A fast natural Newton method

Nicolas Le Roux NICOLAS.LE.ROUX@GMAIL.COM
Andrew Fitzgibbon AWF@MICROSOFT.COM
Microsoft Research, 7 JJ Thomson Avenue, Cambridge, CB3 0FB UK

Abstract

Nowadays, for many tasks such as object recognition or language modeling, data is plentiful. As such, an important challenge has become to find learning algorithms which can make use of all the available data. In this setting, called "large-scale learning" by Bottou & Bousquet (2008), learning and optimization become different and powerful optimization algorithms are suboptimal learning algorithms. While most efforts are focused on adapting optimization algorithms for learning by efficiently using the information contained in the Hessian, Le Roux et al. (2008) exploited the special structure of the learning problem to achieve faster convergence. In this paper, we investigate a natural way of combining these two directions to yield fast and robust learning algorithms.

1. Introduction

Machine learning often looks like optimization: write down the likelihood of some training data under some model and find the model parameters which maximize that likelihood, or which minimize some divergence between the model and the data. In this context, conventional wisdom is that one should find in the optimization literature the state of the art optimizer for one's problem and use it.

Furthermore, many machine learning objective functions are smooth in the optimization sense, so second-order optimizers are the tools of choice. And indeed, comparing second order methods to first order ones shows significant improvements in learning speed.

However, recent research (Le Roux et al., 2008) has shown that, by paying attention to the special structure of machine learning problems (viewing the gradient obtained as a noisy estimate of the true gradient of the function we are really interested in), one could obtain faster convergence speeds than first order gradient descent methods without using the Hessian. We investigate whether this improvement is due to the similarity of these methods to approximate second-order methods or, if this is not the case, if we can combine these improvements with the ones obtained when using the information contained in the Hessian.

The paper is organized as follows: section 2 explores the differences between the optimization and the learning frameworks, section 3 describes our proposed algorithm combining Newton method and natural gradient, which is the basis for the experiments in section 4.

2. Optimization versus learning

2.1. Optimization methods

The goal of optimization is to minimize a function f, which we will assume to be twice differentiable and defined from a space E to $\mathbb{R}$, over E. This is a problem with a considerable literature (Nocedal & Wright, 2006). It is well known that second order descent methods, which rely on the Hessian of f (or approximations thereof), enjoy much faster theoretical convergence than first order methods (quadratic versus linear), even when accounting for the potential complexity of computing and inverting the Hessian. Such methods include the Newton method, Gauss-Newton, Levenberg-Marquardt and Quasi-Newton methods such as BFGS.

2.2. Online learning

The learning framework differs slightly from the optimization one. The function f we wish to minimize (which we call the "cost function") is defined as the expected value of a function $\mathcal{L}$ under a distribution p, that is

$$f(\theta) = \int_x \mathcal{L}(\theta, x) p(x) \, dx \tag{1}$$

Appearing in *Proceedings of the 27^{th} International Conference on Machine Learning*, Haifa, Israel, 2010.

and we have access only to samples x_i drawn from p. If we have n samples, we can define a new function

$$\widehat{f}(\theta) = \frac{1}{n} \sum_i \mathcal{L}(\theta, x_i) \quad . \tag{2}$$

Let us call f the *test cost*, and $\widehat{f}$ the *training cost*. The x_i are the *training data*. As n goes to infinity, the difference between f and $\widehat{f}$ vanishes.

Bottou & Bousquet (2008) study the case where one has access to a potentially infinite amount of training data but only a finite amount of time. This setting, which they dub **large-scale learning**, calls for a tradeoff between the quality of the optimization for each datapoint and the number of datapoints treated. They showed that:

1. good optimization algorithms may be poor learning algorithms
2. stochastic gradient descent enjoys a faster convergence rate than batch gradient descent
3. introducing second order information can win us a constant factor (the condition parameter).

Therefore, the choice lies between first and second order stochastic gradient descent, depending on the additional cost of taking second order information into account and the condition parameter. Recently, several authors have developed algorithms allowing for efficient use of this second order information in a stochastic setting (Schraudolph et al., 2007; Bordes et al., 2009). However, we argue, all of these methods are derived from optimization methods without taking into account the particular nature of the learning problem.

2.3. Taking uncertainty into account

To our knowledge, the first paper explicitly accounting for the uncertainty of the gradient computed on the training set is (Le Roux et al., 2008). The argument is as follows. With f and $\widehat{f}$ as above, we write the gradient of f as

$$g = \frac{df}{d\theta} = \int_x \frac{\partial \mathcal{L}(\theta, x)}{\partial \theta} p(x)\, dx \tag{3}$$

and the gradient of $\widehat{f}$ as

$$\widehat{g} = \frac{d\widehat{f}}{d\theta} = \frac{1}{n} \sum_i \frac{\partial \mathcal{L}(\theta, x_i)}{\partial \theta} \tag{4}$$

where the dependence of g and $\widehat{g}$ on θ has been omitted to keep the notation uncluttered. We may think of g as the "true" gradient of f, and of $\widehat{g}$ as an "empirical" gradient of f, which we view as the mean of a set of samples drawn from a distribution with true mean g. If the training samples are iid, and given the "large-scale" assumption of large n, we can use the central-limit theorem, which yields

$$\widehat{g} \mid g \sim \mathcal{N}\left(g, \frac{C}{n}\right) \tag{5}$$

where

$$C = \int_x \left(\frac{\partial \mathcal{L}(\theta, x)}{\partial \theta} - g\right)\left(\frac{\partial \mathcal{L}(\theta, x)}{\partial \theta} - g\right)^T p(x)\, dx \tag{6}$$

is the true covariance matrix of the gradients. Relaxing eq. 5 to finite n and defining an isotropic Gaussian prior over g:

$$g \sim \mathcal{N}(0, \sigma^2 I), \tag{7}$$

yields the following posterior distribution over the true gradient:

$$g \mid \widehat{g} \sim \mathcal{N}\left(\left[I + \frac{C}{n\sigma^2}\right]^{-1} \widehat{g}, \left[nC^{-1} + \sigma^{-2} I\right]^{-1}\right). \tag{8}$$

Replacing the true covariance C by the empirical covariance $\widehat{C}$ defined as:

$$\widehat{C} = \frac{1}{n} \sum_i \left(\frac{\partial \mathcal{L}(\theta, x_i)}{\partial \theta} - \widehat{g}\right)\left(\frac{\partial \mathcal{L}(\theta, x_i)}{\partial \theta} - \widehat{g}\right)^T, \tag{9}$$

the direction of maximum expected gain becomes

$$\Delta\theta \propto \left[I + \frac{\widehat{C}}{n\sigma^2}\right]^{-1} \widehat{g}, \tag{10}$$

reminiscent of the natural gradient (Amari, 1998), with two differences:

- $\widehat{C}$ is here the centered covariance matrix, whereas (Amari, 1998) uses the uncentered one;
- when the number of datapoints n goes to infinity, the effect of the covariance matrix vanishes. This is understandable as, in that case, f and $\widehat{f}$ are equal, and so are g and $\widehat{g}$.

Le Roux et al. (2008) report large speedups on various neural network problems. Intuitively, one can understand why using the covariance may be beneficial. Indeed, it seems wasteful to compute the gradient over many data points and only keep their mean. While this allows for greater accuracy, one would think that more could (and should) be kept from these computations.

2.4. Natural gradient is not an approximation to Newton

Before moving on to the core of the paper, we clarify the links between natural gradient and Newton method as this should help the reader understand the advantage one can gain from using both.

2.4.1. Similarities

Maximum likelihood: Let us assume that we are training a density model by minimizing the negative log-likelihood. The cost function f_{nll} is defined by

$$f_{\text{nll}}(\theta) = -\int_x \log[L(\theta, x)]p(x)\, dx. \tag{11}$$

Note that this L is not the same as the $\mathcal{L}$ used in sections 2.2 and 2.3. Let us assume that there is a set of parameters θ **such that our model is perfect** and that we are at this θ. Then the covariance matrix of the gradients at that point is equal to the Hessian of f_{nll}. In the general case, this equality does not hold.

Gauss-Newton: Gauss-Newton is an approximation to the Newton method when f can be written as a sum of residuals:

$$f(\theta) = \frac{1}{2}\sum_i f_i(\theta)^2. \tag{12}$$

Computing the Hessian of f yields

$$\frac{\partial^2 f(\theta)}{\partial \theta^2} = \sum_i f_i(\theta)\frac{\partial^2 f_i}{\partial \theta^2} + \sum_i \frac{\partial f_i}{\partial \theta}\frac{\partial f_i}{\partial \theta}^T. \tag{13}$$

If the f_i get close to 0 (relative to their gradient), the first term may be ignored, yielding the following approximation to the Hessian:

$$H \approx \sum_i \frac{\partial f_i}{\partial \theta}\frac{\partial f_i}{\partial \theta}^T. \tag{14}$$

One, however, must be aware that:

- this approximation is only interesting when the f_i are **residuals** (that is, when the approximation is valid close to the optimum)
- the gradients involved are those of f_i and not of f_i^2
- the term on the right-hand side is the **uncentered** covariance of these gradients.

In order to compare the result of eq. 14 to the natural gradient, we will assume that the sum in eq. 12 is over datapoints, that is

$$f(\theta) = \frac{1}{2}\sum_i f_i(\theta)^2 = \frac{1}{N}\sum_i \mathcal{L}(\theta, x_i) \tag{15}$$

with the cost for each datapoint being

$$\mathcal{L}(\theta, x_i) = \frac{N}{2} f_i(\theta)^2. \tag{16}$$

The gradient of this cost with respect to θ is

$$g_i = \frac{\partial \mathcal{L}(\theta, x_i)}{\partial \theta} = N f_i(\theta)\frac{\partial f_i(\theta)}{\partial \theta}. \tag{17}$$

At the optimum (where the average of the gradients is zero and the centered and uncentered covariance matrices are equal), the covariance matrix of the g_i's is

$$C = \sum_i g_i g_i^T = N^2 \sum_i f_i(\theta)^2 \frac{\partial f_i}{\partial \theta}\frac{\partial f_i}{\partial \theta}^T, \tag{18}$$

which is a weighted sum of the terms involved in eq. 14. Thus the natural gradient and the Gauss-Newton approximation, while related, are different quantities, and (as we show) have very different properties.

2.4.2. Differences

Remember what the Hessian is: a measure of the change in gradient when we move in **parameter space**. In other words, the Hessian helps to answer the question: *if I were at a slightly different position in parameter space, how different would the gradient be?* It is a quantity defined for any (twice differentiable) function.

On the other hand, the covariance matrix of the gradients captures the uncertainty around this particular choice of training data, i.e. the change in gradient when we move in **input space**. In other words, the covariance helps us to answer the question: *If I had slightly different training data, how different would the gradient be?* This quantity only makes sense when there are training data.

Whereas the Hessian seems naturally suited to optimization problem (it allows us to be less short-sighted when minimizing a function), the covariance matrix unleashes its power in the learning setting, where we are only given a subset of the data. From this observation, it seems natural to combine these two matrices.

3. Combining Newton method and natural gradient

Building upon (Le Roux et al., 2008), we now show how to use Hessian information within the natural gradient. The Newton method assumes that the cost function is locally quadratic, i.e:

$$f(\theta) = \frac{1}{2}(\theta - \theta^*)^T H(\theta - \theta^*) \tag{19}$$

for some value of θ^*. In the case of a learning problem, this would translate to

$$f(\theta) = \int_x \mathcal{L}(\theta, x) p(x)\, dx = \int_x \frac{1}{2}(\theta - x)^T H (\theta - x) p(x)\, dx \tag{20}$$

with $\theta^* = \int_x x p(x)\, dx$. Here we make the assumption that H depend only weakly on x, a common assumption in online second-order methods. The derivative of this cost is:

$$g(\theta) = \int_x \frac{\partial \mathcal{L}(\theta, x)}{\partial \theta} p(x)\, dx = H(\theta - \theta^*). \tag{21}$$

We can see that, in the context of a quadratic function, the isotropic prior over g proposed in eq. 7 is erroneous as g is clearly influenced by H. We shall rather consider an isotropic Gaussian prior on the quantity $\theta - \theta^*$ as we do not have any information about the position of θ relative to θ^*. The resulting prior distribution over g is

$$g \sim \mathcal{N}\left(0, \sigma^2 H^2\right) \tag{22}$$

where we omitted the dependence on θ to keep the notation uncluttered. In a similar fashion to section 2.3, we will suppose that we are only given a finite training set composed of n datapoints x_i with associated gradients g_i. The empirical gradient $\widehat{g}$ is the mean of the g_i's. Using the central-limit theorem, we again have

$$\widehat{g} \mid g \sim \mathcal{N}\left(g, \frac{C}{n}\right) \tag{23}$$

where C is the true covariance of the gradients, i.e.

$$C = \int_x \left(\frac{\partial \mathcal{L}(\theta, x)}{\partial \theta} - g\right)\left(\frac{\partial \mathcal{L}(\theta, x)}{\partial \theta} - g\right)^T p(x)\, dx. \tag{24}$$

Therefore, the posterior distribution over g is

$$g \mid \widehat{g} \sim \mathcal{N}\left(\left[I + \frac{CH^{-2}}{n\sigma^2}\right]\widehat{g}, \left[\frac{H^{-2}}{\sigma^2} + nC^{-1}\right]^{-1}\right) \tag{25}$$

Since the function is locally quadratic, we wish to move in the direction $H^{-1}g$. This direction follows a Gaussian distribution with mean

$$\left[I + \frac{H^{-1}CH^{-1}}{n\sigma^2}\right]^{-1} H^{-1}\widehat{g} \tag{26}$$

and covariance

$$\left[\frac{I}{\sigma^2} + nHC^{-1}H\right]^{-1}. \tag{27}$$

Once again, we shall replace the true covariance matrix C of the gradients by its empirical counterpart, $\widehat{C}$. Since eq. 26 appears complicated, we shall explain it. Let us define by d_i the Newton directions:

$$d_i = H^{-1} g_i. \tag{28}$$

Since $\widehat{C}$ is the covariance matrix of the gradients g_i, $H^{-1}\widehat{C}H^{-1} = \widehat{D}$ is the covariance matrix of the d_i's. We can therefore rewrite

$$H^{-1} g \mid \widehat{g} \sim \mathcal{N}\left(\left[I + \frac{\widehat{D}}{n\sigma^2}\right]^{-1} \widehat{d}, \left[\frac{I}{\sigma^2} + n\widehat{D}^{-1}\right]^{-1}\right) \tag{29}$$

where $\widehat{d}$ is the average of the Newton directions, i.e. $\widehat{d} = H^{-1}\widehat{g}$. The direction which maximizes the expected gain is thus

$$\delta\theta \propto -\left[I + \frac{\widehat{D}}{n\sigma^2}\right]^{-1} \widehat{d} \quad . \tag{30}$$

This formula is exactly the (regularized) natural gradient, but on the Newton directions. This is good news as it means that one may choose his favorite second-order gradient descent method (for instance, SGD-QN (Bordes et al., 2009)) to compute the Newton directions, and then his favorite natural gradient algorithm (for instance, TONGA (Le Roux et al., 2008)) to apply to these Newton directions, to yield an algorithm combining the advantages of both methods.

As a side note, one can see that, as the number n of data points used to compute the mean increases, the prior vanishes and the posterior distribution concentrates around the empirical Newton direction.

As mentioned in section 2.2, online methods are faster than batch methods. Thus, we will update the parameters of our model after each datapoint, replacing the empirical mean $\widehat{d}$ and covariance $\widehat{D}$ in eq. 30 by running averages, as detailed in section 3.2.

We shall now analyze several components of this method.

3.1. Setting a zero-centered prior at each timestep

Eq. 29 has been obtained using the zero-centered Gaussian prior defined in eq. 22. Except for the first update, one may wonder why we would use such a distribution rather than the posterior distribution at the previous timestep as our prior. The reason is that, whenever we update the parameters of our model, the distribution over the gradients changes. If the function to optimize were truly quadratic, we could exactly quantify the change in gradient using our approximation of the Hessian. Unfortunately, this is not

the case. Thus, while acknowledging that using the prior of eq. 22 at every timestep is a suboptimal strategy, we believe there is still something to be gained while retaining the simplicity of the algorithm.

3.2. Exponentially moving covariance matrix

Since efficiency is our main goal, we need a fast way to update the covariance matrix of the data points which progressively "forgets" about older data. For that purpose, we shall use exponentially moving mean and covariance, namely:

$$\gamma_n = \sum_{i=1}^{n} \gamma^{n-i} \tag{31}$$

$$\mu_n = \frac{\sum_{i=1}^{n} \gamma^{n-i} d_i}{\gamma_n} \tag{32}$$

$$= \frac{(\gamma_n - 1)\mu_{n-1} + d_n}{\gamma_n} \tag{33}$$

$$\widehat{D}_n = \frac{\sum_{i=1}^{n} \gamma^{n-i}(d_i - \mu_n)(d_i - \mu_n)^T}{\gamma_n - \frac{\sum_{i=1}^{n} \gamma^{-2i}}{\gamma_n}} \tag{34}$$

where d_i is the Newton direction obtained at timestep i and γ is the discount factor. The closer γ is to 1, the longer examples will influence the means and covariance.

Introducing U_n, the uncentered covariance matrix at timestep n, we can easily update $\widehat{D}_n$ using:

$$U_n = \frac{\sum_{i=1}^{n} \gamma^{-i} d_i d_i^T}{\gamma_n} \tag{35}$$

$$= \frac{(\gamma_n - 1)U_{n-1} + d_n d_n^T}{\gamma_n} \tag{36}$$

$$\widehat{D}_n = U_n - \mu_n \mu_n^T \tag{37}$$

Therefore, to update $\widehat{D}_n$, one first computes the new γ_n, then computes μ_n and U_n which will be combined to yield C_n.

If the number of parameters is large, computing a full covariance matrix would be too costly. Le Roux et al. (2008) propose an efficient way of computing a low-rank approximation of the covariance matrix. Though their method is for an uncentered covariance matrix, it can be modified to accommodate centered covariance matrices.

3.3. Frequency of updates

The covariance matrix of the gradients changes very slowly. Therefore, one does not need to update it as often as the Hessian approximation. In the SGD-QN algorithm, the authors introduce a counter *skip* which specifies how many gradient updates are done before the approximation to the Hessian is updated. We introduce an additional variable *skipC* which specifies how many Hessian approximation updates are done before updating the covariance approximation. The total number of gradient updates between two covariance approximation updates is therefore $skip \cdot skipC$.

Experiments using the validation set showed that using values of *skipC* lower than 8 did not yield any improvement while increasing the cost of each update. We therefore used this value in all our experiments. This allows us to use the information contained in the covariance with very little computation overhead.

3.4. Limiting the influence of the covariance

Eq. 29 tells us that the direction to follow is

$$\left[I + \frac{\widehat{D}}{n\sigma^2} \right]^{-1} \widehat{d}. \tag{38}$$

The only unknown in this formula is σ^2, which is the variance of our Gaussian prior on $\theta - \theta^*$. To avoid having to set this quantity by hand at every time step, we will devise a heuristic to find a sensible value of σ^2. While this will lack the purity of a full Bayesian treatment, it will allow us to reduce the number of parameters to be set by hand, which we think is a valuable feature of any gradient descent algorithm.

If we knew the distance from our position in parameter space, θ, to the optimal solution, θ^*, then the optimal value for σ^2 would be $\|\theta - \theta^*\|^2$. Of course, this information is not available to us. However, if the function to optimize were truly quadratic, the squared norm of the Newton direction would be exactly $\|\theta - \theta^*\|^2$. We shall therefore replace σ^2 by the squared norm of the last computed Newton direction. Since this estimate may be too noisy, we will replace it by the squared norm of the running average of the Newton directions, i.e. $\|\mu_n\|^2$.

However, even then, we may still get undesirable variations. We shall therefore adopt a conservative strategy: we will set an upper bound on the correction to the Newton method brought by eq. 38. More precisely, we will bound the eigenvalues of $\frac{\widehat{D}}{n\|\mu_n\|^2}$ by a positive number B. The parameter update then becomes

$$\theta_n - \theta_{n-1} = -\left[I + \min\left(B, \frac{C_d}{n\|\mu_n\|^2} \right) \right]^{-1} H^{-1} g_n \tag{39}$$

where B is a hyperparameter and $\min(B, M)$ is defined for symmetric matrices M with eigenvectors $u_1, \ldots, u_n$

and eigenvalues $\lambda_1, \ldots, \lambda_n$ as

$$\min(B, M) = \sum_{i=1}^{n} \min(B, \lambda_i) u_i u_i^T \quad , \qquad (40)$$

(we bound each eigenvalue of M by B). If we set $B = 0$, we recover the standard Newton method. This modification transforms the algorithm in a conservative way, trading off potential gains brought by the covariance matrix with guarantees that the parameter update will not differ too much from the Newton direction.

The pseudo-code for the algorithm is shown in Algorithm 1.

4. Experiments

4.1. Algorithms chosen

Our algorithm requires two independent components:

1. an approximation to the Newton method, to get the Newton directions
2. an approximation to the natural gradient to be applied to the Newton directions.

In these experiments, the former was chosen to be SGD-QN (Bordes et al., 2009), since it recently won the Wild Track competition at the Pascal Large Scale Learning Challenge. Since this method uses a diagonal approximation to the Hessian, we decided to use a diagonal approximation to the covariance matrix. Though this was not required and we could have used a low-rank covariance matrix, using a diagonal approximation shows the improvements over the original method one can obtain with little extra effort.

4.2. Experimental setup

Experiments have been led on datasets of the Pascal Large Scale Learning Challenge, namely Alpha, Gamma, Delta, Epsilon, Zeta and Face datasets. Labels were only available for the training examples of the challenge. We therefore split these examples into several sets:

- the first 100K (1M for the Face dataset) examples constituted our training set
- the last 100K (1M for the Face dataset) examples constituted our test set

The last 50K (500K for the Face dataset) examples of the training set were used as validation examples to tune the bound B defined in eq. 39. The same value of B was used for TONGA.

Algorithm 1 Simplified pseudo-code of the Natural-Newton algorithm

Require: : skip (number of gradient updates between Hessian updates)
Require: : skipC (number of Hessian updates between covariance updates)
Require: : θ_0 (the original set of parameters)
Require: : γ (the discount factor for the moving covariance)
Require: : T (the total number of epochs)
Require: : t_0, λ (the weight decay)

t = 0, count = skip, countC = skipC
$\gamma_0 = 0$, $\zeta_0 = 0$
$\mathbf{H} = \lambda^{-1}\,\mathbf{I}$, $\mathbf{D} = \mathbf{H}$
$\mu_1 = 0$ (the running mean vector), $C_1 = 0$ (the running covariance matrix)
while $t \neq T$ **do**
  $g_t \leftarrow \frac{\partial \mathcal{L}(\theta_t, x_t, y_t)}{\partial \theta_t}$
  $\theta_{t+1} \leftarrow \theta_t - (t + t_0)^{-1} \mathbf{D} g_t$
  if count == 0 **then**
    count ← skip
    Update **H**, the approximation to the Hessian, according to the SGD-QN algorithm
    if countC == 0 **then**
      countC ← skipC
      $\gamma_{t+1} \leftarrow \gamma_t * \gamma + 1$
      $\zeta_{t+1} \leftarrow \zeta_t * \gamma^2 + 1$
      $\mu_{t+1} \leftarrow \frac{(\gamma_{t+1} - 1)\mu_t + d_t}{\gamma_{t+1}}$
      $C_{t+1} \leftarrow \frac{(\gamma_{t+1} - 1) C_t + d_t d_t^T}{\gamma_{t+1}}$
      N ← $1 - \frac{\zeta_{t+1}}{\gamma_{t+1}^2}$
      $\mathbf{D} = \left(I + \frac{C_{t+1} - \mu_{t+1} \mu_{t+1}^T}{N \cdot \|\mu_{t+1}\|^2} \right)^{-1}$
    else
      countC ← countC - 1
    end if
  else
    count ← count - 1
  end if
end while

4.3. Parameter tuning

In all the experiments, γ has been set to 0.995, following (Le Roux et al., 2008). To test the sensitivity of the algorithm to this parameter, we tried other values (0.999, 0.992, 0.99 and 0.9) without noticing any significant difference in validation errors.

We optimized the bound on the covariance (§3.4) on the validation set. The best value was chosen for the test set, but we found that a value of 2 yielded near-optimal results on all datasets, the difference between $B = 1$, $B = 2$ and $B = 5$ being minimal, as shown in

figure 1 in the case of the Alpha dataset.

4.4. Results

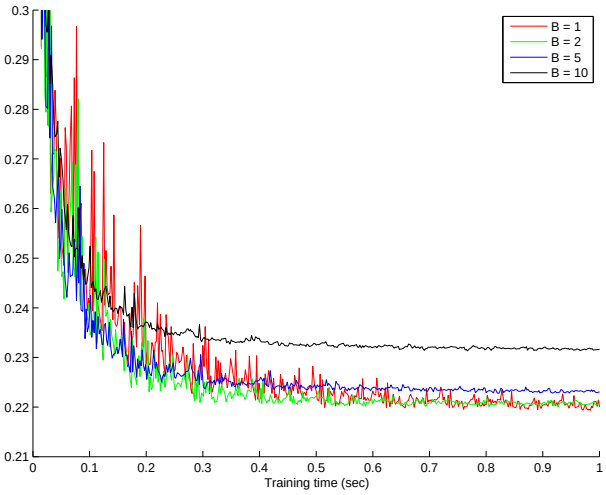

Figure 1. Validation error vs. time on the Alpha dataset, for various values of B.

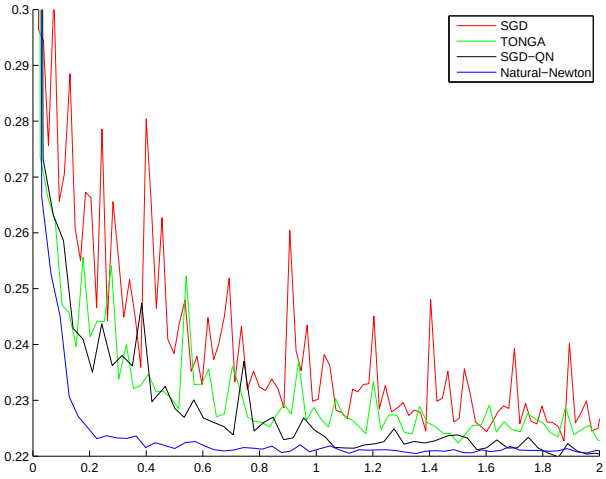

Figure 2. Test error vs. time on the Alpha dataset

Several conclusions may be drawn from these experiments:

- Natural-Newton never performs worse than SGD-QN and always better than TONGA. Using a large value of *skipC* ensures that the overhead of using the covariance matrix is negligible
- on the Alpha dataset, using the information contained in the covariance resulted in significantly faster convergence, with or without second-order information
- on the Epsilon, Zeta and Face datasets, using the covariance information stabilizes the results while

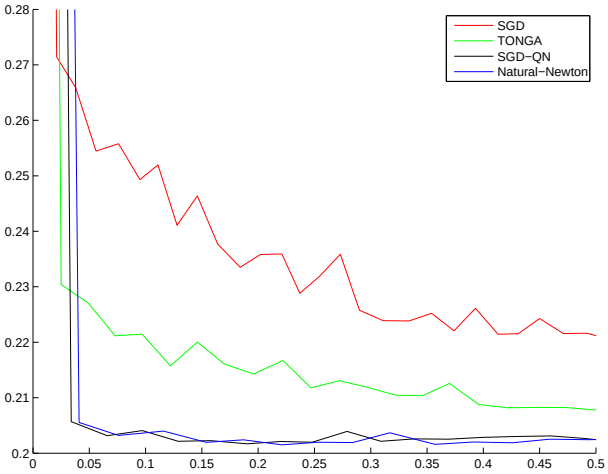

Figure 3. Test error vs. time on the Gamma dataset

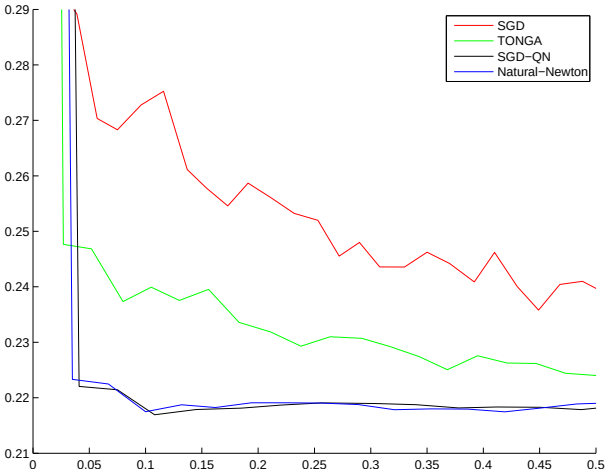

Figure 4. Test error vs. time on the Delta dataset

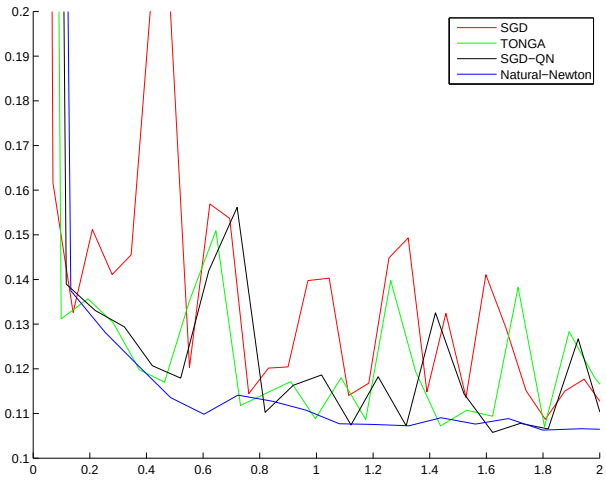

Figure 5. Test error vs. time on the Epsilon dataset

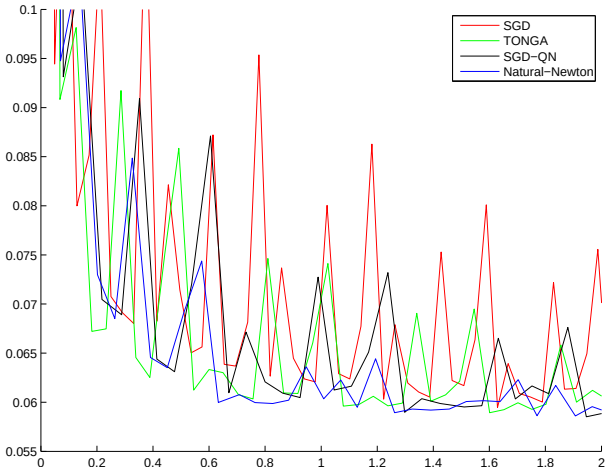

Figure 6. Test error vs. time on the Zeta dataset

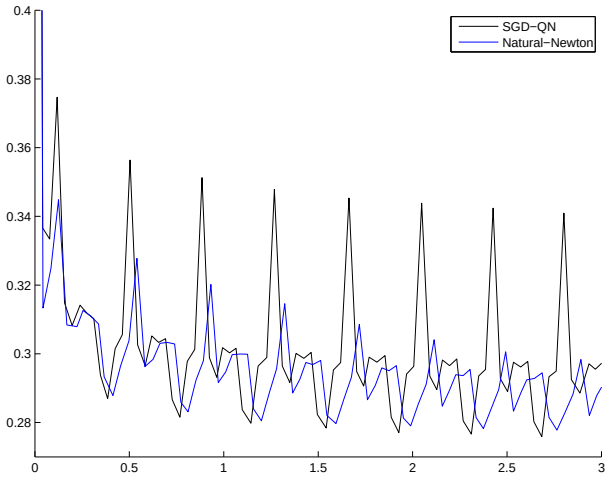

Figure 7. Test error vs. time on the Face dataset

yielding the same convergence speed. This is in accordance with the use of the covariance, which reduces the influence of directions where gradients vary wildly

- on the Gamma and the Delta dataset, the covariance information helped a lot when the Hessian was not used, yielding no improvement otherwise.

5. Conclusion

A lot of effort has been put into designing efficient online optimization algorithms, with great results. Most of these algorithms rely on some approximation to the Hessian or to the covariance matrix of the gradients. While the latter is commonly believed to be an approximation of the former, we proved that they encode very different kinds of information. Based on this, we proposed a way of combining information contained in the Hessian and in the covariance matrix of the gradients.

Experiments showed that, on most datasets, our method offered either faster convergence or increased robustness compared to the original algorithm. Furthermore, our algorithm never performed worse than the Newton algorithm is was built upon.

Moreover, our algorithm is able to use any existing second-order algorithm as base method. Therefore, while we used SGD-QN for our experiments, one may pick any algorithm best suited for a given task.

We hope to have shown two things. Firstly, the covariance matrix of the gradients is usefully viewed, not as an approximation to the Hessian, but as a source of additional information about the problem, for typical "machine learning" objective functions. Secondly, it is possible with little extra effort to use this information in addition to that provided by the Hessian matrix, yielding faster or more robust convergence.

Despite all these successes, we believe that our algorithm may be improved in several ways, whether it is by retaining some of the information contained in the posterior distribution between timesteps or in the selection of the parameter σ^2.

References

Amari, Shun-ichi. Natural gradient works efficiently in learning. *Neural Computation*, 10(2):251–276, 1998.

Bordes, Antoine, Bottou, Léon, and Gallinari, Patrick. SGD-QN: Careful quasi-newton stochastic gradient descent. *Journal of Machine Learning Research*, 10: 1737–1754, July 2009.

Bottou, Léon and Bousquet, Olivier. The tradeoffs of large scale learning. In Platt, J.C., Koller, D., Singer, Y., and Roweis, S. (eds.), *Advances in Neural Information Processing Systems*, volume 20, pp. 161–168. 2008.

Le Roux, Nicolas, Manzagol, Pierre-Antoine, and Bengio, Yoshua. Topmoumoute online natural gradient algorithm. In *Advances in Neural Information Processing Systems 20*, pp. 849–856. MIT Press, Cambridge, MA, 2008.

Nocedal, J. and Wright, S. J. *Numerical Optimization, Second Edition*. Springer Verlag, New York, 2006.

Schraudolph, Nicol N., Yu, Jin, and Günter, Simon. A stochastic quasi-newton method for online convex optimization. In *Proceedings of AISTATS 2007, Puerto Rico*. 2007.