Learning to Optimize using Reinforcement Learning

Viktor Yanush

Moscow State University

November 9, 2017

Motivation

Why learn to learn?

Motivation

Why learn to learn?

Drawbacks of existing algorithms:

Motivation

Why learn to learn?

Drawbacks of existing algorithms:

- Need to choose optimization algorithm (e.g. Adam, RMSProp etc.)

Motivation

Why learn to learn?

Drawbacks of existing algorithms:

- Need to choose optimization algorithm (e.g. Adam, RMSProp etc.)
- Need to tune hyperparameters (especially learning rate)

Motivation

Why learn to learn?

Drawbacks of existing algorithms:

- Need to choose optimization algorithm (e.g. Adam, RMSProp etc.)
- Need to tune hyperparameters (especially learning rate)

It's time-consuming!

Motivation

What we can get if we learn optimization algorithm:

Motivation

What we can get if we learn optimization algorithm:

- No parameters!

Motivation

What we can get if we learn optimization algorithm:

- No parameters!
- Algorithm is fit for particular class of optimization problems

Optimization algorithm structure

General optimizer structure:

$$\Delta x_i = \phi(f, \{x^{(0)}, \dots, x^{(i-1)}\}, \theta)$$
$$x^{(i)} = x^{(i-1)} + \Delta x_i$$

To learn an optimizer = to learn good parameters θ

Problem setup

- $\mathcal{F}$ — distribution over functions
- $f_1, f_2 \dots f_n \sim \mathcal{F}$ — training set
- $\mathcal{D}$ — distribution over initial states
- $\mathcal{L}$ — meta-loss
- θ — parameters of optimization algorithm
- T — number of iterations

Objective:

$$\mathbb{E}_{f \sim \mathcal{F}, x^{(0)} \sim \mathcal{D}} \left[\mathcal{L}(f, x^{(1)}(\theta, x^{(0)}) \dots x^{(T)}(\theta, x^{(0)})) \right] \to \min_{\theta}$$

Problem setup

- $\mathcal{F}$ — distribution over functions
- $f_1, f_2 \dots f_n \sim \mathcal{F}$ — training set
- $\mathcal{D}$ — distribution over initial states
- $\mathcal{L}$ — meta-loss
- θ — parameters of optimization algorithm
- T — number of iterations

Objective:

$$\mathbb{E}_{f \sim \mathcal{F}, x^{(0)} \sim \mathcal{D}} \left[\mathcal{L}(f, x^{(1)}(\theta, x^{(0)}) \dots x^{(T)}(\theta, x^{(0)})) \right] \to \min_{\theta}$$

Wait, and why do we need RL?

Why RL?

Why not use supervised approach?

Why RL?

Why not use supervised approach?

Key ideas:

- step which optimizer takes affects the future.

Why RL?

Why not use supervised approach?

Key ideas:

- step which optimizer takes affects the future.
- consequence: examples are **not** i.i.d.

Why RL?

Why not use supervised approach?

Key ideas:

- step which optimizer takes affects the future.
- consequence: examples are **not** i.i.d.
- we get compounding errors.

Why RL?

Why not use supervised approach?

Key ideas:

- step which optimizer takes affects the future.
- consequence: examples are **not** i.i.d.
- we get compounding errors.
- as a result supervised learning does not generalize successfully.

Why RL?

Why not use supervised approach?

Key ideas:

- step which optimizer takes affects the future.
- consequence: examples are **not** i.i.d.
- we get compounding errors.
- as a result supervised learning does not generalize successfully.

On the other hand, RL seems reasonable to use here.

RL problem setup

POMDP: $(\mathcal{S}, \mathcal{O}, \mathcal{A}, p_i, p, p_o, c, T)$

- $\mathcal{S} \subseteq \mathbb{R}^D, \mathcal{O} \subseteq \mathbb{R}^{D'}, \mathcal{A} \subseteq \mathbb{R}^d$
- $p_i(s_0)$ — probability density over initial states
- $p(s_{t+1} \mid s_t, a_t)$ — dynamics of environment
- $p_o(o_t \mid s_t)$ — probability density over observations given state
- $c : \mathcal{S} \to \mathbb{R}$ — cost function
- T — time horizon length

RL problem setup

In optimization case:

- $s_t = (x^{(t)}, \Phi(x^{(1:t)}, \nabla f(x^{(1:t)}), f(x^{(1:t)}))$
- $o_t = \Psi(x^{(1:t)}, \nabla f(x^{(1:t)}), f(x^{(1:t)})$
- $a_t = \Delta x$
- $c(s_t) = f(x^{(t)})$

We look for policy π^* such that:

$$\pi^* = \underset{\pi}{\operatorname{argmin}} \mathbb{E}_{s_0, a_0, \ldots, s_T} \left[\sum_{t=0}^{T} c(s_t) \right]$$

Guided Policy Search[1]

Idea of Guided Policy Search (GPS):

- Reinforcement learning is hard
- Supervised learning is easier
- Let's convert RL to SL!

[1] End-to-End Training of Deep Visuomotor Policies

Guided Policy Search

- GPS was originally invented to train robots with RL
- Samples are way more expensive than in simulator
- Need sample-efficient RL algorithm

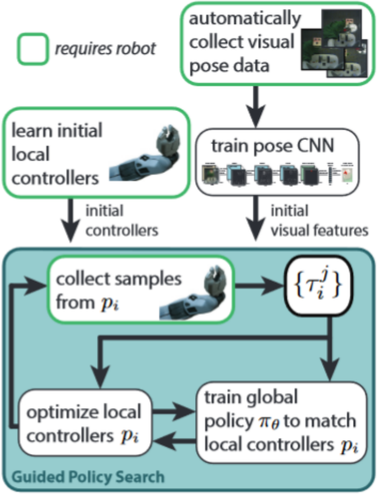
Guided Policy Search

- Fitting complex policy directly (e.g. with model-free RL) is hard
- To use supervised learning — samples should come from policy's own state distribution
- Therefore, guiding distribution should be easy to find but give samples close to $\pi_\theta(a_t \mid o_t)$

Guided Policy Search

Two components:

- Reinforcement learning algorithm (to generate guiding distribution)
- Supervised learning algorithm (to fit global policy)



GPS Derivation

In RL we want to solve:

$$\mathbb{E}_{\pi_\theta}[c(\tau)] \to \min_{\pi}, \ \tau = \{s_1, a_1, ...s_T, a_T\},$$
$$c(\tau) = \sum_{t=1}^{T} c(s_t, a_t)$$

We can rewrite it as follows:

$$\mathbb{E}_p\left[c(\tau)\right] \to \min_{p,\pi_\theta}$$
$$\text{s.t.} p(a_t \mid s_t) = \pi_\theta(a_t \mid s_t), \forall s_t, a_t, t$$

GPS Derivation

In RL we want to solve:

$$\mathbb{E}_{\pi_\theta}[c(\tau)] \to \min_{\pi}, \ \tau = \{s_1, a_1, ...s_T, a_T\},$$
$$c(\tau) = \sum_{t=1}^{T} c(s_t, a_t)$$

We can rewrite it as follows:

$$\mathbb{E}_p\left[c(\tau)\right] \to \min_{p,\pi_\theta}$$
$$\text{s.t.} p(a_t \mid s_t) = \pi_\theta(a_t \mid s_t), \forall s_t, a_t, t$$

This optimization problem is equivalent to the original.

GPS Derivation

In RL we want to solve:

$$\mathbb{E}_{\pi_\theta}[c(\tau)] \to \min_{\pi}, \ \tau = \{s_1, a_1, ...s_T, a_T\},$$
$$c(\tau) = \sum_{t=1}^{T} c(s_t, a_t)$$

We can rewrite it as follows:

$$\mathbb{E}_p\left[c(\tau)\right] \to \min_{p,\pi_\theta}$$
$$\text{s.t.} p(a_t \mid s_t) = \pi_\theta(a_t \mid s_t), \forall s_t, a_t, t$$

This optimization problem is equivalent to the original.
But the number of constraints is infinite!

GPS Derivation

Tractable version:

$$\mathbb{E}_p\left[c(\tau)\right] \to \min_{p,\pi_\theta}$$
$$\text{s.t.}\mathbb{E}_{p(a_t|s_t)p(s_t)}[a_t] = \mathbb{E}_{\pi_\theta(a_t|s_t)p(s_t)}[a_t], \forall t$$

To solve this optimization problem we can use Bregman ADMM.

Bregman ADMM

Problem:

$$\min_{x \in \mathcal{X}, z \in \mathcal{Z}} f(x) + g(z), \text{s.t. } Ax + Bz = c$$

Bregman divergence induced by convex function ϕ:

$$B_\phi(x, y) = \phi(x) - \phi(y) - \langle \nabla \phi(y), x - y \rangle$$

Algorithm:

$$x_{t+1} = \operatorname*{argmin}_{x \in \mathcal{X}} f(x) + \langle y_t, Ax + Bz_t - c \rangle + \rho B_\phi(c - Ax, Bz_t)$$

$$z_{t+1} = \operatorname*{argmin}_{z \in \mathcal{Z}} g(z) + \langle y_t, Ax_{t+1} + Bz - c \rangle + \rho B_\phi(Bz, c - Ax_{t+1})$$

$$y_{t+1} = y_t + \rho(Ax_{t+1} + Bz_{t+1} - c)$$

GPS Derivation

Denote

$$\phi_t^\theta(\theta, p) = \mathbb{E}_{p(s_t)}[KL(p(a_t \mid s_t)||\pi_\theta(a_t \mid s_t))]$$
$$\phi_t^p(p, \theta) = \mathbb{E}_{p(s_t)}[KL(\pi_\theta(a_t \mid s_t)||p(a_t \mid s_t))]$$

BADMM iteration:

$$\theta \leftarrow \underset{\theta}{\operatorname{argmin}} \sum_{t=1}^{T} \mathbb{E}_{p(s_t)\pi_\theta(a_t|s_t)}[a_t^T \lambda_{\mu t}] + \nu_t \phi_t^\theta(\theta, p)$$
$$p \leftarrow \underset{p}{\operatorname{argmin}} \sum_{t=1}^{T} \mathbb{E}_{p(s_t, a_t)}[c(s_t, a_t) - a_t^T \lambda_{\mu t}] + \nu_t \phi_t^p(p, \theta)$$
$$\lambda_{\mu t} \leftarrow \lambda_{\mu t} + \alpha \nu_t \left(\mathbb{E}_{\pi_\theta(a_t|s_t)p(s_t)}[a_t] - \mathbb{E}_{p(a_t|s_t)p(s_t)}[a_t]\right)$$

Trajectory optimization

In GPS $p(\tau)$ is chosen to be Gaussian distribution $p_i(\tau)$

$p_i(\tau)$ — Gaussian $\rightarrow$ conditionals are Gaussian as well:

$$p_i(a_t \mid s_t) = \mathcal{N}(K_t s_t + k_t, C_t)$$
$$p_i(s_{t+1} \mid s_t, a_t) = \mathcal{N}(f_{st} s_t + f_{at} a_t + f_{ct}, F_t)$$

Such policy can be learned efficiently with few samples.

Trajectory optimization

- If dynamics $p(s_{t+1} \mid s_t, a_t)$ are known then $p(a_t \mid s_t)$ can be optimized with iLQG algorithm.
- If not we can fit $p(s_{t+1} \mid s_t, a_t)$ to sample trajectories from distribution $\hat{p}(\tau)$ from previous iteration.

However, optimization can diverge if $p(\tau)$ and $\hat{p}(\tau)$ are too different.

Trajectory optimization

- If dynamics $p(s_{t+1} \mid s_t, a_t)$ are known then $p(a_t \mid s_t)$ can be optimized with iLQG algorithm.
- If not we can fit $p(s_{t+1} \mid s_t, a_t)$ to sample trajectories from distribution $\hat{p}(\tau)$ from previous iteration.

However, optimization can diverge if $p(\tau)$ and $\hat{p}(\tau)$ are too different.

Solution:

$$\min_{p(\tau) \in \mathcal{N}(\tau)} \mathcal{L}_p(p, \theta),$$
$$\text{s.t. } KL(p(\tau) || \hat{p}(\tau)) \leq \varepsilon$$

This problem can be efficiently solved using dual gradient descent with iLQG for primal optimization.

iLQG

We can rewrite cost:

$$\hat{c}(s_t, a_t) = c(s_t, a_t) - a_t^T \lambda_{\mu t} - \nu_t \log \pi_\theta(a_t \mid s_t)$$

Lagrangian looks like:

$$\mathcal{L}(p) = \mathbb{E}_{p(\tau)}[\hat{c}(\tau) - \eta \log \hat{p}(\tau)] - (\eta + \nu_t)\mathcal{H}(p(\tau)) - \eta\varepsilon$$

We can rewrite our problem:

$$\mathbb{E}_{p(\tau)} \underbrace{\left[\frac{1}{\eta + \nu_t}\hat{c}(\tau) - \frac{\eta}{\eta + \nu_t} \log \hat{p}(\tau)\right]}_{\widetilde{c}(\tau)} - \mathcal{H}(p(\tau)) \to \min_{p(\tau)}$$

Final problem:

$$\mathbb{E}_{p(\tau)}[\widetilde{c}(\tau)] - \mathcal{H}(p(\tau)) \to \min_{p}$$

iLQG

Dynamics are estimated as linear-Gaussian:

$$p(s_{t+1} \mid s_t, a_t) = \mathcal{N}(f_{st} s_t + f_{at} a_t + f_{ct}, F_t)$$

We can write quadratic approximations to cost function:

$$\widetilde{c}(s_t, a_t) \approx \frac{1}{2}[s_t; a_t]^T \widetilde{c}_{sa,sa,t}[s_t; a_t] + [s_t; a_t]^T \widetilde{c}_{sa,t} + const$$

Optimal controller can be computed by recursive computation of quadratic Q-function and value function:

$$V(s_t) = \frac{1}{2} s_t^T V_{s,s,t} s_t + s_t^T V_{s,t} + const$$
$$Q(s_t, a_t) = \frac{1}{2}[s_t; a_t]^T Q_{sa,sa,t}[s_t; a_t] + [s_t; a_t]^T Q_{sa,t} + const$$

iLQG

Recursive computation starting from $t = T$:

$$
\begin{aligned}
Q_{sa,sa,t} &= \widetilde{c}_{sa,sa,t} + f_{sa,t}^T V_{s,s,t+1} f_{sa,t} \\
Q_{sa,t} &= \widetilde{c}_{sa,t} + f_{sa,t}^T V_{s,t+1} + f_{sa,t}^T V_{s,s,t+1} f_{ct} \\
V_{s,s,t} &= Q_{s,s,t} - Q_{a,s,t}^T Q_{a,a,t}^{-1} Q_{a,s,t} \\
V_{s,t} &= Q_{s,t} - Q_{a,s,t}^T Q_{a,a,t}^{-1} Q_{a,t}
\end{aligned}
$$

Optimal control is given by:

$$
\begin{aligned}
g(s_t) &= K_t s_t + k_t \\
K_t &= -Q_{a,a,t}^{-1} Q_{a,s,t} \\
k_t &= -Q_{a,a,t}^{-1} Q_{a,t}
\end{aligned}
$$

Maximum entropy policy is given by:

$$
p(a_t \mid s_t) = \mathcal{N}(K_t s_t + k_t, Q_{a,a,t}^{-1})
$$

Dynamics fitting

Linear Gaussian dynamics: $p_i(s_{t+1} \mid s_t, a_t) = \mathcal{N}(f_{st}s_t + f_{at}a_t + f_{ct}, F_t)$
Simple way to fit: just use linear regression on pairs
$(x', y') = ([s_t^i, a_t^i], s_{t+1}^i)$

Better way: we can fit global model to all the transitions $([s_t^i, a_t^i], s_{t+1}^i)$ and use it as a prior for linear regression.

Supervised optimization

With Gaussian policy $\pi(a_t \mid o_t) = \mathcal{N}(\mu^\pi(o_t), \Sigma^\pi(o_t))$ objective rewrites as:

$$\mathcal{L}_\theta(\theta, p) = \frac{1}{2N} \sum_{i=1}^{N} \sum_{i=1}^{T} \mathbb{E}_{p_i(s_t, o_t)}[\mathrm{tr}[C_{ti}^{-1} \Sigma^\pi(o_t)] - \log |\Sigma^\pi(o_t)|$$
$$+ (\mu^\pi(o_t) - \mu_{ti}^p(s_t))^T C_{ti}^{-1} (\mu^\pi(o_t) - \mu_{ti}^p(s_t)) + 2\lambda_{\mu t}^T \mu^\pi(o_t)]$$

Guided Policy Search

Approach summary:

- get N rollouts from guiding distribution
- run alternating optimization; for T steps:
  - update policy
  - update guiding distribution
  - update dual variables
- repeat until convergence

Tips and tricks for GPS

- Dynamics model can be shared between elements of guiding distribution
- During supervised policy optimization we can use samples from previous iterations and account for them with importance sampling
- Neural network can be pretrained for example by predicting o_t from s_t
- Guiding distribution can be also pretrained to get basic level of competence at task

Learning to Optimize[1]

- First work on RL for optimization
- Paper only tells about non-stochastic optimization
- Learns both step direction and step size
- Works in fully-observable MDP

[1] Learning to Optimize

Implementation details

State contains various information from $H = 25$ previous steps:

- x^t
- $\frac{f(x^{t-i}) - f(x^t)}{f(x^t)}, i \in \{2, \ldots, H+1\}$
- $\nabla f(x^{t-i}), i \in \{2, \ldots, H+1\}$

Policy:

$$\pi(a_t \mid s_t) = \mathcal{N}(\mu_\theta(s_t), \Sigma)$$

Mean $\mu_\theta(s_t)$ is 2-layer neural net with 50 hidden units.

Implementation details

Training:

- Guiding distribution: mixture of 20 Gaussians
- Time horizon $T = 40$
- Samples from preceding iterations are discarded

Evaluation:

- Objective value on sample functions
- Mean margin of victory — difference between current and best

Experiments on logistic regression

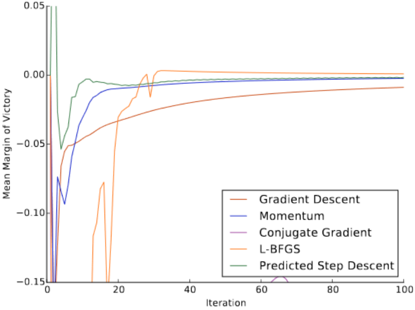


Figure: Mean margin of victory. Higher is better.

- Objective: Logistic regression with L2 regularization (convex)
- Data: Two random multivariate Gaussians correspond to classes
- Meta-train set: 90 random functions
- Meta-test set: 100 random functions

Results on logistic regression

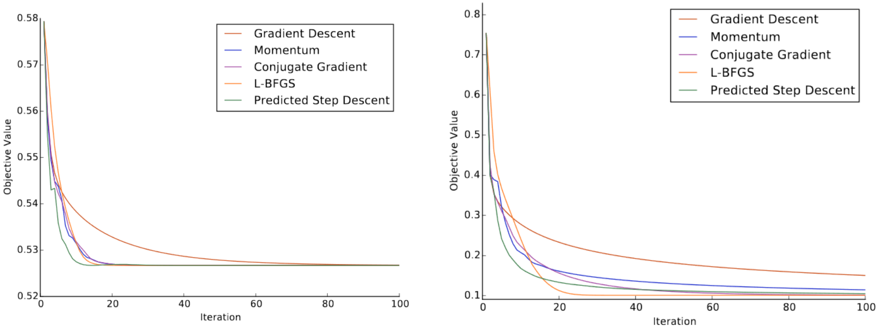


Figure: Logistic regression objective values on two test functions

Experiments on robust linear regression

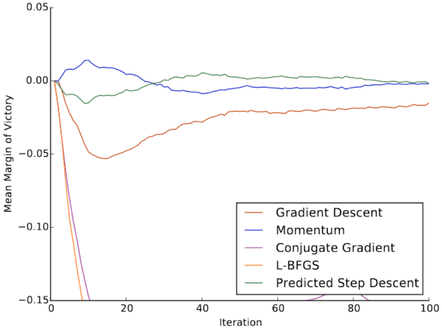
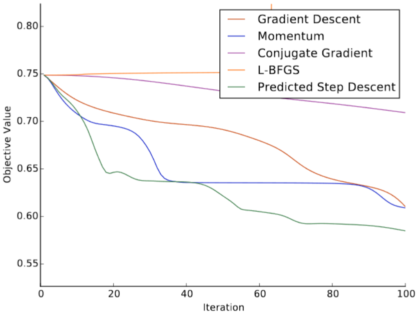


Figure: Mean margin of victory. Higher is better.

- Objective: Robust linear regression (not convex) $\min_{w,b} \frac{1}{n} \sum_{i=1}^{n} \frac{(y_i - w^T x_i - b)^2}{c^2 + (y_i - w^T x_i - b)^2}$
- Data: Four random multivariate Gaussians
- Meta-train set: 120 random functions
- Meta-test set: 100 random functions

Results on robust linear regression



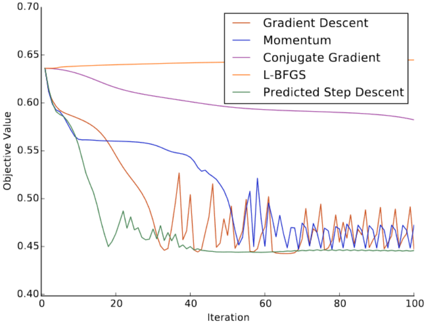


Figure: Robust linear regression objective values on two test functions

Experiments on neural net classifier

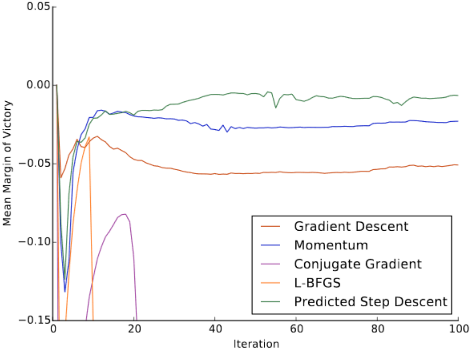


Figure: Mean margin of victory. Higher is better.

- Objective: Two-layer ReLU FC binary classifier
- Data: Four random multivariate Gaussians
- Meta-train set: 80 random functions
- Meta-test set: 100 random functions

Results on neural net classifier

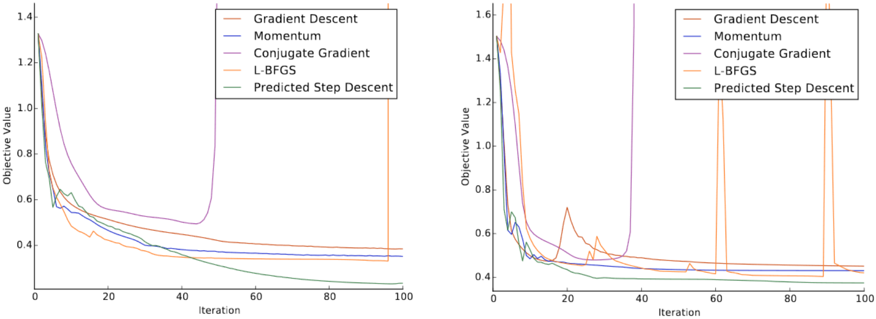


Figure: Neural net classifier objective values on two test functions

Learning to Optimize Neural Networks[1]

- Successor to "Learning to Optimize"
- Works with stochastic optimization and neural networks in particular
- Block-coordinatewise optimization

[1] Learning to Optimize Neural Networks

Implementation details

Policy:

- $\pi_\theta(a_t \mid o_t) = \mathcal{N}(\mu_\theta(o_t), \Sigma_\theta(o_t))$
- $\mu_\theta(o_t)$ — 1-layer LSTM with 128 units
- $\Sigma_\theta(o_t) = \Sigma$ — learned as a parameter

We run GPS for each coordinate group (e.g. layer in NN) separately.
That imposes block-diagonal structure on all matrices in GPS.

State features $\Phi(\cdot)$:

- $\left\{ \frac{\overline{f(x^{(t-5i)})} - \overline{f(x^{(t-5(i+1))})}}{\overline{f(x^{(t-5(i+1))})}} \right\}_{i=0}^{24}$
- $\left\{ \frac{\overline{\nabla f(x^{(t-5i)})}}{|\overline{\nabla f(x^{(\max(t-5(i+1),t \mod 5))})}|+1} \right\}_{i=0}^{24}$
- $\left\{ \frac{|\overline{x^{(\max(t-5(i+1),t \mod 5+5))}} - \overline{x^{(\max(t-5(i+2),t \mod 5))}})|}{|\overline{x^{(t-5i)}} - \overline{x^{(t-5(i+1))}}|+0.1} \right\}_{i=0}^{24}$

Observation features $\Psi(\cdot)$:

- $\frac{f(x^{(t)}) - f(x^{(t-1)})}{f(x^{(t-1)})}$
- $\frac{\nabla f(x^{(t)})}{|\nabla f(x^{(\max(t-1,0))})|+1}$
- $\frac{|x^{(\max(t-1,1))} - x^{(\max(t-2,0))}|}{|x^{(t)} - x^{(t-1)}|+0.1}$

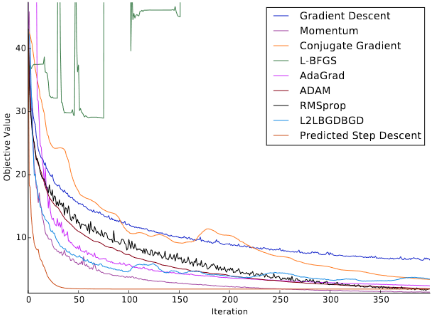
Implementation details

Optimizer was trained on the following objective:

- Architecture: two-layer neural net with 48 input units, 48 hidden units, 10 output units
- Objective: Classification
- Dataset: Randomly projected and normalized MNIST
- Batch size: 64
- Horizon length: $T = 400$

Results on TFD

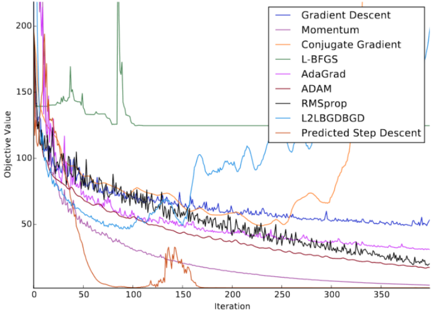
Data: Toronto Faces Database (TFD) — 3300 images, 7 categories.



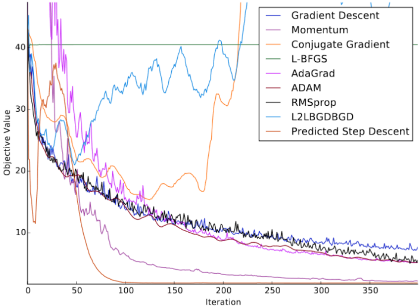
TFD 48 inputs units, 48 hidden units,
minibatch size: 64

Results on TFD

Data: Toronto Faces Database (TFD) — 3300 images, 7 categories.



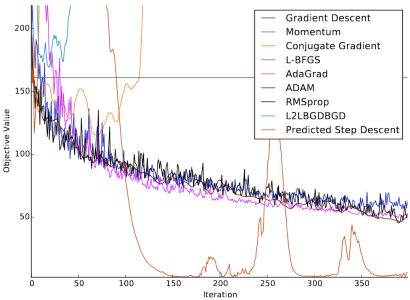
100 inputs units, 200 hidden units, minibatch size: 64



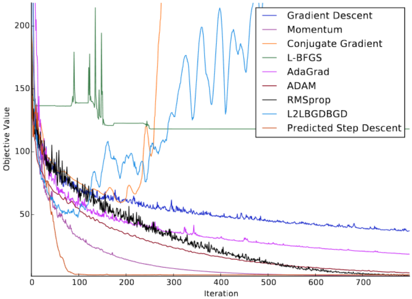
48 inputs units, 48 hidden units, minibatch size: 10

Results on TFD

Data: Toronto Faces Database (TFD) — 3300 images, 7 categories.



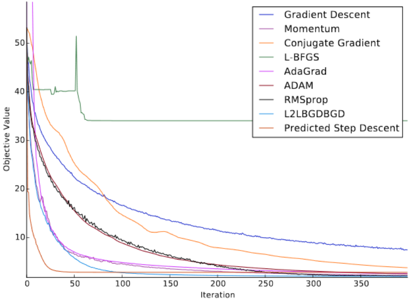
100 inputs units, 200 hidden units, minibatch size: 10



100 inputs units, 200 hidden units, minibatch size: 64, 2x iterations

Results on CIFAR-10

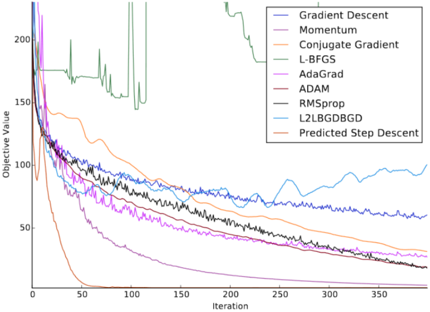
Data: CIFAR-10 — 50000 images, 10 categories.



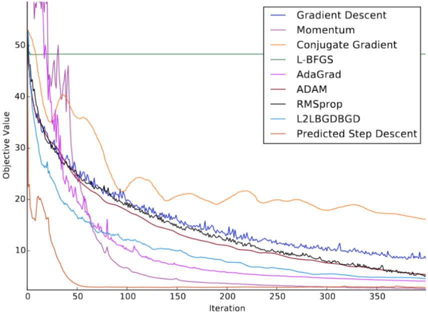
CIFAR-10 48 inputs units, 48 hidden units, minibatch size: 64

Results on CIFAR-10

Data: CIFAR-10 — 50000 images, 10 categories.



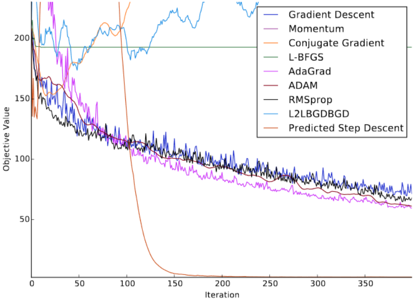
100 inputs units, 200 hidden units, minibatch size: 64



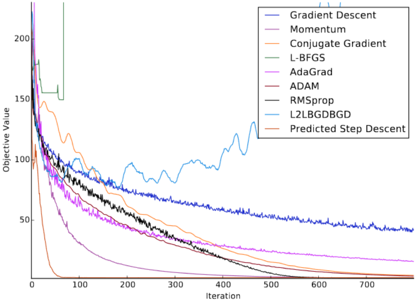
48 inputs units, 48 hidden units, minibatch size: 10

Results on CIFAR-10

Data: CIFAR-10 — 50000 images, 10 categories.



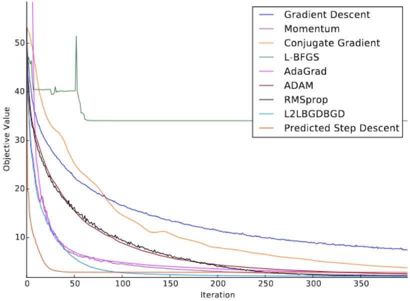
100 inputs units, 200 hidden units, minibatch size: 10



100 inputs units, 200 hidden units, minibatch size: 64, 2x iterations

Results on CIFAR-100

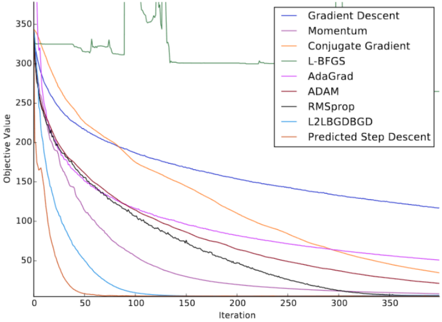
Data: CIFAR-100 — 50000 images, 100 categories.



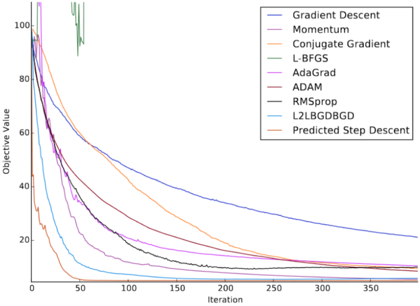
CIFAR-10 48 inputs units, 48 hidden units, minibatch size: 64

Results on CIFAR-100

Data: CIFAR-100 — 50000 images, 100 categories.



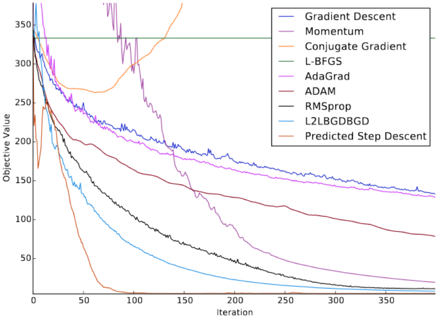
100 inputs units, 200 hidden units, minibatch size: 64



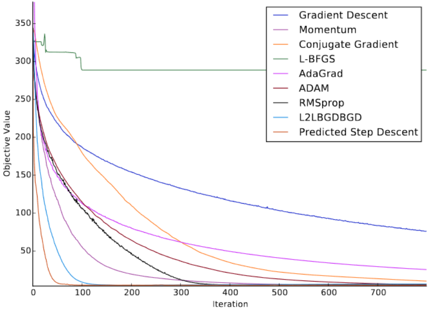
48 inputs units, 48 hidden units, minibatch size: 10

Results on CIFAR-100

Data: CIFAR-100 — 50000 images, 100 categories.



100 inputs units, 200 hidden units, minibatch size: 10



100 inputs units, 200 hidden units, minibatch size: 64, 2x iterations

Summary

We have discussed how to use RL for optimization.
Resulting algorithm has several advantages:

- Good generalization
- Good performance
- No need to tune any parameters
- Quite fast training