# Raiders of the Lost Architecture: Kernels for Bayesian Optimization in Conditional Parameter Spaces

Kevin Swersky, David Duvenaud, Jasper Snoek, Frank Hutter, Michael A Osborne

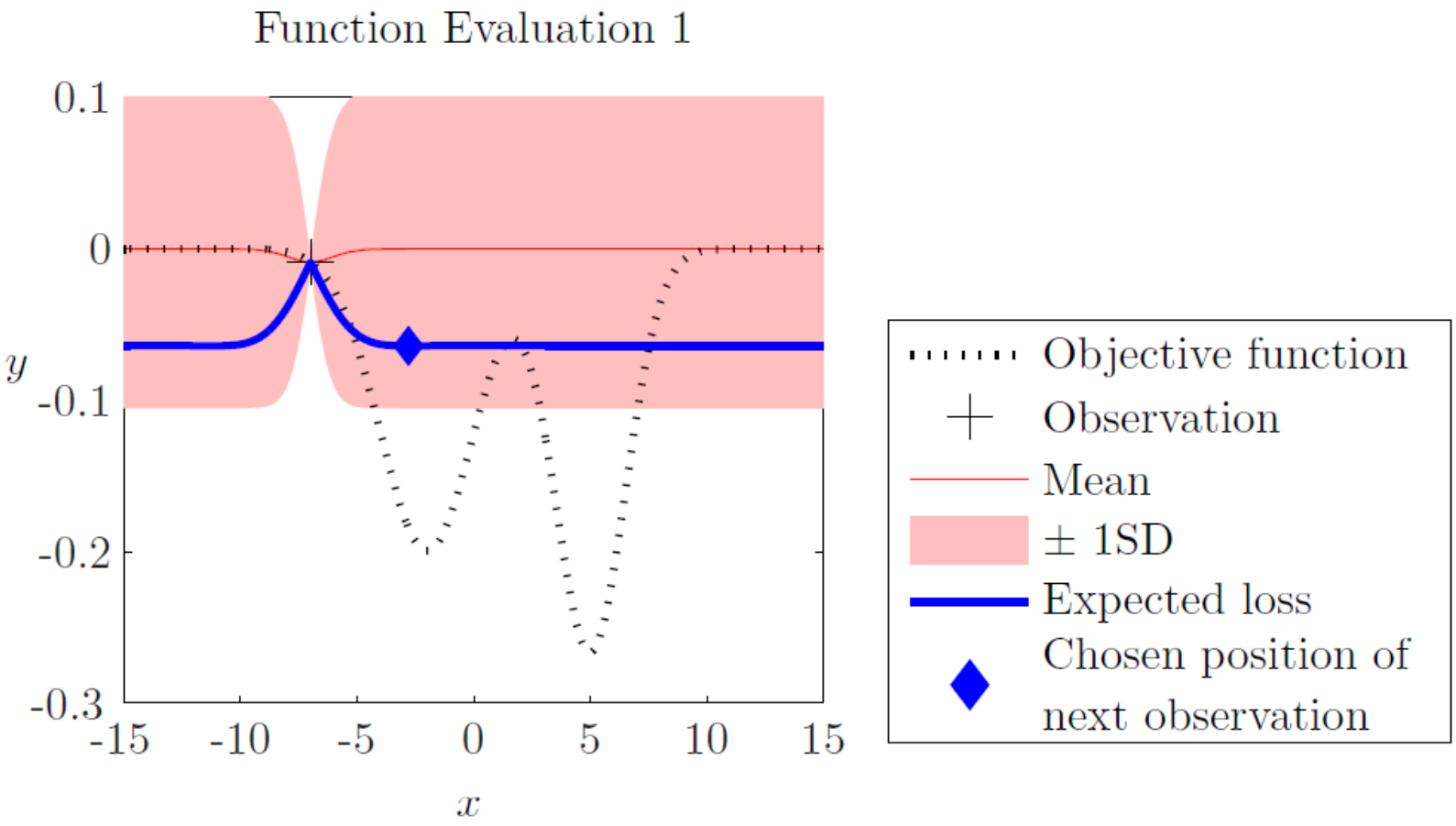
February 23, 2014

In Bayesian optimization, we often search over structures with differing numbers of parameters.

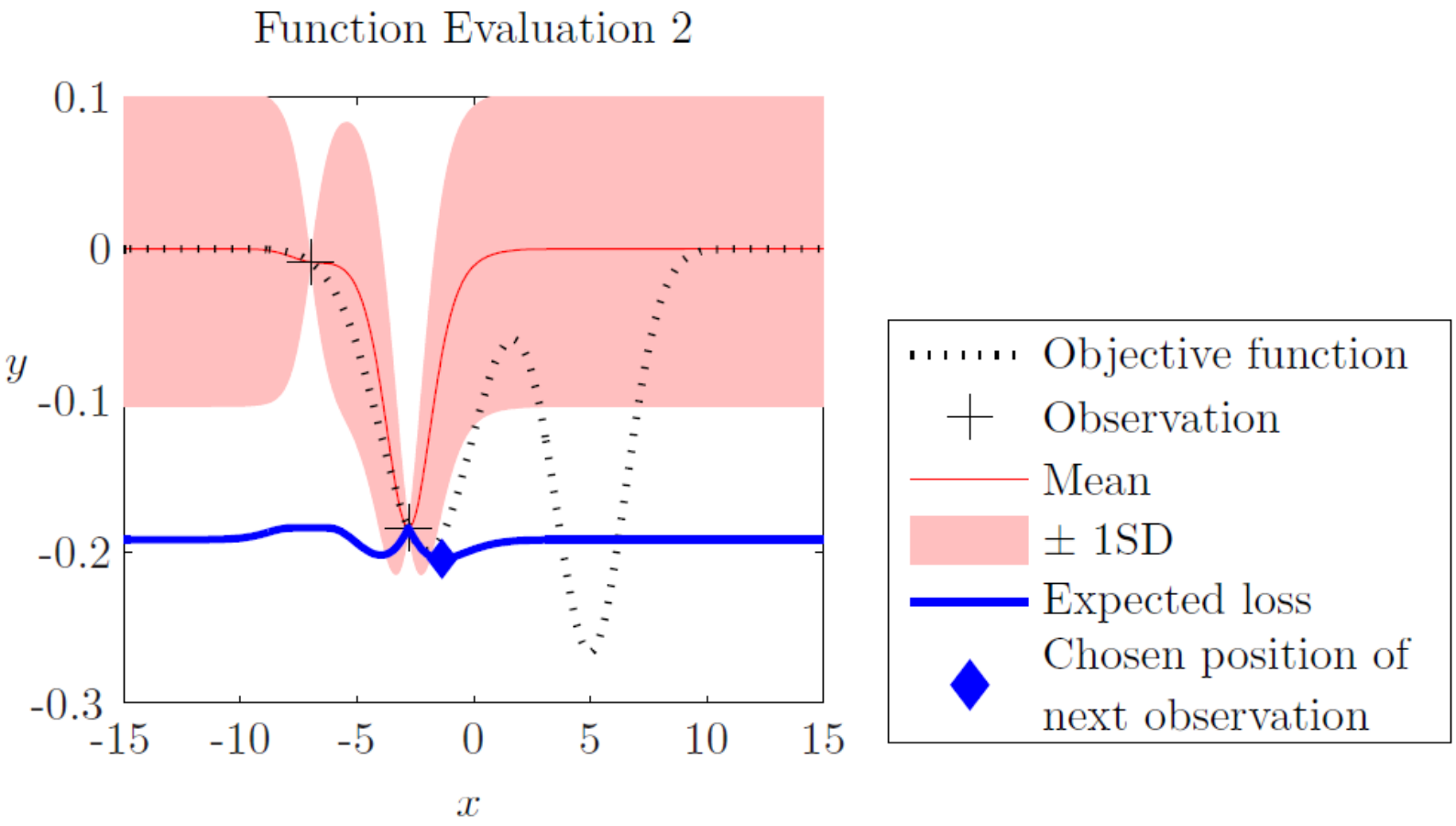
e.g. We may wish to search over neural network architectures with an unknown number of layers.

To relate performance data gathered for different architectures, we define a new kernel for conditional parameter spaces that explicitly includes information about which parameters are relevant in a given structure.

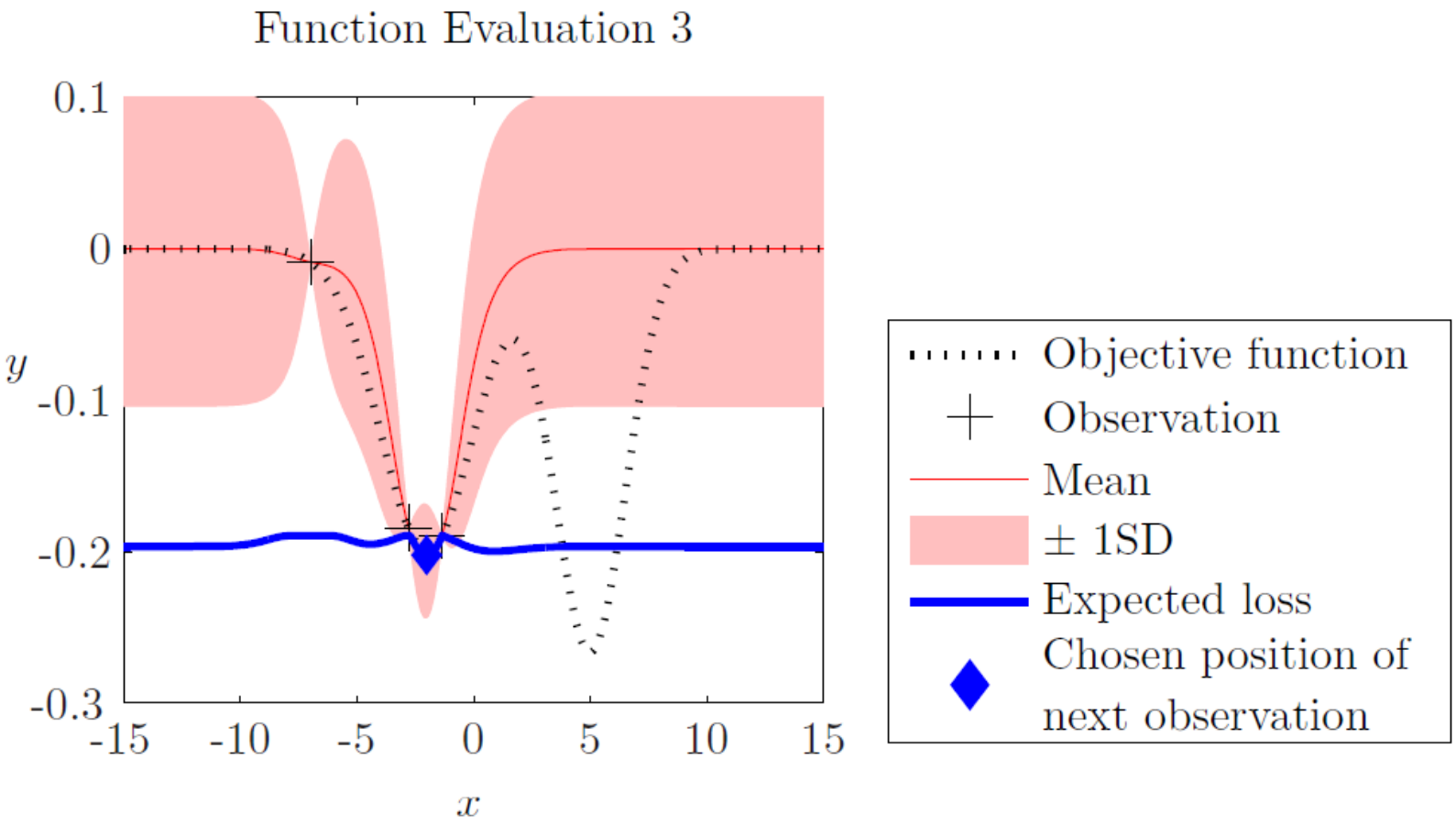
# Bayesian optimization is efficient for expensive and multimodal functions.



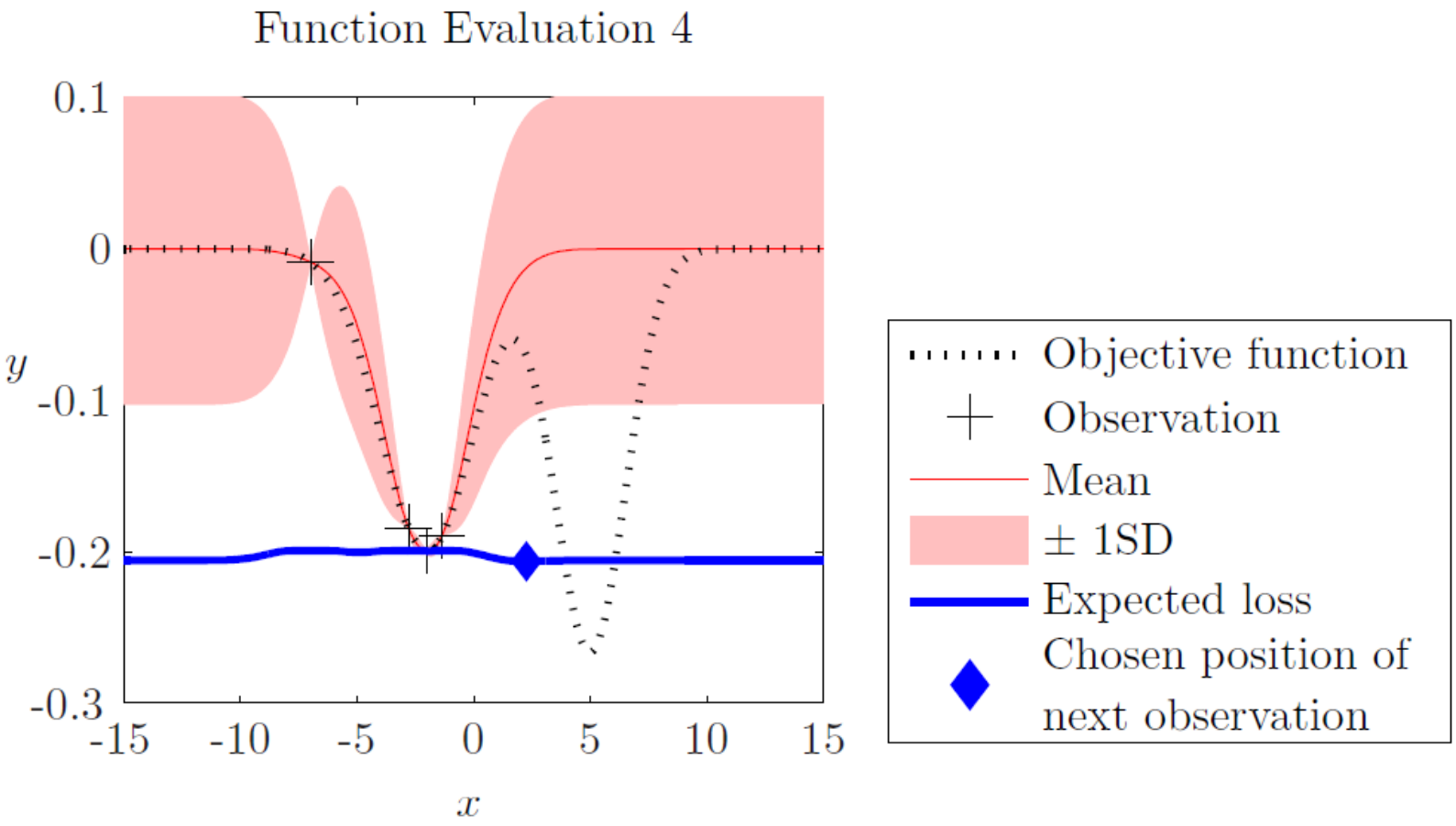
# Bayesian optimization is efficient for expensive and multimodal functions.



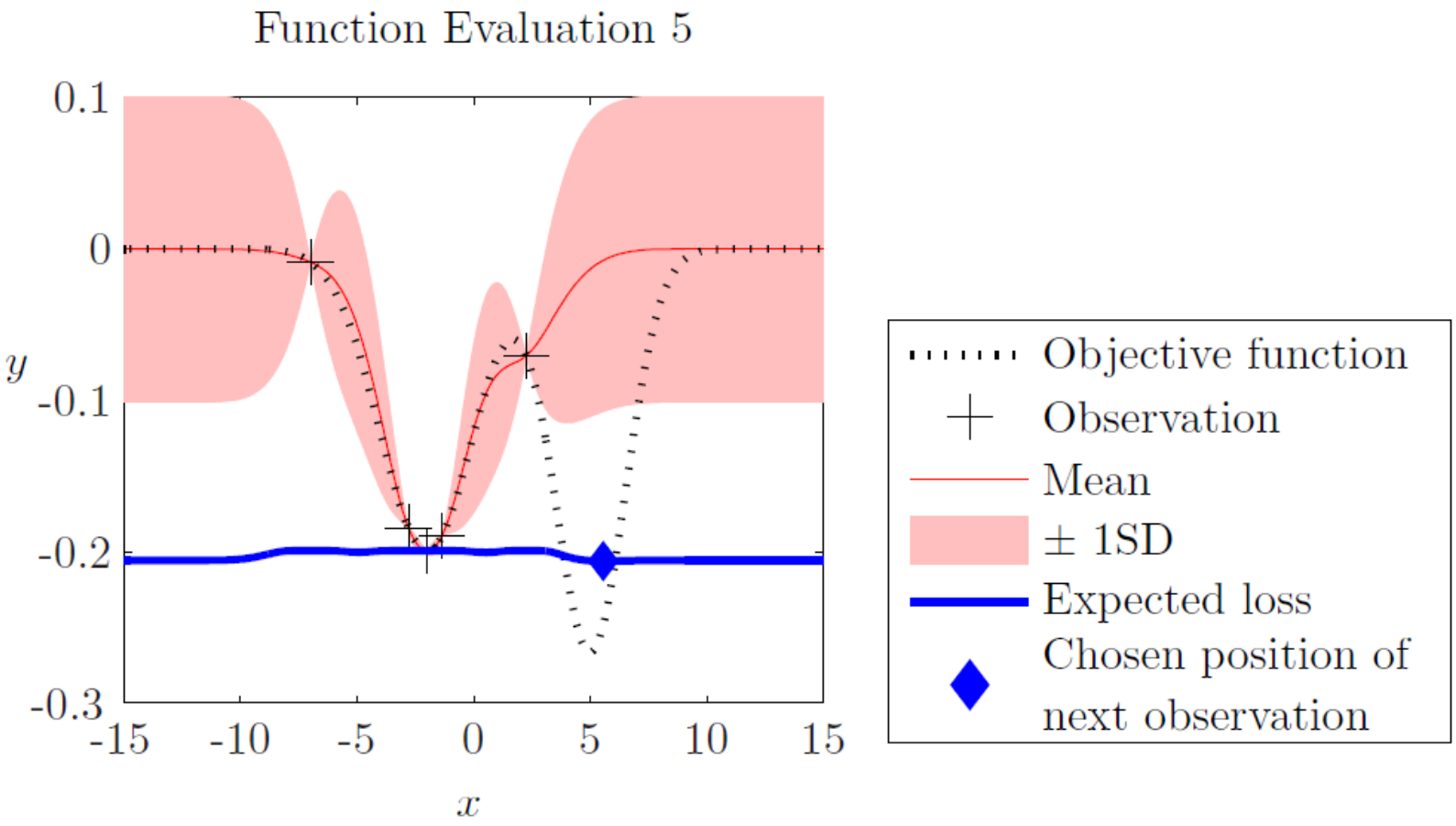
# Bayesian optimization is efficient for expensive and multimodal functions.



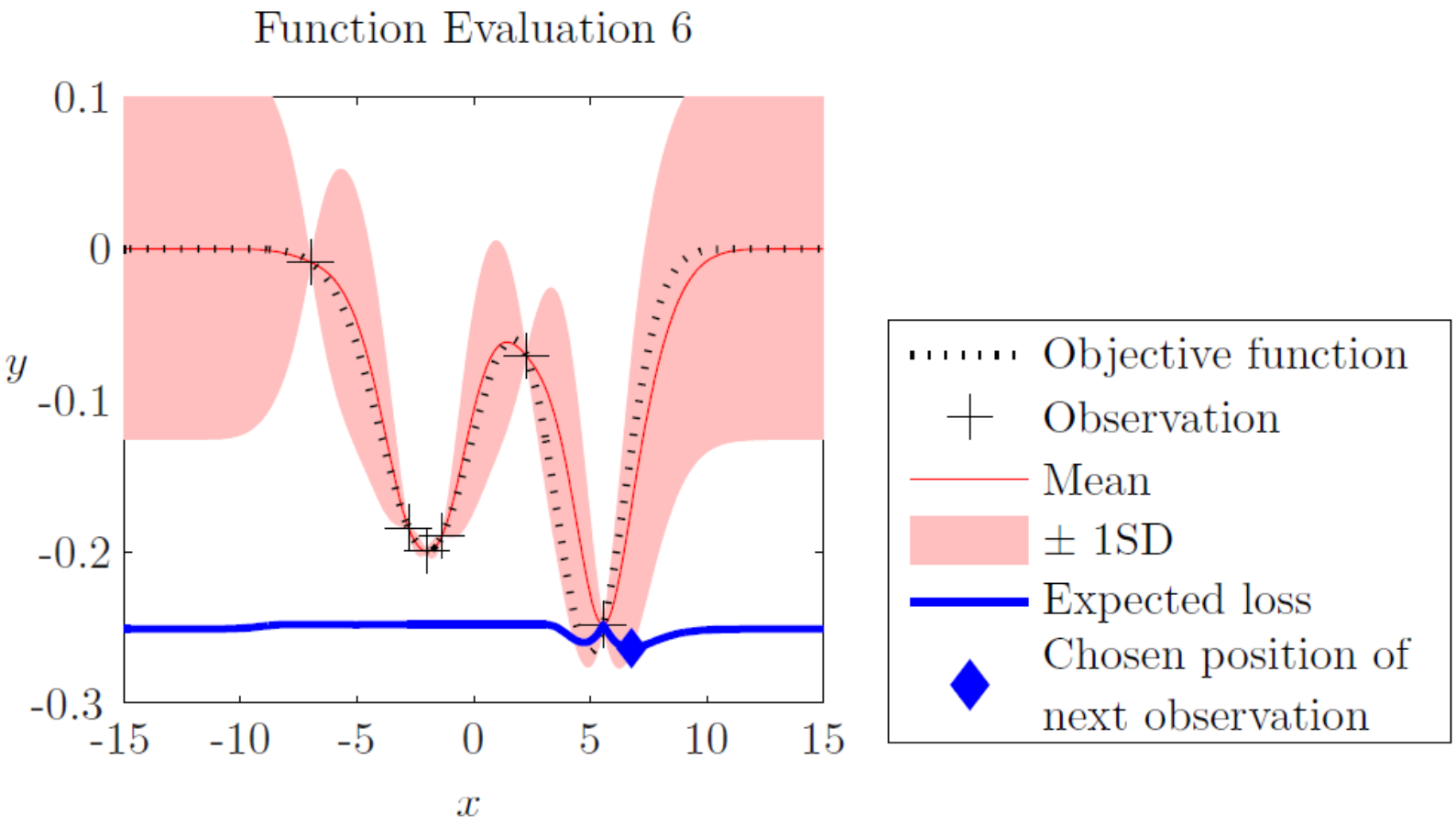
# Bayesian optimization is efficient for expensive and multimodal functions.



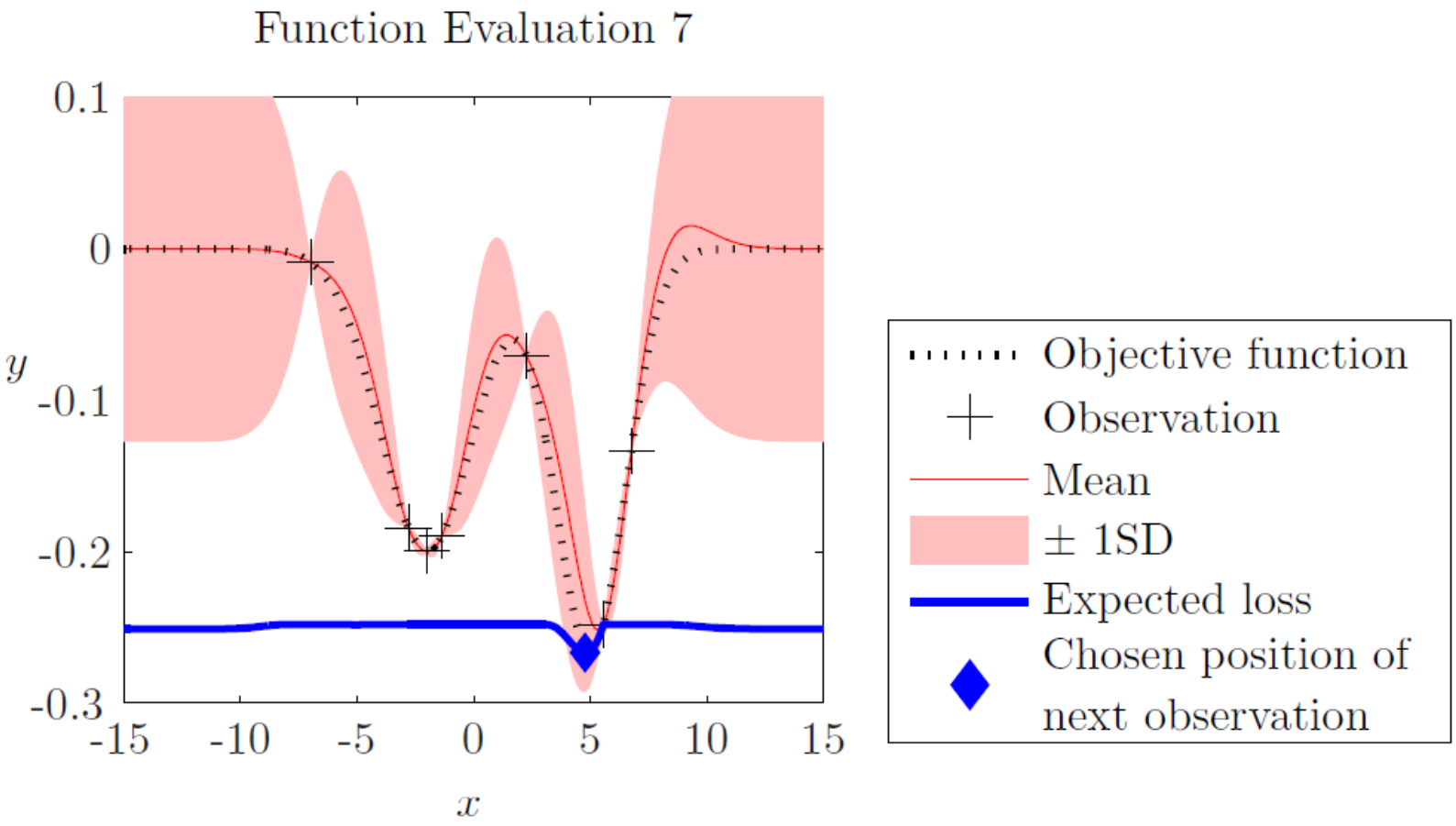
# Bayesian optimization is efficient for expensive and multimodal functions.



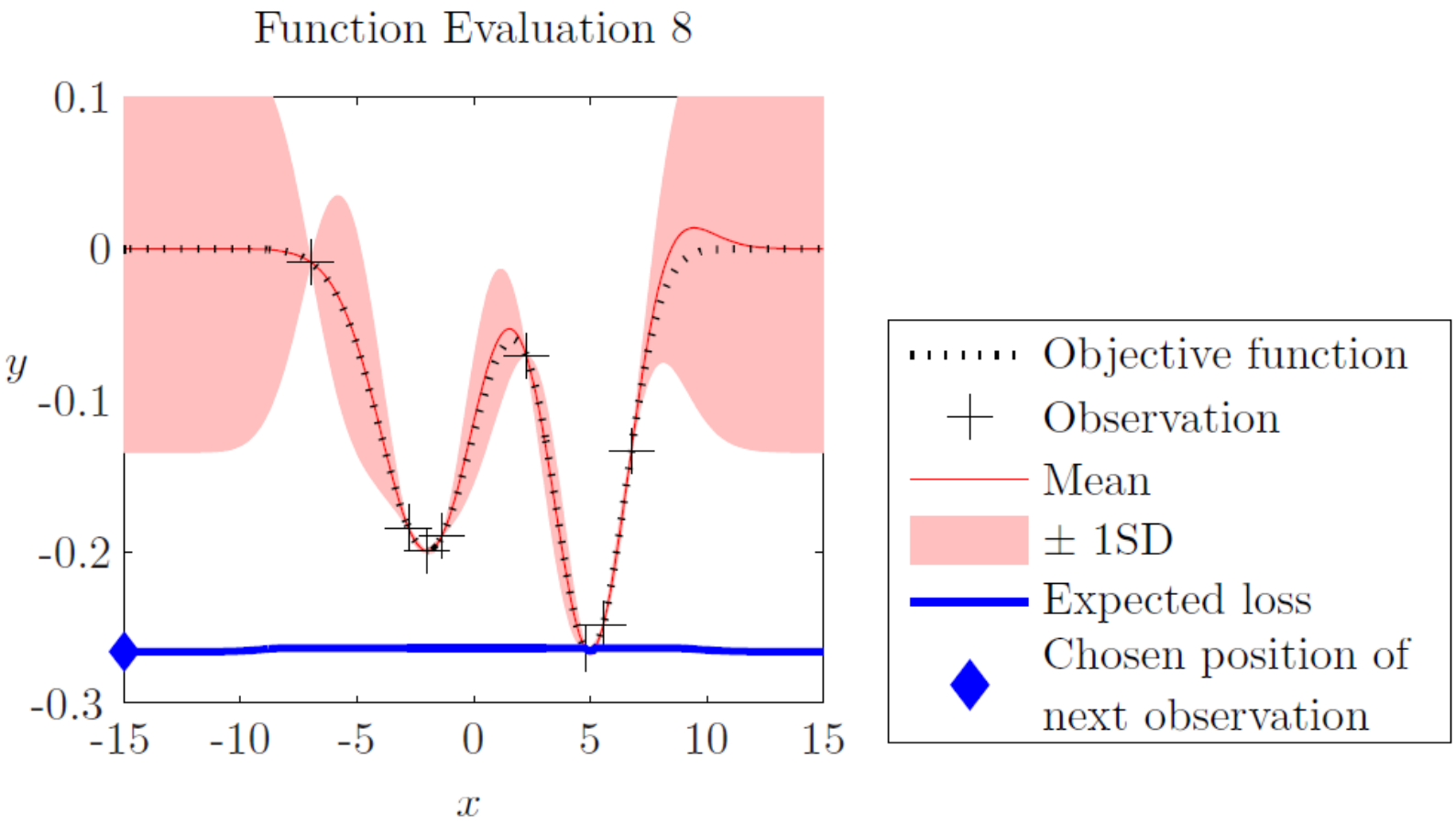
# Bayesian optimization is efficient for expensive and multimodal functions.



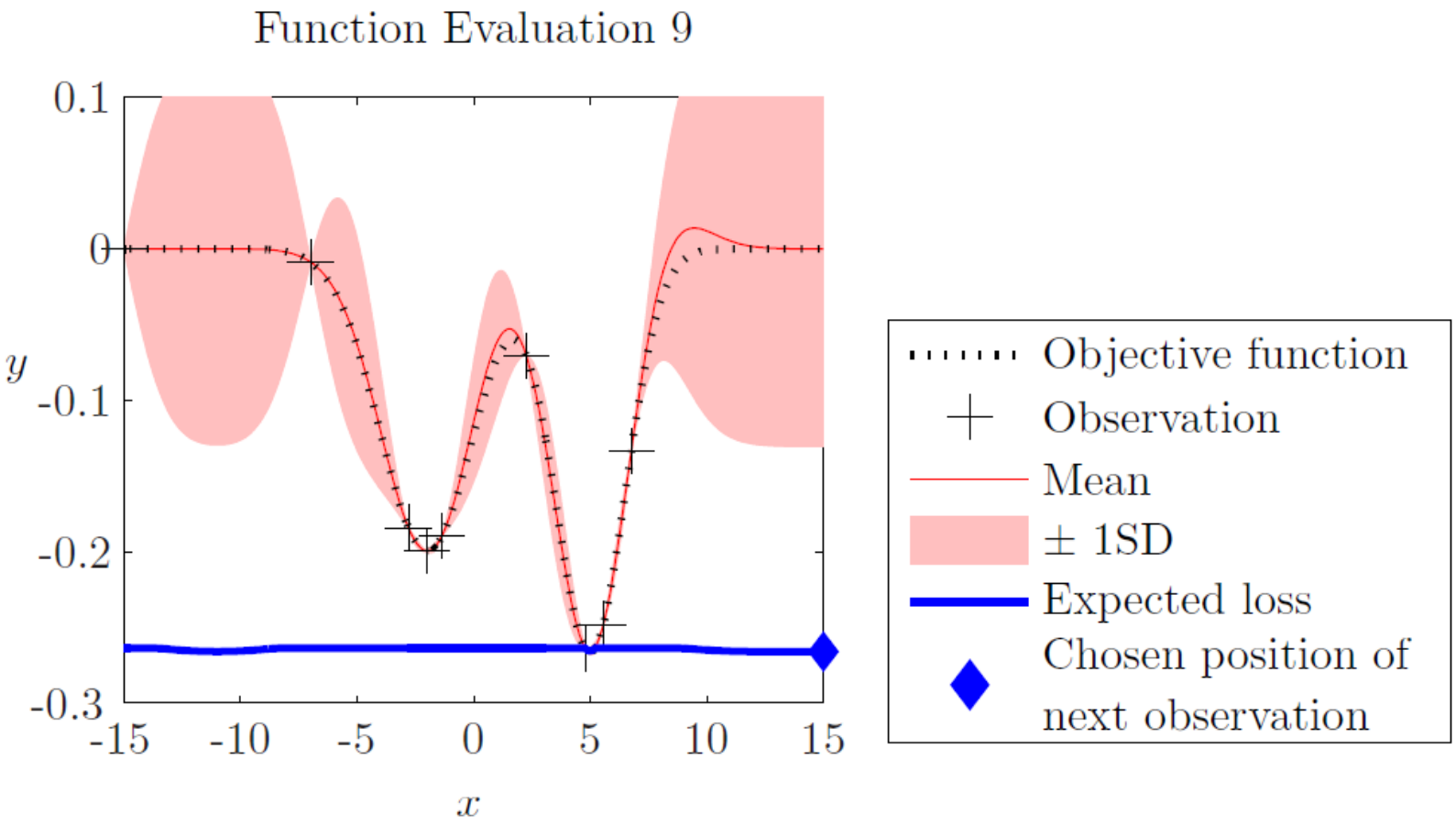
# Bayesian optimization is efficient for expensive and multimodal functions.



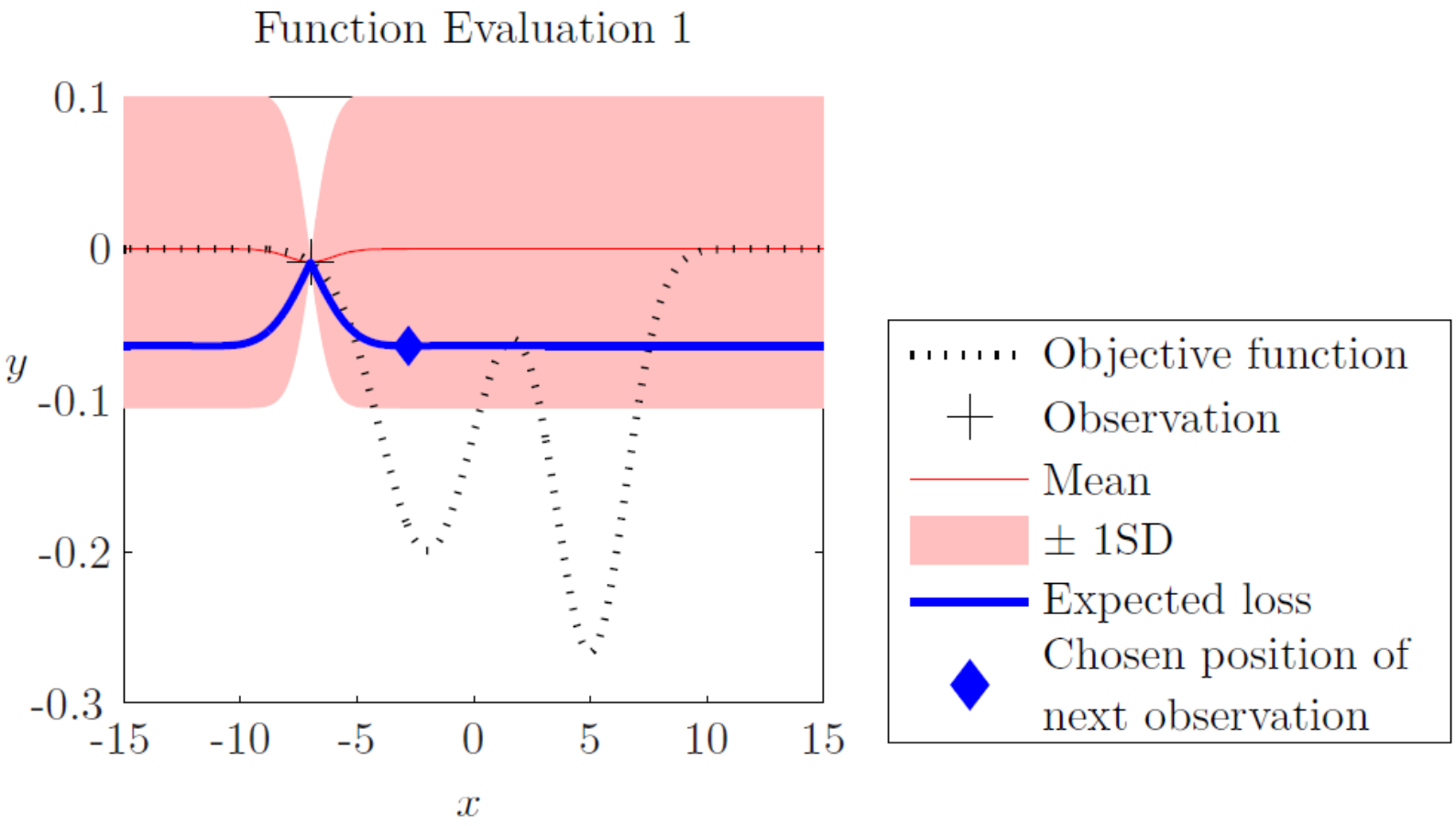
# Bayesian optimization is efficient for expensive and multimodal functions.



# Bayesian optimization is efficient for expensive and multimodal functions.



# Bayesian optimization is efficient for expensive and multimodal functions.



# An increasingly important application for BayesOpt is the configuration of algorithms.

Many algorithms have parameters that are laboriously tuned by hand: instead, why not use Bayesian Optimization?

See, for example, Programming by Optimisation, AutoML and Spearmint.

# Many algorithms have parameters that are only relevant if other inputs take certain values.

This problem arises in:

- deep neural networks [Hinton et al., 2006];
- flexible computer vision architectures [Bergstra et al., 2013];
- and the combined selection and hyperparameter optimization of machine learning algorithms [Thornton et al., 2013].

As an explicit example, we consider a deep neural network: if we set the network depth to 2 we know that the 3rd layer's hyperparameters do not have any effect (as there is no 3rd layer).

# In order to use BayesOpt, we need to define an appropriate covariance (kernel) function.

Formally, we aim to do inference about some function $f$ with domain $\mathcal{X}$.

$\mathcal{X} = \prod_{i=1}^{D} \mathcal{X}_i$ is a $D$-dimensional input space, where each individual dimension is bounded real, that is, $\mathcal{X}_i = [l_i, u_i] \subset \mathbb{R}$ (with lower and upper bounds $l_i$ and $u_i$, respectively).

We define functions $\delta_i : \mathcal{X} \to \{\text{true}, \text{false}\}$, for $i \in \{1, \ldots, D\}$. $\delta_i(\mathbf{x})$ stipulates the relevance of the $i$th feature $x_i$ to $f(\mathbf{x})$.

## Imagine a neural network with either one or two hidden layers and regularization parameters for each layer, $x_1$ and $x_2$.

If $y$ represents the performance of a one layer-net with parameters $x_1$ and $x_2$, then the value $x_2$ doesn't matter, since there is no second layer to the network.

We assume that $\delta_1(\mathbf{x}) =$ true; that is, the regularization parameter for the first layer is always relevant.

Recall that $\delta_2(\mathbf{x})$ specifies the relevance of $x_2$.

Hence we'll write an input triple as $(x_1, \delta_2(\mathbf{x}), x_2)$.

# A kernel $k$ needs to assign a measure of covariance between two performances.

That is, if

- $y$ is the performance of network with parameters $(x_1, \delta_2(\mathbf{x}), x_2)$ and
- $y'$ is the performance of network with parameters $(x_1', \delta_2(\mathbf{x}'), x_2')$,

$$\mathsf{cov}(y, y') = k\Big((x_1, \delta_2(\mathbf{x}), x_2), (x_1', \delta_2(\mathbf{x}'), x_2')\Big).$$

We want $k$ to be dependent on which parameters are relevant, and the values of relevant parameters for both points.

- If we are comparing two points for which the same parameters are relevant, the value of any unused parameters shouldn't matter,

$$k\big((x_1, \mathsf{false}, x_2), (x_1', \mathsf{false}, x_2')\big) = k\big((x_1, \mathsf{false}, x_2''), (x_1', \mathsf{false}, x_2''')\big), \ \forall x_2, x_2', x_2'', x_2'''; \quad (1)$$

- the covariance between a point using both parameters and a point using only one should again only depend on their shared parameters,

$$k\big((x_1, \mathsf{false}, x_2), (x_1', \mathsf{true}, x_2')\big) = k\big((x_1, \mathsf{false}, x_2''), (x_1', \mathsf{true}, x_2''')\big), \ \forall x_2, x_2', x_2'', x_2'''. \quad (2)$$

We want $k$ to be dependent on which parameters are relevant, and the values of relevant parameters for both points.

This implies that

$$k\big((x_1, \mathsf{false}, x_2), (x_1, \mathsf{false}, x_2')\big) = k_{\mathsf{FF}}, \ \forall x_2, x_2'$$
$$k\big((x_1, \mathsf{false}, x_2), (x_1, \mathsf{true}, x_2')\big) = k_{\mathsf{FT}}, \ \forall x_2, x_2'.$$

We usually additionally want

$$k_{\mathsf{FF}} > k_{\mathsf{FT}},$$

expressing the fact that points that have identical relevance $\delta_2(\mathbf{x})$ are more similar than points that differ in relevance $\delta_2(\mathbf{x})$.

# We embed all points in Euclidean space, $g : \mathcal{X} \mapsto \mathbb{R}^N$ (for our choice of $N$).

This general strategy solves the chief problem of designing kernels: ensuring that they are positive semi definite (PSD).

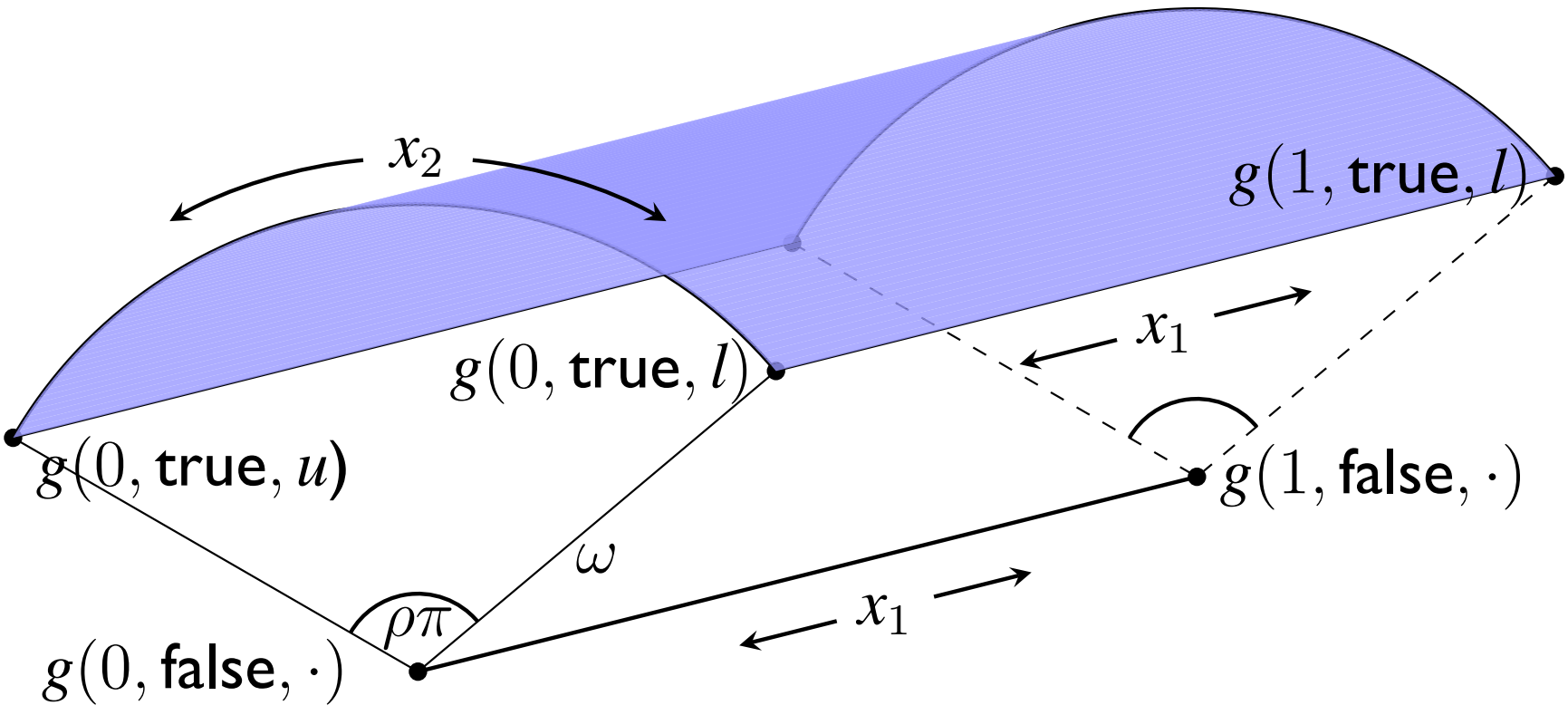
That is, a standard Euclidean kernel (e.g. $\exp\left(-(z - z')^2\right)$) can be used in the embedded space, which we know is PSD.

We consider each original dimension $\mathcal{X}_i$ separately, and use the embedding $g_i : \mathcal{X}_i \mapsto \mathbb{R}^2$ defined by

$$g_i(\mathbf{x}) = \begin{cases} [0, 0]^\intercal & \text{if } \delta_i(\mathbf{x}) = \text{ false} \\ \omega_i[\sin \pi \rho_i \frac{x_i}{u_i - l_i}, \cos \pi \rho_i \frac{x_i}{u_i - l_i}]^\intercal & \text{otherwise.} \end{cases}$$

where $\omega_i \in \mathbb{R}^+$ and $\rho_i \in [0, 1]$.

# This embedding and a standard Euclidean kernel satisfies our desiderata.



# This embedding and a standard Euclidean kernel satisfies our desiderata.

In our embedded space, we have the Euclidean distance,

$$d_i(\mathbf{x}, \mathbf{x}') = ||g_i(\mathbf{x}) - g_i(\mathbf{x}')||_2$$

$$= \begin{cases} 0 & \text{if } \delta_i(\mathbf{x}) = \delta_i(\mathbf{x}') = \text{false} \\ \omega_i & \text{if } \delta_i(\mathbf{x}) \neq \delta_i(\mathbf{x}') \\ \omega_i \sqrt{2} \sqrt{1 - \cos(\pi \rho_i \frac{x_i - x_i'}{u_i - l_i})} & \text{if } \delta_i(\mathbf{x}) = \delta_i(\mathbf{x}') = \text{true}. \end{cases}$$

We can then use our favourite Euclidean covariance, e.g.

- $k_i(\mathbf{x}, \mathbf{x}') = \sigma^2 \exp\left(-\frac{1}{2} d_i(\mathbf{x}, \mathbf{x}')\right)^2$ or
- $k_i(\mathbf{x}, \mathbf{x}') = \sigma^2 (1 + \frac{1}{2\alpha} d_i(\mathbf{x}, \mathbf{x}')^2)^{-\alpha}$.

# Our embedding has two parameters: $\rho$ and $\omega$.

These two parameters weight how the relative importances of

- differing in $x_2$ while having $\delta_2(\mathbf{x}) = \text{true}$ and
- having different values for $\delta_2(\mathbf{x})$.

Explicitly, $\omega$ controls the arc radius and $\rho$ controls the arc angle.

If $\rho$ is sufficiently small, we return to the original distance over $x_2$.

We fit these hyperparameters using maximum likelihood or MCMC, as with any other kernel hyperparameters.

We embed each dimension separately in $\mathbb{R}^2$, so that our final embedded space is $\mathbb{R}^{2D}$.

This gives us a kernel over the entire D-dimensional input space,

$$k(\mathbf{x}, \mathbf{x}') = \prod_{i=1}^{D} k_i(\mathbf{x}, \mathbf{x}').$$

We call it the arc kernel.

# To test our kernel, we attempt to model the performance of neural networks for classification.

We consider the canonical MNIST digits dataset [Lecun et al., 1998] where the task is to classify handwritten digits.

We also consider the CIFAR-10 object recognition dataset [Krizhevsky, 2009]. We pre-processed CIFAR-10 by extracting features according to the pipeline given in Coates et al. [2011].

# We tested the regression performance of our kernel on MNIST data.

951 datapoints (performance evaluations of neural networks for different parameter settings) were generated on MNIST.

We then tested regression performance of different models for that data; Results are averaged over 10-fold train/test splits.

# We tested the regression performance of our kernel on MNIST data.

The 'separate' models assume that performance for classifiers with different numbers of parameters (i.e. different numbers of hidden layers, different architectures) are independent.

Table: Normalized Mean Squared Error

| Method | Original data | Log outputs |
|---|---|---|
| Separate Linear | $0.812 \pm 0.045$ | $0.737 \pm 0.049$ |
| Separate gp | $0.546 \pm 0.038$ | $0.446 \pm 0.041$ |
| Separate Arc gp | $0.535 \pm 0.030$ | $0.440 \pm 0.031$ |
| Linear | $0.876 \pm 0.043$ | $0.834 \pm 0.047$ |
| gp | $0.481 \pm 0.031$ | $0.401 \pm 0.028$ |
| Arc gp | $\mathbf{0.421 \pm 0.033}$ | $\mathbf{0.335 \pm 0.028}$ |

# We next evaluated using the arc kernel to perform Bayesian optimisation.

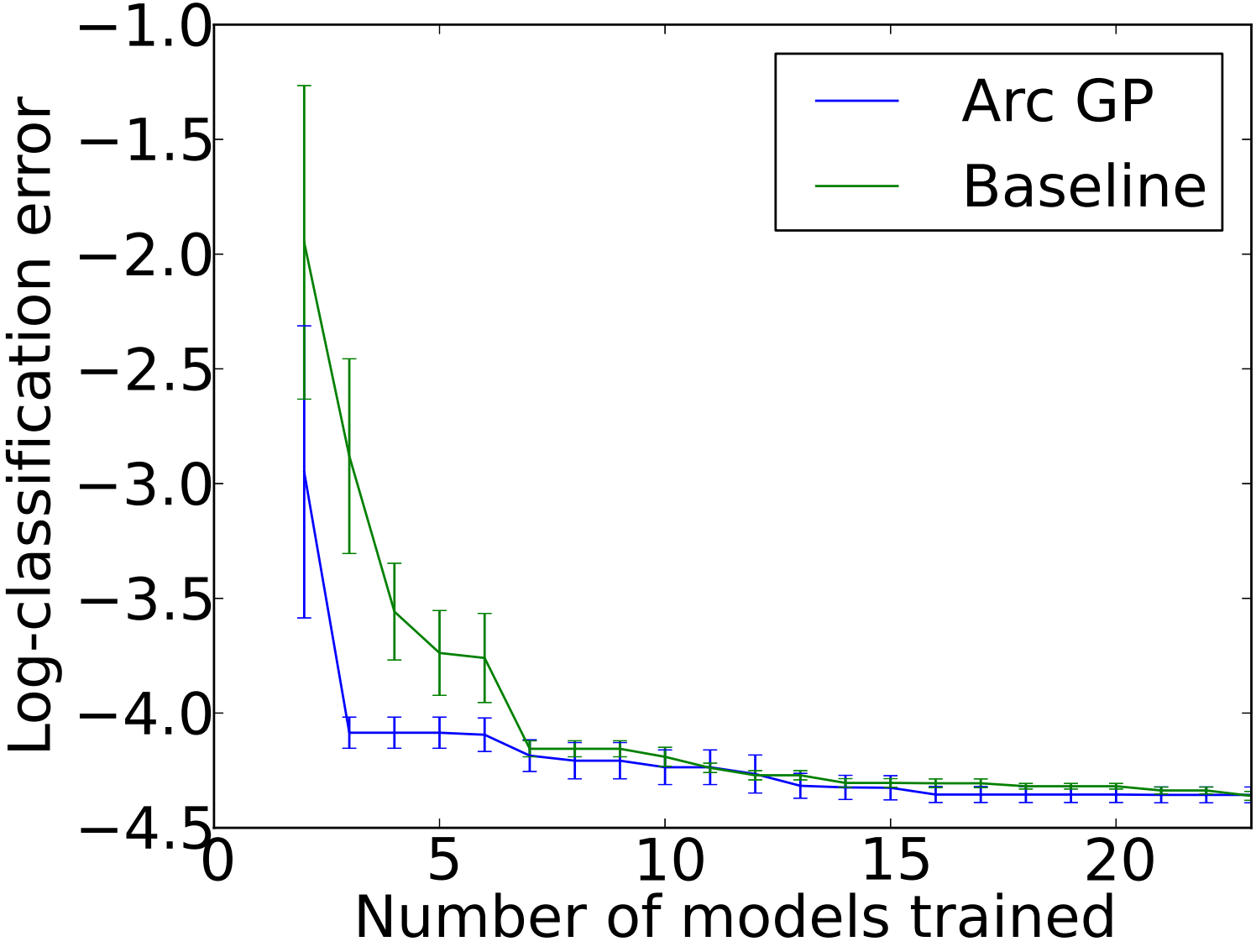
We test the ability of Bayesian optimization to tune the hyperparameters of each layer of a deep neural network.

We allow the neural networks for these problems to use up to $5$ hidden layers (or no hidden layer).

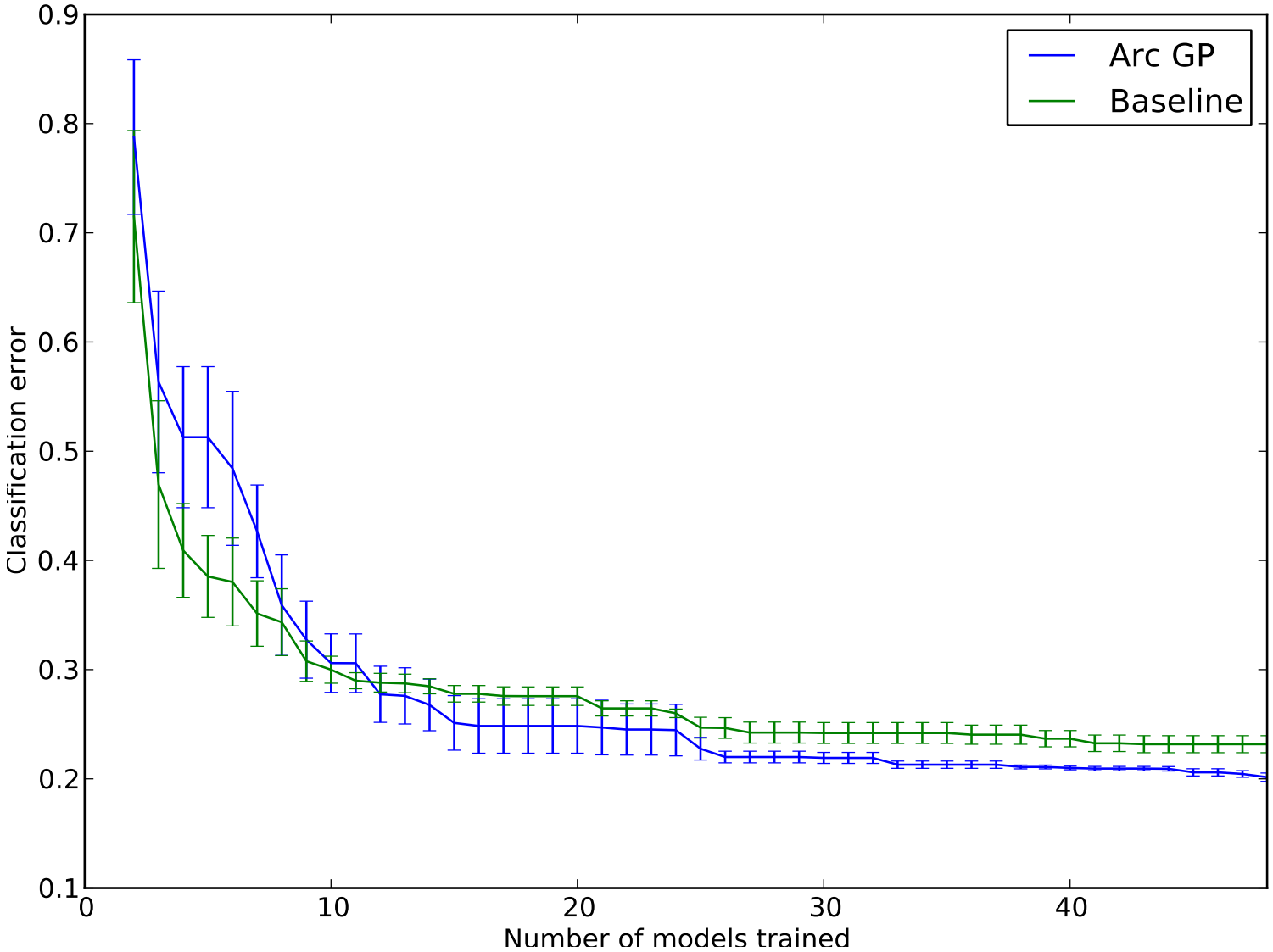
We optimize over learning rates, L2 weight constraints, dropout rates Hinton et al. [2012], and the number of hidden units per layer leading to a total of up to $23$ hyperparameters and $6$ architectures.

Our baseline is BayesOpt with a GP that assumes random values for irrelevant parameters.

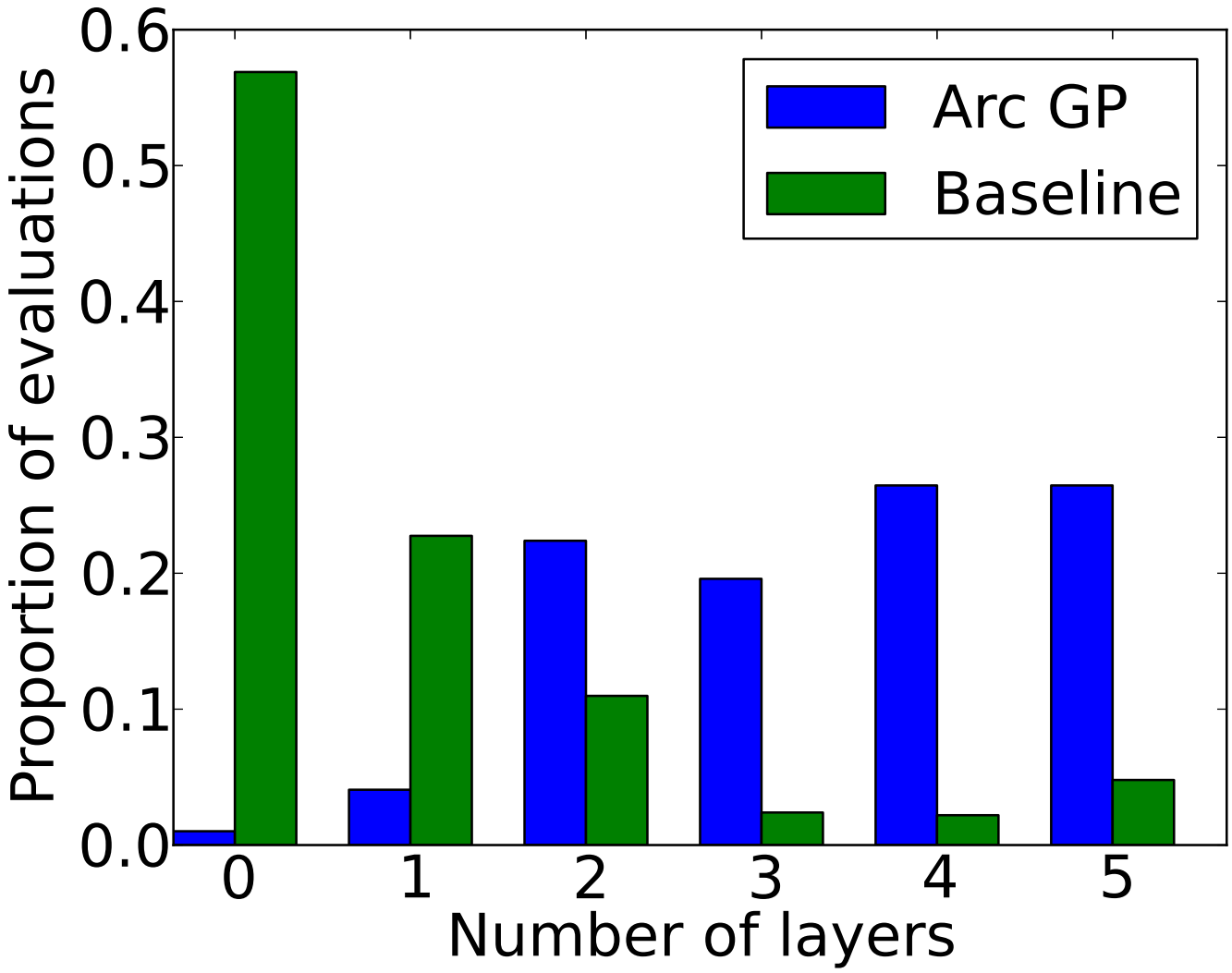
# We have results on the MNIST problem.



# We have results on the CIFAR-10 problem.



# We search over deeper architectures for the CIFAR-10 experiment.



# The arc kernel finds better solutions faster.

Allowing information to be shared across architectures improves the efficiency of Bayesian optimization and removes the need to manually search for good architectures.

The resulting models perform favourably compared to established benchmarks by domain experts.