Topmoumoute Online Natural Gradient Algorithm

Nicolas Le Roux, Pierre-Antoine Manzagol and Yoshua Bengio
Dept. IRO, Université de Montréal
C.P. 6128, Montreal, Qc, H3C 3J7, Canada
{*lerouxni,manzagop,bengioy*}*@iro.umontreal.ca*
http://www.iro.umontreal.ca/∼lisa

Natural gradient is a gradient descent technique which uses the inverse of the covariance matrix of the gradient. Using the central-limit theorem, we prove that it yields the direction that minimizes the probability of overfitting. However, its prohibitive computational cost makes it impractical for online training. Here, we present a new online version of the natural gradient which we coin TONGA (Topmoumoute Online Natural Gradient Algorithm).

Introduction

Successful learning posesses two qualities: it leads to good generalization and it is fast. With respect to the choice of an optimization algorithm, while the latter is important to many, few seem to consider the former. This is surprising considering the plethora of situations where the same cost function can lead to several solutions depending on the path chosen, such as non-convex optimization or early stopping.

This work aims at improving the quality of solutions through better optimization techniques. Consider an optimization on a train set with access to a validation set. As the end objective is a good solution with respect to generalization, one often uses early stopping: optimizing the train error, while monitoring the validation error to fight overfitting. This approach makes the underlying assumption that overfitting happens last. A better perspective is that overfitting happens all through the learning, but starts being detrimental only at the point it overtakes the "true" learning. In terms of gradients, the gradient on the training set is never colinear with the true gradient, and the dot product between those actually becomes negative at a certain point. Early stopping is designed to determine that point. One can thus wonder: can one limit overfitting before that point? Would this actually postpone that point?

Minimizing the probability of overfitting

Starting from these considerations and using the central limit theorem, we derive an estimate of the distribution of the true gradient. Based upon that distribution and the fact that what matters during gradient descent (in terms of generalization error) is the value of the dot product between the descent direction and the true gradient, we can now consider two policies for the choice of the descent direction:

- the direction that minimizes the expected value of the dot product. This direction turns out to be the **empirical gradient**.
- the direction that minimizes the probability of the dot product being positive (meaning an increase in the generalization error). This is the **natural gradient** $\widehat{\Sigma}^{-1}\hat{\mu}$, where $\hat{\mu}$ is the empirical gradient and $\widehat{\Sigma}$ the covariance matrix of the empirical gradient.

The natural gradient

Amari introduced the natural gradient as the direction of steepest descent in a Riemanian space. If the space is actually Euclidean, the natural gradient boils down to the usual gradient. Amari showed that the parameter space is a Riemanian space of metric Σ. The natural gradient somewhat ressembles the Newton method with two important differences: Σ is positive-definite, which makes the technique more stable, but contains no second order information. The Hessian allows to account for slight variations in the parameters. The covariance matrix accounts for slight variations in the set of training samples. It also means that, if the gradients highly disagree in one direction, one should not go in that direction, even if the mean suggests otherwise. In that sense, it is a conservative gradient.

Fast batch formulation

Even in the batch setting, the computation of $\widehat{\Sigma}^{-1}$ is impractical. Fortunately, we do not need this matrix explicitly, only its product with $\hat{\mu}$. Yang and Amari (1998) exploit this, but as Schraudolph (2002) notes, the resulting algorithm is rather complicated and Press, Flannery, Teukolsky, and Vetterling (1986) give a preferable iterative procedure

$$v_{t+1} = v_t - D(\hat{\mu} - \widehat{\Sigma} v_t)$$

which converges to $\widehat{\Sigma}^{-1}\hat{\mu}$ for all v_0 and any matrix D. The convergence is faster if D is close to $\widehat{\Sigma}^{-1}$ (in some sense). Since $\widehat{\Sigma} = GG'$, with G the matrix with the centered gradients concatenated, we have $\widehat{\Sigma}^{-1}v_t = G(G'v_t)$ and $\widehat{\Sigma}$ need not be stored at any time. However, even though each step needs $O(nd)$ computations and $O(nd)$ memory space, the number of steps to achieve convergence is undetermined. Since this iterative procedure is indeed a gradient descent over a quadratic cost function, we apply conjugate gradient descent and find this attains convergence in usually less than five steps (for a toy problem with 600 parameters).

Experiments on a toy dataset have shown that this procedure performs better (both in train and in test) than stochastic gradient descent, batch gradient descent and conjugate gradient descent. One reason might be that the inverse covariance prevents it from taking a decision too soon (i.e. this algorithm is very conservative and tries to postpone its decision as much as possible).

Even though this procedure is surprisingly fast and totally appropriate for the batch setting, its computational cost in $O(nd)$ leaves it inappropriate for the online setting.

Online formulation

In the online setting, we cannot afford to compute the product $G'v_t$ for every new sample. Furthermore, the definition of the covariance matrix needs to be rethought. Indeed, we will use a moving mean and a moving covariance matrix with the update formula:

$$\widehat{\mu}_{t+1} = \gamma\widehat{\mu}_t + (1-\gamma)g_{t+1} \quad (1)$$

$$\widehat{\Sigma}_{t+1} = \gamma\widehat{\Sigma}_t + (1-\gamma)(g_{t+1}-\mu_{t+1})(g_{t+1}-\mu_{t+1})' \quad (2)$$

Performing only one step of the iterative procedure mentioned above and starting with $v_0 = d_t$, the direction d_{t+1} at time $t+1$ is given by

$$\begin{aligned} d_{t+1} &= d_t + \nu\left(g_{t+1} - \widehat{\Sigma}_{t+1}d_t\right) \\ &= d_t + \nu\left(g_{t+1} - \gamma\widehat{\Sigma}_t d_t - (1-\gamma)(g_{t+1}-\mu_{t+1}) < g_{t+1}-\mu_{t+1}, d_t >\right) \\ d_{t+1} &= d_t + \nu\left(g_{t+1} - \gamma g_t - (1-\gamma)(g_{t+1}-\mu_{t+1}) < g_{t+1}-\mu_{t+1}, d_t >\right) \quad (3) \end{aligned}$$

One might wonder why v_0 is taken to be equal to d_t and not $g_t{+}1$. This allows the computation of $\widehat{\Sigma}_{t+1}v_0$ to be straightforward. This also introduces a form of smoothing.

Conclusion and ongoing work

We presented here another justification for the natural gradient of Amari. Besides, we provided a fast computation of that gradient, making it practical for online use while retaining most of its interesting properties. However, several improvements can still be made:

- instead of using a moving covariance matrix, we could use the covariance matrix of the last k gradients, with $k << n$, and thus be able to perform the online procedure exactly
- it would be interesting to use a diagonal matrix D instead of a scalar. This is similar to the work of Schraudolph (2002) with the Stochastic Meta-Descent algorithm

References

Press, W., Flannery, B., Teukolsky, S., & Vetterling, W. (1986). *Numerical Recipes*. Cambridge University Press, Cambridge.

Schraudolph, N. N. (2002). Fast curvature matrix-vector products for second-order gradient descent. *Neural Computation*, *14*(7), 1723–1738.

Yang, H. H., & Amari, S. (1998). Complexity issues in natural gradient descent method for training multi-layer perceptrons. *Neural Computation*, *10*(8), 2137–2157.