Robust nonlinear Optimization

Maren Mahsereci

Workshop on Uncertainty Quantification
09 / 15 / 2016, Sheffield

MAX-PLANCK-GESELLSCHAFT

Emmy Noether Group on Probabilistic Numerics
Department of Empirical Inference
Max Planck Institute for Intelligent Systems
Tübingen, Germany

Emmy Noether-Programm
DFG Deutsche Forschungsgemeinschaft

Robust optimization

...outline

- basics about greedy optimizers
  - GD and SGD: (stochastic) gradient descent
- robust stochastic optimization
  - example: step size adaptation
  - extending line searches
  - robust search directions

Typical scheme

. . . greedy and gradient based optimizer

$$x^* = \arg\min_x \mathcal{L}(x)$$

$$x_{i+1} \leftarrow x_i - \alpha_i s_i$$

1. s_i – which direction? → model objective function locally
2. α_i – how far? → prevent blow ups and stagnation
3. repeat

- needs to work for many different $\mathcal{L}(x)$

Typical scheme

...greedy and gradient based optimizer

$$x^* = \arg\min_x \mathcal{L}(x)$$

$$x_{i+1} \leftarrow x_i - \alpha_i s_i$$

1. s_i – which direction? → model objective function locally
2. α_i – how far? → prevent blow ups and stagnation
3. repeat

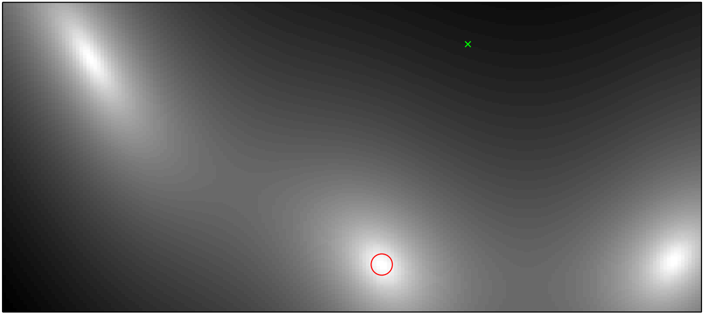
- needs to work for many different $\mathcal{L}(x)$

The steepest way downhill

... gradient descend finds local minimum

$$x^* = \arg\min_x \mathcal{L}(x)$$

$$x_{i+1} \leftarrow x_i - \alpha \nabla \mathcal{L}(x_i), \quad \alpha = const.$$

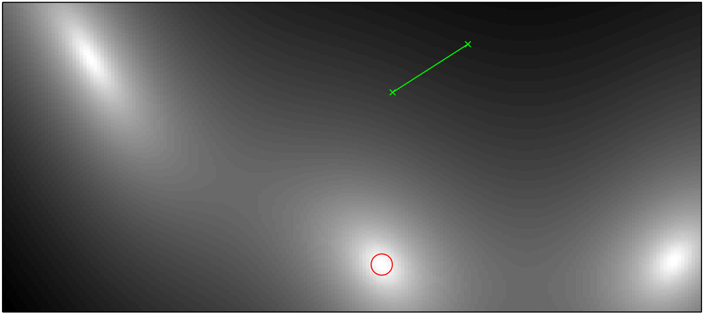


The steepest way downhill

. . . gradient descend finds local minimum

$$x^* = \arg\min_x \mathcal{L}(x)$$

$$x_{i+1} \leftarrow x_i - \alpha \nabla \mathcal{L}(x_i), \quad \alpha = const.$$

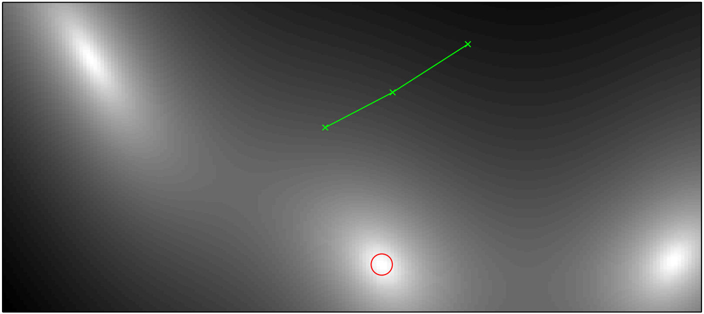


The steepest way downhill

. . . gradient descend finds local minimum

$$x^* = \arg\min_x \mathcal{L}(x)$$

$$x_{i+1} \leftarrow x_i - \alpha \nabla \mathcal{L}(x_i), \quad \alpha = const.$$

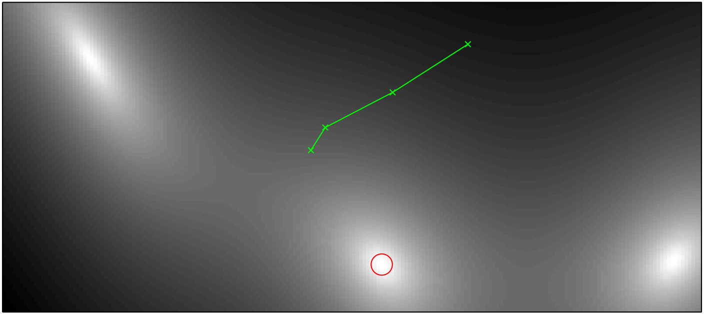


The steepest way downhill

... gradient descend finds local minimum

$$x^* = \arg\min_x \mathcal{L}(x)$$

$$x_{i+1} \leftarrow x_i - \alpha \nabla \mathcal{L}(x_i), \quad \alpha = const.$$

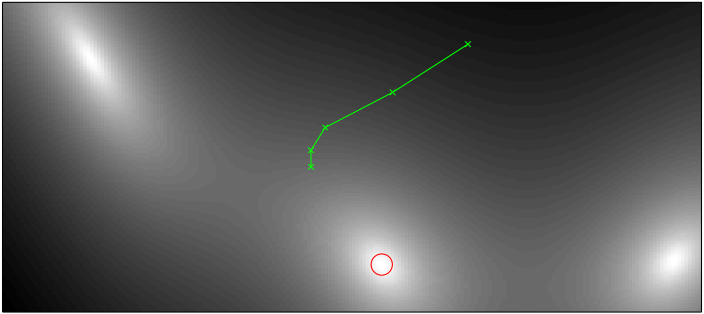


The steepest way downhill

. . . gradient descend finds local minimum

$$x^* = \arg\min_x \mathcal{L}(x)$$

$$x_{i+1} \leftarrow x_i - \alpha \nabla \mathcal{L}(x_i), \quad \alpha = const.$$

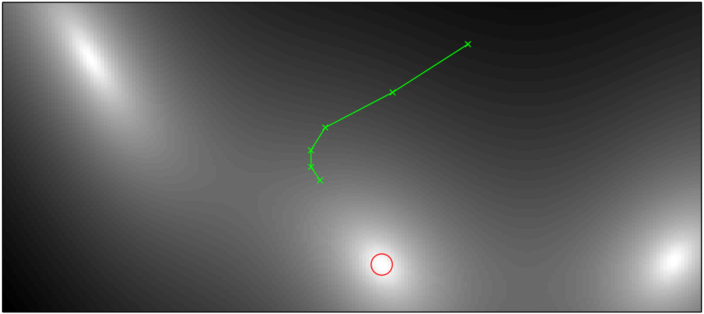


The steepest way downhill

. . . gradient descend finds local minimum

$$x^* = \arg\min_x \mathcal{L}(x)$$

$$x_{i+1} \leftarrow x_i - \alpha \nabla \mathcal{L}(x_i), \quad \alpha = const.$$

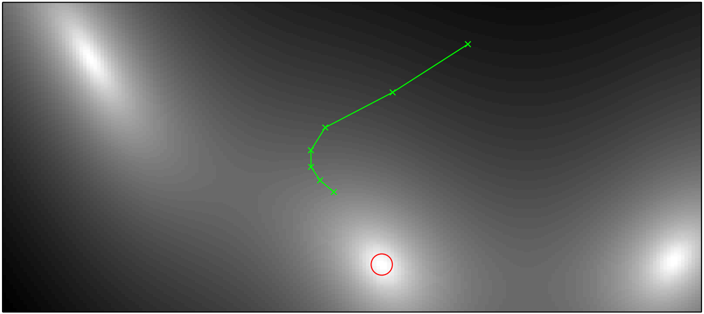


The steepest way downhill

. . . gradient descend finds local minimum

$$x^* = \arg\min_x \mathcal{L}(x)$$

$$x_{i+1} \leftarrow x_i - \alpha \nabla \mathcal{L}(x_i), \quad \alpha = const.$$

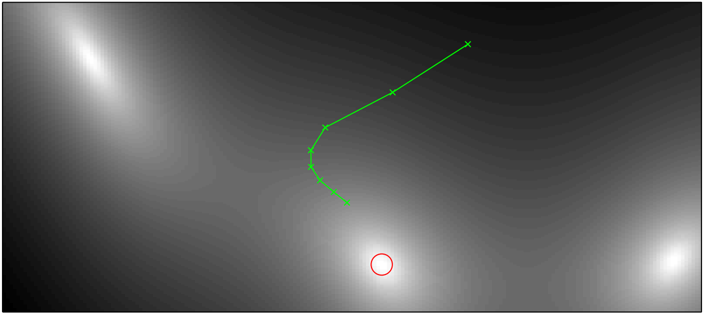


The steepest way downhill

. . . gradient descend finds local minimum

$$x^* = \arg\min_x \mathcal{L}(x)$$

$$x_{i+1} \leftarrow x_i - \alpha \nabla \mathcal{L}(x_i), \quad \alpha = const.$$

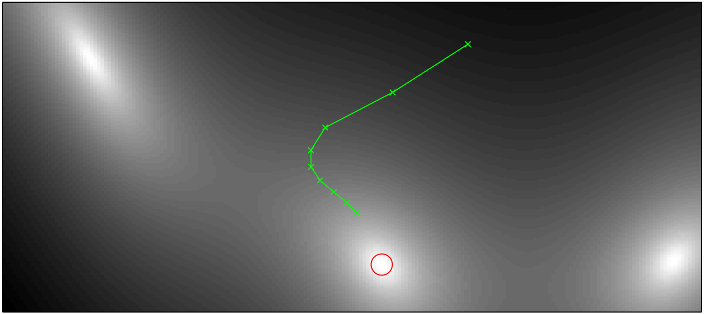


The steepest way downhill

. . . gradient descend finds local minimum

$$x^* = \arg\min_x \mathcal{L}(x)$$

$$x_{i+1} \leftarrow x_i - \alpha \nabla \mathcal{L}(x_i), \quad \alpha = const.$$

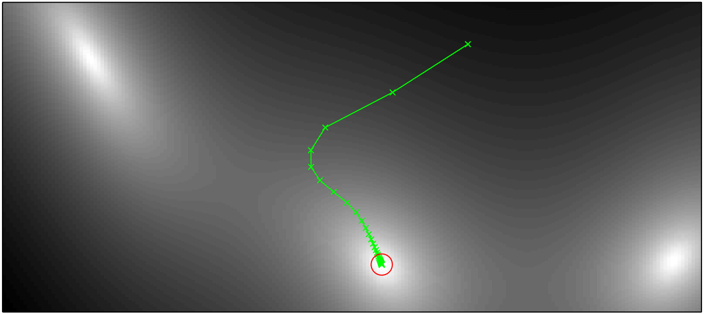


The steepest way downhill

... gradient descend finds local minimum

$$x^* = \arg\min_x \mathcal{L}(x)$$

$$x_{i+1} \leftarrow x_i - \alpha \nabla \mathcal{L}(x_i), \qquad \alpha = const.$$



Additional difficulty

.. noisy functions by mini-batching

$x^* = \arg\min_x \mathcal{L}(x)$

sometimes we do not know $-\nabla\mathcal{L}(x_i)$ precisely!

Additional difficulty

.. noisy functions by mini-batching

$$\mathcal{L}(x) := \frac{1}{M}\sum_{i=1}^{M} \ell(x, y_i) \approx \frac{1}{m}\sum_{j=1}^{m} \ell(x, y_j) =: \hat{\mathcal{L}}(x), \quad m \ll M$$

- compute only smaller sum over m
- hope that $\hat{\mathcal{L}}(x)$ approximates $\mathcal{L}(x)$ well
- smaller m means higher noise on $\nabla \mathcal{L}(x)$

Additional difficulty

.. noisy functions by mini-batching

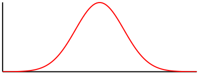
$$\mathcal{L}(x) := \frac{1}{M}\sum_{i=1}^{M}\ell(x, y_i) \approx \frac{1}{m}\sum_{j=1}^{m}\ell(x, y_j) =: \hat{\mathcal{L}}(x), \qquad m \ll M$$

- compute only smaller sum over m
- hope that $\hat{\mathcal{L}}(x)$ approximates $\mathcal{L}(x)$ well
- smaller m means higher noise on $\nabla\mathcal{L}(x)$

for iid. mini-batches, noise is approximately Gaussian

$$\mathcal{L}(x) = \hat{\mathcal{L}}(x) + \epsilon, \qquad \epsilon \sim \mathcal{N}\left(0, \mathcal{O}\left(\frac{M-m}{m}\right)\right)$$

$\hat{\mathcal{L}}(x) \sim$

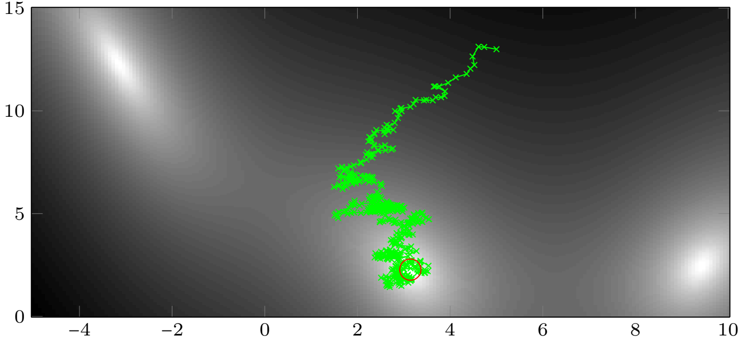


The steepest way downhill

... in expectation: SGD finds local minimum, too.

$$x^* = \arg\min_x \mathcal{L}(x)$$

$$x_{i+1} \leftarrow x_i - \alpha \hat{\nabla \mathcal{L}}(x_i), \qquad \alpha = const.$$

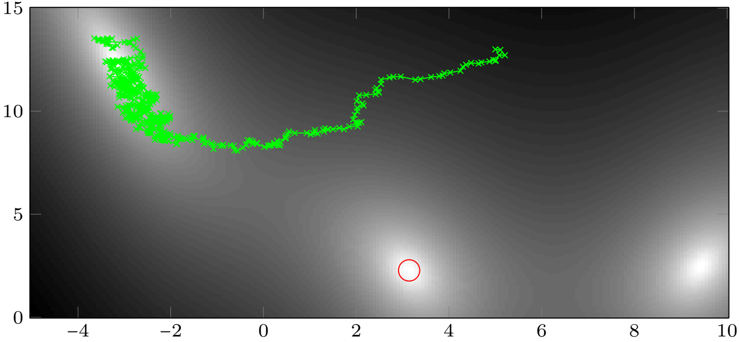


The steepest way downhill

... in expectation: SGD finds local minimum, too.

$$x^* = \arg\min_x \mathcal{L}(x)$$

$$x_{i+1} \leftarrow x_i - \alpha \hat{\nabla \mathcal{L}}(x_i), \quad \alpha = const.$$



Step size adaptation

... by line searches

$$x_{i+1} \leftarrow x_i - \alpha_i s_i$$

so far α was constant and hand-chosen!

- line searches automatically choose step sizes

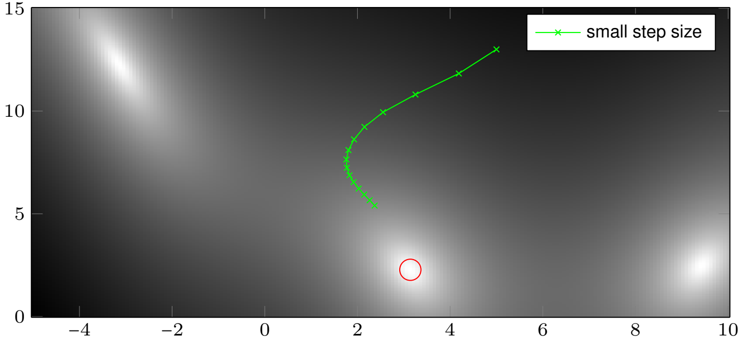
Line searches

automated learning rate adaptation

$x^* = \arg\min_x \mathcal{L}(x)$

$$x_{i+1} \leftarrow x_i - \alpha_i \nabla \mathcal{L}(x_i)$$

set scalar step size α_i given direction $-\nabla \mathcal{L}(x_i)$



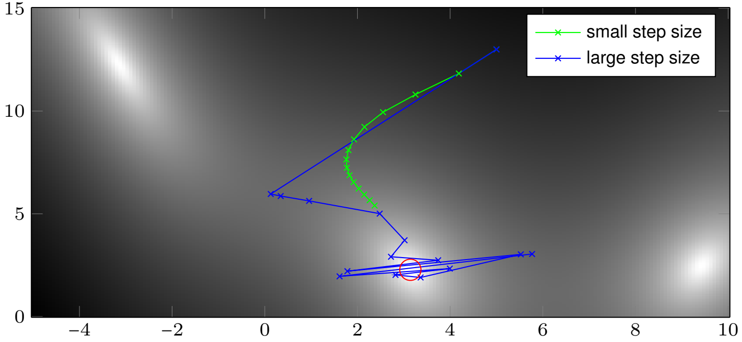
Line searches

automated learning rate adaptation

$x^* = \arg\min_x \mathcal{L}(x)$

$$x_{i+1} \leftarrow x_i - \alpha_i \nabla \mathcal{L}(x_i)$$

set scalar step size α_i given direction $-\nabla \mathcal{L}(x_i)$



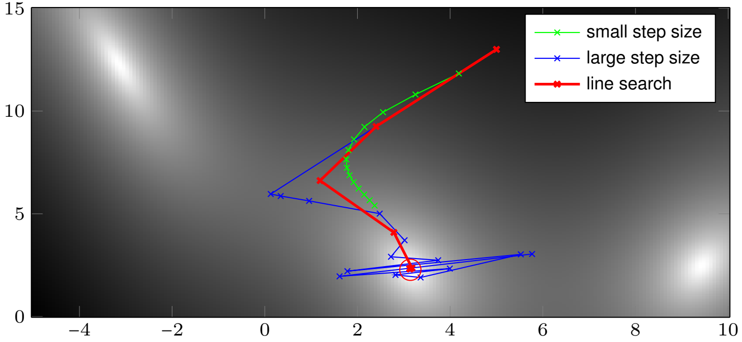
Line searches

automated learning rate adaptation

$x^* = \arg\min_x \mathcal{L}(x)$

$$x_{i+1} \leftarrow x_i - \alpha_i \nabla \mathcal{L}(x_i)$$

set scalar step size α_i given direction $-\nabla \mathcal{L}(x_i)$



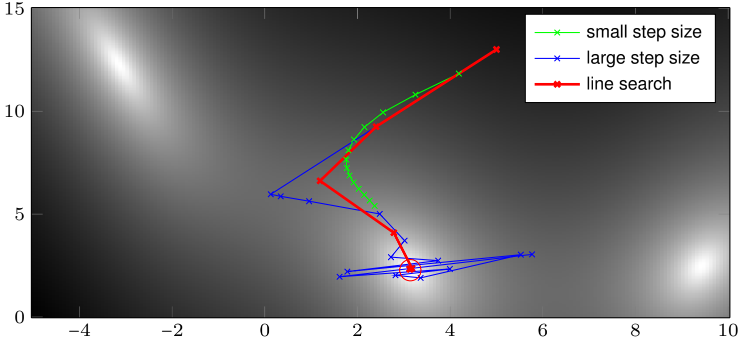
Line searches

automated learning rate adaptation

$x^* = \arg\min_x \mathcal{L}(x)$

$$x_{i+1} \leftarrow x_i - \alpha_i \nabla \mathcal{L}(x_i)$$

set scalar step size α_i given noisy direction $-\nabla \hat{\mathcal{L}}(x_i)$



Line searches break in stochastic setting!

Step size adaptation

... by line searches

$$x_{i+1} \leftarrow x_i - \alpha_i s_i$$

- line searches automatically choose step sizes
- very fast subroutines called in each optimization step
- control blow up or stagnation
- they do not work in stochastic optimization problems!

Step size adaptation

... by line searches

$$x_{i+1} \leftarrow x_i - \alpha_i s_i$$

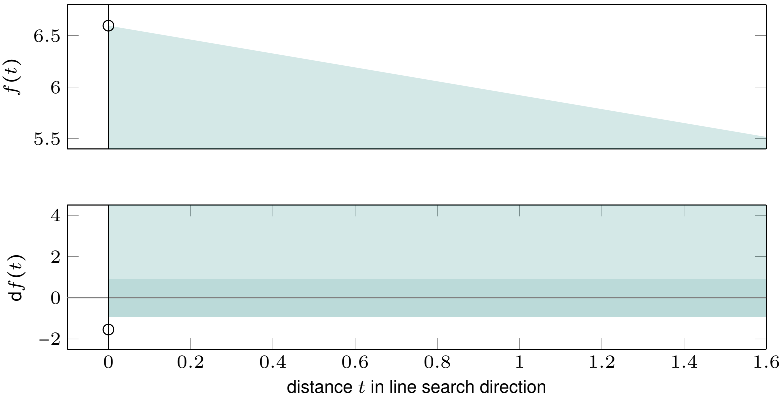
- line searches automatically choose step sizes
- very fast subroutines called in each optimization step
- control blow up or stagnation
- they do not work in stochastic optimization problems!

small outline

- introduce classic (noise free) line searches
- translate concept to language of probability
- get a new algorithm robust to noise

Classic line searches

Initial evaluation ≡ current position of optimizer $x^* = \arg\min_x f(x), \quad x_{i+1} \leftarrow x_i - t s_{i+1}$



Wolfe conditions: accept when [Wolfe, SIAM Review, 1969]

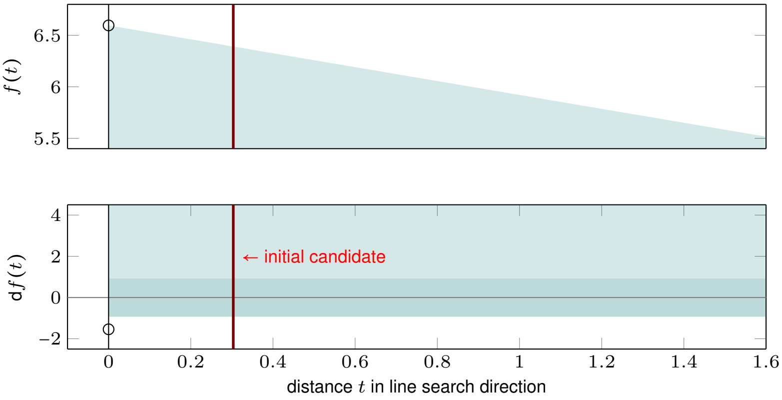
$$f(t) \leq f(0) + c_1 t f'(0) \quad \text{(W-I)}$$

$$f'(t) \geq c_2 f'(0) \quad \text{(W-IIa)}$$

$$|f'(t)| \leq c_2 |f'(0)| \quad \text{(W-IIb)}$$

Classic line searches

Search: candidate # 1 $\quad x^* = \arg\min_x f(x), \qquad x_{i+1} \leftarrow x_i - t s_{i+1}$



Wolfe conditions: accept when [Wolfe, SIAM Review, 1969]

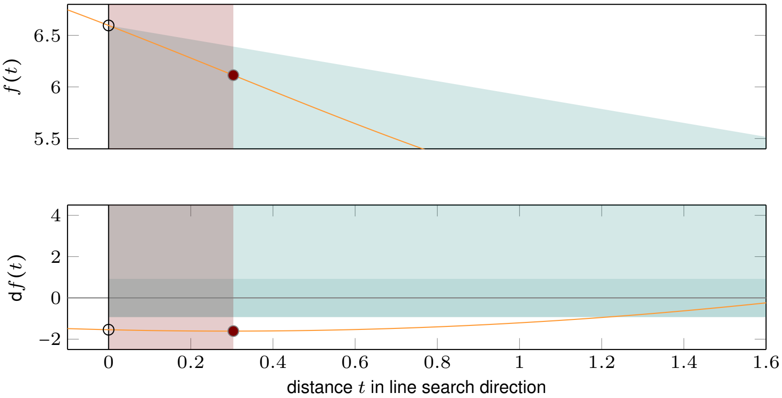
$$f(t) \leq f(0) + c_1 t f'(0) \quad \text{(W-I)}$$

$$f'(t) \geq c_2 f'(0) \quad \text{(W-IIa)}$$

$$|f'(t)| \leq c_2 |f'(0)| \quad \text{(W-IIb)}$$

Classic line searches

Collapse search space $\quad x^* = \arg\min_x f(x), \qquad x_{i+1} \leftarrow x_i - t s_{i+1}$



Wolfe conditions: accept when [Wolfe, SIAM Review, 1969]

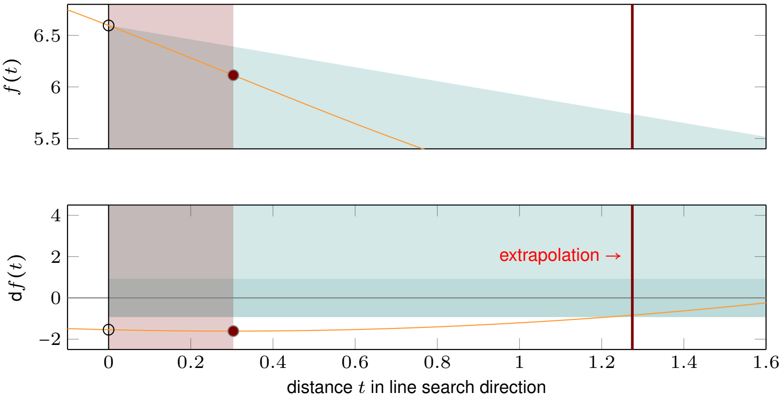
$$f(t) \leq f(0) + c_1 t f'(0) \quad \text{(W-I)}$$

$$f'(t) \geq c_2 f'(0) \quad \text{(W-IIa)}$$

$$|f'(t)| \leq c_2 |f'(0)| \quad \text{(W-IIb)}$$

Classic line searches

Search: candidate # 2 $x^* = \arg\min_x f(x), \qquad x_{i+1} \leftarrow x_i - t s_{i+1}$



Wolfe conditions: accept when [Wolfe, SIAM Review, 1969]

$$f(t) \leq f(0) + c_1 t f'(0) \quad \text{(W-I)}$$

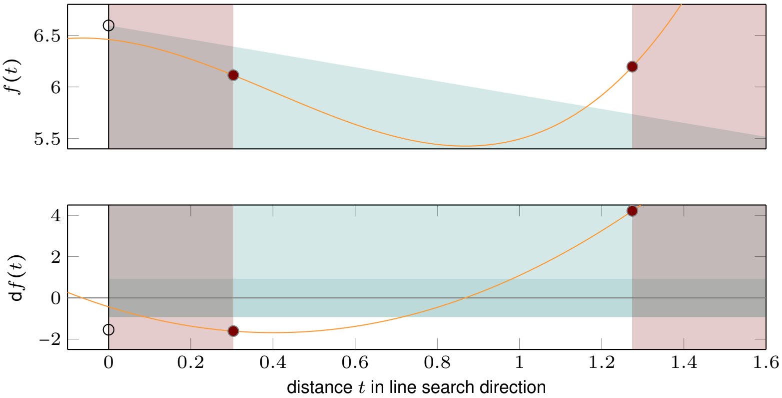
$$f'(t) \geq c_2 f'(0) \quad \text{(W-IIa)}$$

$$|f'(t)| \leq c_2 |f'(0)| \quad \text{(W-IIb)}$$

Classic line searches

Collapse search space

$x^* = \arg\min_x f(x), \qquad x_{i+1} \leftarrow x_i - t s_{i+1}$



Wolfe conditions: accept when [Wolfe, SIAM Review, 1969]

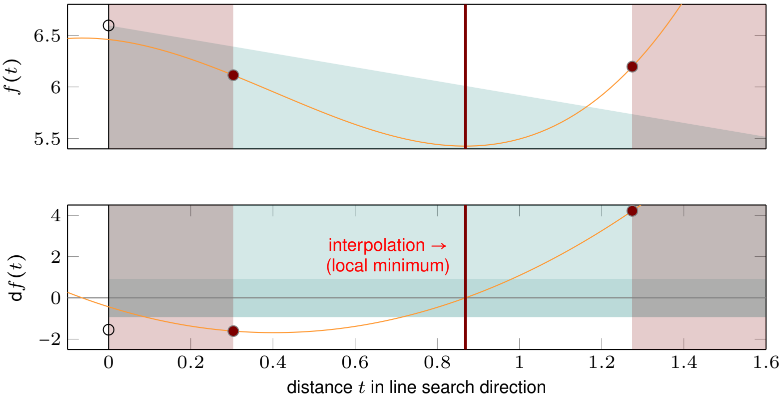
$f(t) \leq f(0) + c_1 t f'(0)$ (W-I)

$f'(t) \geq c_2 f'(0)$ (W-IIa)

$|f'(t)| \leq c_2 |f'(0)|$ (W-IIb)

Classic line searches

Search: candidate # 3 $x^* = \arg\min_x f(x), \qquad x_{i+1} \leftarrow x_i - t s_{i+1}$



Wolfe conditions: accept when [Wolfe, SIAM Review, 1969]

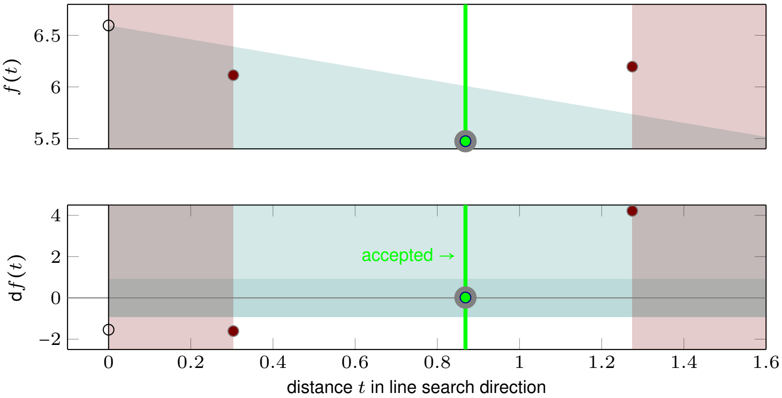
$$f(t) \leq f(0) + c_1 t f'(0) \quad \text{(W-I)}$$

$$f'(t) \geq c_2 f'(0) \quad \text{(W-IIa)}$$

$$|f'(t)| \leq c_2 |f'(0)| \quad \text{(W-IIb)}$$

Classic line searches

Accept: datapoint # 3 fulfills Wolfe conditions $\quad x^* = \arg\min_x f(x), \qquad x_{i+1} \leftarrow x_i - t s_{i+1}$



Wolfe conditions: accept when [Wolfe, SIAM Review, 1969]

$$f(t) \leq f(0) + c_1 t f'(0) \quad \text{(W-I)}$$

$$f'(t) \geq c_2 f'(0) \quad \text{(W-IIa)}$$

$$|f'(t)| \leq c_2 |f'(0)| \quad \text{(W-IIb)}$$

Classic line searches

Choosing meaningful step-sizes, at very low overhead

many classic line searches

1. model the 1D objective with cubic spline
2. search candidate points by collapsing search space
3. accept if Wolfe conditions fulfilled

Classic line searches

Fail in the presence of noise.

many classic line searches

1. model the 1D objective with cubic spline
2. search candidate points by collapsing search space
3. accept if Wolfe conditions fulfilled

Classic line searches break in stochastic optimization problems!

many classic line searches

1. model the 1D objective with cubic spline
2. search candidate points by collapsing search space
3. accept if Wolfe conditions fulfilled

Classic line searches break in stochastic optimization problems!

extending the line search paradigm:

1. model: cubic spline GP surrogate
2. search: Bayesian optimization for exploration
3. accept: probabilistic Wolfe termination conditions

Building a probabilistic line search

1. model: cubic spline GP (integrated Wiener process)

$$p(f) = \mathcal{GP}(f, 0; k), \quad k(t,t') = \left[\tfrac{1}{3}\min^3(t,t') + \tfrac{1}{2}|t - t'|\min^2(t,t')\right]$$

- robust and flexible
- has analytic minima (root of quadratic equation)

Building a probabilistic line search

Step 1: cubic spline GP surrogate, Step 2: BO for exploration

1. model: cubic spline GP (integrated Wiener process)

$$p(f) = \mathcal{GP}(f, 0; k), \quad k(t, t') = \left[\tfrac{1}{3}\min^3(t, t') + \tfrac{1}{2}|t - t'|\min^2(t, t')\right]$$

    - robust and flexible
    - has analytic minima (root of quadratic equation)

2. search: Bayesian optimization (expected improvement)

$$u_{\mathrm{EI}}(t) = \mathrm{E}_{p(f_t \mid \boldsymbol{y}, \boldsymbol{y}')}[\min\{0, \eta - f(t)\}] \qquad \text{[Jones et al., 1998]}$$

    - only evaluated at few candidate points:
        - analytic minima of posterior mean
        - one extrapolation point

Building a probabilistic line search

Step 3: probabilistic Wolfe termination conditions

3. accept: probabilistic Wolfe termination conditions:
   - Wolfe conditions are positivity constraints on two variables a_t, b_t

$$f(t) \leq f(0) + c_1 t f'(0) \quad \text{(W-I)} \qquad \text{and} \qquad f'(t) \geq c_2 f'(0) \quad \text{(W-II)}$$

$$\begin{bmatrix} a_t \\ b_t \end{bmatrix} = \begin{bmatrix} 1 & c_1 t & -1 & 0 \\ 0 & -c_2 & 0 & 1 \end{bmatrix} \begin{bmatrix} f(0) \\ f'(0) \\ f(t) \\ f'(t) \end{bmatrix} \geq 0.$$

Building a probabilistic line search

Step 3: probabilistic Wolfe termination conditions

3. accept: probabilistic Wolfe termination conditions:
   - Wolfe conditions are positivity constraints on two variables a_t, b_t

$$f(t) \leq f(0) + c_1 t f'(0) \quad \text{(W-I)} \qquad \text{and} \qquad f'(t) \geq c_2 f'(0) \quad \text{(W-II)}$$

$$\begin{bmatrix} a_t \\ b_t \end{bmatrix} = \begin{bmatrix} 1 & c_1 t & -1 & 0 \\ 0 & -c_2 & 0 & 1 \end{bmatrix} \begin{bmatrix} f(0) \\ f'(0) \\ f(t) \\ f'(t) \end{bmatrix} \geq 0.$$

3. accept: probabilistic Wolfe termination conditions:
   - Wolfe conditions are positivity constraints on two variables a_t, b_t

$$f(t) \leq f(0) + c_1 t f'(0) \quad \text{(W-I)} \qquad \text{and} \qquad f'(t) \geq c_2 f'(0) \quad \text{(W-II)}$$

$$\begin{bmatrix} a_t \\ b_t \end{bmatrix} = \begin{bmatrix} 1 & c_1 t & -1 & 0 \\ 0 & -c_2 & 0 & 1 \end{bmatrix} \begin{bmatrix} f(0) \\ f'(0) \\ f(t) \\ f'(t) \end{bmatrix} \geq 0.$$

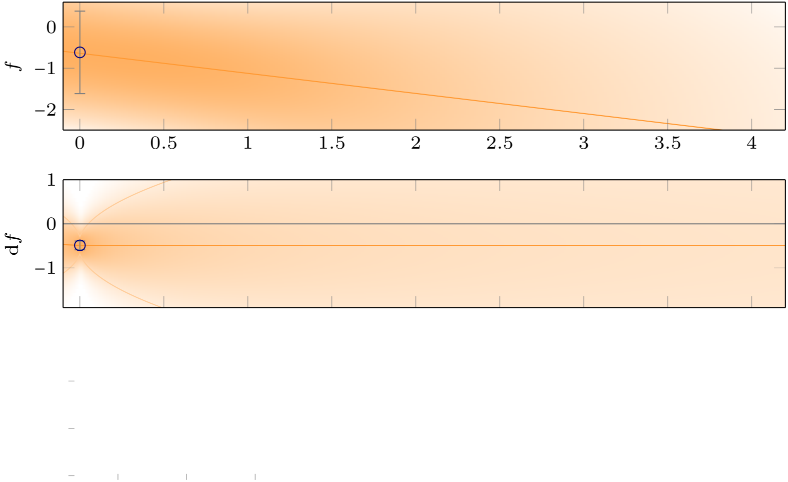
   - GP on f implies, at each t, a bivariate Gaussian distribution:

$$p(a_t, b_t) = \mathcal{N}\left(\begin{bmatrix} a_t \\ b_t \end{bmatrix}; \begin{bmatrix} m_t^a \\ m_t^b \end{bmatrix}, \begin{bmatrix} C_t^{aa} & C_t^{ab} \\ C_t^{ba} & C_t^{bb} \end{bmatrix} \right)$$

probability for weak Wolfe conditions : $p_t^{\text{Wolfe}} = p(0 \leq a_t \wedge 0 \leq b_t)$
approximate strong conditions : $p_t^{\text{Wolfe}} = p(0 \leq a_t \wedge 0 \leq b_t \leq \bar{b})$

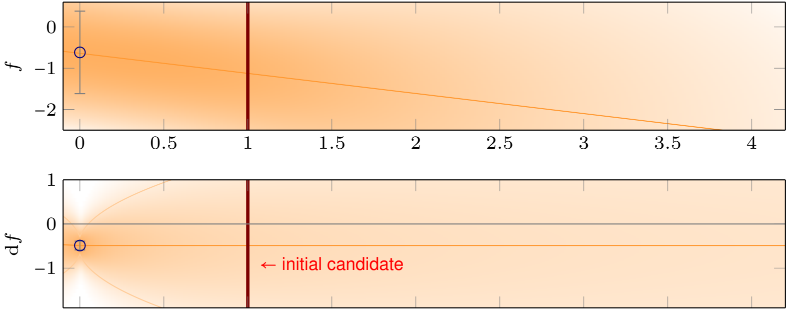
Probabilistic line search routine

Initial belief: first evaluation ≡ current position of optimizer



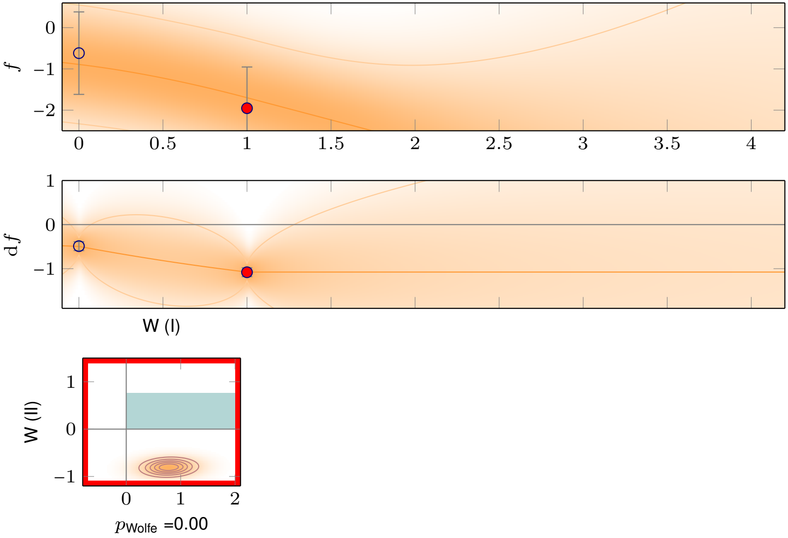
Probabilistic line search routine

Search: candidate # 1



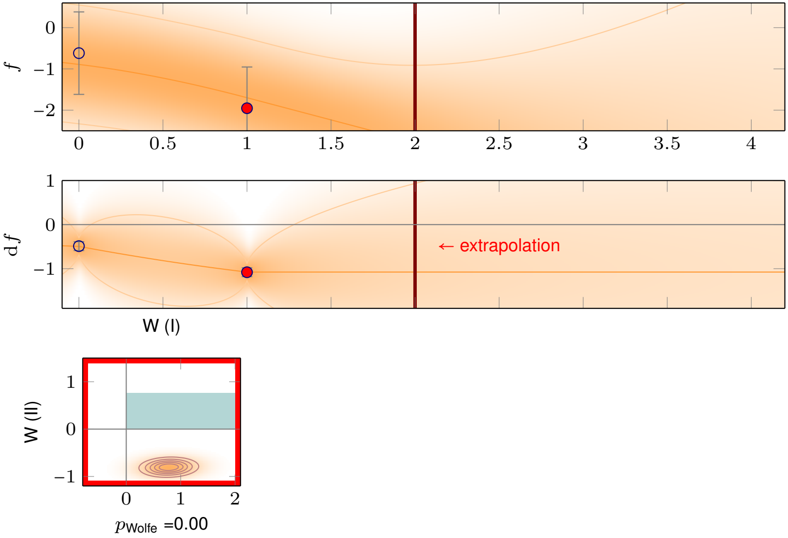
Probabilistic line search routine

Accept: Check p_{Wolfe} for first datapoint



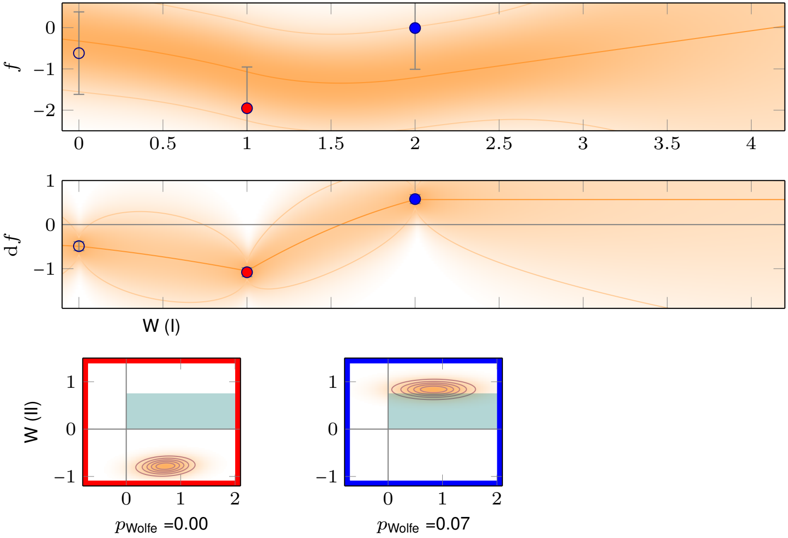
Probabilistic line search routine

Search: candidate # 2



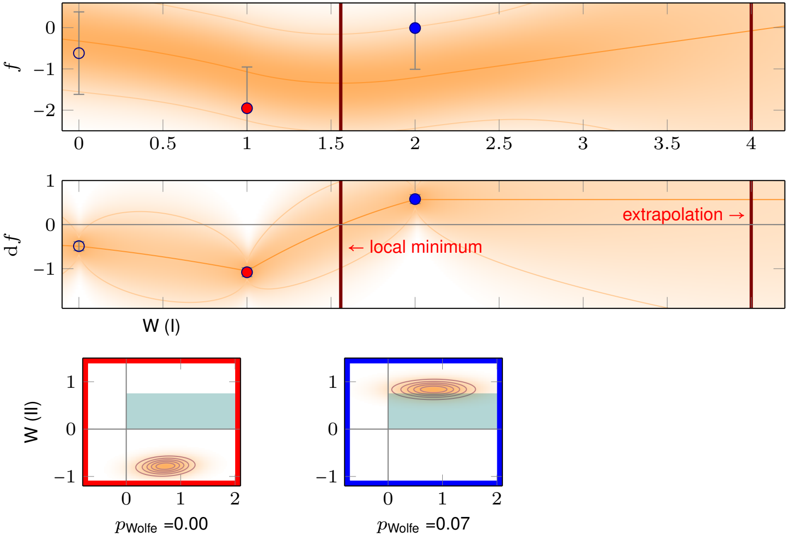
Probabilistic line search routine

Accept: check p_{Wolfe} for datapoints # 1 and # 2



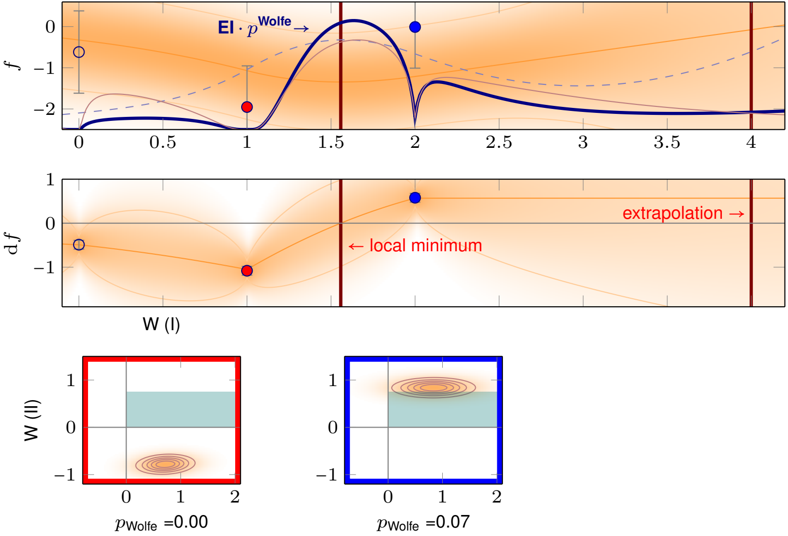
Probabilistic line search routine

Search: candidates # 3



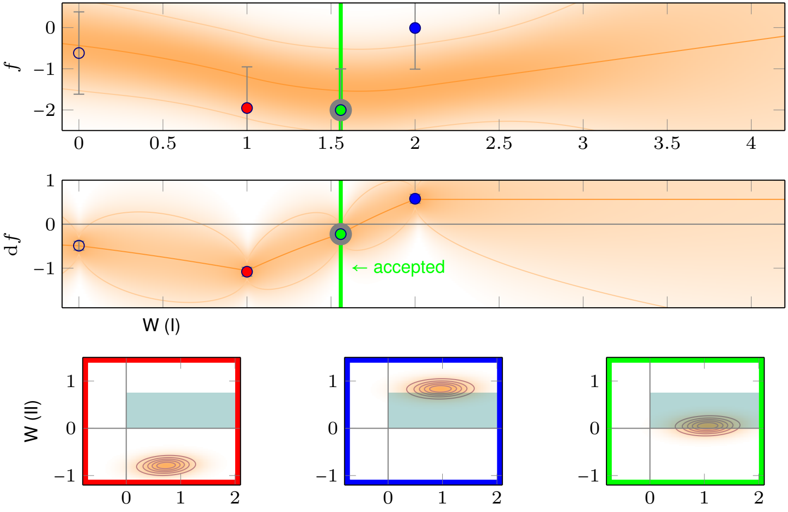
Probabilistic line search routine

Search: candidates # 3 (discriminate through EI)



Probabilistic line search routine

Accept: check p_{Wolfe} for datapoints # 1, # 2 and # 3



p_{Wolfe} =0.00 p_{Wolfe} =0.08 p_{Wolfe} =0.68

small summary

...probabilistic line searches

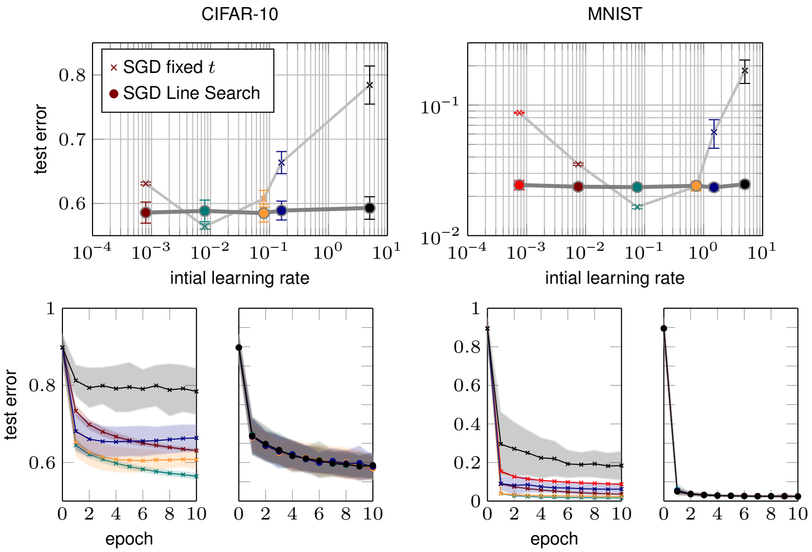
make new from old:

1. model cubic spline → GP with cubic spline means
2. search collapsing search space → Bayesian optimization
3. accept binary Wolfe conditions → probabilistic Wolfe conditions

→ lightweight inner optimization routine
→ robust stochastic optimization

Line search finds learning rates

SGD on 2-layer neural net: mini-batch size: 10



small summary

... about line searches and others

take away

- optimizer are learning machines
- *data*: noisy gradient
- *prior* encodes structure of the objective
- prob. line search: *infers* approximate minimum

small summary

... about line searches and others

take away

- optimizer are learning machines
- *data*: noisy gradient
- *prior* encodes structure of the objective
- prob. line search: *infers* approximate minimum

there is more

- the field is much broader than 'only' line searches
- search directions can also be learned
- classic search directions are MAP estimator of Gaussian inference
- robust second order search directions are still needed!

Probabilistic line searches

... in Tensorflow

We implement in:

TensorFlow

Have a beer with Lukas!

We implement in:

TensorFlow

Have a beer with Lukas!

Thank you!