Learning to learn by gradient descent by gradient descent

Chen Zhao

NIPS 2016 (`https://arxiv.org/pdf/1606.04474.pdf`)

1 Introduction

An algorithm is said to learn to learn if its performance at each task improves with experience and with the number of tasks. Frequently, tasks in machine learning can be expressed as the problem of optimizing an objective function defined over some domain. The goal is to find the minimizer $\theta^* = \arg\min_{\theta \in \Theta} f(\theta)$. The standard approach for differentiable functions is some form of gradient descent, resulting in a sequence of updates.

2 Motivation

- Most of the modern work is based around designing update rules for specific classes of problems, it might perform poorly on other class of problems.
- In this work authors take a different tack and instead propose to replace hand-designed update rules with a learned update rule.

3 Main Idea

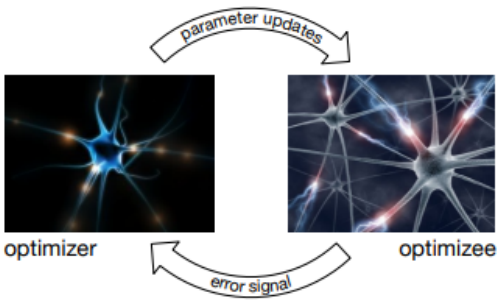

Figure 1: The optimizer (left) is provided with performance of the optimizee (right) and proposes updates to increase the optimizee's performance. [photos: Bobolas, 2009, Maley, 2011]

In this work, they proposed to replace hand-designed update rules with a learned update rule, which is called the optimizer (LSTM) with its own parameter ϕ. This results in updates to the optimizee of the form

$$\theta_{t+1} = \theta_t + g_t(\nabla f(\theta_t), \phi)$$

where g_t is the output of LSTM. For training the optimizer, we have an objective that depends on the trajectory for a time horizon T.

$$\mathcal{L}(\phi) = \mathbb{E}[\sum_{t=1}^{T} w_t f(\theta_t)]$$

where

- $\theta_{t+1} = \theta_t + g_t$
- $[g_t \quad h_{t+1}] = m(\nabla_t, h_t, \phi)$, m is the LSTM.

- $\nabla_t = \nabla_\theta f(\theta_t)$
- θ is the optimizee parameters; ϕ is the optimizer parameters; f is the function in question.

One challenge in applying RNNs in our setting is that we want to be able to optimize at least tens of thousands of parameters. Optimizing at this scale with a fully connected RNN is not feasible as it would require a huge hidden state and an enormous number of parameters. To avoid this difficulty we will use an optimizer m which operates coordinatewise on the parameters of the objective function, similar to other common update rules like RMSprop and ADAM. This coordinatewise network architecture allows us to use a very small network that only looks at a single coordinate to define the optimizer and share optimizer parameters across different parameters of the optimizee. Different behavior on each coordinate is achieved by using separate activations for each objective function parameter. In addition to allowing us to use a small network for this optimizer, this setup has the nice effect of making the optimizer invariant to the order of parameters in the network, since the same update rule is used independently on each coordinate.

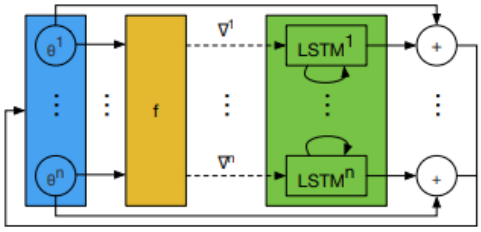

Figure 3: One step of an LSTM optimizer. All LSTMs have shared parameters, but separate hidden states.

Information Sharing Between Coordinates:

- global average cells(GAC) designate a subset of the cells in each LSTM layer for communication. their outgoing activations are averages at each step across all coordinates.
- allowing different LSTMs to communicate with each other