maximum likelihood

- Consider maximum *a posteriori*, where prior on covariance parameters biases towards low off-diagonal covariances

Covariance selection in Bayesian optimization

Look for a Goldilocks covariance:

1. Models the prior belief on the function well in order to speed up computation.

2. Does not overestimate its confidence with large covariances.

Impact on parameter settings:

- Most typically set with maximum likelihood

- Consider maximum *a posteriori*, where prior on covariance parameters biases towards low off-diagonal covariances

Covariance selection in Bayesian optimization

Look for a Goldilocks covariance:

1. Models the prior belief on the function well in order to speed up computation.

2. Does not overestimate its confidence with large covariances.

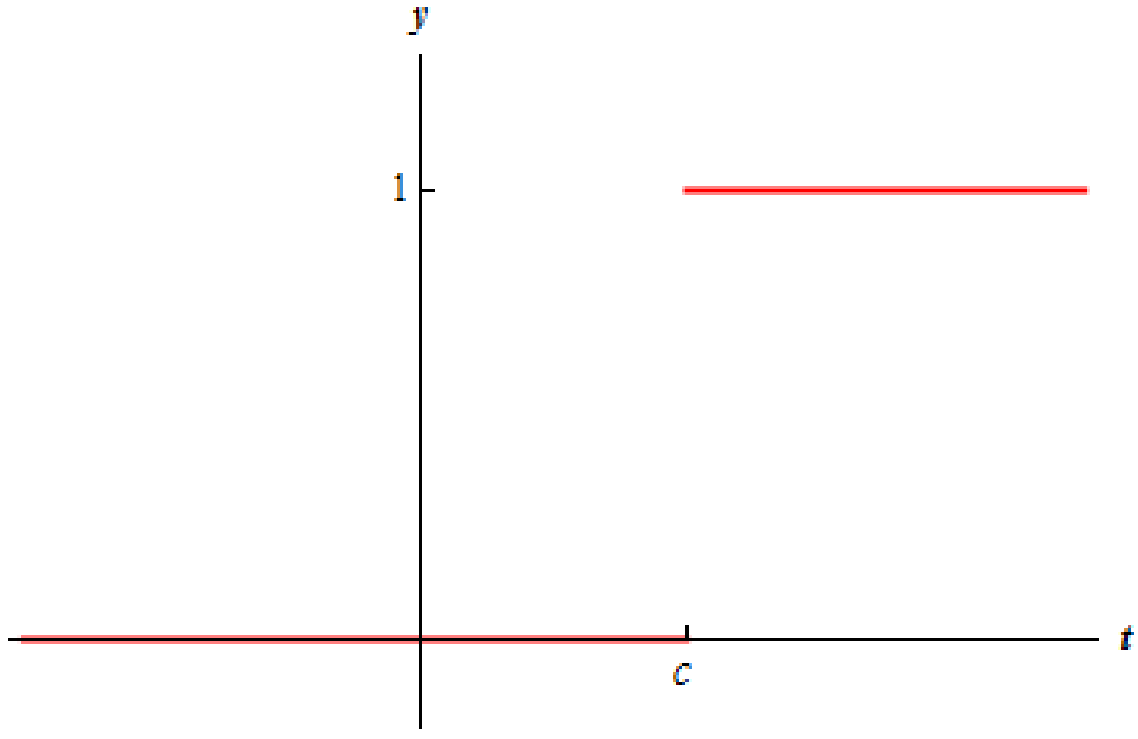
Impact on parameter settings:

- Most typically set with maximum likelihood

- Consider maximum *a posteriori*, where prior on covariance parameters biases towards low off-diagonal covariances

Structured parameter spaces

Consider the case that we are asked to optimize the step function $f(t) = [t > c]$:



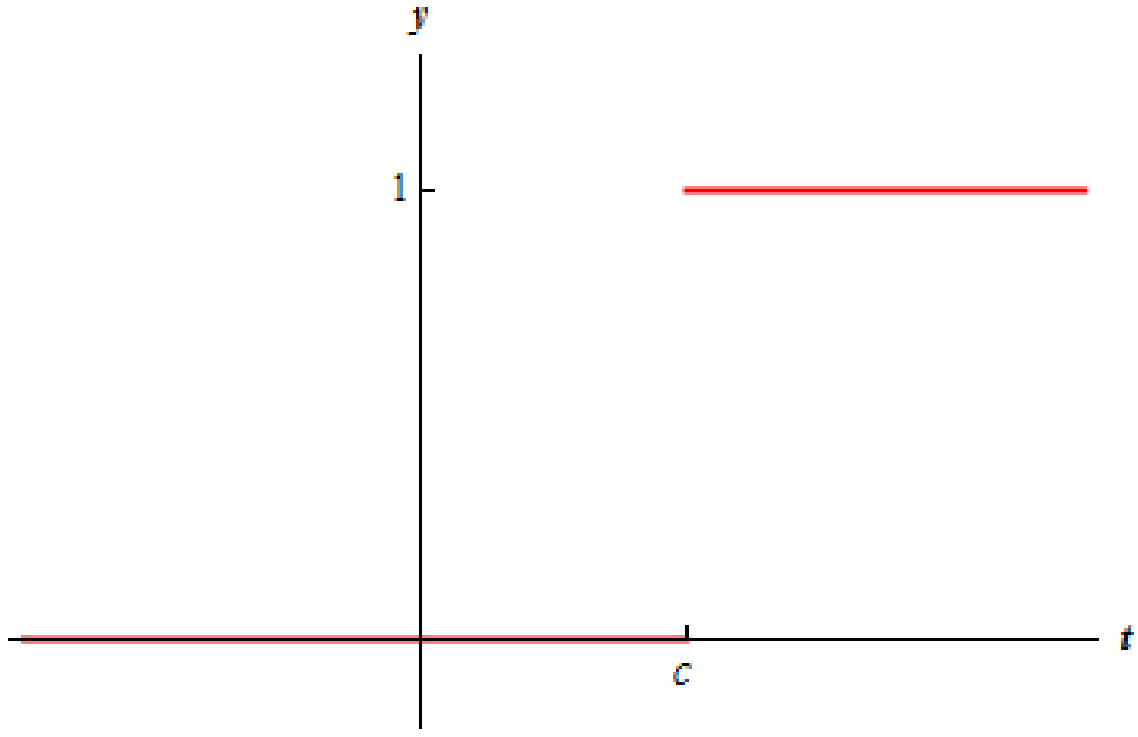
The default behavior in e.g. GPyOpt is to pretend we have a continuous variable.

Is this a good idea?

What if we need to optimize $f_1(x_2)[t \leq c] + f_2(x_2)[t > c]$?

Structured parameter spaces

Consider the case that we are asked to optimize the step function $f(t) = [t > c]$:



The default behavior in e.g. GPyOpt is to pretend we have a continuous variable.

Is this a good idea?

What if we need to optimize $f_1(x_2)[t \leq c] + f_2(x_2)[t > c]$?

K-Lipschitz functions

You are optimizing a function and you expect it is K-Lipschitz for some (upper bound on) K. How would this influence the design of your covariance function?

Piecewise linear functions

Suppose you think your function is piecewise linear. What does that imply about the choice of a covariance function? Your acquisition function?

Periodic data

Suppose your data have some known structure, e.g. periodic. Should you modify your acquisition function to better learn the parameters of the covariance?

Local optima of the acquisition function

Select an acquisition function, e.g. UCB, and determine how the number of local optima grow with the number of observed samples.

Summary

- Bayesian optimization defines state of the art in many settings of model selection / AutoML / Neural Architecture Search
- You now know (or know where to look up) the predictive equations for Gaussian processes
- You know intuititvely what the desired properties of a covariance function are (and some of their standard choices)
- You can set a better termination criterion than is implemented in most Bayesian optimization software packages
- You can begin to design covariance functions and optimization strategies for structured parameter spaces

Summary

- Bayesian optimization defines state of the art in many settings of model selection / AutoML / Neural Architecture Search
- You now know (or know where to look up) the predictive equations for Gaussian processes
- You know intuititvely what the desired properties of a covariance function are (and some of their standard choices)
- You can set a better termination criterion than is implemented in most Bayesian optimization software packages
- You can begin to design covariance functions and optimization strategies for structured parameter spaces

Summary

- Bayesian optimization defines state of the art in many settings of model selection / AutoML / Neural Architecture Search
- You now know (or know where to look up) the predictive equations for Gaussian processes
- You know intuititvely what the desired properties of a covariance function are (and some of their standard choices)
- You can set a better termination criterion than is implemented in most Bayesian optimization software packages
- You can begin to design covariance functions and optimization strategies for structured parameter spaces

Summary

- Bayesian optimization defines state of the art in many settings of model selection / AutoML / Neural Architecture Search
- You now know (or know where to look up) the predictive equations for Gaussian processes
- You know intuititvely what the desired properties of a covariance function are (and some of their standard choices)
- You can set a better termination criterion than is implemented in most Bayesian optimization software packages
- You can begin to design covariance functions and optimization strategies for structured parameter spaces

Summary

- Bayesian optimization defines state of the art in many settings of model selection / AutoML / Neural Architecture Search
- You now know (or know where to look up) the predictive equations for Gaussian processes
- You know intuititvely what the desired properties of a covariance function are (and some of their standard choices)
- You can set a better termination criterion than is implemented in most Bayesian optimization software packages
- You can begin to design covariance functions and optimization strategies for structured parameter spaces

You have what it takes to apply Bayesian optimization to your model selection problems, and make methodological contributions to the Bayesian optimization literature

Resources (non exhaustive)

Video tutorials: (this lecture borrowed heavily from Richard Turner & Javier Gonzalez)

- David MacKay
- Richard Turner
- Javier Gonzalez
- Peter Frazier
- Cedric Archambeau
- Hutter & Vanschoren

Literature:

- Rasmussen & Williams: Gaussian Processes for Machine Learning, MIT Press, 2006. (available online)
- Brochu et al.: A Tutorial on Bayesian Optimization ..., arXiv:1012.2599.
- Frazier: A Tutorial on Bayesian Optimization, arXiv:1807.02811.
- Elsken et al.: Neural Architecture Search: A Survey, JMLR, 2019.

Software (non-exhaustive)

- `https://github.com/fmfn/BayesianOptimization` - small and easy to get started
- GPflow: Gaussian Processes in TensorFlow - `https://github.com/GPflow/GPflow`
- GPyOpt - `https://github.com/SheffieldML/GPyOpt`
- Spearmint - `https://github.com/HIPS/Spearmint`
- SMAC - `https://github.com/automl/SMAC3`
- etc.

No offense intended if your favorite is not on the list. Google, look at other tutorials, ask your colleagues what they think...

We're hiring PhD students!

Just send your CV + a paper / masters thesis (if possible) + motivation statement to matthew.blaschko@esat.kuleuven.be

Thank you.

Matthew Blaschko
http://homes.esat.kuleuven.be/~mblaschk/
matthew.blaschko@esat.kuleuven.be

Amal Rannen Triki
http://amal.rannen.triki.me/
amal.rannen@esat.kuleuven.be