A new formulation for deep neural net optimisation

Miguel Á. Carreira-Perpiñán and **Weiran Wang**

Electrical Engineering and Computer Science

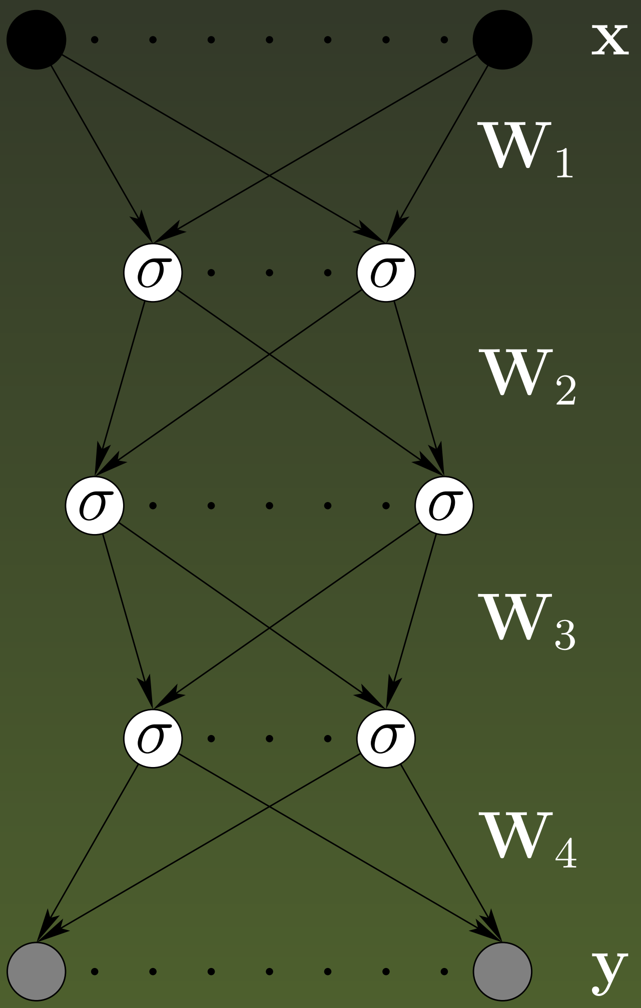
University of California, Merced

`http://eecs.ucmerced.edu`

Training deep neural nets: difficulties

A deep neural net is a deeply nested mapping from inputs to outputs.

Net with $K = 3$ hidden layers



Deep nets are difficult to optimise:

- Large problems:
  - Many weights: can't fully benefit from second-order methods.
  - Many data points: costly to compute exact gradients.
- Vanishing gradients caused by squashing nonlinearities σ: can get stuck.
- Ill-conditioning with multiple layers: lower layers' weights have less influence than higher layers' weights.

These difficulties worsen as the number of hidden layers K increases.

Training deep neural nets: existing work

- Gradient computed using backprop.
- Standard optimisation algorithms:

  gradient descent $\xrightarrow[\text{minibatches, CG, Hessian-free/autodiff, preconditioning. . .}]{\text{NCG, L-BFGS, GN/LM, quasi-Newton. . .}}$ Newton's method

- State-of-the-art: no clear consensus:
  - Carefully tuned stochastic gradient descent usually best, particularly with large problems (but hard to parallelise).
  - Heavily engineered large-scale methods (L-BFGS, Hessian free, etc.) using minibatches can do well too.
- Plus tricks, heavily dependent on architecture/dataset: rescaling weights; fan-in rules; initialisation; etc.
- Getting a method to work best (or to work at all) requires much expert user intervention. Learning rates, minibatch size, etc.
- Training takes very long, most methods take tiny steps.

Training deep neural nets: recent progress

- Large computers and GPUs.
- Good initialisation strategies (followed by fine tuning):
  - RBM pretraining (Hinton et al. 2006)
  - greedy layerwise training (Bengio et al. 2007)

  Again, these heavily depend on the architecture/dataset, require know-how and careful parameter tuning, and do not always work anyway.

In summary:

- Training deep nets remains an art.

  Hard to replicate results from others.
- Long run times even with GPUs and parallel processing.
- Model selection is even slower.

The method of auxiliary coordinates (MAC)

The nested problem

Consider a regression problem of mapping inputs $\mathbf{x}$ to outputs $\mathbf{y}$:

$$E_1(\mathbf{W}) = \sum_{n=1}^{N} \left\| \mathbf{y}_n - \mathbf{f}(\mathbf{x}_n; \mathbf{W}) \right\|^2$$

$$\mathbf{f}(\mathbf{x}; \mathbf{W}) = \mathbf{f}_{K+1}(\dots \mathbf{f}_2(\mathbf{f}_1(\mathbf{x}; \mathbf{W}_1); \mathbf{W}_2) \dots; \mathbf{W}_{K+1})$$

$$\mathbf{f}_k(\mathbf{x}; \mathbf{W}_k) = \sigma(\mathbf{W}_k \mathbf{x}), \ k = 1, \dots, K+1 \qquad \text{(including biases)}$$

The basic issue we focus on is the deep nesting of the mapping $\mathbf{f}$. This causes ill-conditioning and makes most methods take tiny steps, slowly zigzagging down a curved valley.

Also applicable to other loss functions, fully or sparsely connected layers each with a different number of hidden units, with weights shared across layers, and with regularization terms on the $\mathbf{f}_k$.

The method of auxiliary coordinates (MAC) (cont.)

The equality-constrained problem

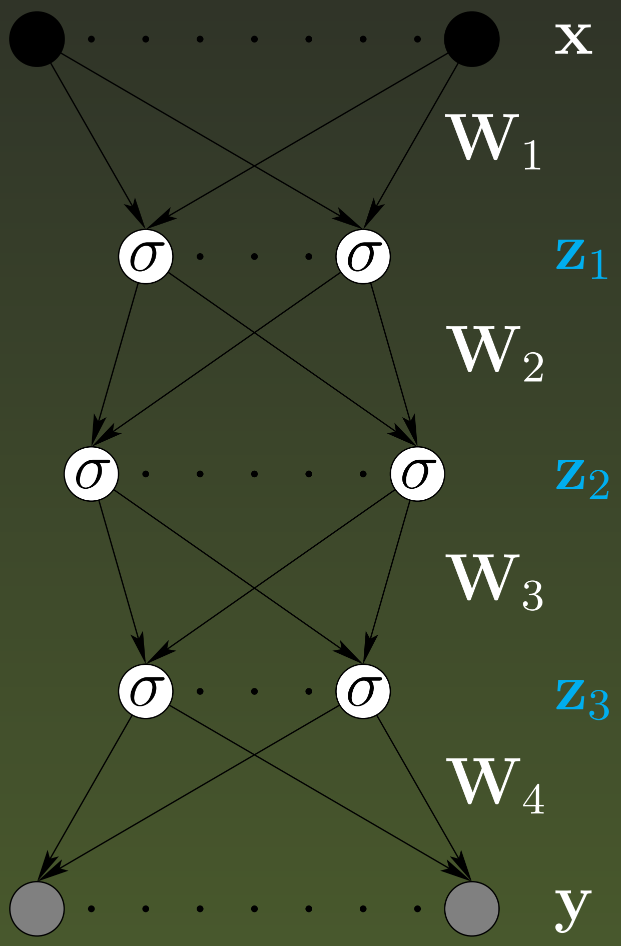
We introduce one auxiliary variable per data point and per hidden unit:

$$E(\mathbf{W},\mathbf{Z}) = \sum_{n=1}^{N} \|\mathbf{y}_n - \mathbf{f}_{K+1}(\mathbf{z}_{K,n};\mathbf{W}_{K+1})\|^2 \text{ s.t.}$$

$$\left.\begin{cases} \mathbf{z}_{K,n} = \mathbf{f}_K(\mathbf{z}_{K-1,n};\mathbf{W}_K) \\ \dots \\ \mathbf{z}_{1,n} = \mathbf{f}_1(\mathbf{x}_n;\mathbf{W}_1) \end{cases}\right\} n = 1,\dots,N.$$

- Equivalent to the nested problem, but in an augmented space without nesting. Shortcuts across $(\mathbf{W},\mathbf{Z})$ space rather than crawling along $\mathbf{W}$'s valley.
- Each term (objective & constraints) involves only a small subset of parameters.
- Partially decouples many variables: efficient optimization, trivial to parallelise unlike SGD

Net with $K = 3$ hidden layers



$\mathbf{z}_{k,n}$ = "coordinates" of $\mathbf{x}_n$ in intermediate feature space k.

The method of auxiliary coordinates (MAC) (cont.)

MAC offers a framework for many different optimisation approaches.

MAC with quadratic-penalty (QP) optimization

We optimize the following over $(\mathbf{W}, \mathbf{Z})$ for fixed $\mu > 0$ and drive $\mu \to \infty$:

$$E(\mathbf{W}, \mathbf{Z}; \mu) = \sum_{n=1}^{N} \left\| \mathbf{y}_n - \mathbf{f}_{K+1}(\mathbf{z}_{K,n}; \mathbf{W}_{K+1}) \right\|^2 + \frac{\mu}{2} \sum_{n=1}^{N} \sum_{k=1}^{K} \left\| \mathbf{z}_{k,n} - \mathbf{f}_k(\mathbf{z}_{k-1,n}; \mathbf{W}_k) \right\|^2.$$

- Continuous path $(\mathbf{W}^*(\mu), \mathbf{Z}^*(\mu))$ converges to a minimum of the constrained (and nested) problem, under some mild assumptions.

  In practice, we follow this path loosely.
- $E(\mathbf{W}, \mathbf{Z}; \mu)$ breaks the functional dependences in the nested mapping $\mathbf{f}$ and unfolds them over layers.
- Every squared term involves only a shallow mapping; all variables $(\mathbf{W}, \mathbf{Z})$ are equally scaled; simpler derivatives (no backprop).

Optimisation of MAC/QP

Alternating optimization over $\mathbf{W}$ and $\mathbf{Z}$:

$\mathbf{W}$-step least-squares regression separately for each weight vector of each hidden unit of the entire net.

Fit $\mathbf{W}_k$ to $\{(\mathbf{z}_{k-1,n}, \mathbf{z}_{k,n})\}_{n=1}^{N}$.

$\mathbf{Z}$-step nonlinear opt. separately for each data point's $\mathbf{Z}_n$, $n = 1, \dots, N$.

$$\min_{\mathbf{z}} \|\mathbf{y} - \mathbf{f}_{K+1}(\mathbf{z}_K)\|^2 + \frac{\mu}{2}\left(\cdots + \|\mathbf{z}_1 - \mathbf{f}_1(\mathbf{x})\|^2\right).$$

We solve each step with a Gauss-Newton approach with backtracking l.s. (exact step in 1–2 iterations).

- Other variations, e.g.:
  - layerwise (with guaranteed convergence)
  - no need to introduce auxiliary variables in each layer
  - etc.
- Each step operates over very large blocks of variables, so large error decrease in each iteration, unlike the tiny decreases achieved in the nested function.

Optimisation of MAC/QP (cont.)

MAC/QP makes a lot of progress initially but eventually slows down:

- Alternating optimisation has relatively slow convergence.
- For large μ, the quadratic-penalty method introduces ill-conditioning.

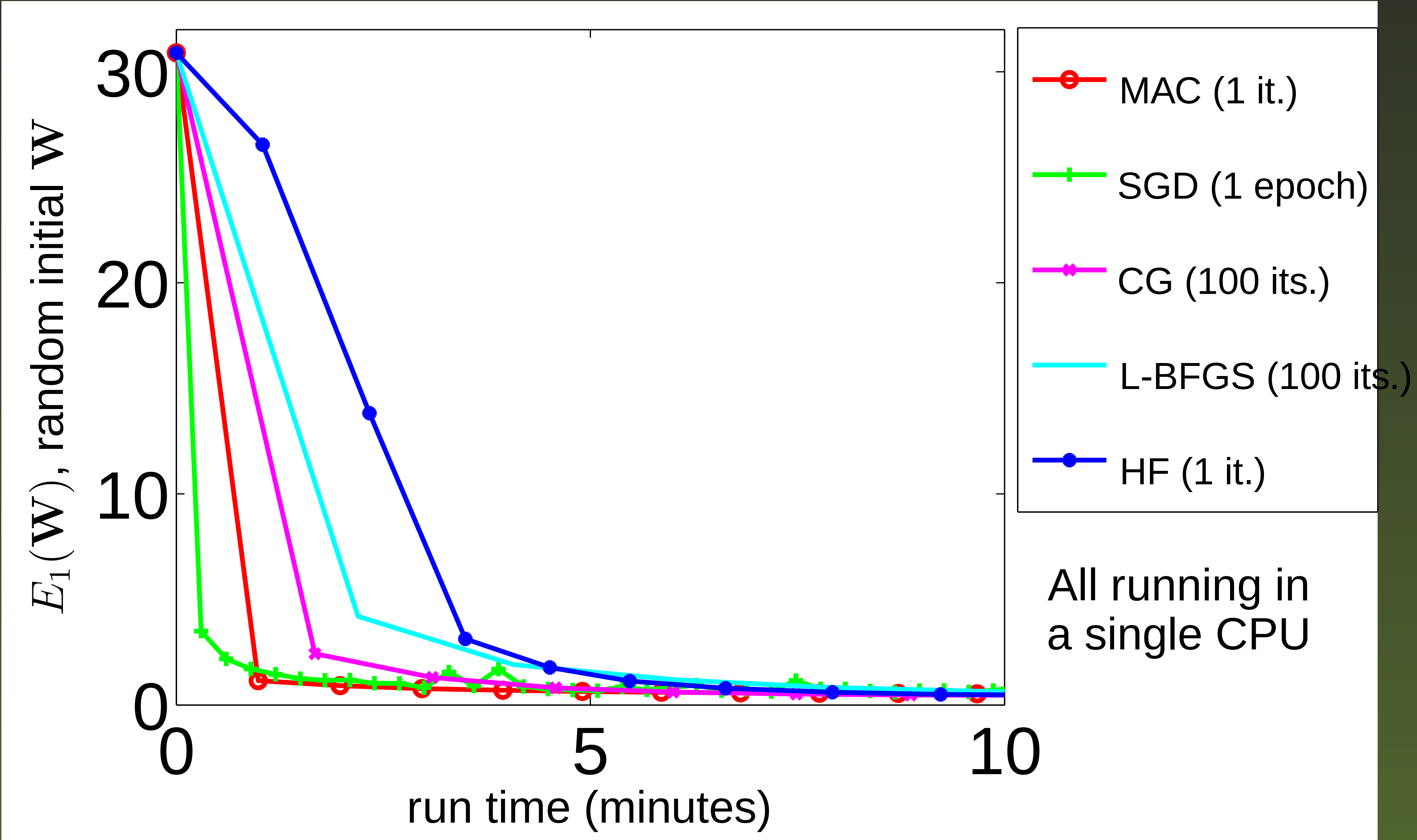
A sensible strategy is to use MAC/QP to achieve a pretty good solution pretty fast and then, if needed, switch to a method with faster convergence on the nested problem.

The postprocessing step:

- Achieves feasibility, eliminates the auxiliary coordinates and provably reduces the nested error.
- We exit for finite μ and simply satisfy the constraints by forcing $\mathbf{Z}_k = \mathbf{f}_k(\mathbf{Z}_{k-1}; \mathbf{W}_k)$, $k = 1, \dots, K$ (forward propagation), and keep all the weights the same except for the last layer, where we set $\mathbf{W}_{K+1}$ by fitting $\mathbf{f}_{K+1}$ to the dataset $(\mathbf{f}_k(\dots(\mathbf{f}_1(\mathbf{X}))), \mathbf{Y})$.
- Fast, and often causes a large error reduction.

MAC/QP experiments with autoencoders (USPS dataset)

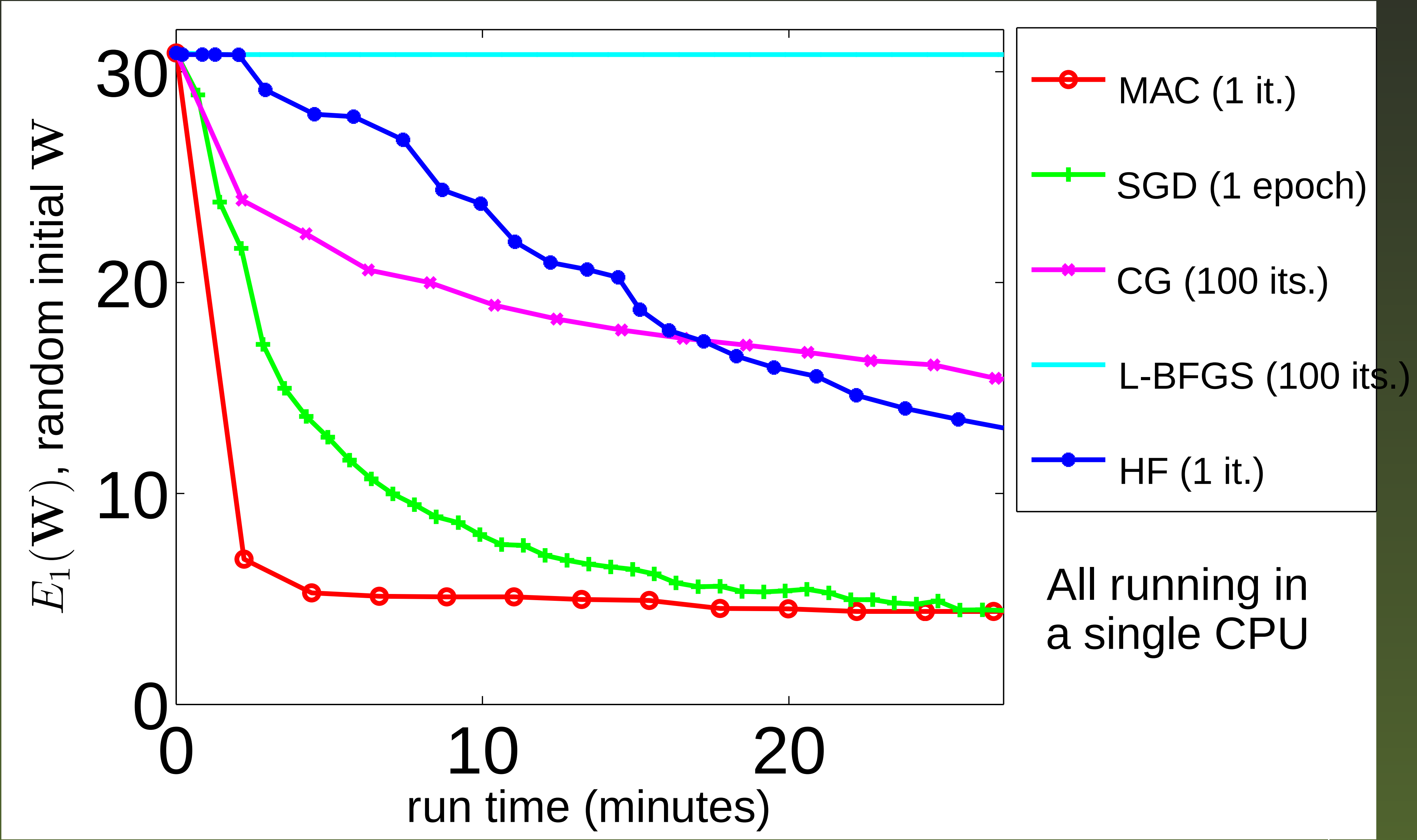
$K = 1$ layer with $H = 180$ hidden sigmoidal units



All running in a single CPU

MAC/QP experiments with autoencoders (cont.)

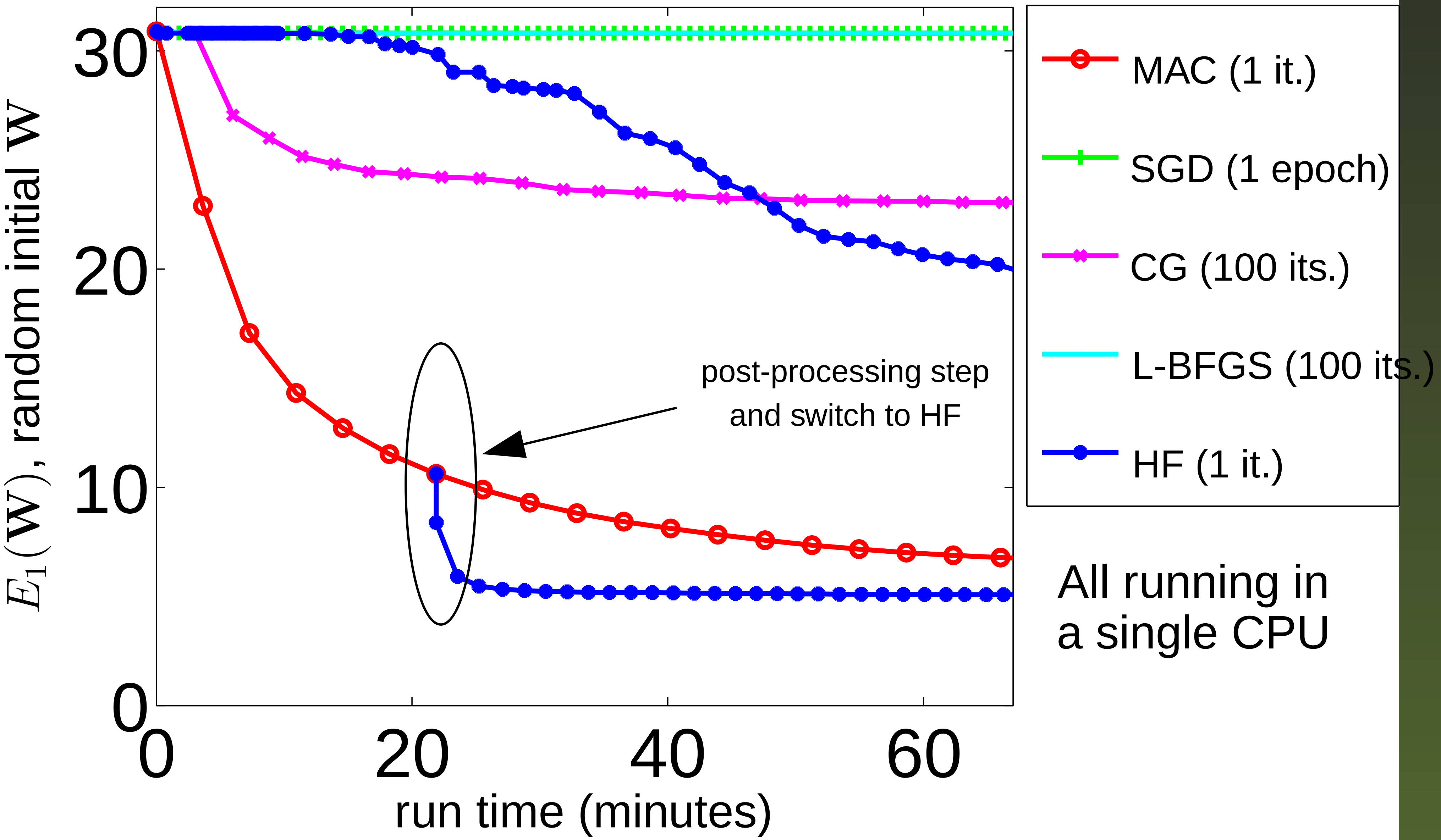
$K = 5$ layers each with $H = 100$ hidden sigmoidal units



All running in a single CPU

MAC/QP experiments with autoencoders (cont.)

$K = 9$ layers each with $H = 80$ hidden sigmoidal units

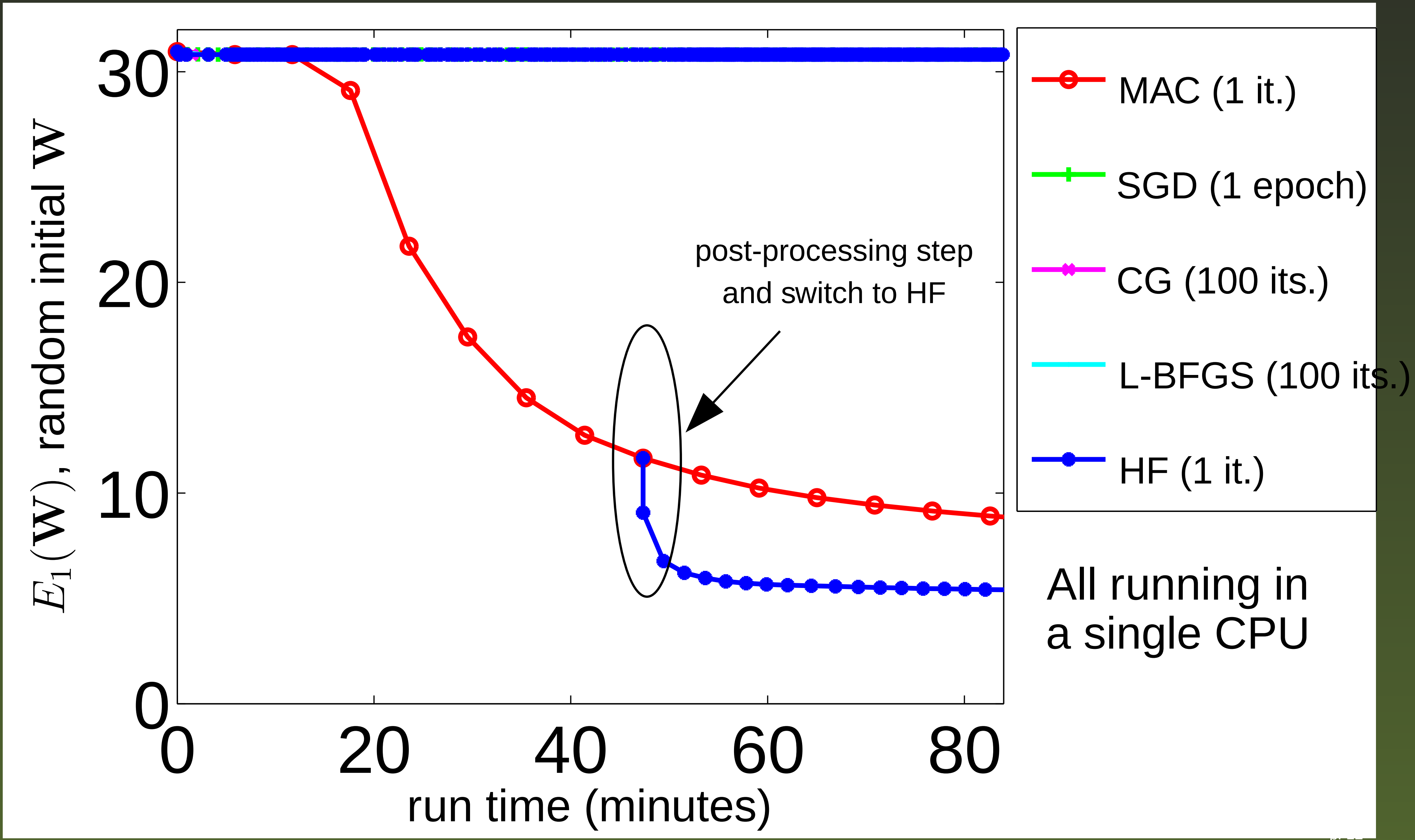


post-processing step and switch to HF

All running in a single CPU

MAC/QP experiments with autoencoders (cont.)

$K = 13$ layers each with $H = 70$ hidden sigmoidal units



All running in
a single CPU

Conclusion

- The method of auxiliary coordinates (MAC):
  - is a framework that applies to nested systems in general

    (e.g. cascade of processes for object recognition)
  - eliminates nesting, decouples many parameters
  - allows deep steps, embarrasingly parallel
  - is particularly useful to get a pretty good solution pretty fast
  - can be optimised in many ways.
- Even with a simple optimisation (quadratic-penalty with exact steps) and without parallelism or GPUs, MAC is competitive with heavily engineered state-of-the-art methods.

  Many large speedups are possible (work in progress):
  - Parallelism: lots of independent subproblems.
  - Fast, inexact $\mathbf{Z}$-step (at similar cost to backprop).
  - Stochastic updates using minibatches.
  - etc.