Learning to Learn

Using Deep Networks with Memory Capacity

Dr. Hien Nguyen V
Aryan Mobiny

Department of Electrical and Computer Engineering
University of Houston
Houston, TX

Abstract

The goal of the proposed work is to replace the hand-designed update rule of the standard optimization algorithms (such as gradient descent, Newton's method, etc.) with a learned update rule, and investigate the associated computational methods to improve the overall performance of the optimization algorithm. The limitations of current optimization algorithms are associated with their hand-designed update rules tailored only to the specific problem in hand, their poor performance for a large number of system parameters, and the need for a lot of data to learn, often through extensive iterative training. We hypothesize that casting the design of optimization algorithm as a learning problem, called learning to learn or meta-learning, allows the algorithm to learn to exploit structure in the problems of interest in an automatic way. Moreover, using neural nets with memory capacity (like recurrent neural networks or LSTM) enables us not only to handle many parameters, but also to rapidly learn never-before-seen functions and make accurate predictions with a low number of data samples.

1 Introduction

Phrases like "I have experience in ...", "This is similar to ..." or "This is a typical case of ..." imply that the person making such statements learns the task at hand faster or more accurately than an inexperienced human. This learning enhancement results from solution regularities in a problem domain. In a conventional machine learning approach, the learning algorithm mostly does not take into account previous learning experiences despite the fact that methods similar to human reasoning are expected to yield better performance. The use of previous learning experiences in inductive reasoning is known as"knowledge transfer" or "inductive bias shifts". Here, we focus on one of the most appealing topics in knowledge transfer research field: "*meta-learning*" or "*learning to learn*".

learning to learn is an exciting new research direction in designing optimization algorithms that can change the way they *generalize*, i.e., practice the task of learning itself and improve on it. To illustrate this, it's worthwhile to compare machine learning to human learning. Humans encounter a continual stream of learning tasks. They don't just learn concepts or motor skills, they also learn bias, i.e., they learn how to generalize. As a result, humans are often able to generalize correctly from extremely few examples; often just a single example is enough to teach us a new thing. Given a family of tasks, an algorithm is said to learn to learn if its performance at each task improves with experience and with the number of tasks. Put differently, an ordinary learning algorithm whose performance doesn't depend on the number of learning tasks, which hence would not benefit from the presence of other learning tasks, is not said learn to learn. For an algorithm to fit this definition, some kind of transfer must occur between multiple tasks that must have a positive impact on expected task-performance. Generally speaking, learning to learn or meta-learning refers to a scenario in which an agent learns at two levels, each associated with different time scales. Rapid learning occurs within a task, for example, when learning to accurately classify emphwithin a particular data set. This learning is guided by knowledge accrued more gradually *across* tasks, which captures the way in which task structure varies across target domains.

2 Methodology

In machine learning, tasks can be expressed as the problem of optimizing a cost function $f(\theta)$ and the goal is to find parameter θ that minimizes the cost ($\theta^* = argmin_\theta\ f(\theta)$). Among all possible methods to be applied, gradient descent is the standard approach for differentiable functions, resulting in the following sequence of updates:

$$\theta_{t+1} = \theta_t - \alpha_t(\nabla f(\theta_t))$$

While there are various types of problems of interest in different research areas, much of the focus in optimization works is based around such hand-designed update rules which make it suitable only to the specific problem in hand. In other words, exploiting the structure of the problem of interest for designing the optimizer comes at the expense of potentially poor performance on problems outside of that scope.

The goal of this work is to use the idea of meta-learning to come up with a procedure for constructing the learning algorithm which performs well on our desired class of problems. To this end, we'll take a different tack and replace the hand-designed update rule of gradient descent with

optimizer g which can be learned. The resulting updates of the objective function f is of the form

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

where φ denotes the set of parameters of optimizer g. This notation enables us to cast the design of optimization algorithm as a learning problem so that the resulting optimizers are specialized to particular classes of functions.

In order to model the update rule g, we can rely on the ability of deep networks to generalize to new examples by learning interesting sub-structures. Specifically, it's been proposed that recurrent neural networks (RNN) are quite suitable to model the update rule g. Having internal memory enables RNN to maintain its own states and learn dynamic update rules which integrate information from the history of gradients.

Later on, we can compare the performance of our learned neural optimizer with state-of-the-art methods used in deep learning such as RMSprop, ADAM, and Nestrov's accelerated gradient (NAG). To provide a meaningful comparison to our proposed learned optimizer, for each problem and each of these methods we have to tune the design parameters such as learning rate, and report results with the set of parameters that gives the best performance for each problem.

Another aspect of high importance which is often overlooked has to do with the speed at which the algorithm converge. In this sense, "rate of convergence", the speed at which a convergent sequence approaches its limit, gives us a measure of the efficiency of the designed optimizer. We believe that while much of the focus in optimization works is on minimizing the cost function, inserting the rate of convergence in the problem of learning the optimizer and maximizing it is the key to achieve a suitable optimization algorithm. Given a sequence $x_1,\ x_2,\ \ldots,\ x_n$ converging to a target value r, the rate of convergence of the sequence (α) is defined as:

$$\alpha \approx \frac{(log|x_{n+1} - r| - \lambda)}{log|x_n - r|}$$

Where λ is a positive real number. Rewriting the formula by inserting the objective function results in

$$\mathbb{E}[\alpha] = \mathbb{E}\left[\frac{(log|f(\theta_{n+1}(f, \varphi)) - r| - \lambda)}{log|f(\theta_n(f, \varphi)) - r|}\right]$$

In this setting, r can be estimated from the current literature or we can set it to zero by default. To evaluate the performance of the proposed optimization algorithm, it needs to be implemented to several classes of experiments to see how well it can derive the learning algorithms from scratch. We believe that our learned algorithm is able to outperform generic, hand-designed competitors on the problems for which they are trained, and also generalize well to new problems with similar structure. Our goal is to demonstrate this on various classes of problems, from optimizing simple convex functions to functions with sparsity and group sparsity constraints, and from learning neural networks to deep networks with certain constraints such as sparsity and group sparsity. Testing and evaluating the generalization of the learned optimizer to different architectures can be done for different applications and purposes, e.g., regression, image classification, image style transfer and synthesis, and by using various sources of data such as MNIST and ImageNet.