Deep Learning via Hessian-free Optimization

James Martens

University of Toronto

June 29, 2010

Computer Science
UNIVERSITY OF TORONTO

Gradient descent is bad at learning deep nets

The common experience:

- gradient descent gets much slower as the depth increases

Gradient descent is bad at learning deep nets

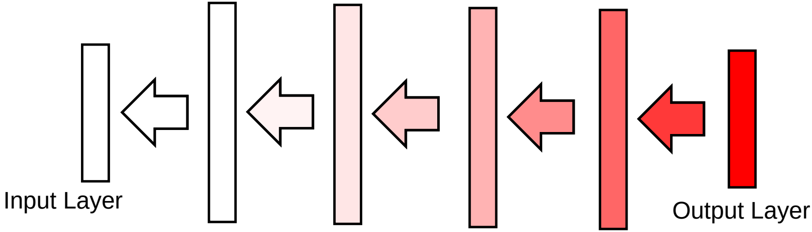
The common experience:

- gradient descent gets much slower as the depth increases
- large enough depth $\longrightarrow$ learning to slow to a crawl or even "stops" $\longrightarrow$ severe under-fitting (poor performance on the *training* set)

Gradient descent is bad at learning deep nets

The common experience:

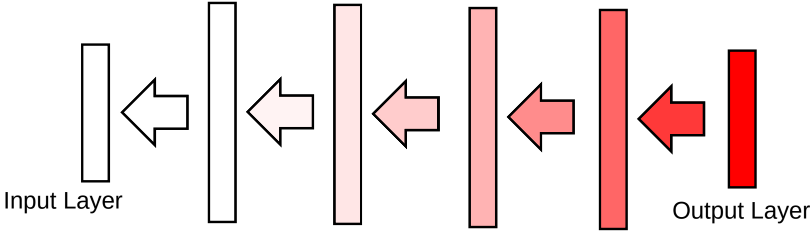
- gradient descent gets much slower as the depth increases
- large enough depth $\longrightarrow$ learning to slow to a crawl or even "stops" $\longrightarrow$ severe under-fitting (poor performance on the *training* set)
- "vanishing-gradients problem": error signal decays as it is backpropagated



Gradient descent is bad at learning deep nets

The common experience:

- gradient descent gets much slower as the depth increases
- large enough depth $\longrightarrow$ learning to slow to a crawl or even "stops" $\longrightarrow$ severe under-fitting (poor performance on the *training* set)
- "vanishing-gradients problem": error signal decays as it is backpropagated

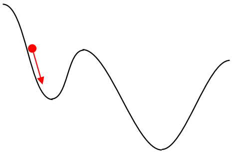


- the gradient is tiny for weights in early layers

Gradient descent is bad at deep learning (cont.)

Two hypotheses for why gradient descent fails:

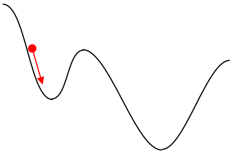
- increased frequency and severity of bad local minima:



Gradient descent is bad at deep learning (cont.)

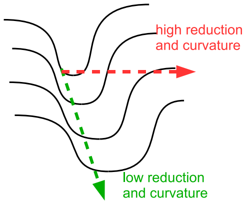
Two hypotheses for why gradient descent fails:

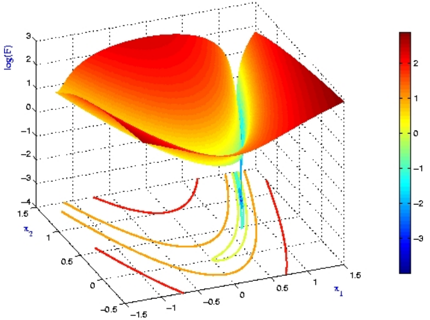
- increased frequency and severity of bad local minima:



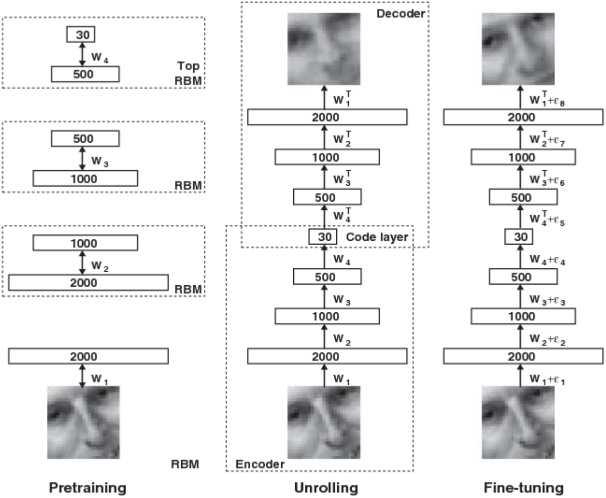
- pathological curvature, like the type seen in the well-known Rosenbrock function:

$$f(x,y) = (1-x)^2 + 100(y-x^2)^2$$





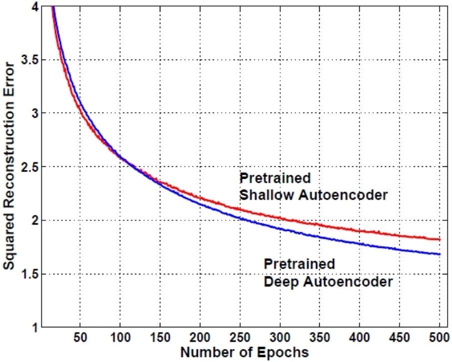
Pre-training for deep auto-encoders



(from Hinton and Salakhutdinov, 2006)

Pre-training (cont.)

- doesn't generalize to all the sorts of deep-architectures we might wish to train
- does it get full power out of deep auto-encoders?



(from Hinton and Salakhutdinov, 2006)

Our contribution

- we develop a very powerful and practical 2nd-order optimization algorithm based on the "Hessian-free" approach

Our contribution

- we develop a very powerful and practical 2nd-order optimization algorithm based on the "Hessian-free" approach
- we show that it can achieve significantly lower test-set reconstruction errors on the deep auto-encoder problems considered in Hinton and Salakhutdinov
  - no pre-training required!

Our contribution

- we develop a very powerful and practical 2nd-order optimization algorithm based on the "Hessian-free" approach
- we show that it can achieve significantly lower test-set reconstruction errors on the deep auto-encoder problems considered in Hinton and Salakhutdinov
  - no pre-training required!
- using pre-training still lowers *generalization* error on 2 of the 3 problems
  - *but critically there isn't a significant benefit on the training set*
- our method provides a better solution to the underfitting problem in deep networks and can be applied to a much larger set of models

2nd-order optimization

If pathological curvature is the problem, this could be the solution

2nd-order optimization

If pathological curvature is the problem, this could be the solution

General framework

- model the objective function by the local approximation:

$$f(\theta + p) \approx q_\theta(p) \equiv f(\theta) + \nabla f(\theta)^\top p + \frac{1}{2} p^\top \mathrm{B} p$$

where B is a matrix which quantifies curvature

2nd-order optimization

If pathological curvature is the problem, this could be the solution

General framework

- model the objective function by the local approximation:

$$f(\theta + p) \approx q_\theta(p) \equiv f(\theta) + \nabla f(\theta)^\top p + \frac{1}{2} p^\top \mathrm{B} p$$

  where B is a matrix which quantifies curvature

- in Newton's method, $\mathrm{B} = \mathrm{H}$ or $\mathrm{H} + \lambda I$

2nd-order optimization

If pathological curvature is the problem, this could be the solution

General framework

- model the objective function by the local approximation:

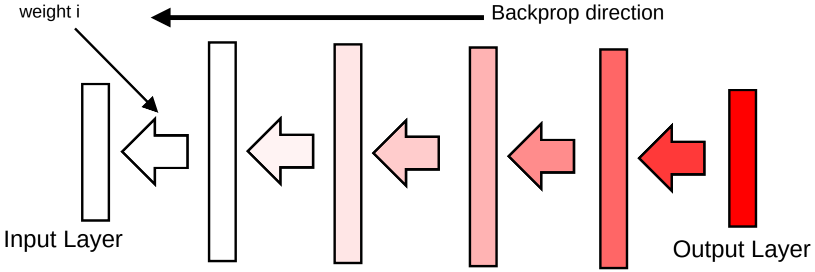
$$f(\theta + p) \approx q_\theta(p) \equiv f(\theta) + \nabla f(\theta)^\top p + \frac{1}{2} p^\top \mathrm{B} p$$

  where B is a matrix which quantifies curvature

- in Newton's method, $\mathrm{B} = \mathrm{H}$ or $\mathrm{H} + \lambda I$
- fully optimizing $q_\theta(p)$ this w.r.t. p gives: $p = -\mathrm{B}^{-1} \nabla f(\theta)$
- update is: $\theta \leftarrow \theta + \alpha p$ for some $\alpha \leq 1$ determined by a line search

Vanishing Curvature

- define the direction d by $d_k = \delta_{ik}$
- low reduction along d: $-\nabla f^\top d = -(\nabla f)_i \approx 0$
- but also low curvature: $d^\top \mathrm{H} d = -\mathrm{H}_{ii} = \frac{\partial^2 f}{\partial \theta_i^2} \approx 0$



- so a 2nd-order optimizer will pursue d at a reasonable rate, an elegant solution to the vanishing gradient problem of 1st-order optimizers

Practical Considerations for 2nd-order optimization

Hessian size problem

- for machine learning models the number of parameter N can be **very** large
- we can't possibly calculate or even store a $N \times N$ matrix, let alone invert one

Practical Considerations for 2nd-order optimization

Hessian size problem

- for machine learning models the number of parameter N can be **very** large
- we can't possibly calculate or even store a $N \times N$ matrix, let alone invert one

Quasi-Newton Methods

- non-linear conjugate gradient (NCG) - a hacked version of the quadratic optimizer linear CG
- limited-memory BFGS (L-BFGS) - a low rank Hessian approximation
- approximate diagonal or block-diagonal Hessian

Unfortunately these don't seem to resolve the deep-learning problem

Hessian-free optimization

- a quasi-newton method that uses no low-rank approximations
- named 'free' because we never explicitly compute B

Hessian-free optimization

- a quasi-newton method that uses no low-rank approximations
- named 'free' because we never explicitly compute B

First motivating observation

- it is relatively easy to compute the matrix-vector product $\mathrm{H}v$ for an arbitrary vectors v

Hessian-free optimization

- a quasi-newton method that uses no low-rank approximations
- named 'free' because we never explicitly compute B

First motivating observation

- it is relatively easy to compute the matrix-vector product $\mathrm{H}v$ for an arbitrary vectors v
- e.g. use finite differences to approximate the limit:

$$\mathrm{H}v = \lim_{\epsilon \to 0} \frac{\nabla f(\theta + \epsilon v) - \nabla f(\theta)}{\epsilon}$$

Hessian-free optimization

- a quasi-newton method that uses no low-rank approximations
- named 'free' because we never explicitly compute B

First motivating observation

- it is relatively easy to compute the matrix-vector product $\mathrm{H}v$ for an arbitrary vectors v
- e.g. use finite differences to approximate the limit:

$$\mathrm{H}v = \lim_{\epsilon \to 0} \frac{\nabla f(\theta + \epsilon v) - \nabla f(\theta)}{\epsilon}$$

- $\mathrm{H}v$ is computed for the *exact* value of H, there is no low-rank or diagonal approximation here!

Hessian-free optimization (cont.)

Second motivating observation

- linear conjugate gradient (CG) minimizes positive definite quadratic cost functions using only matrix-vector products

Hessian-free optimization (cont.)

Second motivating observation

- linear conjugate gradient (CG) minimizes positive definite quadratic cost functions using only matrix-vector products
- more often seen in the context of solving large sparse systems

Hessian-free optimization (cont.)

Second motivating observation

- linear conjugate gradient (CG) minimizes positive definite quadratic cost functions using only matrix-vector products
- more often seen in the context of solving large sparse systems
- directly minimizes the the quadratic $q \equiv p^\top \mathrm{B} p/2 + g^\top p$ and not the residual $\|\mathrm{B}p + g\|^2$ $\rightarrow$ these are related but different!

Hessian-free optimization (cont.)

Second motivating observation

- linear conjugate gradient (CG) minimizes positive definite quadratic cost functions using only matrix-vector products
- more often seen in the context of solving large sparse systems
- directly minimizes the the quadratic $q \equiv p^\top \mathrm{B} p/2 + g^\top p$ and not the residual $\|\mathrm{B}p + g\|^2$ $\rightarrow$ these are related but different!
- but we actually care about the quadratic, so this is good

Hessian-free optimization (cont.)

Second motivating observation

- linear conjugate gradient (CG) minimizes positive definite quadratic cost functions using only matrix-vector products
- more often seen in the context of solving large sparse systems
- directly minimizes the the quadratic $q \equiv p^\top \mathrm{B} p/2 + g^\top p$ and not the residual $\|\mathrm{B}p + g\|^2 \rightarrow$ these are related but different!
- but we actually care about the quadratic, so this is good
- requires $N = \dim(\theta)$ iterations to converge in general, but makes a lot of progress in *far* fewer iterations than that

Standard Hessian-free Optimization

Pseudo-code for a simple variant of damped Hessian-free optimization:

---

for $n = 1$ to max-epochs **do**
    compute gradient $g_n = \nabla f(\theta_n)$
    choose/adapt λ_n according to some heuristic
    define the function $\mathrm{B}_n(v) = \mathbf{H}v + \lambda_n v$
    $p_n = \text{CGMinimize}(B_n, -g_n)$
    $\theta_{n+1} = \theta_n + p_n$
end for

In addition to choosing λ_n, the stopping criterion for the CG algorithm is a critical detail.

A new variant is required

- **the bad news**: common variants of HF (e.g. Steihaug) don't work particular well for neural networks

A new variant is required

- **the bad news**: common variants of HF (e.g. Steihaug) don't work particular well for neural networks

- there are many aspects of the algorithm that are ill-defined in the basic approach which we need to address:
  - how can deal with negative curvature?
  - how should we choose λ?
  - how can we handle large data-sets
  - when should we stop the CG iterations?
  - can CG be accelerated?

Pearlmutter's R-operator method

- finite-difference approximations are undesirable for many reasons

Pearlmutter's R-operator method

- finite-difference approximations are undesirable for many reasons
- there is a better way to compute $\mathrm{H}v$ due to Pearlmutter (1994)

Pearlmutter's R-operator method

- finite-difference approximations are undesirable for many reasons
- there is a better way to compute $\mathrm{H}v$ due to Pearlmutter (1994)
- similar cost to a gradient computation

Pearlmutter's R-operator method

- finite-difference approximations are undesirable for many reasons
- there is a better way to compute $\mathrm{H}v$ due to Pearlmutter (1994)
- similar cost to a gradient computation
- for neural nets, no extra non-linear functions need to be evaluated

Pearlmutter's R-operator method

- finite-difference approximations are undesirable for many reasons
- there is a better way to compute $\mathrm{H}v$ due to Pearlmutter (1994)
- similar cost to a gradient computation
- for neural nets, no extra non-linear functions need to be evaluated
- technique generalizes to almost any twice-differentiable function that is tractable to compute

Pearlmutter's R-operator method

- finite-difference approximations are undesirable for many reasons
- there is a better way to compute $\mathrm{H}v$ due to Pearlmutter (1994)
- similar cost to a gradient computation
- for neural nets, no extra non-linear functions need to be evaluated
- technique generalizes to almost any twice-differentiable function that is tractable to compute
- can be automated (like automatic differentiation)

The Gauss-Newton Matrix ($\mathbf{G}$)

- a well-known alternative to the Hessian that is guaranteed to be positive semi-definite - thus no negative curvature!

The Gauss-Newton Matrix (G)

- a well-known alternative to the Hessian that is guaranteed to be positive semi-definite - thus no negative curvature!
- usually is applied to non-linear least squares problems
- Schraudolph showed in 2002 that it can be generalized beyond just least squares to neural nets with "matching" loss functions and output non-linearities
  - e.g. logistic units with cross-entropy error

The Gauss-Newton Matrix (G)

- a well-known alternative to the Hessian that is guaranteed to be positive semi-definite - thus no negative curvature!
- usually is applied to non-linear least squares problems
- Schraudolph showed in 2002 that it can be generalized beyond just least squares to neural nets with "matching" loss functions and output non-linearities
  - e.g. logistic units with cross-entropy error
- works better in practice than Hessian or other curvature matrices (e.g. empirical Fisher)

The Gauss-Newton Matrix (G)

- a well-known alternative to the Hessian that is guaranteed to be positive semi-definite - thus no negative curvature!
- usually is applied to non-linear least squares problems
- Schraudolph showed in 2002 that it can be generalized beyond just least squares to neural nets with "matching" loss functions and output non-linearities
  - e.g. logistic units with cross-entropy error
- works better in practice than Hessian or other curvature matrices (e.g. empirical Fisher)
- and we can compute $\mathrm{G}v$ using an algorithm similar to the one for $\mathrm{H}v$

CG stopping conditions

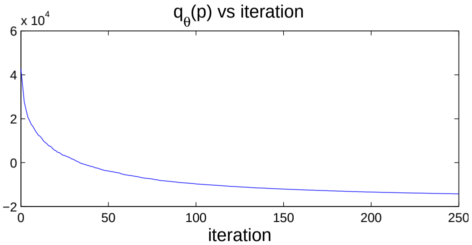
- CG is only guaranteed to converge after N (size of parameter space) iterations $\longrightarrow$ we can't always run it to convergence

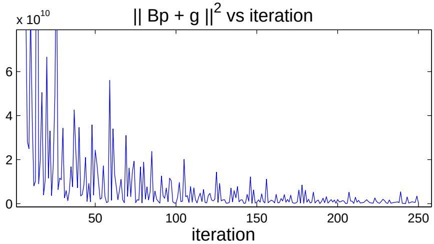
CG stopping conditions

- CG is only guaranteed to converge after N (size of parameter space) iterations $\longrightarrow$ we can't always run it to convergence
- the standard stopping criterion used in most versions of HF is $\|r\| < \min(\frac{1}{2}, \|g\|^{\frac{1}{2}})\|g\|$ where $r = \mathrm{B}p + g$ is the "residual"

CG stopping conditions

- CG is only guaranteed to converge after N (size of parameter space) iterations $\longrightarrow$ we can't always run it to convergence
- the standard stopping criterion used in most versions of HF is $\|r\| < \min(\frac{1}{2}, \|g\|^{\frac{1}{2}})\|g\|$ where $r = \mathrm{B}p + g$ is the "residual"
- strictly speaking $\|r\|$ is *not* the quantity that CG minimizes, nor is it the one we really care about





CG stopping conditions (cont.)

- we found that terminating CG once the relative per-iteration reduction rate fell below some tolerance ϵ worked best

$$\frac{\Delta q}{q} < \epsilon$$

(Δq is the change in the quadratic model averaged over some window of the last k iterations of CG)

Handling large datasets

- each iteration of CG requires the evaluation of the product $\mathrm{B}v$ for some v

Handling large datasets

- each iteration of CG requires the evaluation of the product $\mathrm{B}v$ for some v
- naively this requires a pass over the training data-set

Handling large datasets

- each iteration of CG requires the evaluation of the product $\mathrm{B}v$ for some v
- naively this requires a pass over the training data-set
- but for a sufficiently large subset of the training data - sufficient to capture enough useful curvature information

Handling large datasets

- each iteration of CG requires the evaluation of the product $\mathrm{B}v$ for some v
- naively this requires a pass over the training data-set
- but for a sufficiently large subset of the training data - sufficient to capture enough useful curvature information
- size is related to model and qualitative aspects of the dataset, but critically not its size
  - for very large datasets, mini-batches might be a tiny fraction of the whole

Handling large datasets

- each iteration of CG requires the evaluation of the product $\mathrm{B}v$ for some v
- naively this requires a pass over the training data-set
- but for a sufficiently large subset of the training data - sufficient to capture enough useful curvature information
- size is related to model and qualitative aspects of the dataset, but critically not its size
  - for very large datasets, mini-batches might be a tiny fraction of the whole
- gradient and line-searches can be computed using even larger mini-batches since they are needed much less often

Other enhancements

- using a Levenburg-Marquardt style heuristic for adjusting the damping parameter λ

Other enhancements

- using a Levenburg-Marquardt style heuristic for adjusting the damping parameter λ
- using M-preconditioned CG with the diagonal preconditioner:

$$M = \left[\mathrm{diag}\left(\sum_i \nabla f_i \odot \nabla f_i \right) + \lambda I \right]^{\alpha}$$

Other enhancements

- using a Levenburg-Marquardt style heuristic for adjusting the damping parameter λ
- using M-preconditioned CG with the diagonal preconditioner:

$$M = \left[\mathrm{diag}\left(\sum_i \nabla f_i \odot \nabla f_i\right) + \lambda I\right]^{\alpha}$$

- initializing each run of the inner CG-loop from the solution found by the previous run

Other enhancements

- using a Levenburg-Marquardt style heuristic for adjusting the damping parameter λ
- using M-preconditioned CG with the diagonal preconditioner:

$$M = \left[\mathrm{diag}\left(\sum_i \nabla f_i \odot \nabla f_i\right) + \lambda I\right]^{\alpha}$$

- initializing each run of the inner CG-loop from the solution found by the previous run
- carefully bounding and "back-tracking" the maximum number of CG steps to compensate for the effect of using mini-batches to compute the $\mathrm{B}v$ products

Other enhancements

- using a Levenburg-Marquardt style heuristic for adjusting the damping parameter λ
- using M-preconditioned CG with the diagonal preconditioner:

$$M = \left[\mathrm{diag}\left(\sum_i \nabla f_i \odot \nabla f_i\right) + \lambda I\right]^{\alpha}$$

- initializing each run of the inner CG-loop from the solution found by the previous run
- carefully bounding and "back-tracking" the maximum number of CG steps to compensate for the effect of using mini-batches to compute the $\mathrm{B}v$ products
- (see the paper for further details)

Results

Experimental parameters (K = mini-batch size)

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

Deep auto-encoder experiments

- used precisely the same model architectures and datasets as in Hinton and Salakhutdinov, 2006

Results

Experimental parameters (K = mini-batch size)

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

Deep auto-encoder experiments

- used precisely the same model architectures and datasets as in Hinton and Salakhutdinov, 2006
- CURVES, MNIST and FACES are all image datasets

Results

Experimental parameters (K = mini-batch size)

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

Deep auto-encoder experiments

- used precisely the same model architectures and datasets as in Hinton and Salakhutdinov, 2006
- CURVES, MNIST and FACES are all image datasets
- trained with cross-entropy but performance measured with squared error

Results

Experimental parameters (K = mini-batch size)

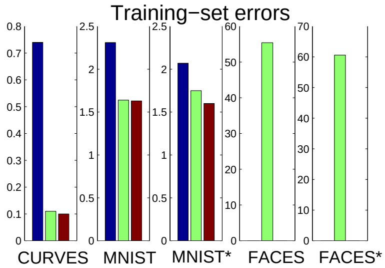
Name	size	K	encoder dims
CURVES	20000	5000	784-400-200-100-50-25-6
MNIST	60000	7500	784-1000-500-250-30
FACES	103500	5175	625-2000-1000-500-30

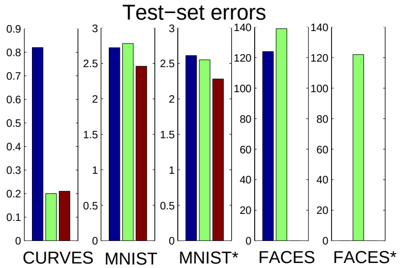
Deep auto-encoder experiments

- used precisely the same model architectures and datasets as in Hinton and Salakhutdinov, 2006
- CURVES, MNIST and FACES are all image datasets
- trained with cross-entropy but performance measured with squared error
- all methods were run using GPU implementations. H&S's pre-training plus NCG fine-tuning method was run for a *lot* longer

Results (cont.)

- **PT + NCG** = pre-trained initialization with non-linear CG optimizer
- **RAND+HF** = random initialization with our Hessian-free method
- **PT + HF** = pre-trained initialization with our Hessian-free method
- * indicates an ℓ_2 prior was used

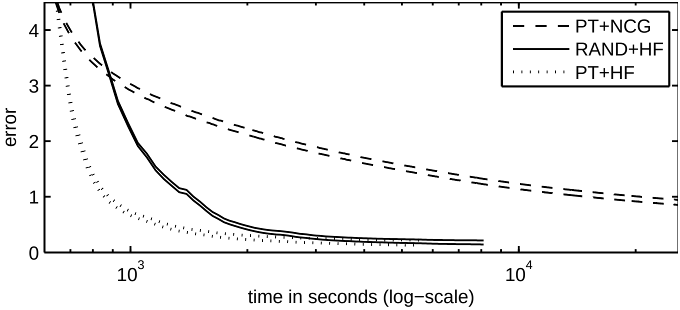




	PT + NCG	RAND+HF	PT + HF	NO EARLY STOP
CURVES	0.74, 0.82	0.11, 0.20	0.10, 0.21	0.1
MNIST	2.31, 2.72	1.64, 2.78	1.63, 2.46	1.4
MNIST*	2.07, 2.61	1.75, 2.55	1.60, 2.28	
FACES	-, 124	55.4, 139	-,-	12.9!
FACES*	-,-	60.6, 122	-,-	

Our HF method is practical

- error on the CURVES task versus GPU time:



Thank you for your attention