

Hessian-free optimization (Truncated-Newton family)

 POLITECNICO DI MILANO



Andrea Sessa

Mat. 850082

Optimization - Prof. E. Amaldi

AA. 2016/2017





Old methods:

- **Gradient Descent:** It uses only first order information! Curvature properties of the function is ignored.
- **Newton Method:** Second order information are taken into consideration but it does not scale well for large instance problems (ie. deep neural nets.)

Many solution have been proposed in literature (Conjugate-gradient method, Quasi-Newton methods etc.).

In this presentation we describe a family of methods known as Hessian-free (or Truncated-Newton).

Moreover we propose an application of such method in the training of *Deep Neural networks*.



Newton Method - Recap - 1

Let $f \in \mathcal{C}^2$ and $B = H(x)$ with $x \in \mathbb{R}^n$

The problem is:

$$\begin{aligned} \min_x \quad & f(x) \\ \text{st.} \quad & x \in \mathbb{R}^n \end{aligned}$$

We consider the quadratic approximation of f at x_k :

$$q_k(\underline{x}) := f(\underline{x}_k) + \nabla^t f(\underline{x}_k)(\underline{x} - \underline{x}_k) + \frac{1}{2}(\underline{x} - \underline{x}_k)^t H(\underline{x}_k)(\underline{x} - \underline{x}_k)$$

We search for the direction \underline{d} that minimizes q_k .

In particular for the Newton method we have:

$$\underline{d} = H^{-1}(\underline{x}_k) \nabla f(\underline{x}_k)$$

and the update becomes:

$$\underline{x}_{k+1} = \underline{x}_k - H^{-1}(\underline{x}_k) \nabla f(\underline{x}_k)$$

An important property of the Newton method is *scale invariance*. Precisely if apply the linear (affine) transformation $\hat{x} = Ax$ (where A is non-singular matrix) then $\hat{d} = Ad$.

This is important in the context of neural network training: **badly scaled** parameters may be much harder to optimize!



Newton Method - Pathological Curvature scenario

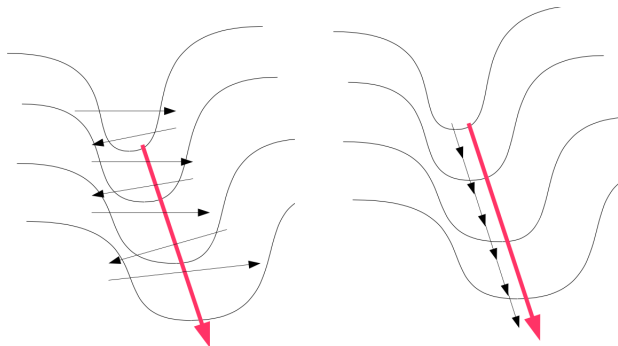


Figure: Optimizing along a narrow valley



The newton method implicitly solve the following linear system
(**Newton's system**):

$$\nabla^2 f(\underline{x}_k) \underline{d} = -\nabla f(\underline{x}_k) \quad (1)$$

Idea: Apply an iterative method for finding the solution of (1).

The iterative method is not run up to convergence but an approximate solution is accepted.



Hessian Free and Conjugate Gradient - 1

A very common choice for the inner procedure is the **Conjugate Gradient Method**.

CG is a very effecting algorithm for optimizing quadratic functions.

In the worst case CG is guaranteed to converge in at most n iterations it is impractical for very large problem to wait for CG to completely converge in general.

In practice the behaviour of CG is that it will make significant improvements well before n iterations.

Moreover despite Quasi-Newton methods, HF does not make any approximation of the Hessian of the function!



Hessian Free and Conjugate Gradient - 2

The classical CG method require some modifications:

The standard CG algorithm is used to solve positive definite systems, but the Hessian of the function may have negative eigenvalues, so CG must stop as soon as a direction of negative curvature is generated.

For every iteration of CG we require the following condition to be satisfied:

1. The starting point $d^{(0)} = \underline{0}$
2. Curvature test. If CG generates a direction d such that

$$p^t \nabla^2 f(x_k) p < 0$$



Hessian Free and Conjugate Gradient - 3

Then if $k = 0$, we complete another iteration and then CG return the solution $d^{(k+1)}$, if $k > 0$ then we stop immediately and return $d^{(k+1)}$.

Theorem 1. Assume that f is continuously differentiable in a neighborhood of a local solution x^* of the original problem. In addition, assume that $\nabla^2 f(x^*)$ is nonsingular and that ∇f is Lipschitz continuous at x^* .

Assume that iteration k of the truncated-Newton method computes a step p_k that satisfies:

$$\|\nabla f(\underline{x}_k) + \nabla^2 f(\underline{x}_k)p_k\| \leq \eta_k \|\nabla f(\underline{x}_k)\| \quad (2)$$

For a specified value of η_k ; the new estimate of the solution is computed using $x_{k+1} \leftarrow x_k + p_k$.

If x_0 is sufficiently close to x^* and $0 \leq \eta_k \leq \eta_{max} < 1$ then x_k converges to x^* linearly in the norm $\|\cdot\|_*$ defined by

$\|v\| := \|\nabla^2 f(x^*)v\|$, with asymptotic rate constant no greater than η_{max} .

If $\lim_{k \rightarrow \infty} \eta_k = 0$; then the convergence is superlinear.

If $k = O(\|f(x_k)\|^r)$ for $0 < r \leq 1$; then the convergence is of order at least $(1 + r)$.

Common choice for η_k (also known as *Forcing Sequence*) are:

$$\eta_k = \min \left\{ \frac{1}{2}, c \|\nabla f(x_k)\|^r \right\}$$

with $c \geq 0$ and $0 \leq r \leq 1$



Hessian Free + CG - Sketch Algorithm

Algorithm 6.1 (Line Search Newton–CG).

Given initial point x_0 ;

for $k = 0, 1, 2, \dots$

 Compute a search direction p_k by applying the CG method to
 $\nabla^2 f(x_k)p = -\nabla f_k$, starting from $x^{(0)} = 0$. Terminate when
 $\|r_k\| \leq \min(0.5, \sqrt{\|\nabla f_k\|})\|\nabla f(x_k)\|$, or if negative curvature is
 encountered, as described in (b);

 Set $x_{k+1} = x_k + \alpha_k p_k$, where α_k satisfies the Wolfe, Goldstein, or
 Armijo backtracking conditions;

end

Figure: Sketch algorithm for HF+CG method



In practice the method results to be very slow when $\nabla^2 f(\underline{x}_k)$ is nearly singular.

A trust-region approach is proposed to partially alleviate this problem.

In the following the autor of [1] propose a series of optimization to effectively use HF+CG in machine learning settings:

1. Damping
2. Matrix-vector Product
3. Termination Conditions
4. Preconditioning



Hessian Free + CG - Damping

The idea is to use CG to solve:

$$(\nabla^2 f(\underline{x}_k)\underline{d} + \lambda\underline{d}) = -\nabla f(\underline{x}_k)$$

where λ is the so-called damping factor. The exact curvature matrix available to HF allows for the identification of extremely low curvature directions.

When such a direction is detected CG will elect to move far from along it and possibly well outside the region where the original quadratic approximation is a sensible approximation.

λ can be interpreted as controlling how *conservative* the approximation is.



Let $\underline{d} \in \mathbb{R}^n$ and $H \in \mathbb{R}^{n \times n}$ then:

$$H\underline{d} = \lim_{\epsilon \rightarrow 0} \frac{\nabla f(\underline{x}_k + \epsilon \underline{d}) - \nabla f(\underline{x}_k)}{\epsilon}$$

Using this formula the product can be calculated using only one extra evaluation of the gradient of f .

Other techniques (with better numerical properties), in case complex arithmetic is available, permit an estimation that is accurate to $O(\epsilon)$.



Hessian Free + CG - Termination Condition

The typical termination condition (as seen before) is of the type:

$$\|r_k\| \leq \min \{0.5, \|\nabla f(x_k)\|^{\frac{1}{2}}\} \|\nabla f(x_k)\| \quad (3)$$

We use CG to solve $Ax = b$, but the method does not optimize $\|Ax - b\|^2$ but $\phi(x) = \frac{1}{2}x^t Ax - b^t x$. In particular we choose $A = H(x)$ and $b = -\nabla f(x)$.

One sub-optimal solution for one method may be terrible for the other.

In practice the authors of [1] found the following condition to be much more efficient:

$$i > k, \phi(x_i) < 0, \frac{\phi(x_i) - \phi(x_{i-k})}{\phi(x_i)} < k\epsilon \quad (4)$$

Preconditioning is a technique to accelerate CG.

We define a linear change of variables $\hat{x} = Px$.

The new quadratic objective is

$$\hat{\phi}(\hat{x}) = \frac{1}{2} \hat{x}^t P^{-t} A P^{-1} \hat{x} - (P^{-1}b)^t \hat{x}$$

To use preconditioning one must specify $M := P^t P$ with the understanding that $My = x$ is easy to solve.

A possible choice:

$$M = \left[\text{diag} \left(\sum_{i=1}^D \nabla f_i(x) \times \nabla f_i(x) \right) + \lambda I \right]^\alpha \quad (5)$$



Trust Region approach

Idea: search for \underline{d}_k and α_k at the current x_k over a *trust region* B_k .

Example: $B_k = \{\underline{x} \in \mathbb{R} : \|\underline{x} - x_k\| < \Delta_k\}$

This results in solving an additional problem referred as the *trust-region subproblem*.

The subproblem in general can be solved in closed form or with low computational requirements.

Moreover the size (or radius) of the trust region is changed adaptively at each iteration.



Trust Region - Subproblem

The trust region subproblem is generally defined as:

$$\min_{\underline{d} \in \mathbb{R}^n} \{m_k(\underline{d}) := f_k + \nabla f_k^t x + \frac{1}{2} \underline{d}^t B_k \underline{d}\} \quad st \quad ||\underline{d}|| < \Delta_k \quad (6)$$

Specifically for the Newton-CG method the computation step consists in setting $B_k = \nabla^2 f(x_k)$ and applying the CG procedure to:

$$B_k d_k = -\nabla f_k$$

and stopping if:

- The size (norm) of the solution exceeds the Δ_k .
- The system has been solved to a required accuracy.
- Negative curvature condition is reached.



Trust Region - Properties

The Trust-Region Newton-CG method has a number of desirable properties:

- It is **globally convergent**: the first step along the direction $-\nabla f_k$ identifies the Cauchy point of 6 and the subsequent directions only improves the model value.
- It requires no matrix factorizations, so we can exploit the sparsity structure of the Hessian without worrying about fill-in during a direct factorization.
- The CG iteration, the most computationally intensive part of the algorithm, may be executed in parallel since the key operation is a matrix-vector product.



Trust Region - Properties

- When the Hessian matrix is positive definite, the Newton-CG method approximates the pure Newton step more and more closely as the solution x^* is approached, so rapid convergence is also possible.
- Two advantages, compared with the line search Newton-CG method, are that the lengths of the steps are controlled by the trust region and that directions of negative curvature are often explored.
- A limitation of the trust-region Newton-CG method is that it accepts any direction of negative curvature, even when this direction gives an insignificant reduction in the model.

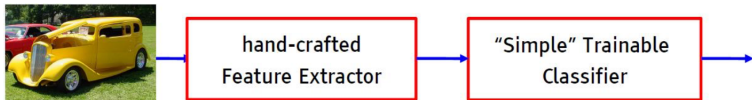
The limitation is overcome by the **Lanczos method**; the idea is to solve (1) but it does not stop at the first negative curvature direction but continues in search for a sufficiently negative curvature direction.



Deep Learning - Introduction

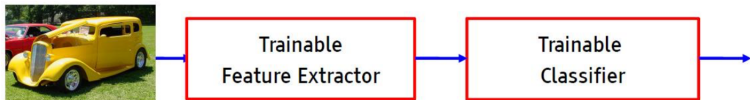
Deep learning can be summarized as learning both the representation and the classifier out of it.

- Fixed engineered features (or kernels) + trainable classifier



VS.

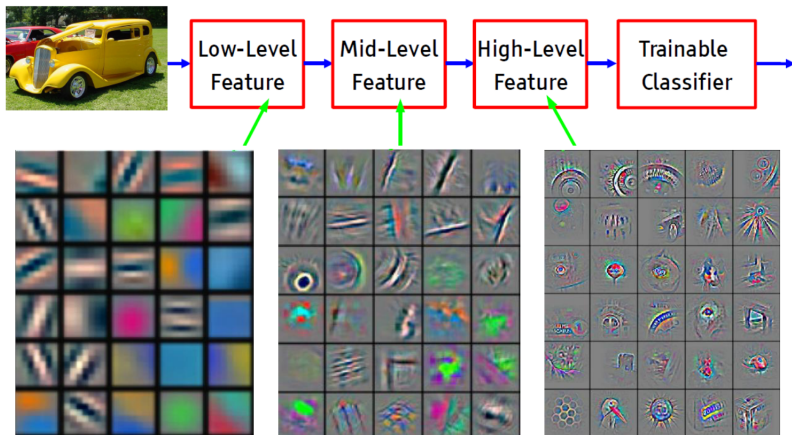
- End-to-end learning / feature learning / deep learning





Deep Learning - Representation

In deep learning we have multiple stages of non linear feature transformation





Deep Learning - Trainable features hierarchy

Deep learning assumes it is possible to *learn* a hierarchy of descriptors with increasing abstraction.

In image recognition

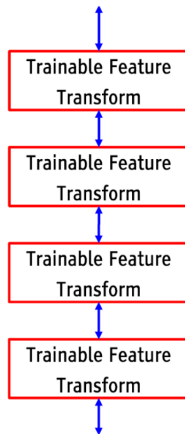
- Pixel → edge → texon → motif → part → object

In text analysis

- Character → word → word group → clause → sentence → story

In speech recognition

- Sample → spectral band → sound → ... → phone → phoneme → word

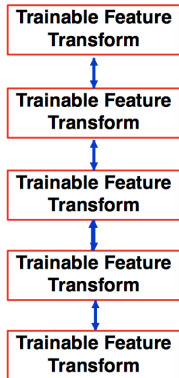




Depending on the direction of the information flow we can have different architectures for the hierarchy of features

- Feed forward (e.g., multilayer neural nets, convolutional nets, etc.)
- Feed back (e.g., Stacked sparse coding, Deconvolutional nets, etc.)
- Bi-directional (e.g., Deep Boltzmann Machines, **Stacked Auto-Encoders**, etc.)

Associated with different types of learning protocols (supervised, semi-supervised, self-taught, etc.)





Given an unlabelled dataset $\{x^{(i)}\}_{i=1}^m$, an auto encoder is a two-layer neural network that learns non-linear codes to represents (or reconstruct) the data.

Specifically we want to learn $h(x^{(i)}; W, b) = \sigma(Wx^{(i)} + b)$ such that $\sigma(W^th(x^{(i)}; W, b) + c)$ is approximately $x^{(i)}$. That is:

$$\min_{W,b,c} \sum_{i=1}^m \|\sigma(W^th(x^{(i)}; W, b) + c) - x^{(i)}\|_2^2$$



Experiments - Autoencoders - 2

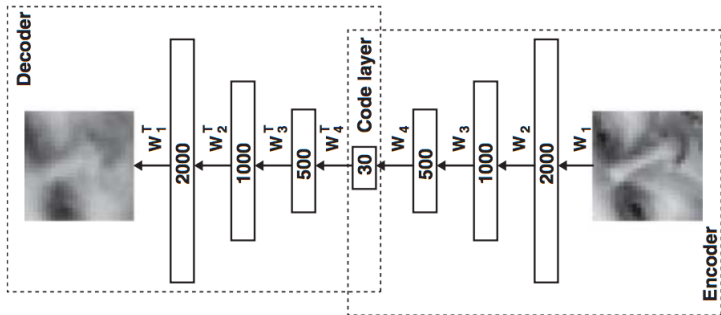


Figure: A stacked autoencoder



NAME	SIZE	K	ENCODER DIMS
CURVES	20000	5000	784-400-200-100-50-25-6
MNIST	60000	7500	784-1000-500-250-30
FACES	103500	5175	625-2000-1000-500-30

Figure: Parameters for different datasets

	PT + NCG	RAND+HF	PT + HF
CURVES	0.74, 0.82	0.11, 0.20	0.10, 0.21
MNIST	2.31, 2.72	1.64, 2.78	1.63, 2.46
MNIST*	2.07, 2.61	1.75, 2.55	1.60, 2.28
FACES	-, 124	55.4, 139	-, -
FACES*	-, -	60.6, 122	-, -

Figure: Results for different datasets



Experiments - Autoencoders - Timings

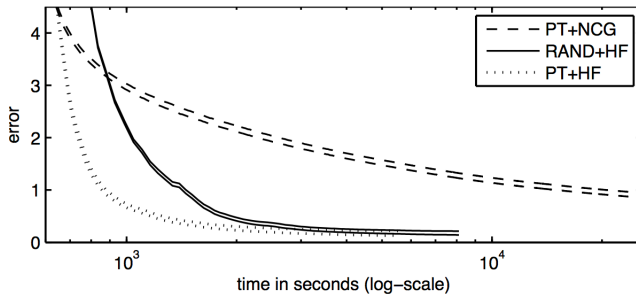


Figure: Timing for different methods and datasets







The most important implications is that the authors have been able to learn a deep-model with a completely generic optimizer without **the need of pretraining**.

A clear theme that emerges from the results is that HF optimized nets have a much lower training error, implying that the HF approach is more effective than pre-training + fine-tuning.

The pre-trained version of HF benefits in terms of speed of convergence but does not gain any significant advantage in terms of underfitting.

It is clear that pre-training helps 1st approximation algorithm in overcoming the underfitting problem by placing the parameter in a region less affected by the pathological-curvature scenario.



-  Martens, James. *Deep learning via Hessian-free optimization*. Proceedings of the 27th International Conference on Machine Learning (ICML-10). 2010.
-  Ngiam, Jiquan, et al. *On optimization methods for deep learning*. Proceedings of the 28th international conference on machine learning (ICML-11). 2011.
-  Wright, Stephen J., and Jorge Nocedal. *Numerical optimization*. Springer Science 35.67-68 (1999): pages 139-142.
-  Hinton, Geoffrey E., and Ruslan R. Salakhutdinov. *Reducing the dimensionality of data with neural networks*. Science 313.5786 (2006): pages 504-507.