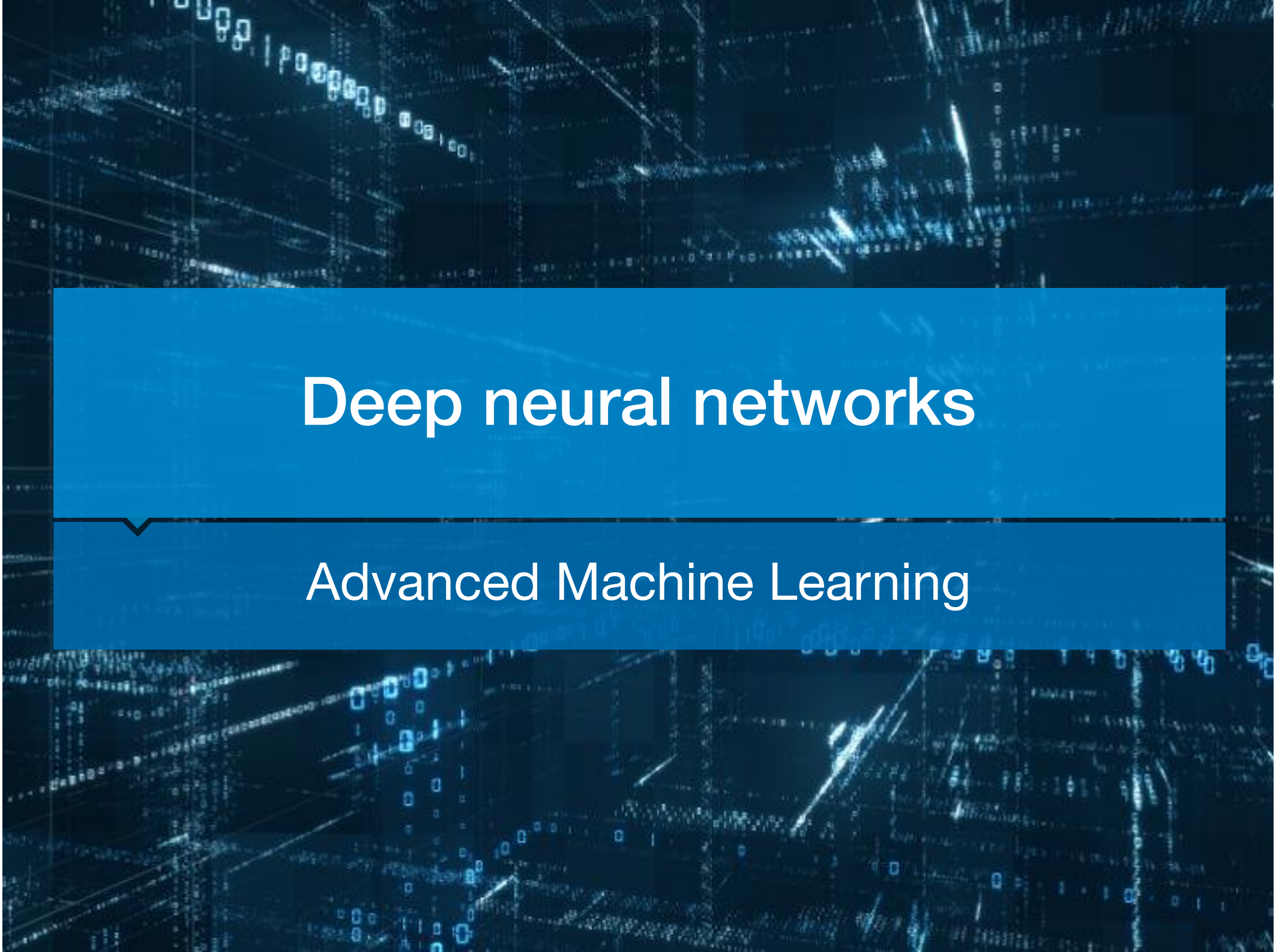
Advanced Machine Learning
Lecture 7: Deep neural networks

Sandjai Bhulai
Vrije Universiteit Amsterdam

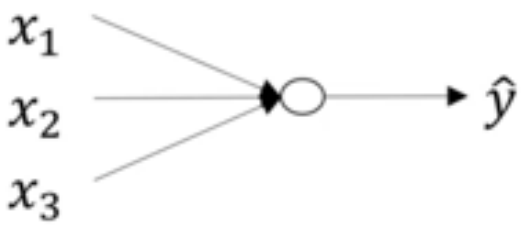
s.bhulai@vu.nl
26 September 2023



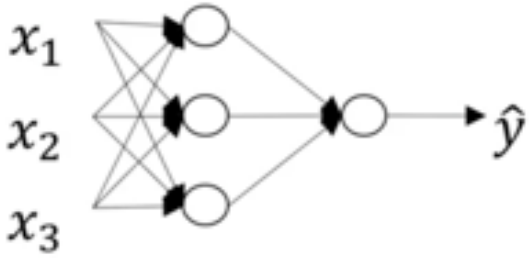
Deep neural networks

Advanced Machine Learning

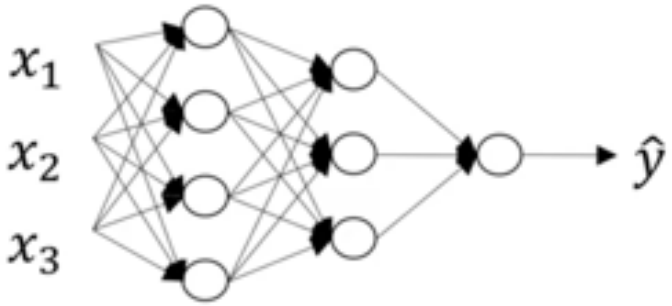
Deep neural networks



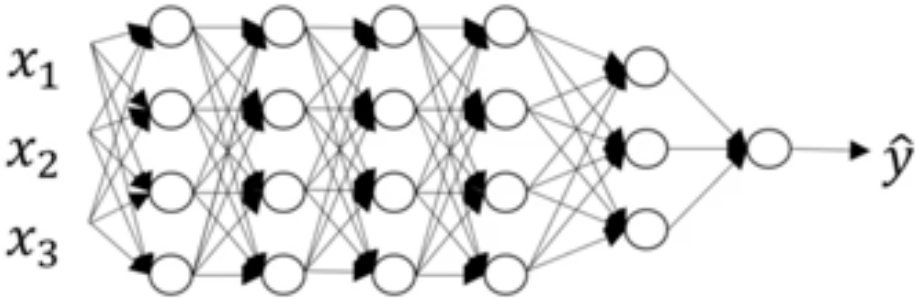
logistic regression



1 hidden layer

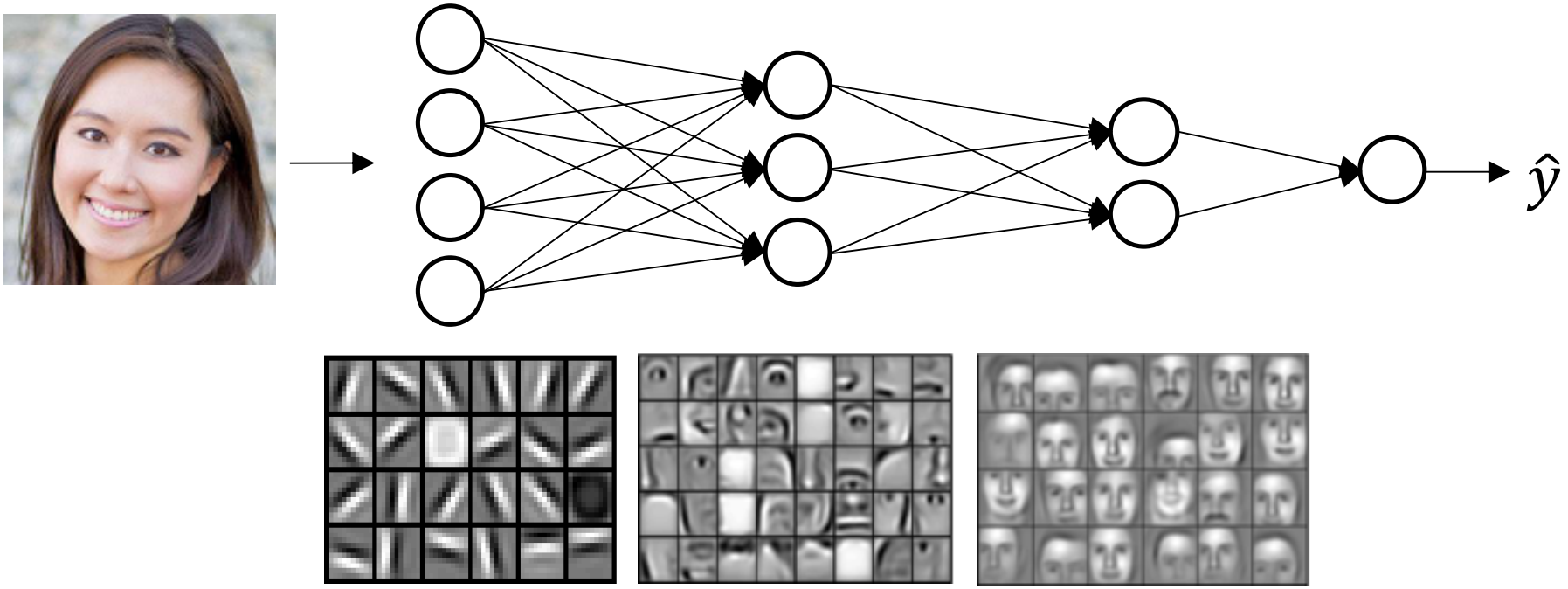


2 hidden layers



5 hidden layers

Deep neural networks



Deep neural networks

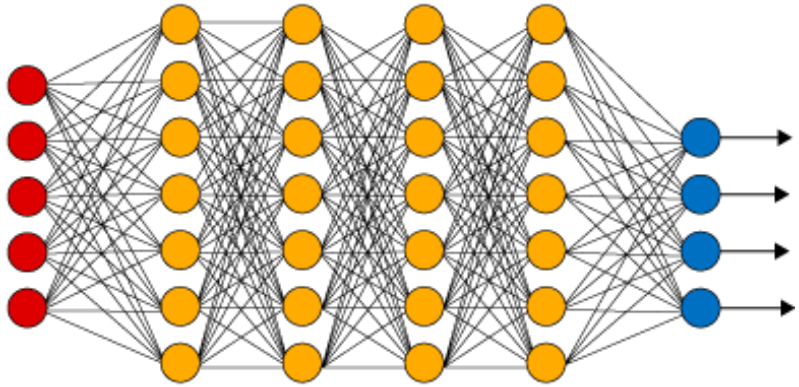
- Why deep neural networks?
  - > Informally: There are many functions you can compute with a “small” -layer deep neural network that shallower networks require exponentially more hidden units to compute
  - > Example: x_1 XOR x_2 XOR x_3 XOR x_4 ⋯

    $O(\log n)$ versus $O(2^n)$

Deep neural networks

Building blocks for forward propagation:

- Consider layer l
- Input to layer l is $a^{[l-1]}$
- Output $a^{[l]}$ given by:



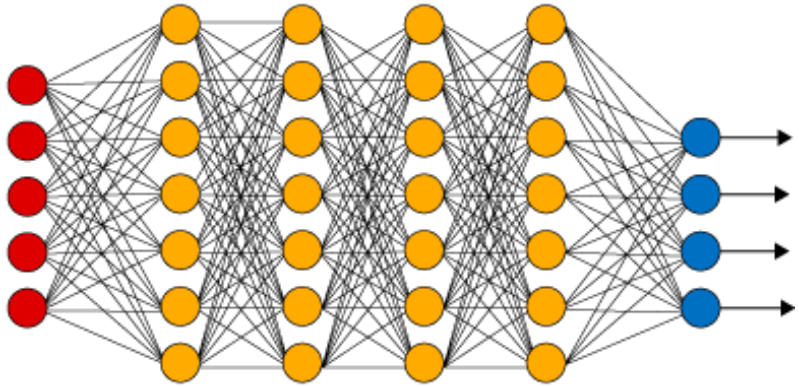
$$z^{[l]} = W^{[l]} a^{[l-1]} + b^{[l]}$$

$$a^{[l]} = g^{[l]}(z^{[l]})$$

Deep neural networks

Building blocks for backward propagation:

- Consider layer l
- Input to layer l is $\mathrm{d}a^{[l]}$
- Output is given by:



$$\mathrm{d}z^{[l]} = \mathrm{d}a^{[l]} \cdot g^{[l]'}(z^{[l]})$$

$$\mathrm{d}W^{[l]} = \mathrm{d}z^{[l]} \cdot a^{[l-1]}$$

$$\mathrm{d}b^{[l]} = \mathrm{d}z^{[l]}$$

$$\mathrm{d}a^{[l-1]} = (W^{[l]})^{\top}\mathrm{d}z^{[l]}$$

Deep neural networks

Forward and backward propagation

$$
\begin{aligned}
Z^{[1]} &= W^{[1]}X + b^{[1]} \\
A^{[1]} &= g^{[1]}(Z^{[1]}) \\
Z^{[2]} &= W^{[2]}A^{[1]} + b^{[2]} \\
A^{[2]} &= g^{[2]}(Z^{[2]}) \\
&\vdots \\
A^{[L]} &= g^{[L]}(Z^{[L]}) = \hat{Y}
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{d}Z^{[L]} &= A^{[L]} - Y \\
\mathrm{d}W^{[L]} &= \frac{1}{m}\mathrm{d}Z^{[L]}(A^{[L]})^{\top} \\
\mathrm{d}b^{[L]} &= \frac{1}{m}\mathrm{np.sum}(\mathrm{d}Z^{[L]}, \mathrm{axis} = 1, \mathrm{keepdims} = \mathrm{True}) \\
\mathrm{d}Z^{[L-1]} &= \mathrm{d}(W^{[L]})^{\top}\mathrm{d}Z^{[L]}g'^{[L]}(Z^{[L-1]}) \\
&\vdots \\
\mathrm{d}Z^{[1]} &= \mathrm{d}(W^{[L]})^{\top}\mathrm{d}Z^{[2]}g'^{[1]}(Z^{[1]}) \\
\mathrm{d}W^{[1]} &= \frac{1}{m}\mathrm{d}Z^{[1]}(A^{[1]})^{\top} \\
\mathrm{d}b^{[1]} &= \frac{1}{m}\mathrm{np.sum}(\mathrm{d}Z^{[1]}, \mathrm{axis} = 1, \mathrm{keepdims} = \mathrm{True})
\end{aligned}
$$

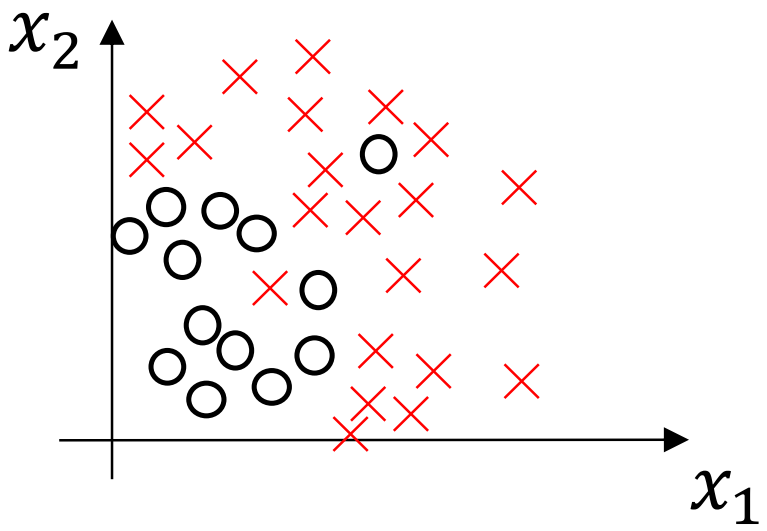
Practical aspects of DNNs

- Deep neural networks have a lot of hyperparameters
  - # layers
  - # hidden units
  - Learning rates
  - Activation functions
  - …
- Need for: train, development, and test set

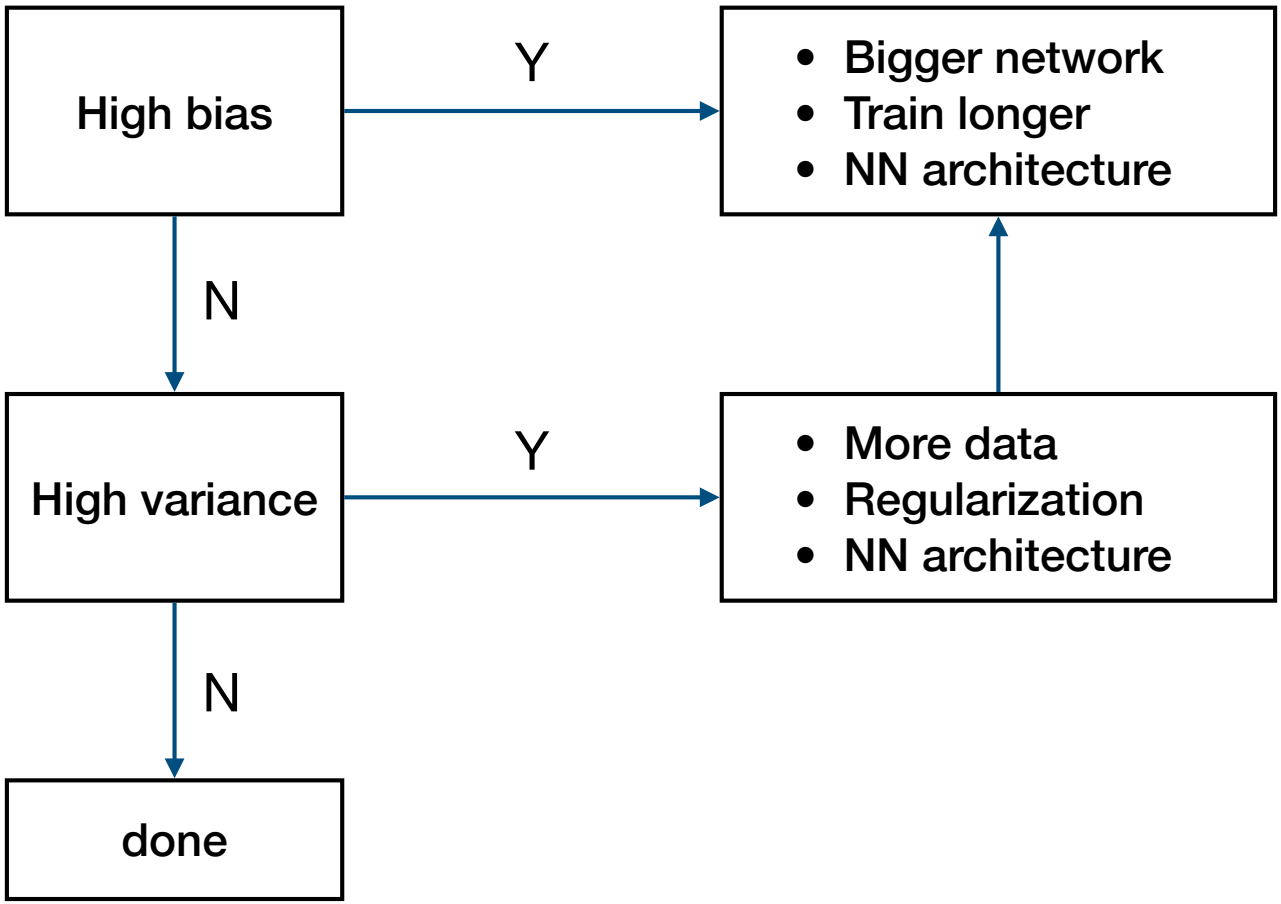
Practical aspects of DNNs

- Bias and variance

	High variance	High bias	High bias High variance	Low Bias Low variance
Train set	1%	15%	15%	0.5%
Dev set	11%	16%	30%	1%

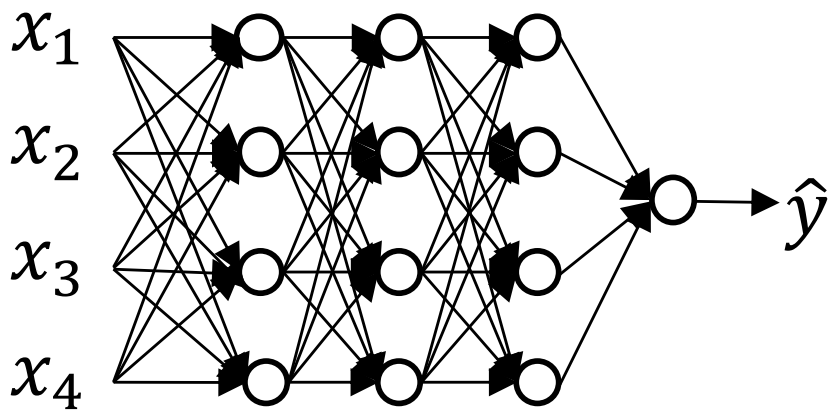


Practical aspects of DNNs

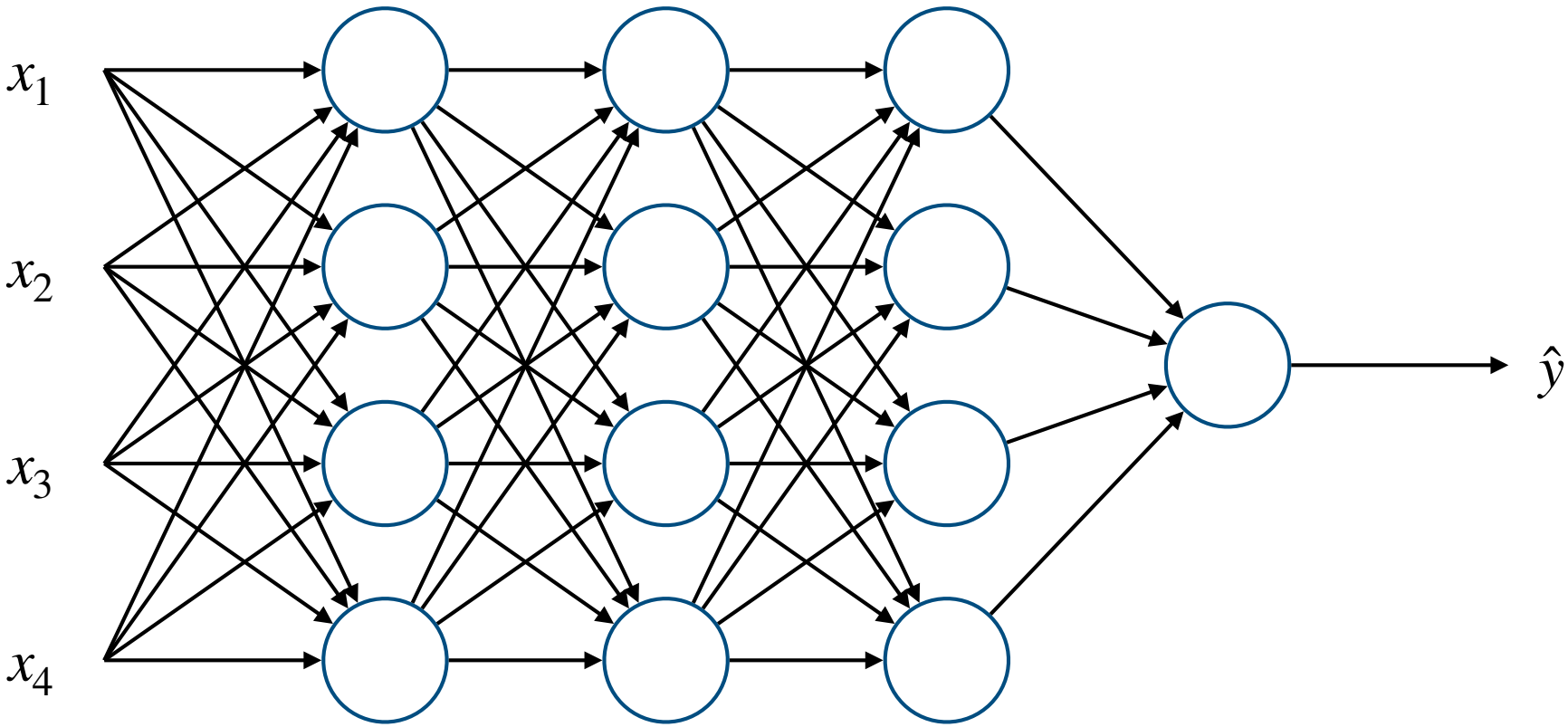


Practical aspects of DNNs

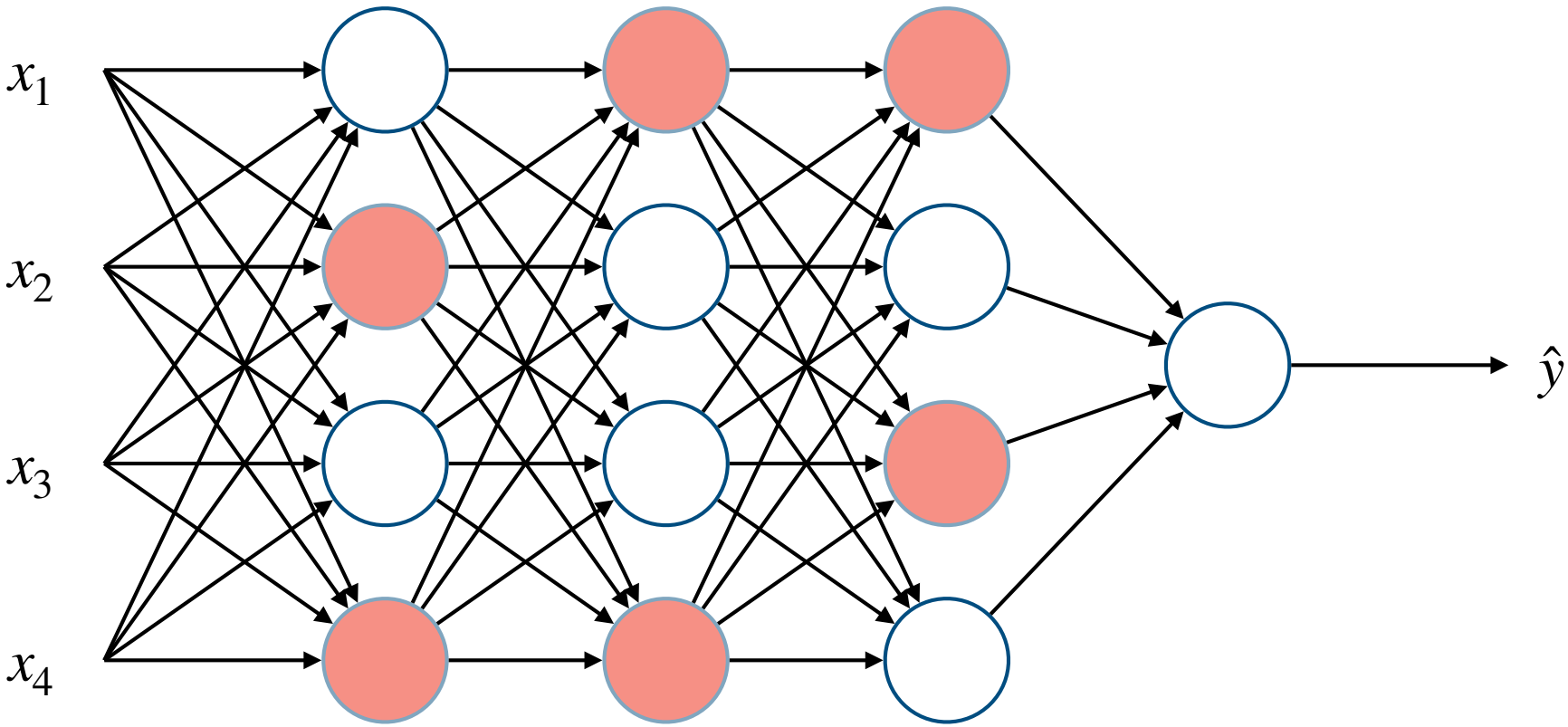
- Regularization can reduce overfitting
- Dropout regularization
- Intuition: cannot rely on any on feature, so have to spread out weights



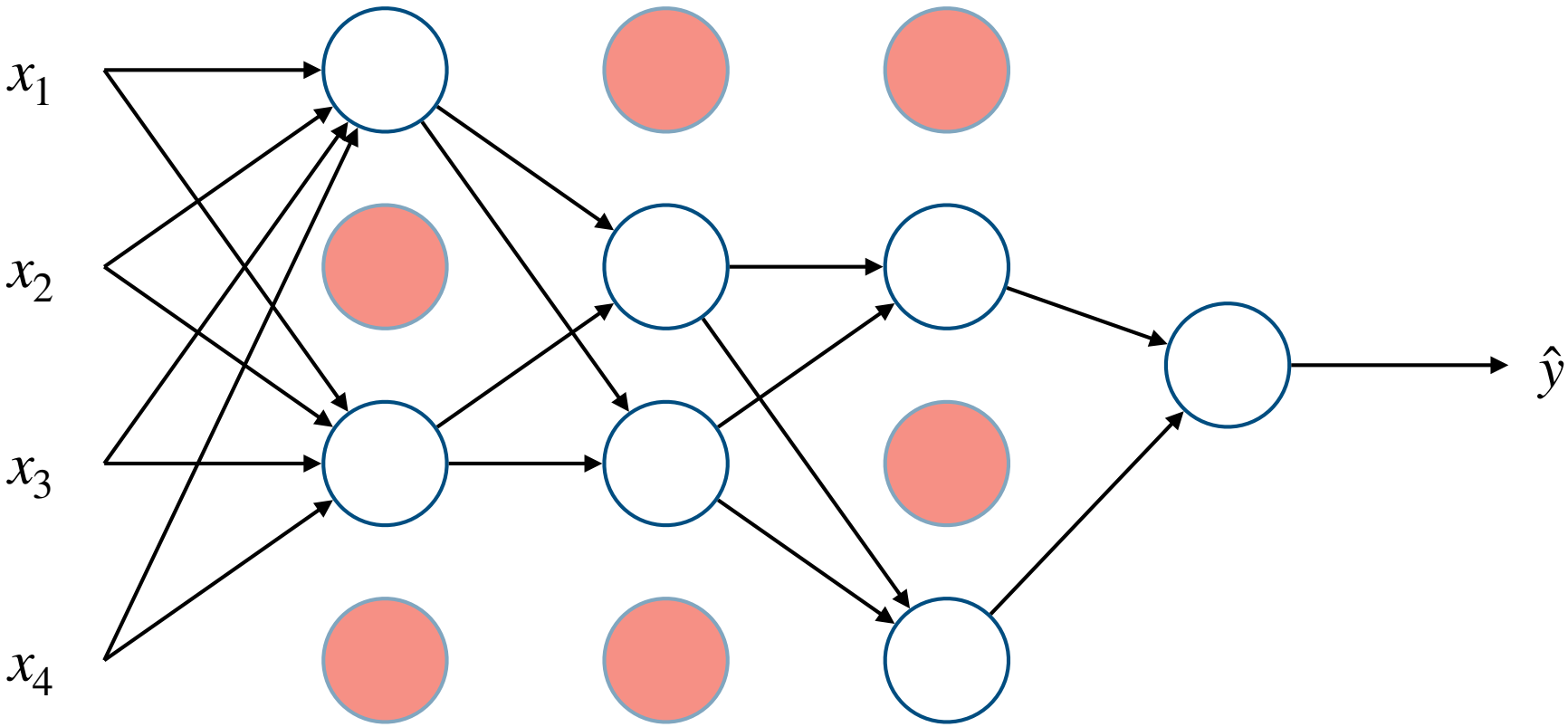
Practical aspects of DNNs



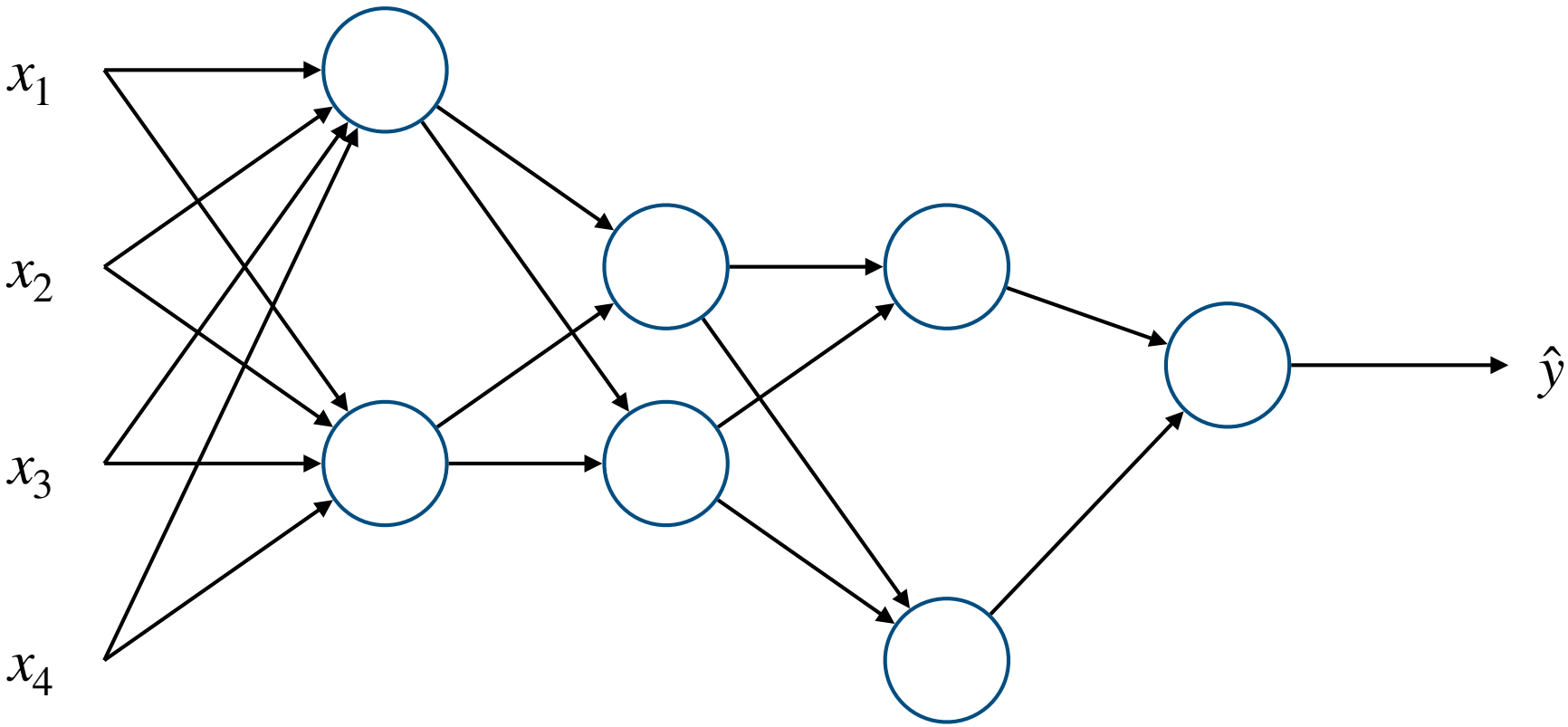
Practical aspects of DNNs



Practical aspects of DNNs



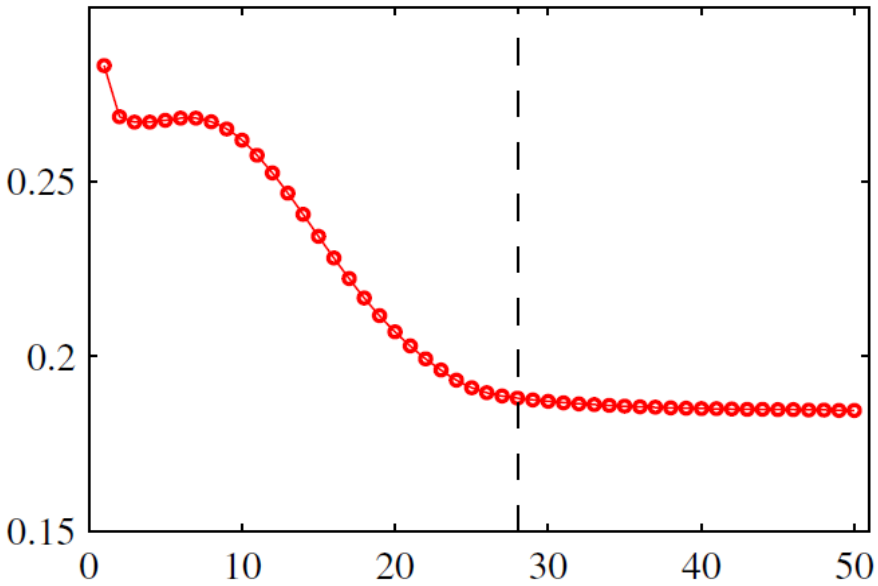
Practical aspects of DNNs



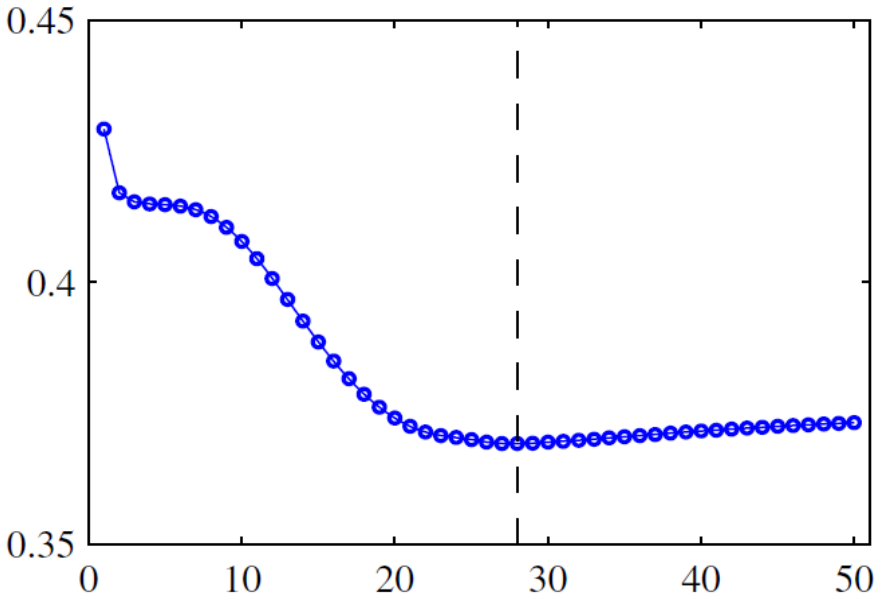
Practical aspects of DNNs

- Alternative to regularization: early stopping
- Stop at smallest error with validation data

training set error



validation set error

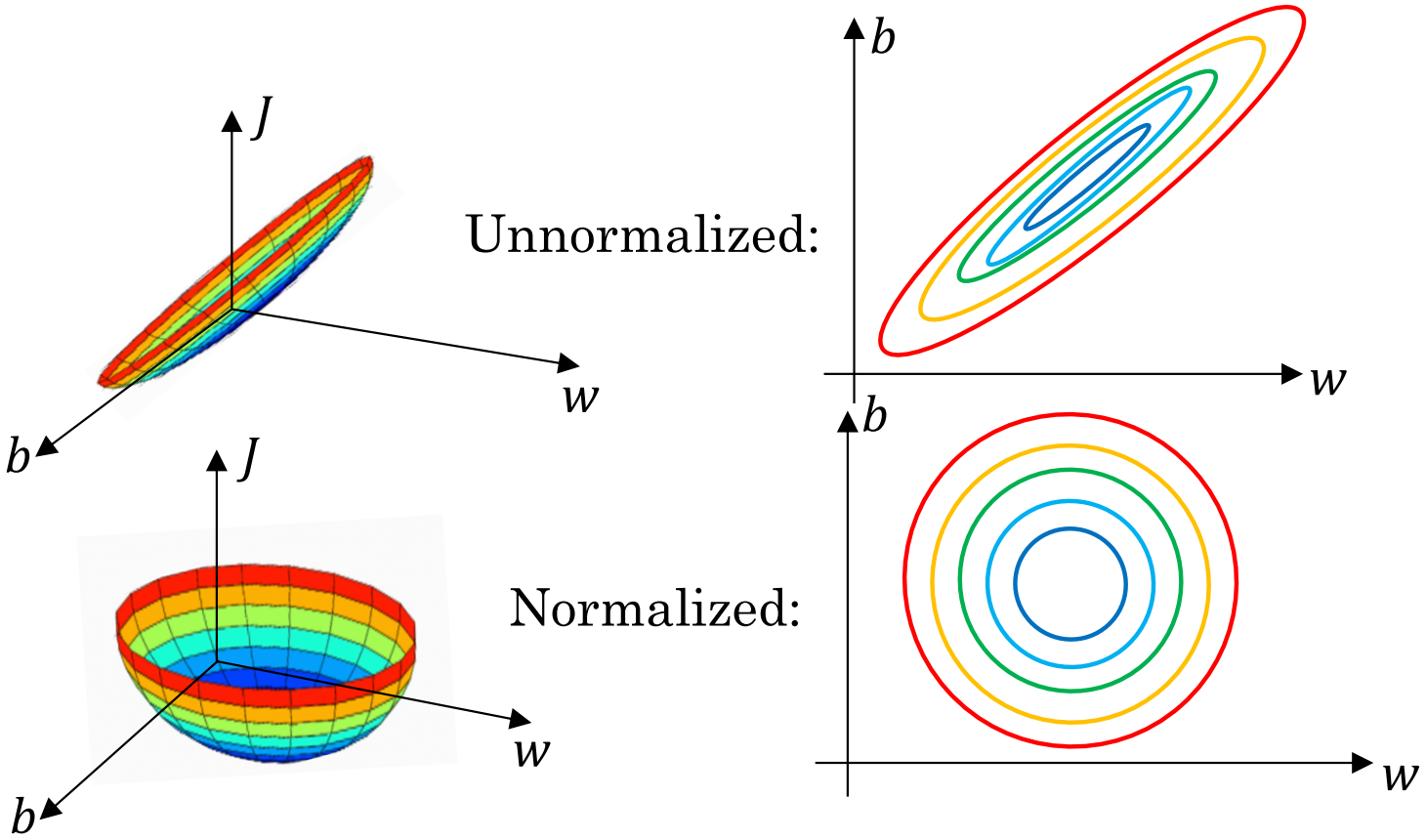


Practical aspects of DNNs

- Data augmentation

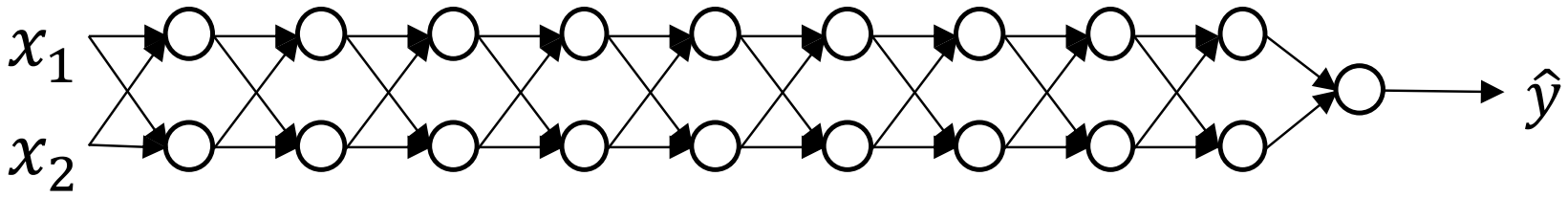
Practical aspects of DNNs

- Normalize inputs



Practical aspects of DNNs

- Vanishing / exploding gradients



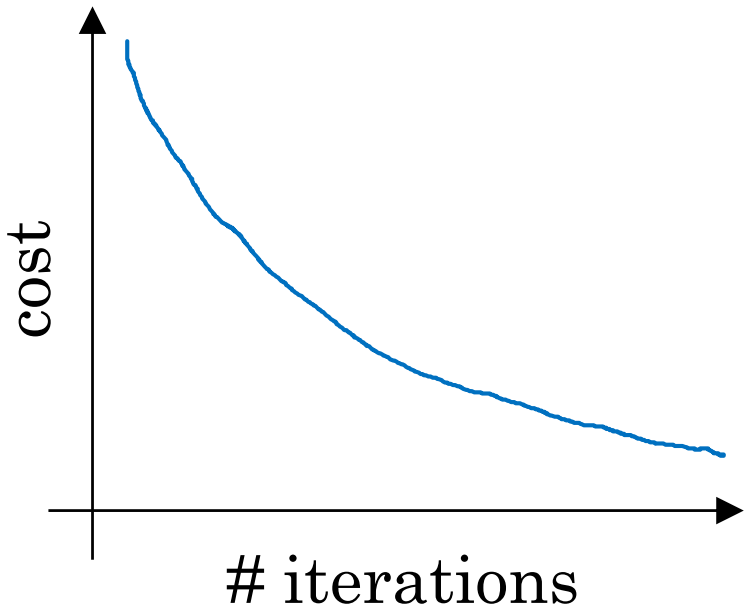
- Use linear activation function
- $\hat{y} = W^{[1]}W^{[2]}\ldots W^{[l-1]}W^{[l]}x$

Optimization algorithms

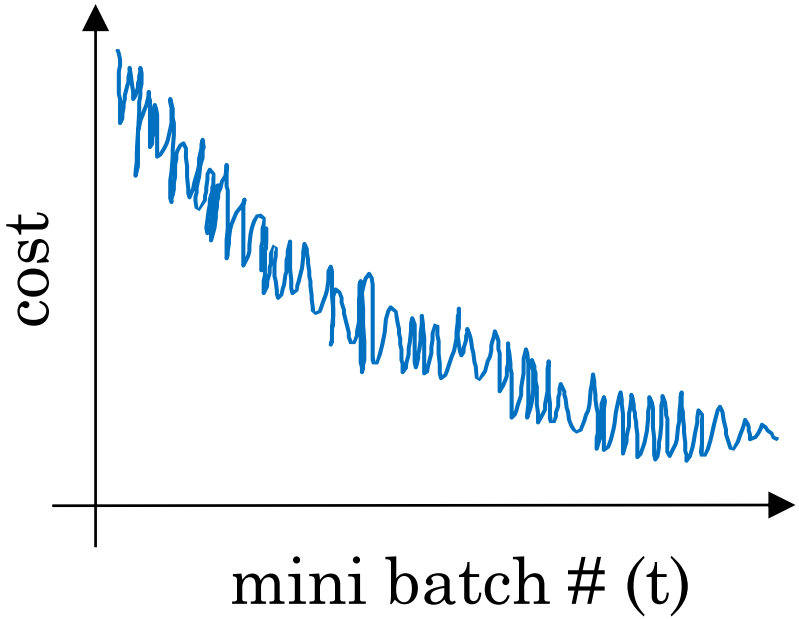
- Mini-batch gradient descent
  - Vectorization allows efficient computation on many examples
  - Forward propagation uses all data
  - Idea: split up training set in batches and apply gradient descent on evaluation of one batch

Optimization algorithms

Batch gradient descent

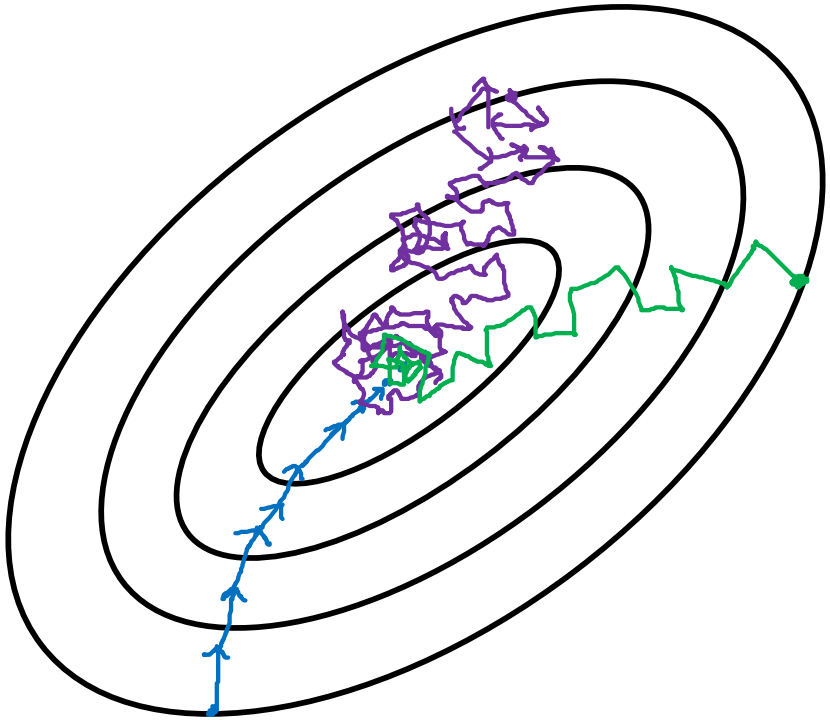


Mini-batch gradient descent



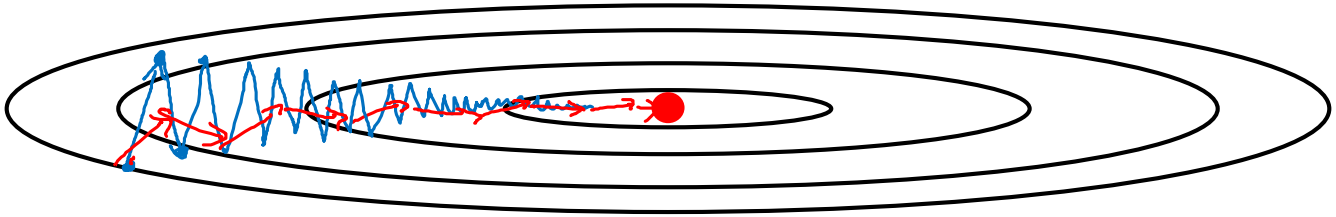
Optimization algorithms

- Purple = stochastic gradient descent
- Green = mini-batch gradient descent
- Blue = batch gradient descent



Optimization algorithms

- Gradient descent with momentum



- In practice, you want to dampen directions in which you should not move

Optimization algorithms

- Gradient descent with momentum

- On iteration t, compute $\mathrm{d}W$, $\mathrm{d}b$ on the current mini-batch

$$v_{\mathrm{d}W} = \beta v_{\mathrm{d}W} + (1-\beta)\mathrm{d}W$$

$$v_{\mathrm{d}b} = \beta v_{\mathrm{d}b} + (1-\beta)\mathrm{d}b$$

$$W = W - \alpha v_{\mathrm{d}W}$$

$$b = b - \alpha v_{\mathrm{d}b}$$

- Hyperparameters: α, β $\qquad (\beta = 0.9)$

Optimization algorithms

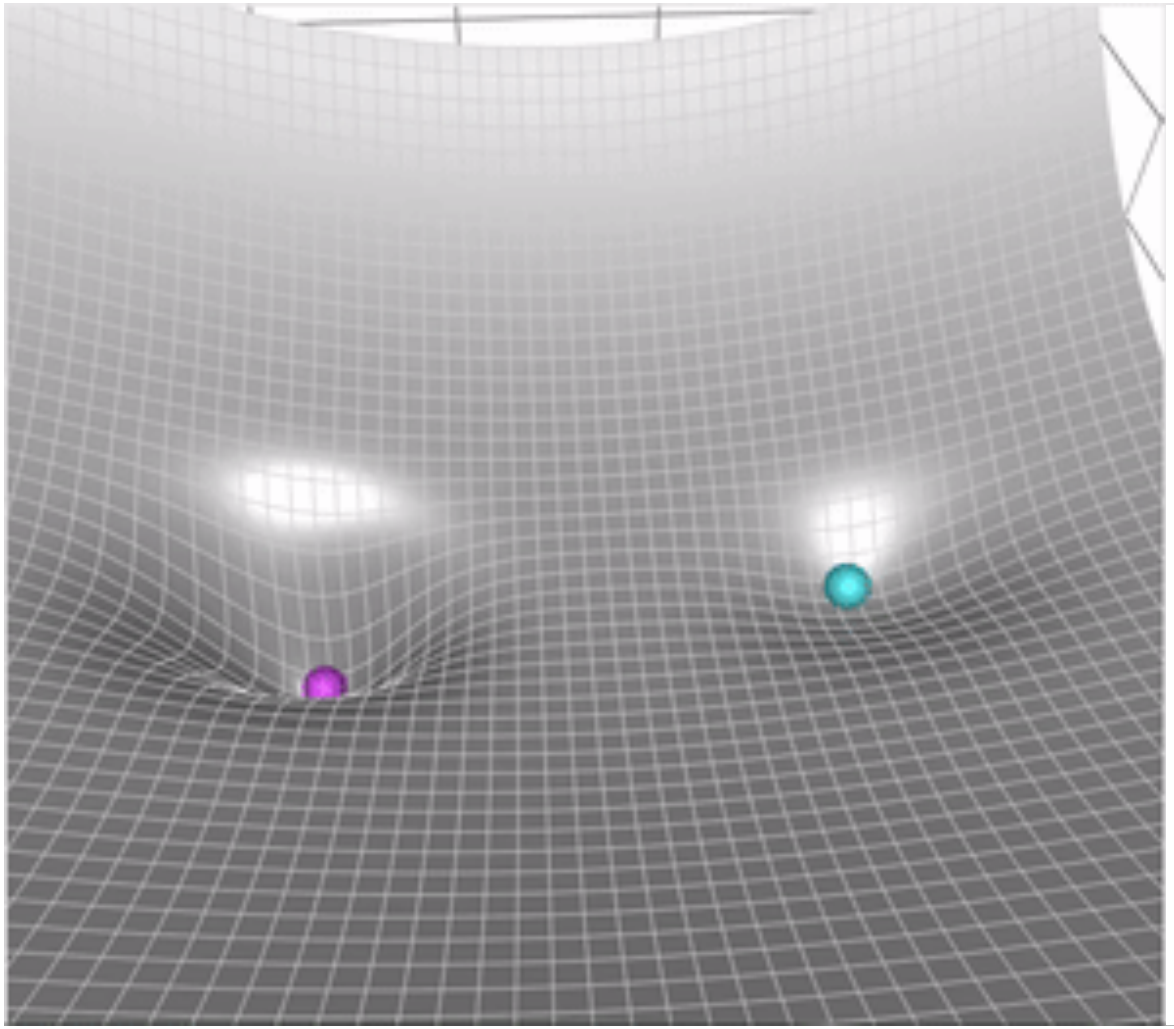
- Taking average is an example of smoothing

$$\bar{x}_{n+1} = \frac{x_1 + \cdots + x_{n+1}}{n+1} = \frac{x_1 + \cdots + x_n}{n+1} + \frac{x_{n+1}}{n+1}$$

$$= \frac{x_1 + \cdots + x_n}{n} \cdot \frac{n}{n+1} + \frac{x_{n+1}}{n+1} = \frac{n}{n+1}\bar{x}_n + \frac{1}{n+1}x_{n+1}$$

$$= \alpha_n \bar{x}_n + (1 - \alpha_n)x_{n+1}, \text{ with } \alpha_n = \frac{n}{n+1}$$

Optimization algorithms



- gradient descent (cyan)
- momentum (magenta)

Optimization algorithms

- RMSprop

- On iteration t, compute $\mathrm{d}W$, $\mathrm{d}b$ on the current mini-batch

$$s_{\mathrm{d}W} = \beta s_{\mathrm{d}W} + (1-\beta)\mathrm{d}W^2$$

$$s_{\mathrm{d}b} = \beta s_{\mathrm{d}b} + (1-\beta)\mathrm{d}b^2$$

$$W = W - \alpha \frac{\mathrm{d}W}{\sqrt{s_{\mathrm{d}W}}}$$

$$b = b - \alpha \frac{\mathrm{d}b}{\sqrt{s_{\mathrm{d}b}}}$$

Optimization algorithms

- Properties of RMSprop

$$s_{\mathrm{d}W} = \beta s_{\mathrm{d}W} + (1-\beta)\mathrm{d}W^2 \rightarrow s_t = \beta s_{t-1} + (1-\beta)g_t^2$$

- Note that $s_{t-1} = \beta s_{t-2} + (1-\beta)g_{t-1}^2$. By substitution, we get

$$s_t = \beta\left[\beta s_{t-2} + (1-\beta)g_{t-1}^2\right] + (1-\beta)g_t^2 = \beta^2 s_{t-2} + (1-\beta)\left[g_t^2 + \beta g_{t-1}^2\right]$$

- By continuing this process, we get

$$s_t = (1-\beta)\sum_{i=1}^{t}\beta^{t-i}g_i^2$$

Optimization algorithms

- If we assume that the g_i's are i.i.d., then

$$\mathbb{E}[s_t] = \mathbb{E}\left[(1-\beta)\sum_{i=1}^{t}\beta^{t-i}g_i^2\right] = (1-\beta)\sum_{i=1}^{t}\beta^{t-i}\mathbb{E}\left[g_i^2\right]$$

$$\mathbb{E}[s_t] = \mathbb{E}\left[g_t^2\right](1-\beta)\sum_{i=1}^{t}\beta^{t-i} = \mathbb{E}\left[g_t^2\right](1-\beta)\sum_{i=0}^{t-1}\beta^{i}$$

$$\mathbb{E}[s_t] = \mathbb{E}\left[g_t^2\right](1-\beta)\frac{1-\beta^t}{1-\beta} = \mathbb{E}\left[g_t^2\right](1-\beta^t)$$

-

Optimization algorithms

- Adam optimization

- On iteration t, compute $\mathrm{d}W$, $\mathrm{d}b$ on the current mini-batch

$$v_{\mathrm{d}W} = \beta_1 v_{\mathrm{d}W} + (1-\beta_1)\mathrm{d}W \qquad s_{\mathrm{d}W} = \beta_2 s_{\mathrm{d}W} + (1-\beta_2)\mathrm{d}W^2$$

$$v_{\mathrm{d}b} = \beta_1 v_{\mathrm{d}b} + (1-\beta_1)\mathrm{d}b \qquad s_{\mathrm{d}b} = \beta_2 s_{\mathrm{d}b} + (1-\beta_2)\mathrm{d}b^2$$

$$V_{\mathrm{d}W}^{\mathrm{cor}} = v_{\mathrm{d}W}/(1-\beta_1^t) \qquad S_{\mathrm{d}W}^{\mathrm{cor}} = s_{\mathrm{d}W}/(1-\beta_2^t)$$

$$V_{\mathrm{d}b}^{\mathrm{cor}} = v_{\mathrm{d}b}/(1-\beta_1^t) \qquad S_{\mathrm{d}b}^{\mathrm{cor}} = s_{\mathrm{d}b}/(1-\beta_2^t)$$

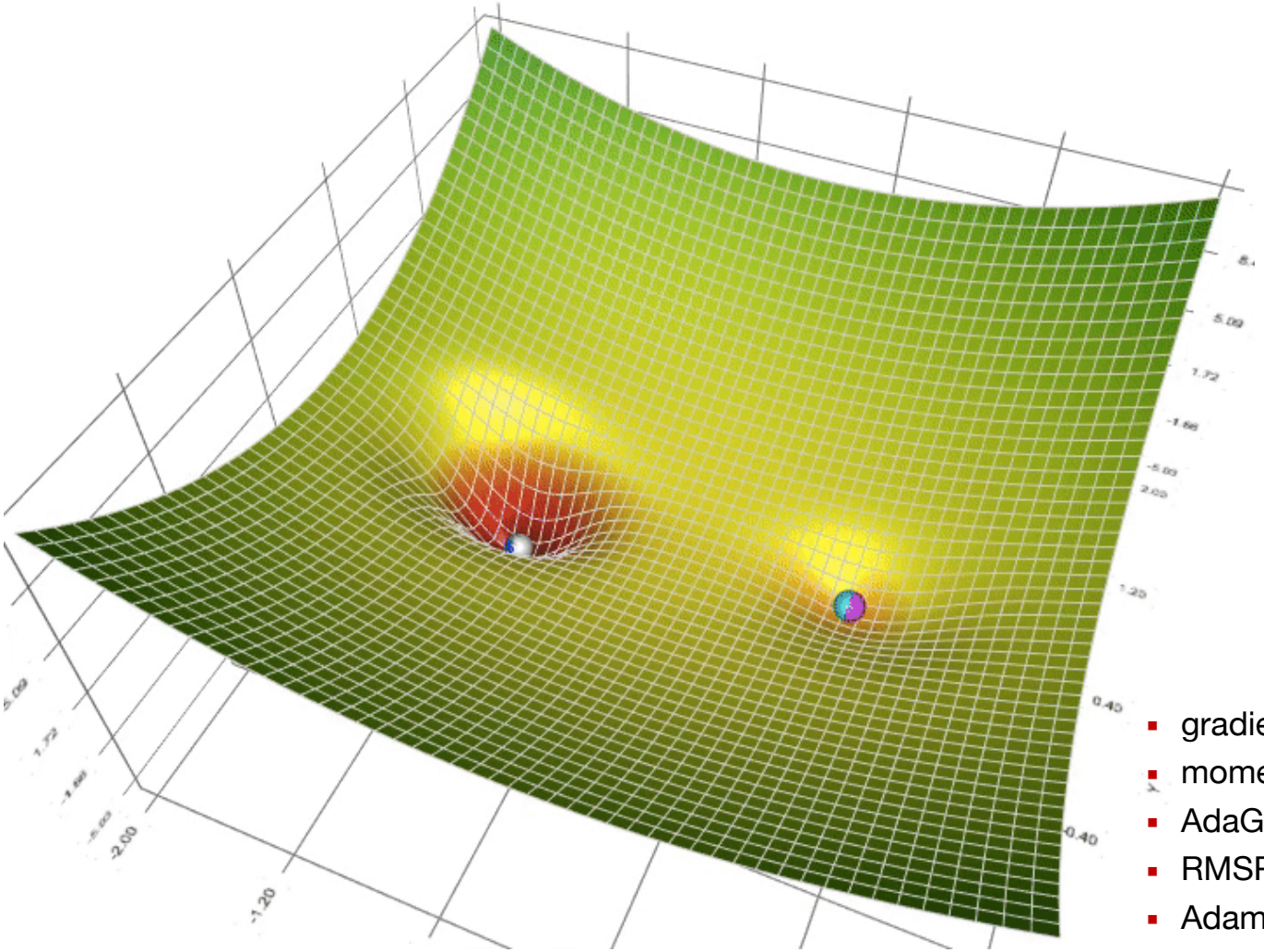
Optimization algorithms

- Adam optimization

$$W = W - \alpha \frac{V_{dW}^{cor}}{\sqrt{S_{dW}^{cor}} + \varepsilon}$$

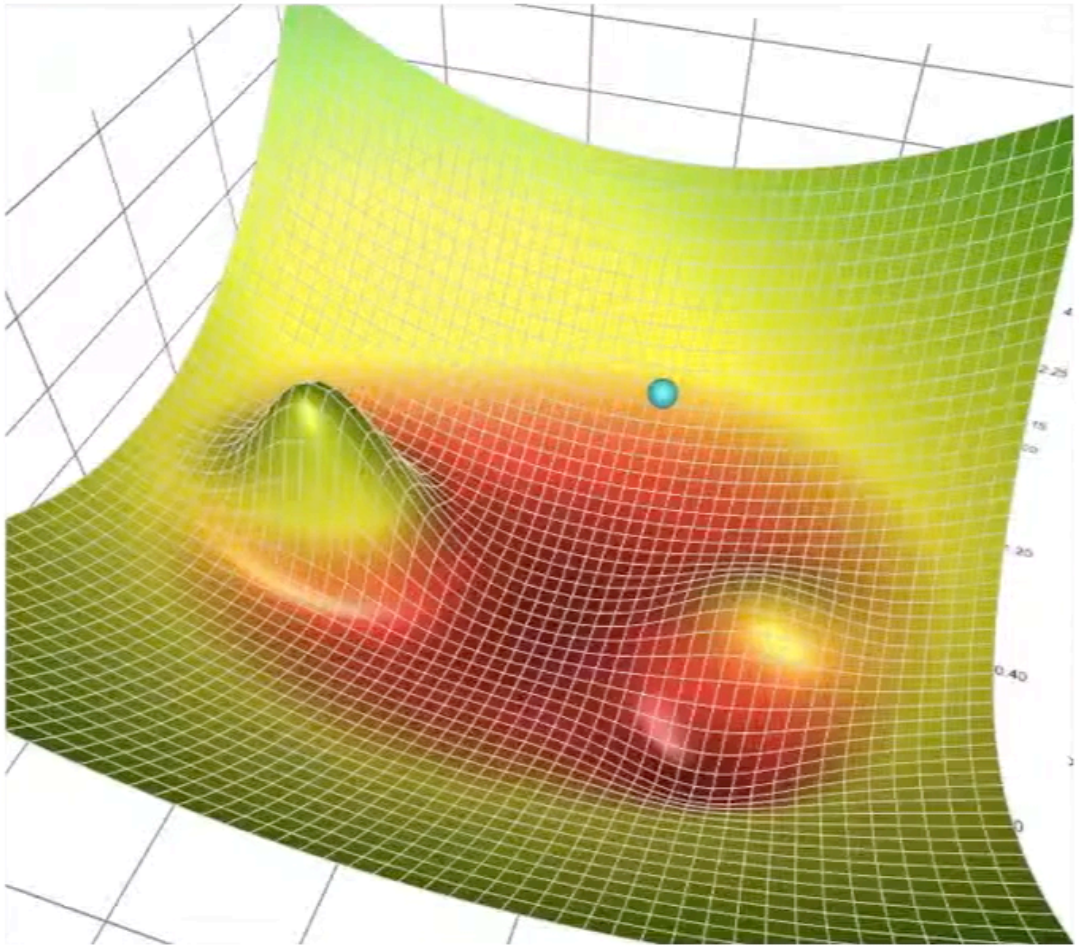
$$b = b - \alpha \frac{V_{db}^{cor}}{\sqrt{S_{db}^{cor}} + \varepsilon}$$

Optimization algorithms



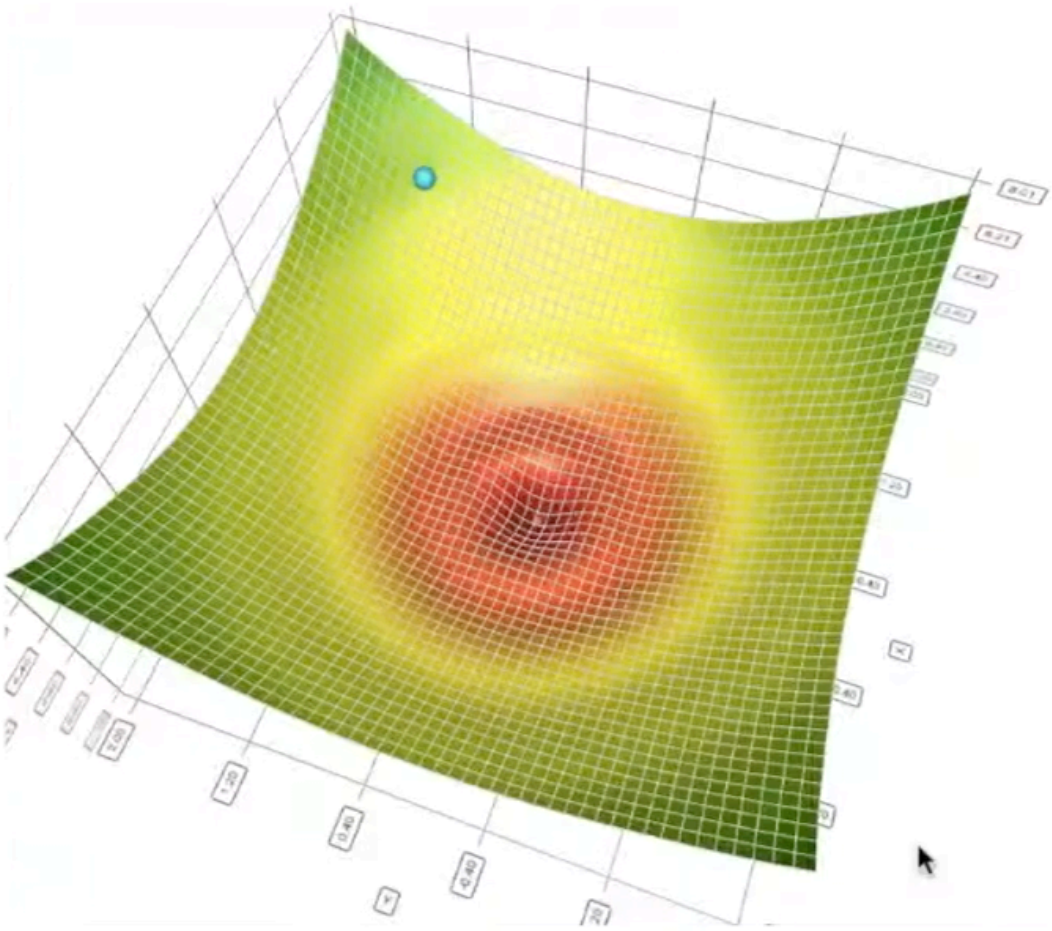
- gradient descent (cyan)
- momentum (magenta)
- AdaGrad (white)
- RMSProp (green)
- Adam (blue)

Optimization algorithms



- gradient descent (cyan)
- momentum (magenta)
- AdaGrad (white)
- RMSProp (green)
- Adam (blue)

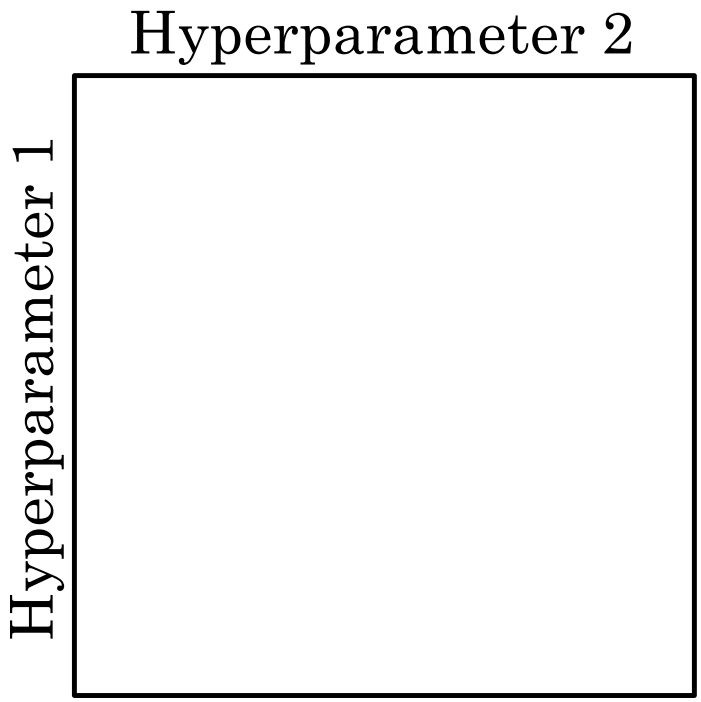
Optimization algorithms



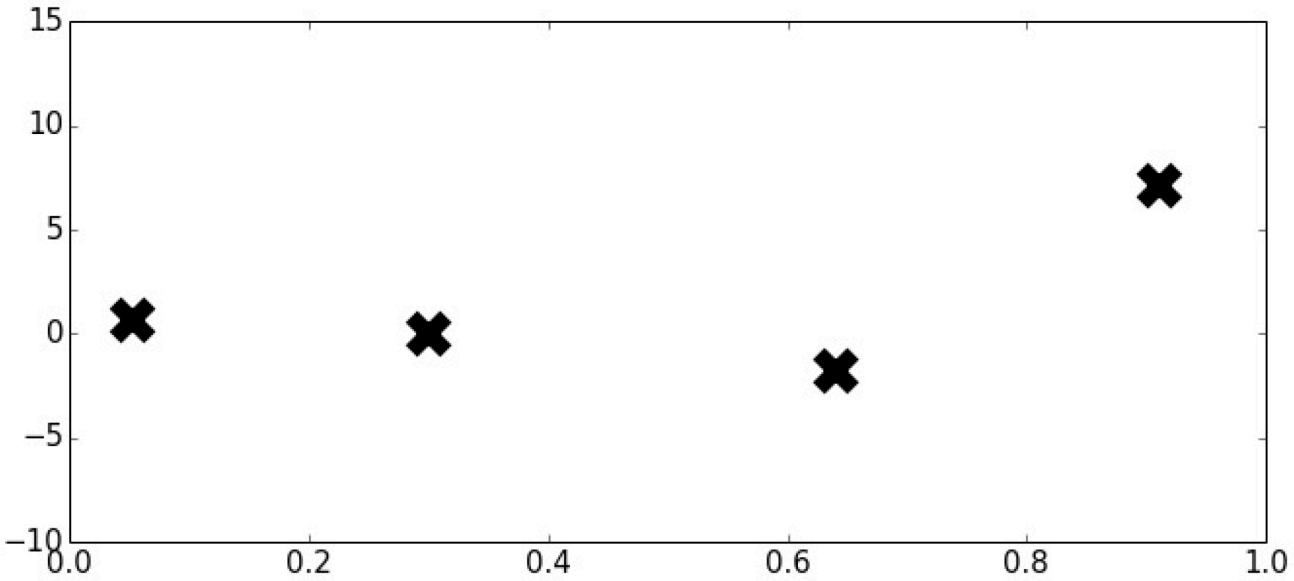
- gradient descent (cyan)
- momentum (magenta)
- AdaGrad (white)
- RMSProp (green)
- Adam (blue)

Optimization algorithms

- Hyperparameter tuning:
  - Do not use a grid
  - Go from coarse to fine
  - Pick appropriate scale
  - Run in parallel



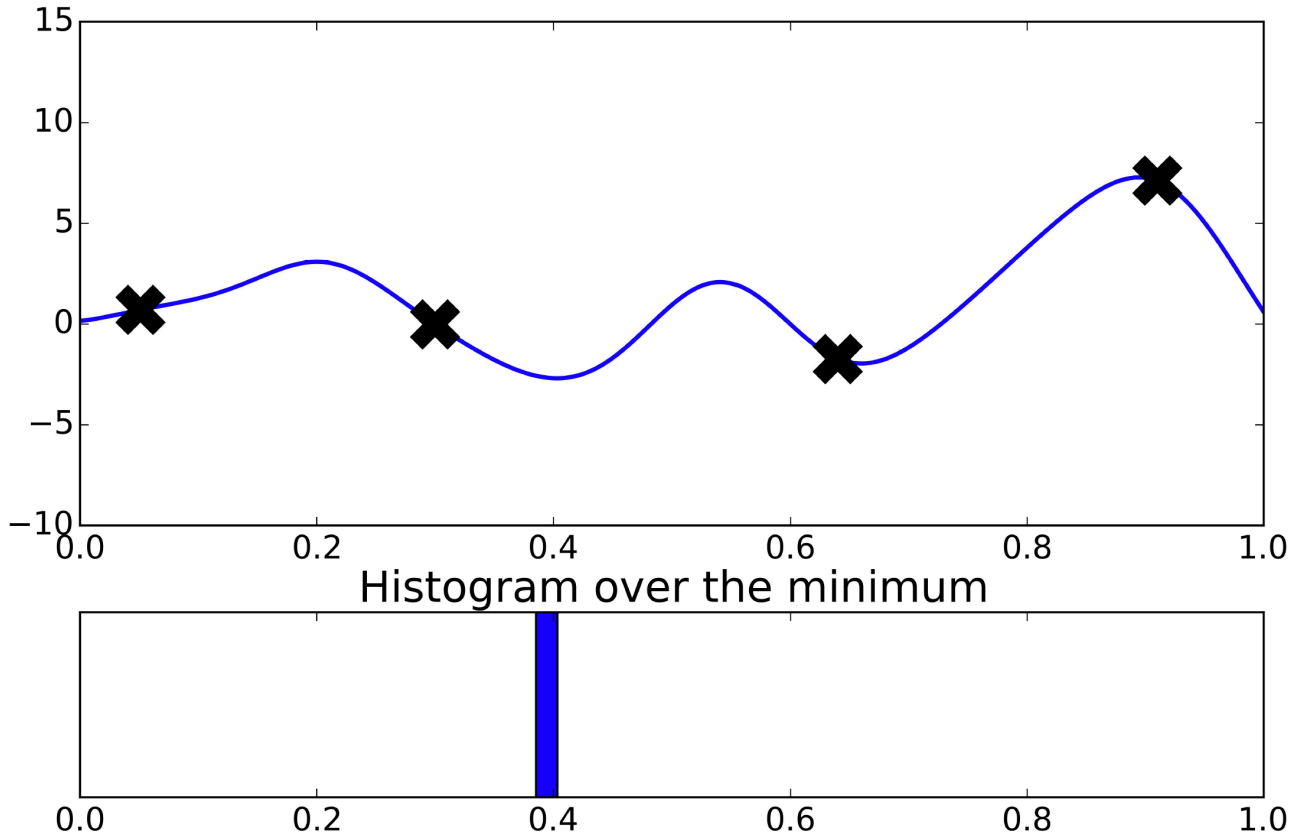
Optimization algorithms



- Where is the minimum of f?
- Where should the next evaluation be taken?

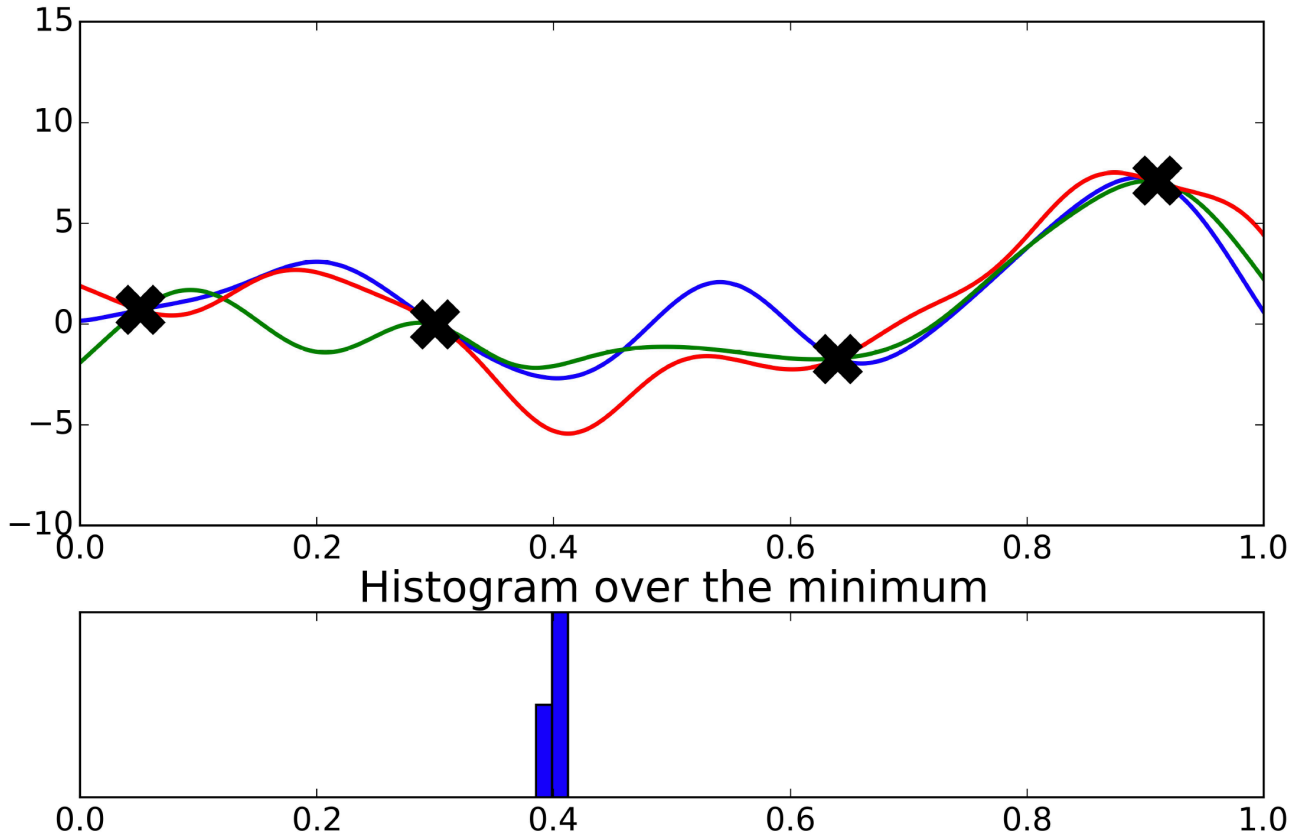
Optimization algorithms

- Hyperparameter tuning: Bayesian optimization — 1 curve



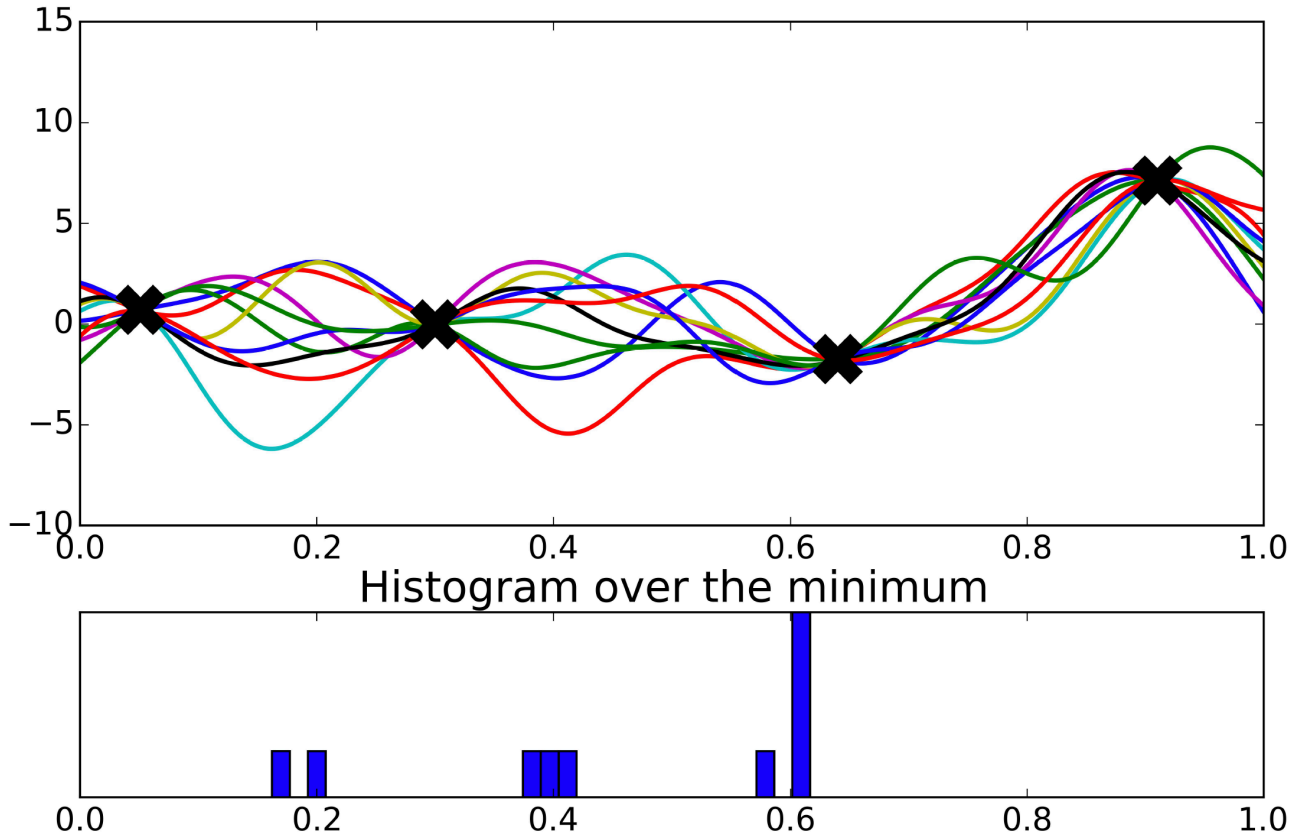
Optimization algorithms

- Hyperparameter tuning: Bayesian optimization — 3 curves



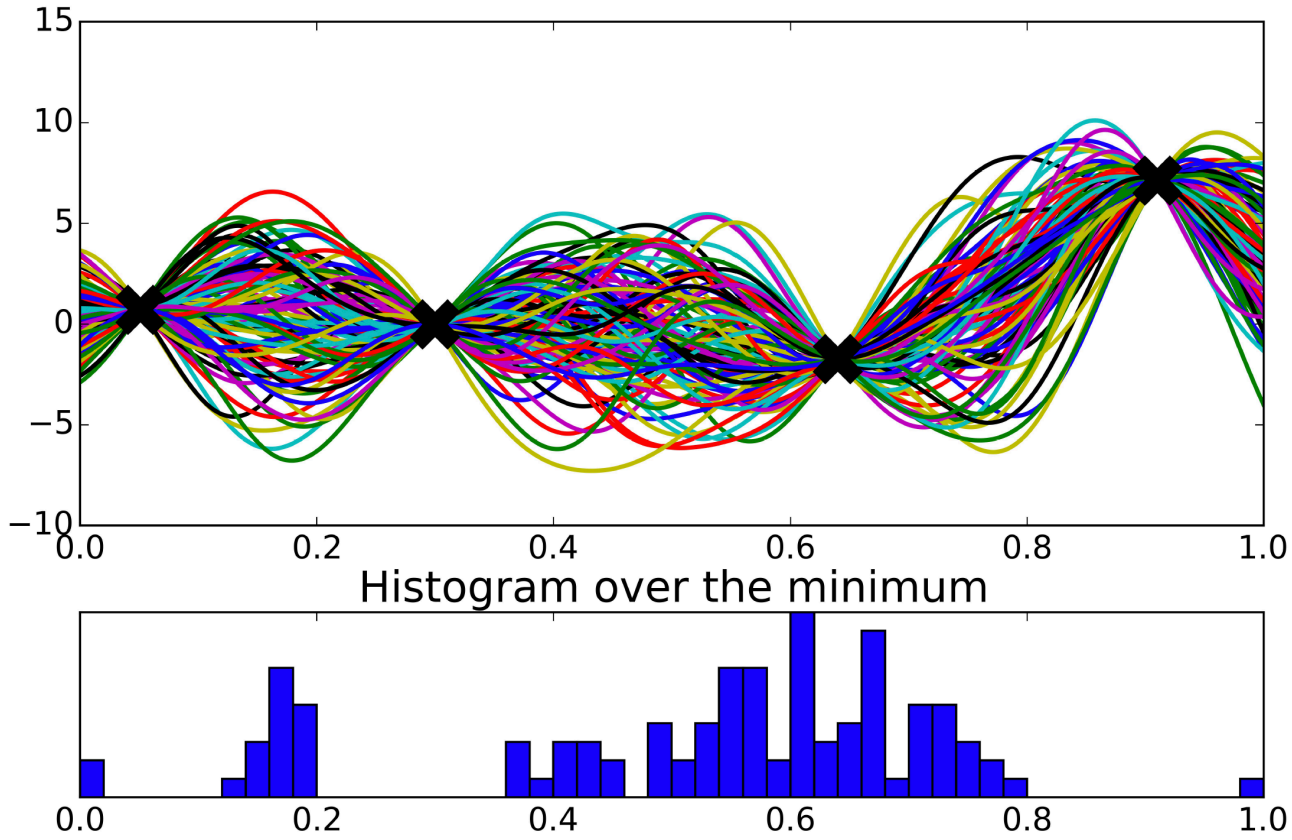
Optimization algorithms

- Hyperparameter tuning: Bayesian optimization — 10 curves



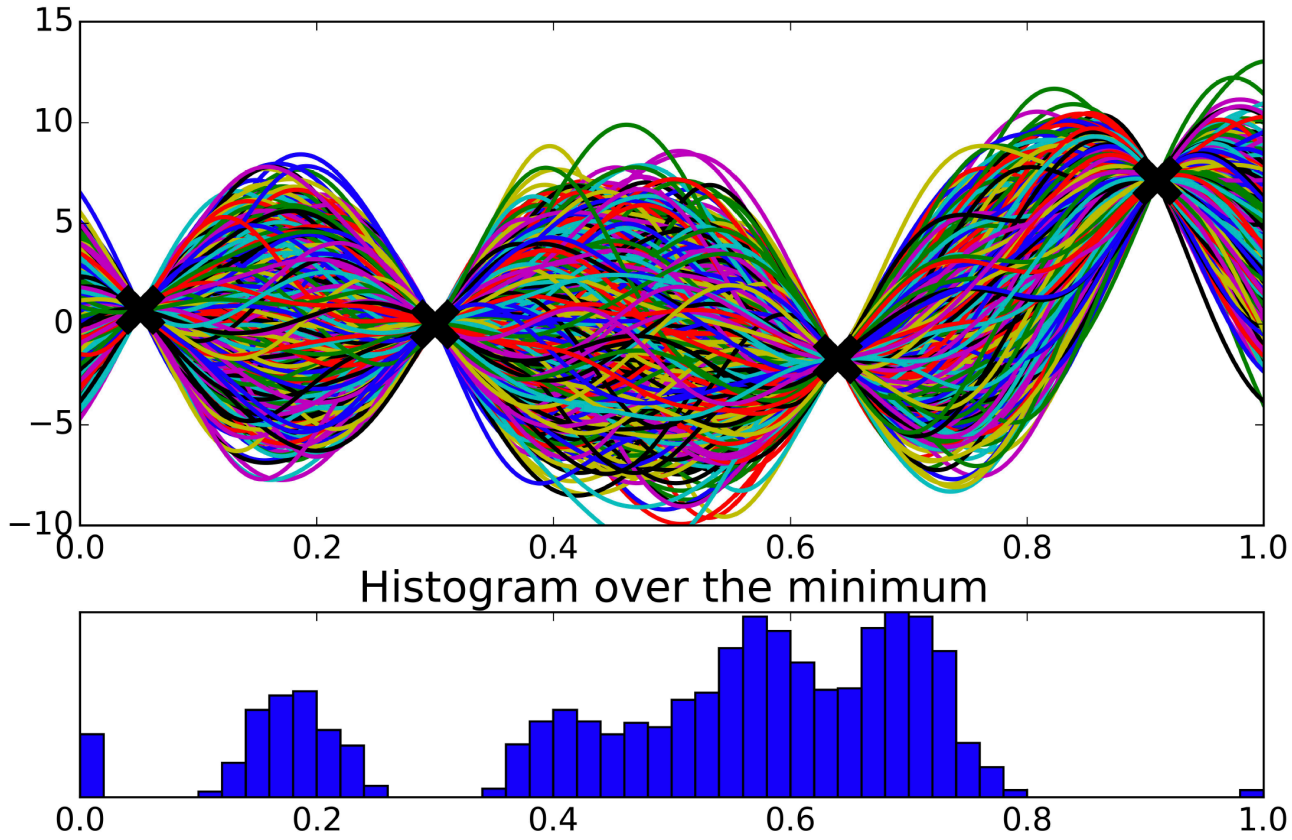
Optimization algorithms

- Hyperparameter tuning: Bayesian optimization — 100 curves



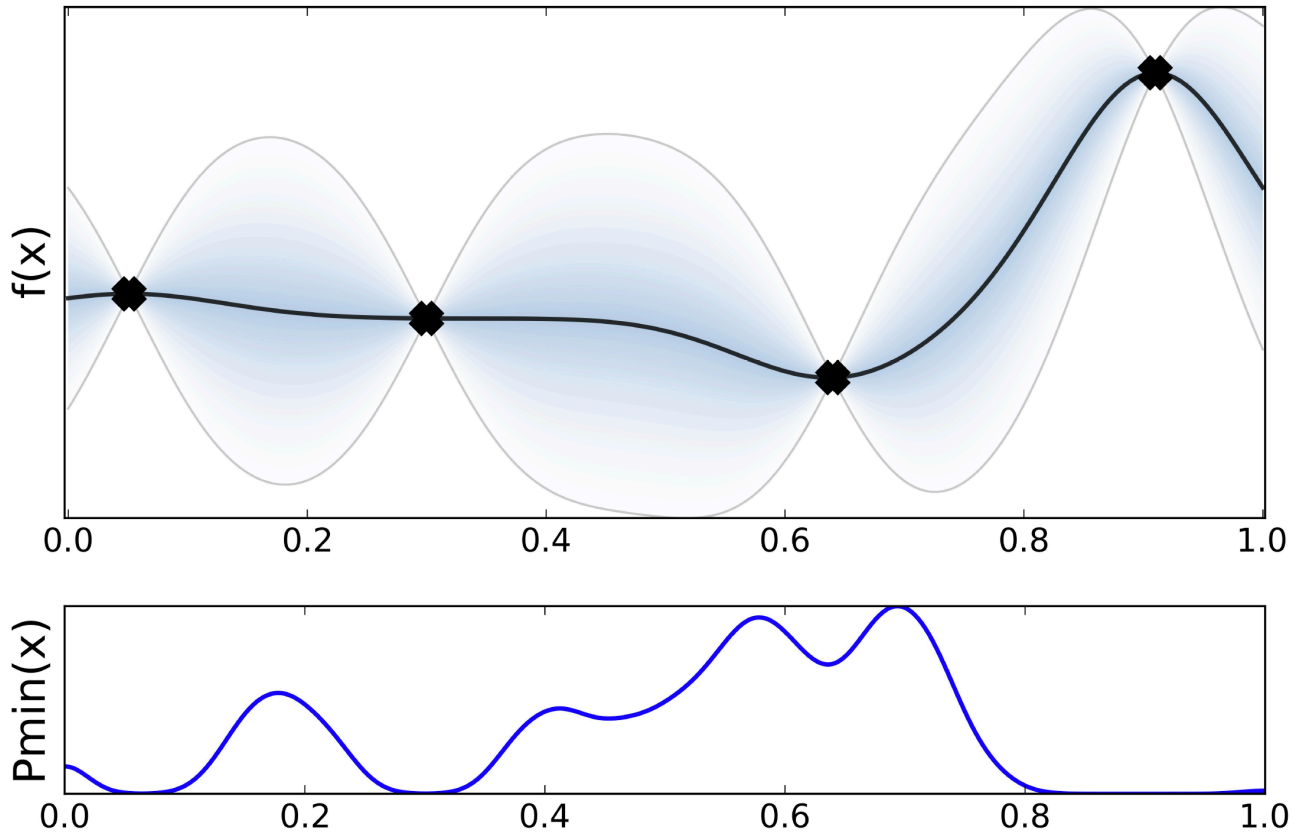
Optimization algorithms

- Hyperparameter tuning: Bayesian optimization — many curves



Optimization algorithms

- Hyperparameter tuning: Bayesian optimization — ∞ curves



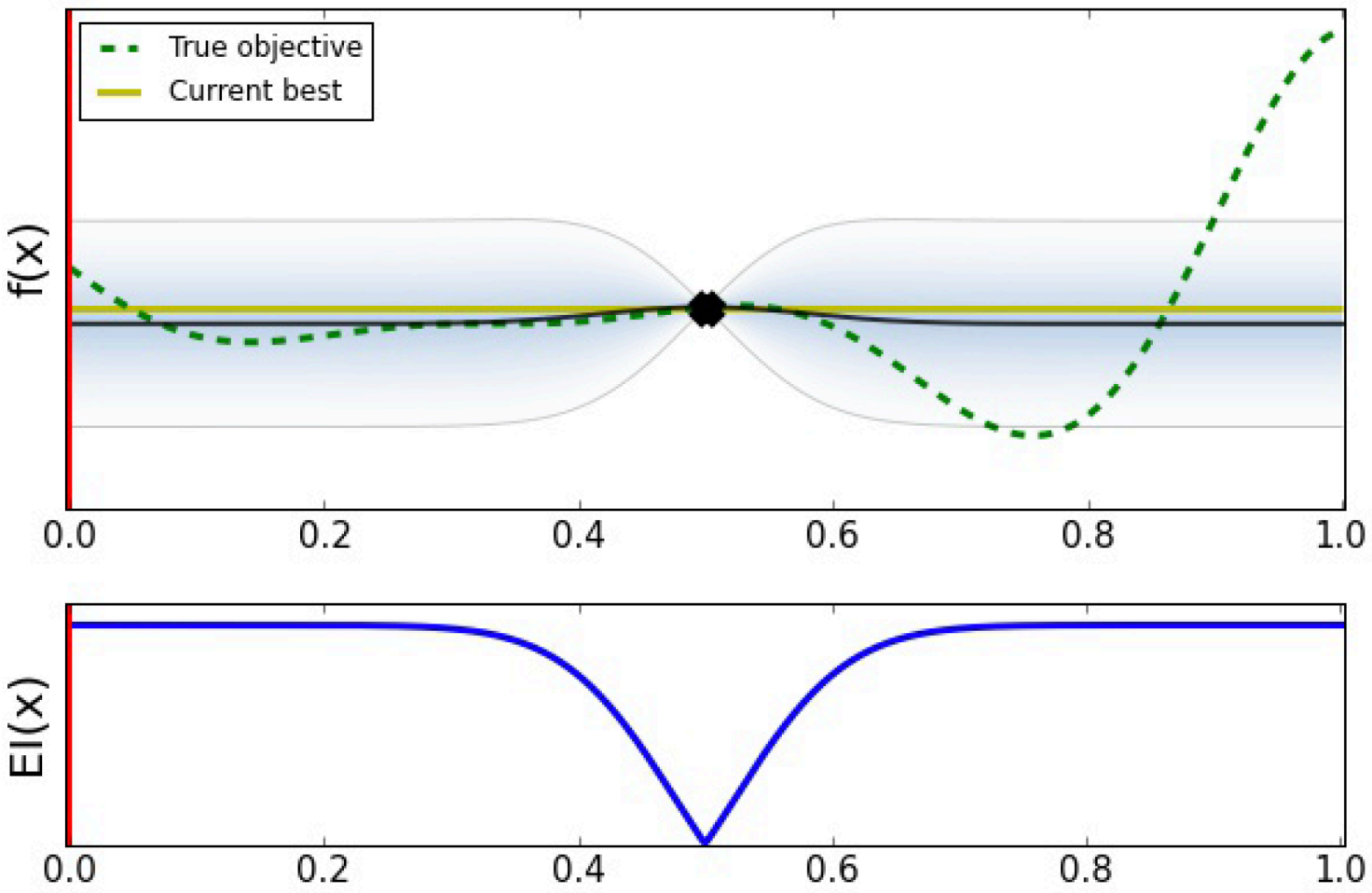
Optimization algorithms

- Use a surrogate model of f to carry out the optimization
  - $f(x) \sim \text{GP}(\mu(x), k(x, x'))$
  - $m(x)$ is the mean function
  - $k(x, x'; \theta)$ is the covariance function

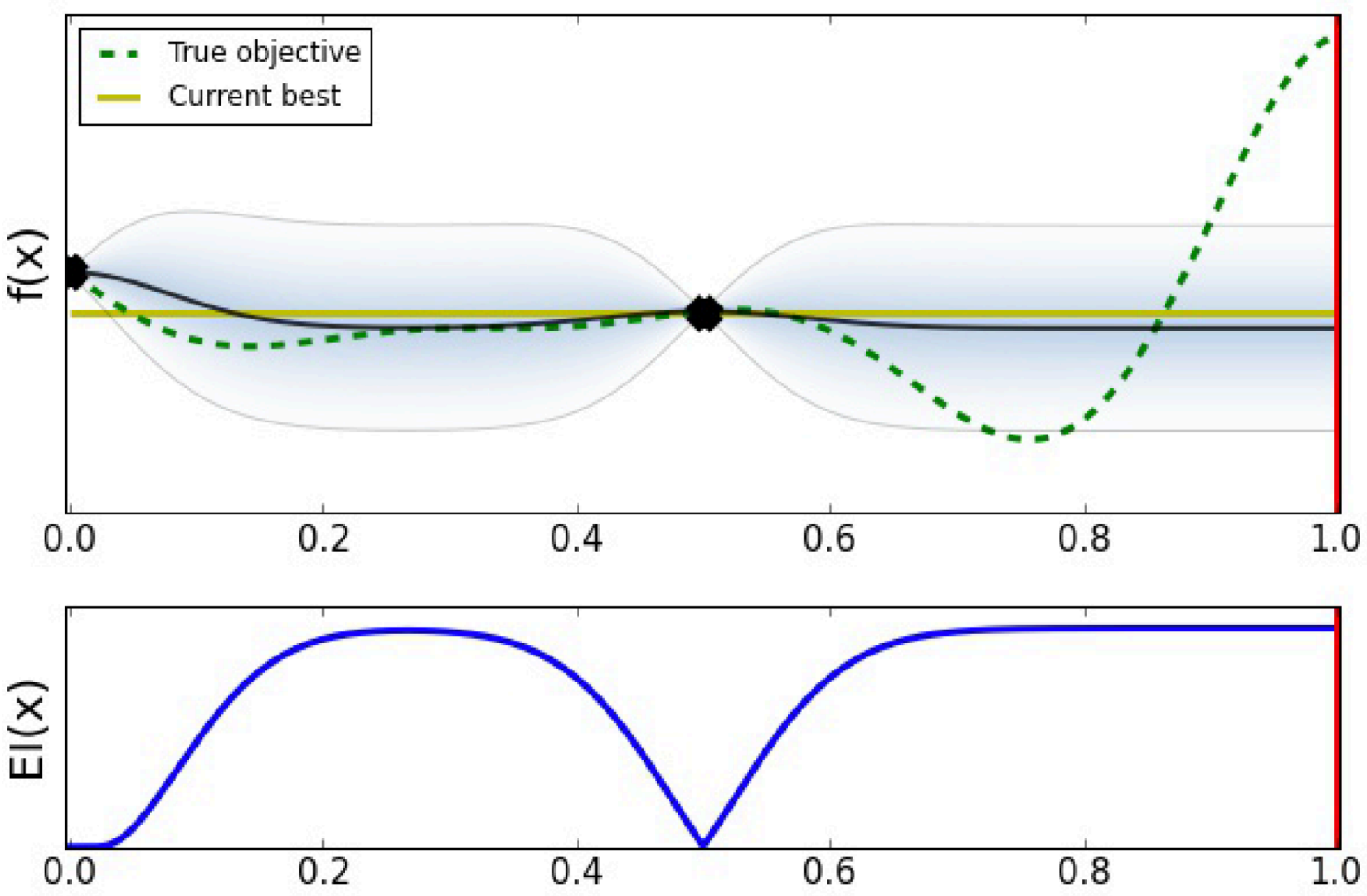
- Expected improvement EI(x) is given by

$$EI(\mathbf{x}; \theta, \mathscr{D}) = \int_y \max(0, y_{\text{best}} - y) p(y \mid \mathbf{x}; \theta, \mathscr{D}) \text{d}y$$

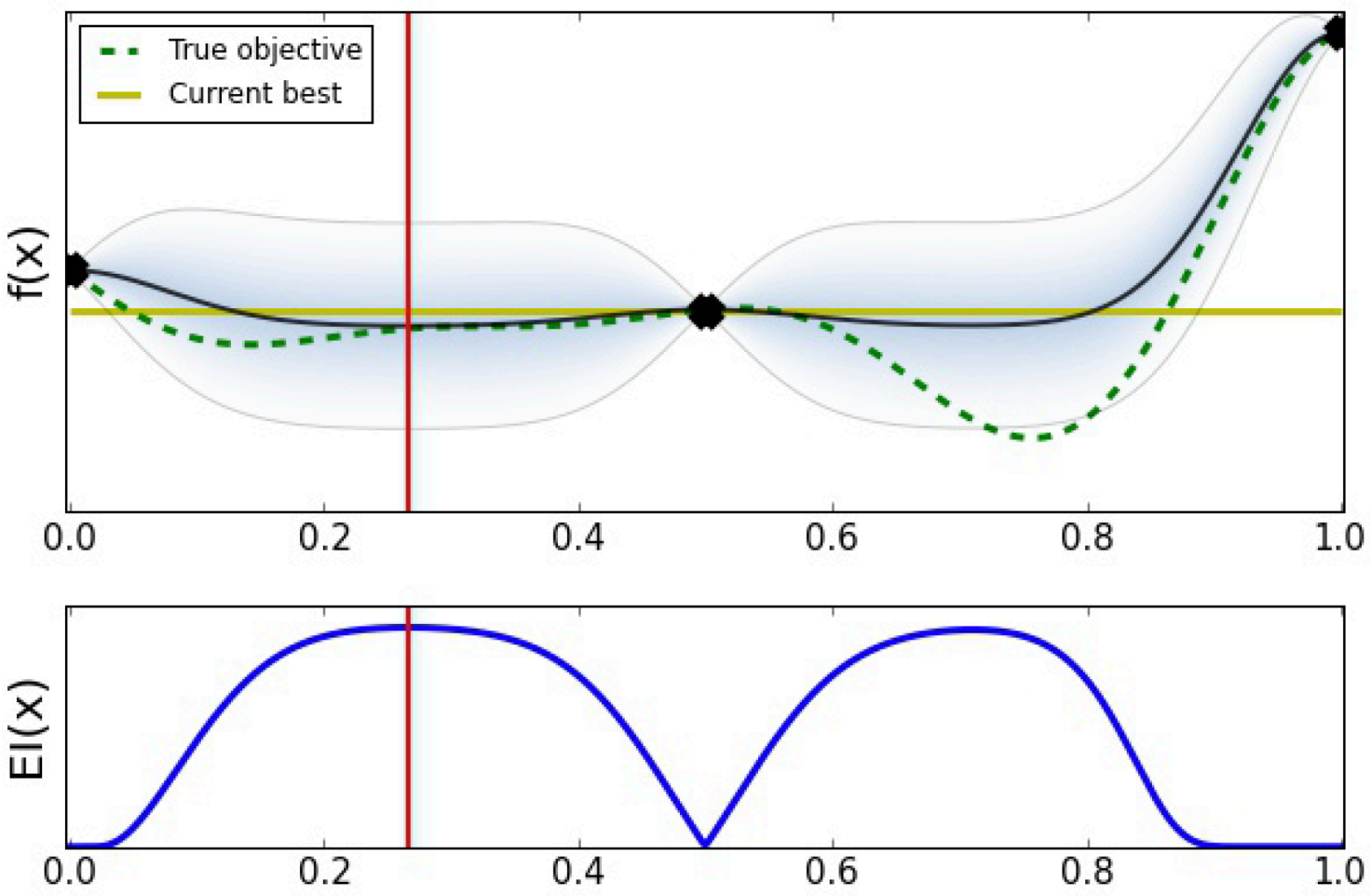
Optimization algorithms



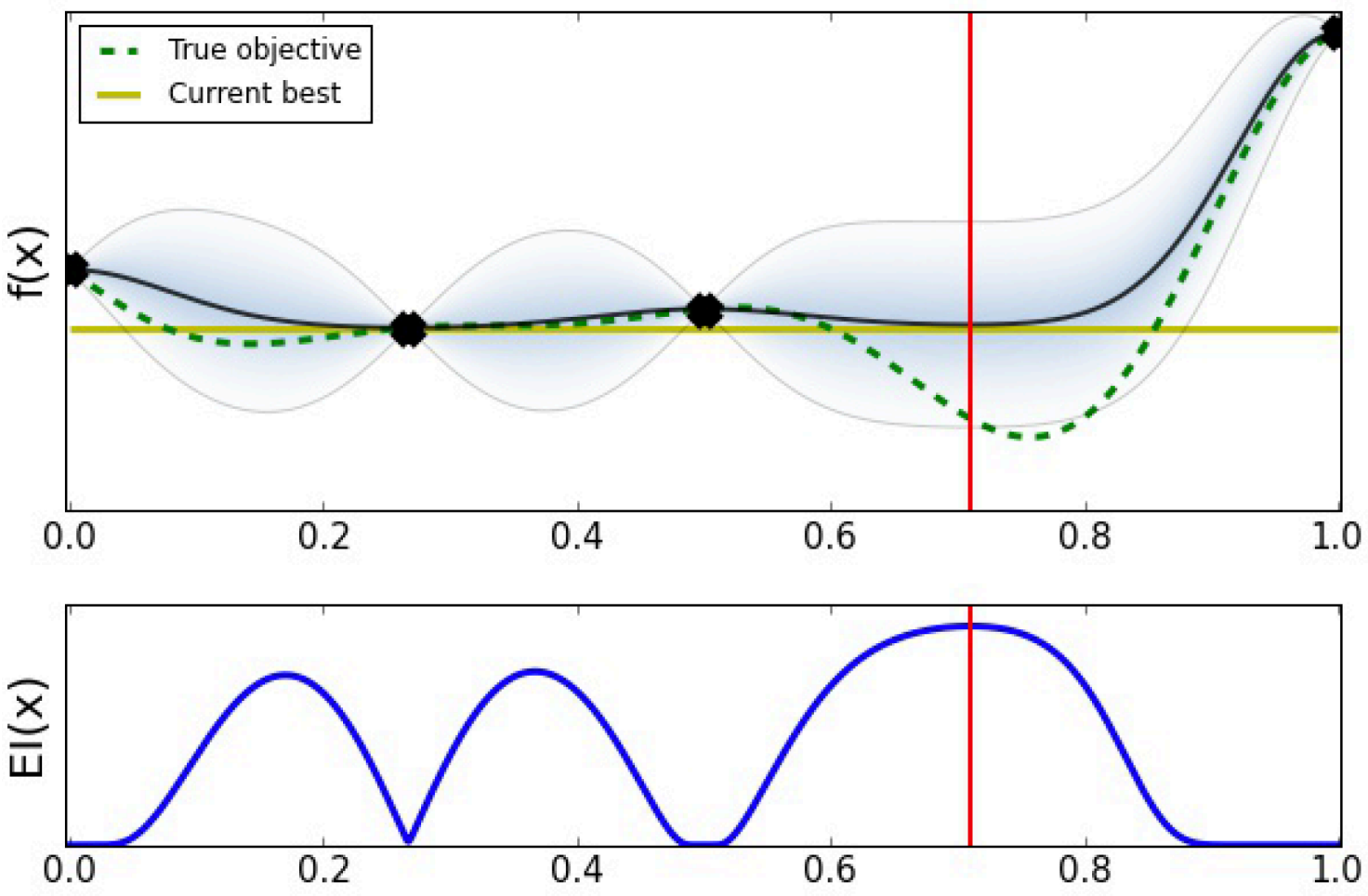
Optimization algorithms



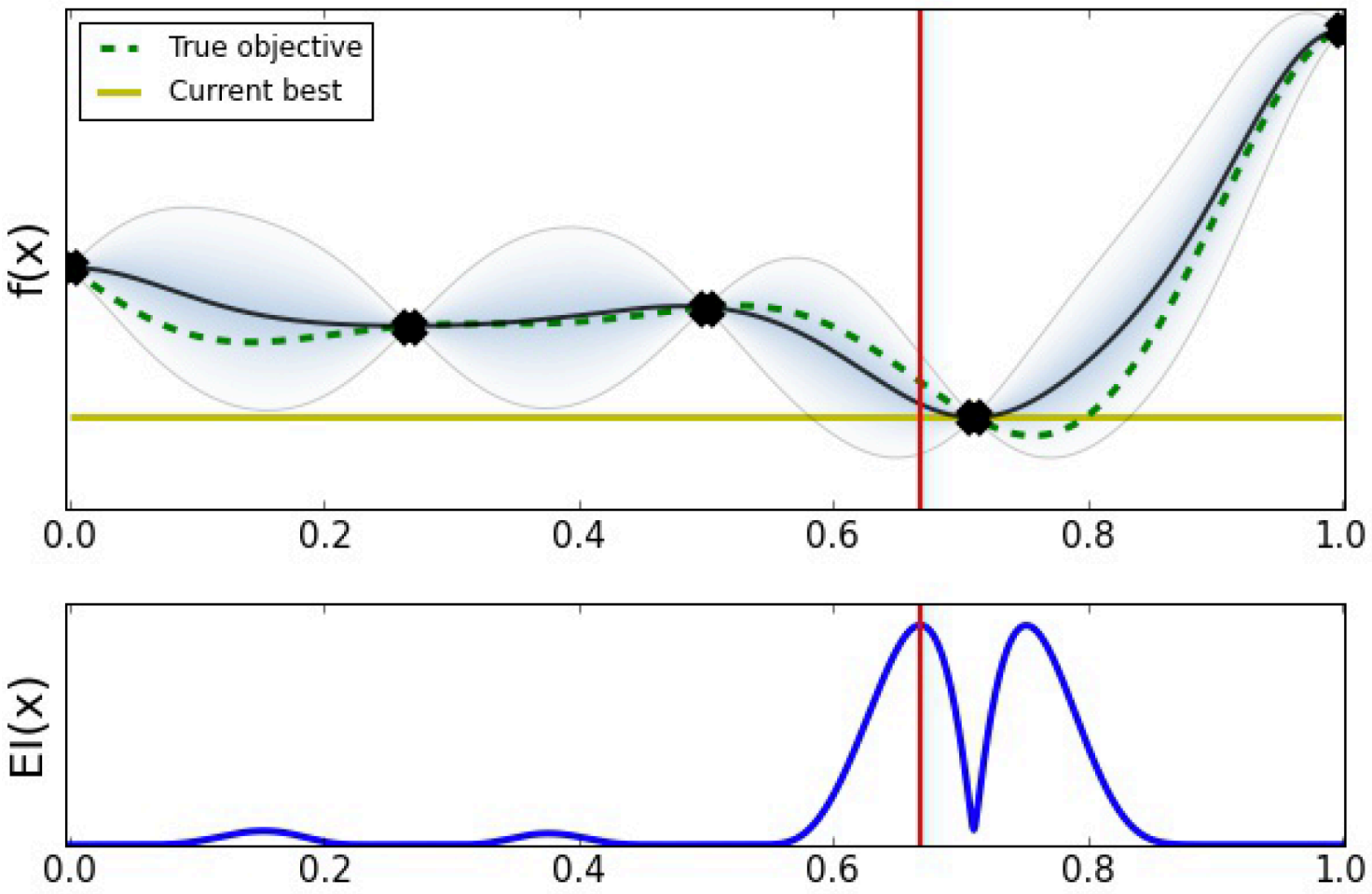
Optimization algorithms



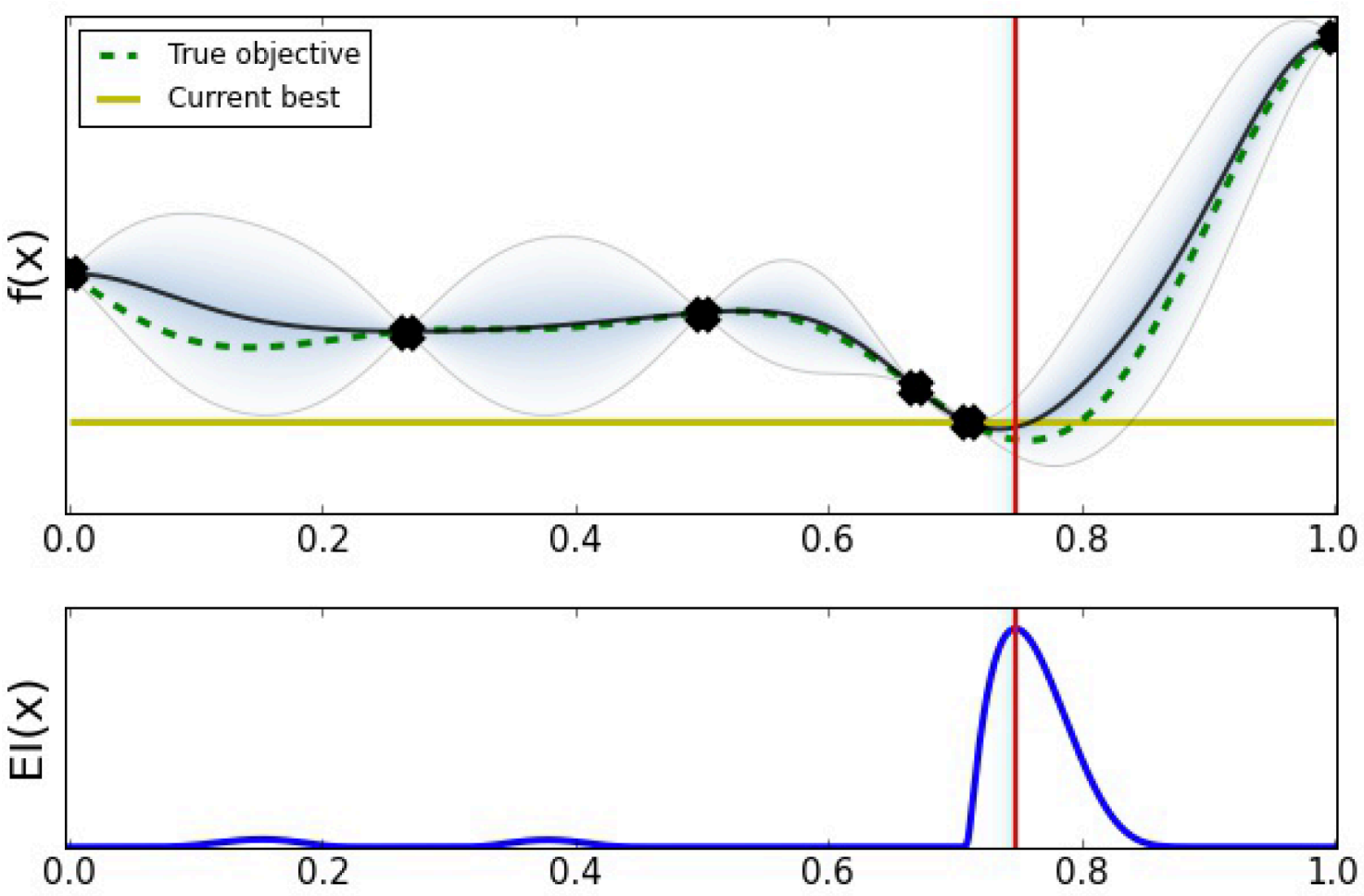
Optimization algorithms



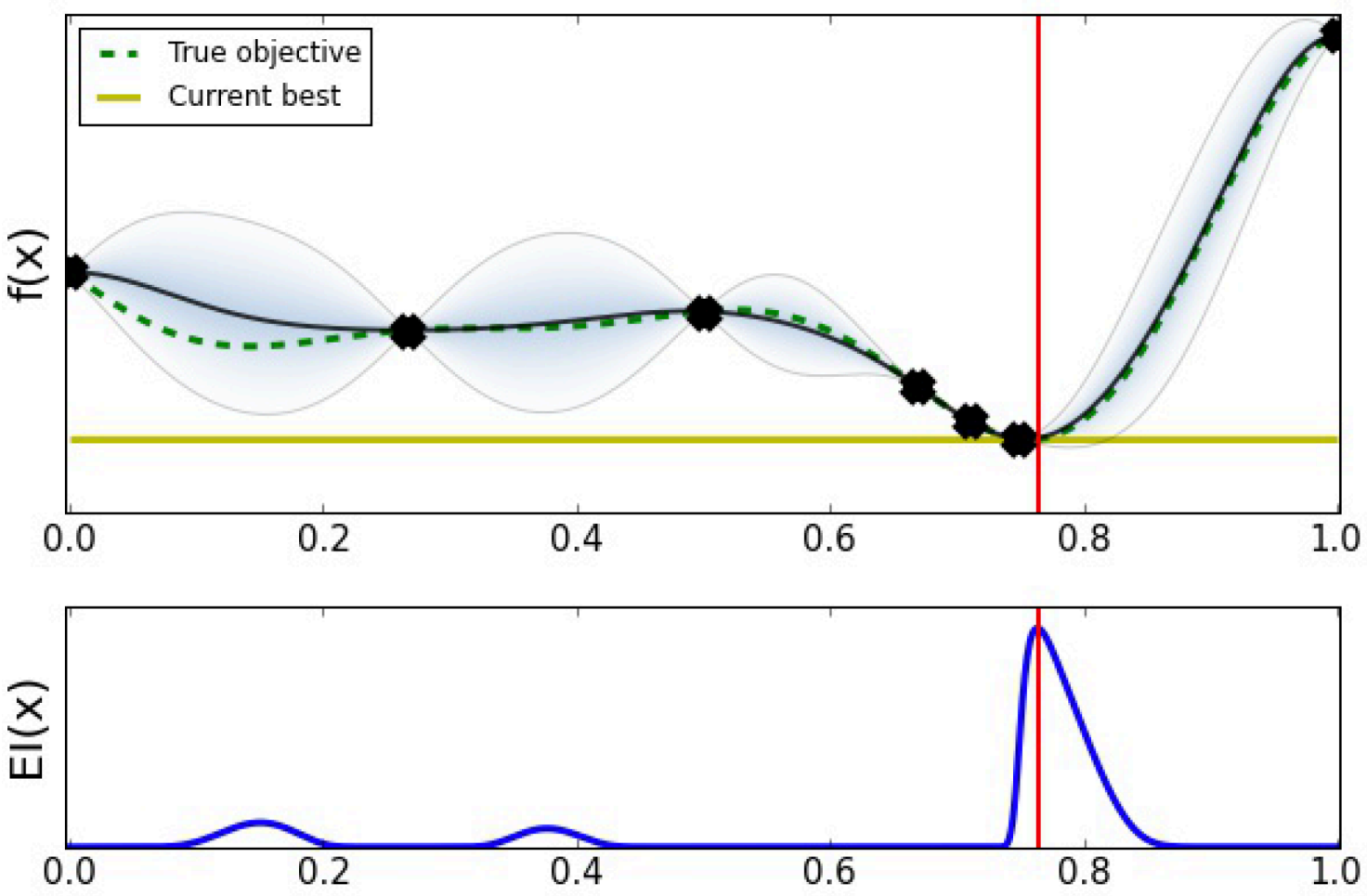
Optimization algorithms



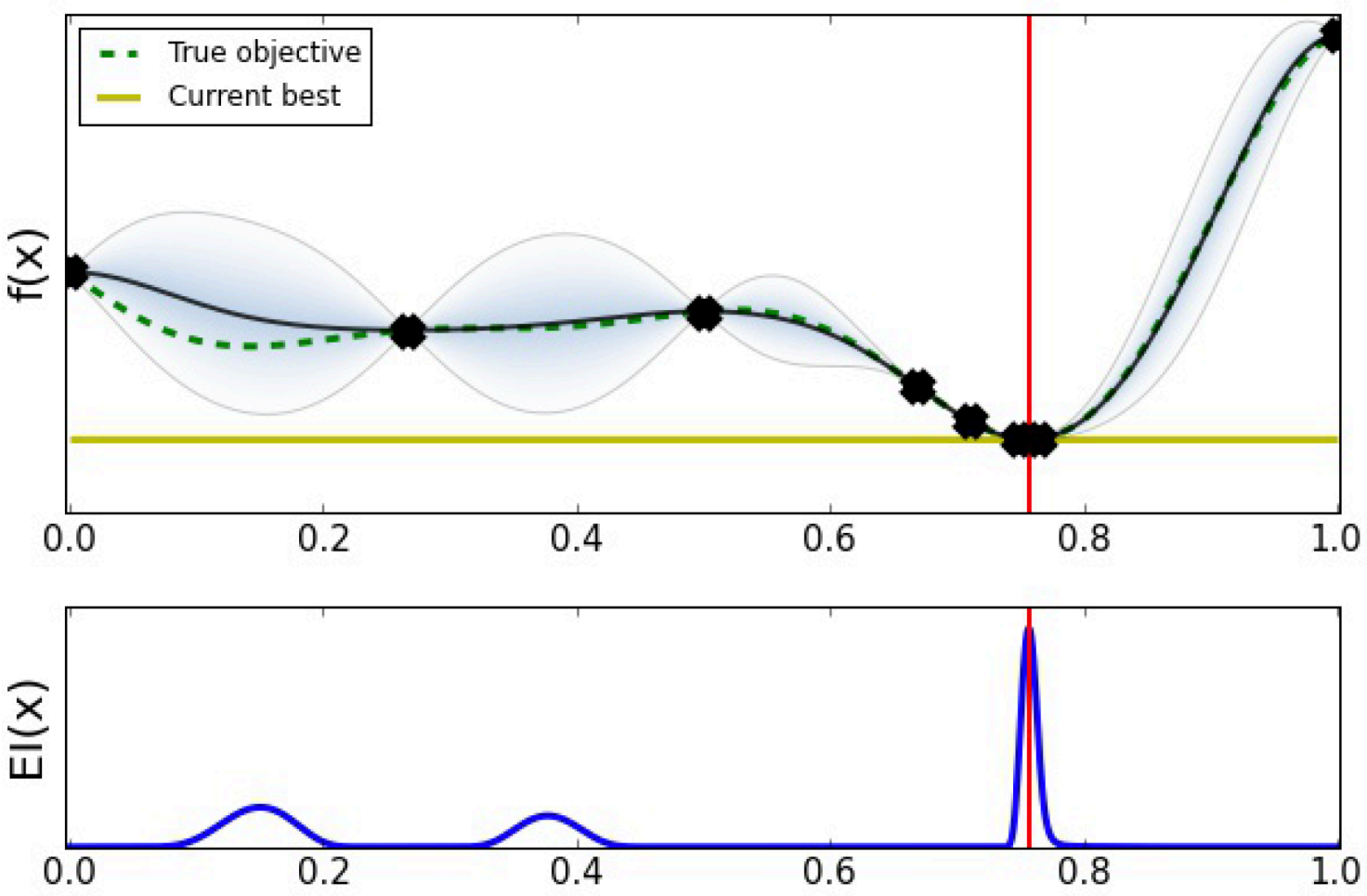
Optimization algorithms



Optimization algorithms



Optimization algorithms



Optimization algorithms

- Caffe/Caffe2
- CNTK
- DL4J
- Keras
- Lasagne
- mxnet
- PaddlePaddle
- TensorFlow
- Theano
- Torch