AutoML: Dynamic Configuration & Learning

Learning to Learn: Supervised

Bernd Bischl   Frank Hutter   Lars Kotthoff
Marius Lindauer   Joaquin Vanschoren

Learning to Learn

Idea

- Learn algorithms directly, e.g., how to search in the weight space
- First idea: learn weight updates of a neural network

Learning to Learn

Idea

- Learn algorithms directly, e.g., how to search in the weight space
- First idea: learn weight updates of a neural network

Learning to learn by gradient descent by gradient descent

[Andrychowicz et al'16]

Weight updates (note: θ denote DNN weights):

$$\theta^{(t+1)} = \theta^{(t)} - \alpha^{(t)} \nabla f(\theta^{(t)})$$

Learning to Learn

Idea

- Learn algorithms directly, e.g., how to search in the weight space
- First idea: learn weight updates of a neural network

Learning to learn by gradient descent by gradient descent

[Andrychowicz et al'16]

Weight updates (note: θ denote DNN weights):

$$\theta^{(t+1)} = \theta^{(t)} - \alpha^{(t)} \nabla f(\theta^{(t)})$$

Even more general:

$$\theta^{(t+1)} = \theta^{(t)} + g^{(t)}(\nabla f(\theta^{(t)}), \phi)$$

where g is the optimizer and ϕ are the parameters of the optimizer g.

Learning to Learn

Idea

- Learn algorithms directly, e.g., how to search in the weight space
- First idea: learn weight updates of a neural network

Learning to learn by gradient descent by gradient descent

[Andrychowicz et al'16]

Weight updates (note: θ denote DNN weights):

$$\theta^{(t+1)} = \theta^{(t)} - \alpha^{(t)} \nabla f(\theta^{(t)})$$

Even more general:

$$\theta^{(t+1)} = \theta^{(t)} + g^{(t)}(\nabla f(\theta^{(t)}), \phi)$$

where g is the optimizer and ϕ are the parameters of the optimizer g.

$\rightsquigarrow$ Goal: Optimize f wrt θ by learning g (resp. ϕ)

Learning to Learn: Objective [Andrychowicz et al'16]

$$L(\phi) = \mathbb{E}\left[f(\theta^*(f, \phi))\right]$$

where L is a loss function and $\theta^*(f, \phi)$ are the optimized weights θ^* by using the optimizer parameterized with ϕ on function f.

Learning to Learn: Objective [Andrychowicz et al'16]

$$L(\phi) = \mathbb{E}\left[f(\theta^*(f,\phi))\right]$$

where L is a loss function and $\theta^*(f,\phi)$ are the optimized weights θ^* by using the optimizer parameterized with ϕ on function f.

$$L(\phi) = \mathbb{E}\left[\sum_{t=1}^{T} w^{(t)} f(\theta^{(t)})\right]$$

Learning to Learn: Objective [Andrychowicz et al'16]

$$L(\phi) = \mathbb{E}\left[f(\theta^*(f, \phi))\right]$$

where L is a loss function and $\theta^*(f, \phi)$ are the optimized weights θ^* by using the optimizer parameterized with ϕ on function f.

$$L(\phi) = \mathbb{E}\left[\sum_{t=1}^{T} w^{(t)} f(\theta^{(t)})\right]$$

where w_t are arbitrary weights associated with each time step and

Learning to Learn: Objective [Andrychowicz et al'16]

$$L(\phi) = \mathbb{E}\left[f(\theta^*(f, \phi))\right]$$

where L is a loss function and $\theta^*(f, \phi)$ are the optimized weights θ^* by using the optimizer parameterized with ϕ on function f.

$$L(\phi) = \mathbb{E}\left[\sum_{t=1}^{T} w^{(t)} f(\theta^{(t)})\right]$$

where w_t are arbitrary weights associated with each time step and

$$\theta^{(t+1)} = \theta^{(t)} + g^{(t)}$$

$$\begin{pmatrix} g^{(t)} \\ h^{(t+1)} \end{pmatrix} = m(\nabla_\theta f(\theta^{(t)}), h^{(t)}, \phi)$$

Learning to Learn: Objective [Andrychowicz et al'16]

$$L(\phi) = \mathbb{E}\left[f(\theta^*(f,\phi))\right]$$

where L is a loss function and $\theta^*(f,\phi)$ are the optimized weights θ^* by using the optimizer parameterized with ϕ on function f.

$$L(\phi) = \mathbb{E}\left[\sum_{t=1}^{T} w^{(t)} f(\theta^{(t)})\right]$$

where w_t are arbitrary weights associated with each time step and

$$\theta^{(t+1)} = \theta^{(t)} + g^{(t)}$$

$$\begin{pmatrix} g^{(t)} \\ h^{(t+1)} \end{pmatrix} = m(\nabla_\theta f(\theta^{(t)}), h^{(t)}, \phi)$$

$\rightsquigarrow$ Goal: Learn m via ϕ by using gradient descent by optimizing L

Learning to Learn: Objective [Andrychowicz et al'16]

$$L(\phi) = \mathbb{E}\left[f(\theta^*(f, \phi))\right]$$

where L is a loss function and $\theta^*(f, \phi)$ are the optimized weights θ^* by using the optimizer parameterized with ϕ on function f.

$$L(\phi) = \mathbb{E}\left[\sum_{t=1}^{T} w^{(t)} f(\theta^{(t)})\right]$$

where w_t are arbitrary weights associated with each time step and

$$\theta^{(t+1)} = \theta^{(t)} + g^{(t)}$$

$$\begin{pmatrix} g^{(t)} \\ h^{(t+1)} \end{pmatrix} = m(\nabla_\theta f(\theta^{(t)}), h^{(t)}, \phi)$$

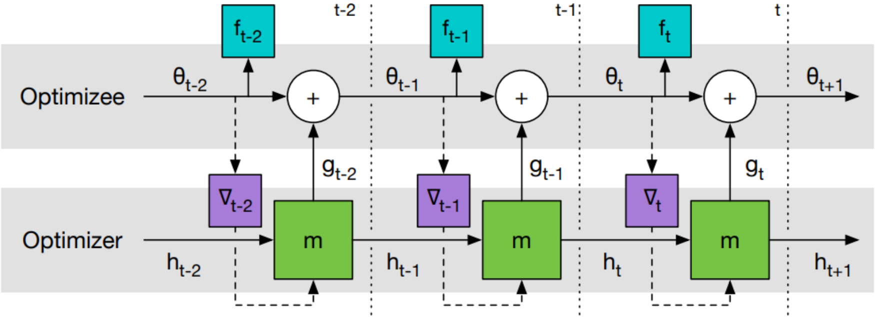
$\rightsquigarrow$ Goal: Learn m via ϕ by using gradient descent by optimizing L

$\rightsquigarrow$ "Learning to learn gradient descent by gradient descent"

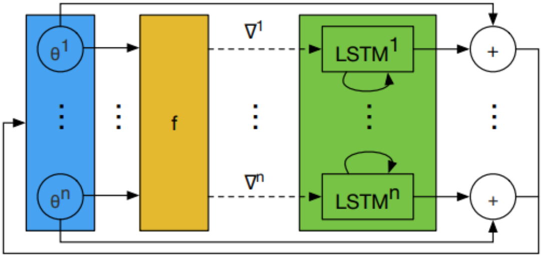
Learning to Learn: LSTM approach [Andrychowicz et al'16]

Optimizee Target network to be trained

Optimizer LSTM with hidden state h_t that predicts weight updates g_t

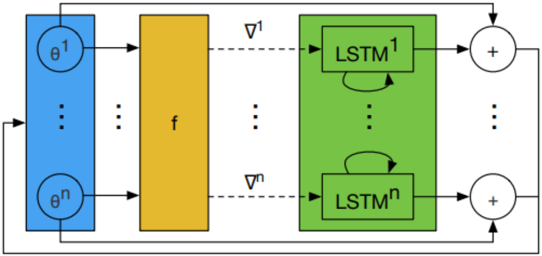


Learning to Learn: Coordinatewise LSTM optimizer [Andrychowicz et al'16]



- One LSTM for each coordinate (i.e., weight)
- All LSTMs have shared parameters ϕ
- Each coordinate has its own separate hidden state

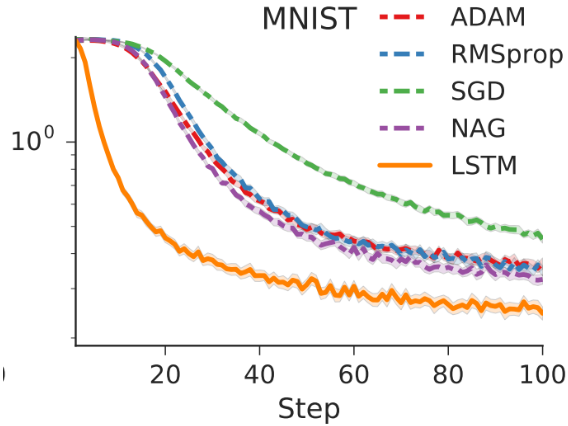
Learning to Learn: Coordinatewise LSTM optimizer [Andrychowicz et al'16]



- One LSTM for each coordinate (i.e., weight)
- All LSTMs have shared parameters ϕ
- Each coordinate has its own separate hidden state

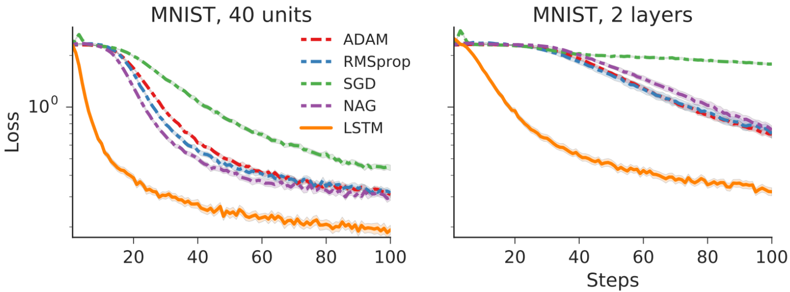
$\rightsquigarrow$ We can train the LSTM on k weights and apply it larger DNNs with k' weights, where $k \leq k'$

Learning to Learn with LSTM: Results [Andrychowicz et al'16]



Learning to Learn with LSTM: Results [Andrychowicz et al'16]

Changing the original architecture of the DNN:

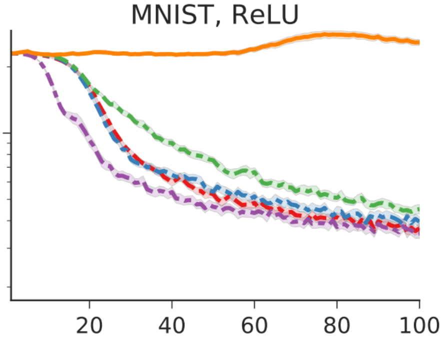


$\rightsquigarrow$ learnt optimizer is robust against some architectural changes

Learning to Learn with LSTM: Results

[Andrychowicz et al'16]

Changing the activation function to ReLU:



$\rightsquigarrow$ fails on other activation functions

Learning Black-box Optimization [Chen et al'17]

Black Box Optimization Setting

$$\mathbf{x}^* \in \arg\min_{\mathbf{x}\in\mathcal{X}} f(\mathbf{x})$$

1. Given the current state of knowledge $h^{(t)}$ propose a query point $\mathbf{x}^{(t)}$
2. Observe the response $y^{(t)}$
3. Update any internal statistics to produce $h^{(t+1)}$

Learning Black-box Optimization [Chen et al'17]

Learning Black Box Optimization

Essentially, a similar idea as before:

$$\begin{aligned} h^{(t)}, \mathbf{x}^{(t)} &= \mathsf{RNN}_\phi(h^{(t-1)}, \mathbf{x}^{(t-1)}, y^{(t)}) \\ y^{(t)} &\sim p(y|\mathbf{x}^{(t)}) \end{aligned}$$

- Using recurrent neural network (RNN) to predict next x_t.
- $h^{(t)}$ is the internal hidden state

Learning Black-box Optimization: Loss Functions

[Chen et al'17]

- Sum loss: Provides more information than final loss

$$L_{\mathsf{sum}}(\phi) = \mathbb{E}_{f,y^{(1:T-1)}} \left[\sum_{t=1}^{T} f(\mathbf{x}^{(t)}) \right]$$

Learning Black-box Optimization: Loss Functions

[Chen et al'17]

- Sum loss: Provides more information than final loss

$$L_{\mathsf{sum}}(\phi) = \mathbb{E}_{f,y^{(1:T-1)}}\left[\sum_{t=1}^{T} f(\mathbf{x}^{(t)})\right]$$

- EI loss: Try to learn behavior of Bayesian optimizer based on expected improvement (EI)
  - requires model (e.g., GP)

$$L_{\mathsf{EI}}(\phi) = -\mathbb{E}_{f,y^{(1:T-1)}}\left[\sum_{t=1}^{T} \mathsf{EI}(\mathbf{x}^{(t)}|y^{(1:t-1)})\right]$$

Learning Black-box Optimization: Loss Functions

[Chen et al'17]

- Sum loss: Provides more information than final loss

$$L_{\mathsf{sum}}(\phi) = \mathbb{E}_{f,y^{(1:T-1)}}\left[\sum_{t=1}^{T} f(\mathbf{x}^{(t)})\right]$$

- EI loss: Try to learn behavior of Bayesian optimizer based on expected improvement (EI)
  - requires model (e.g., GP)

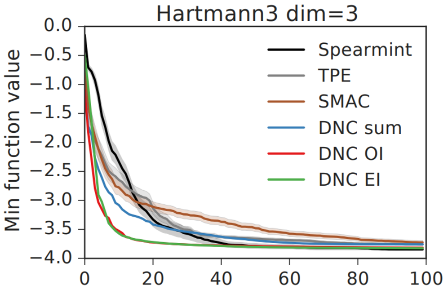
$$L_{\mathsf{EI}}(\phi) = -\mathbb{E}_{f,y^{(1:T-1)}}\left[\sum_{t=1}^{T} \mathsf{EI}(\mathbf{x}^{(t)}|y^{(1:t-1)})\right]$$

- Observed Improvement Loss:

$$L_{\mathsf{OI}}(\phi) = \mathbb{E}_{f,y^{(1:T-1)}}\left[\sum_{t=1}^{T} \min\left\{f(\mathbf{x}^{(t)}) - \min_{i<t}(f(\mathbf{x}^{(i)})), 0\right\}\right]$$

Learning Black-box Optimization: Results

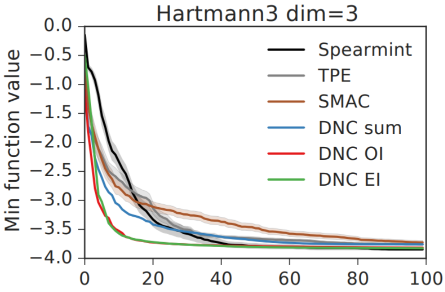
[Chen et al'17]



- Hartmann3 is an artificial function with 3 dimensions

Learning Black-box Optimization: Results

[Chen et al'17]



- Hartmann3 is an artificial function with 3 dimensions

⇝ L_{OI} and L_{EI} perform best

⇝ L_{OI} easier to compute than L_{EI}
because we need a predictive model to compute EI