Learning nested systems using auxiliary coordinates

Miguel Á. Carreira-Perpiñán

Electrical Engineering and Computer Science

University of California, Merced

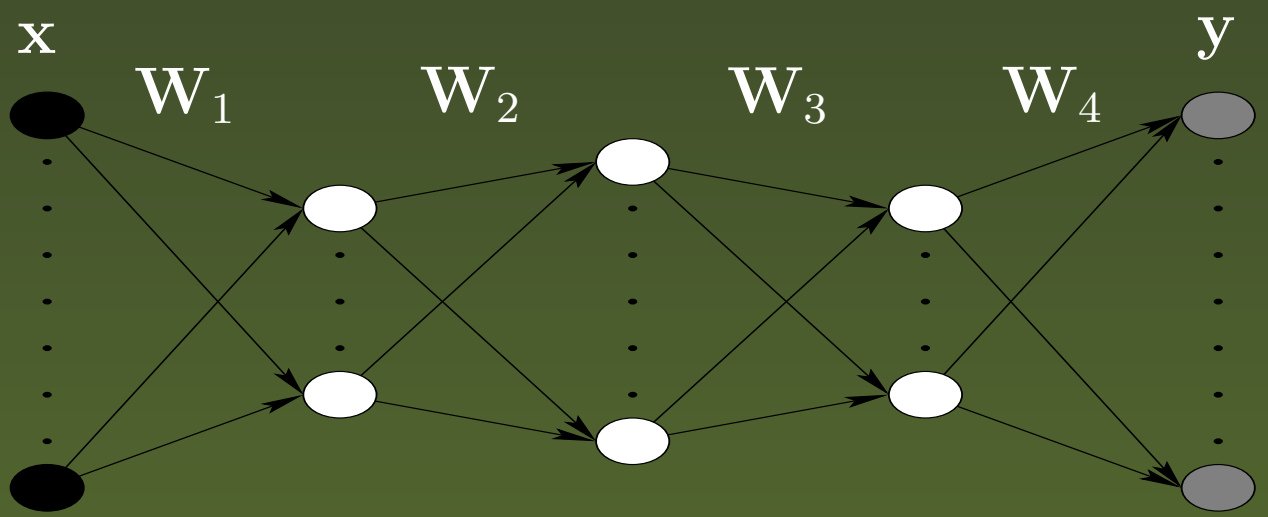
http://eecs.ucmerced.edu

work with **Weiran Wang**

Nested (hierarchical) systems: examples

Common in computer vision, speech processing, machine learning. . .

- Object recognition pipeline:
  pixels $\rightarrow$ SIFT/HoG $\rightarrow$ $\begin{matrix}\text{k-means}\\ \text{sparse coding}\end{matrix}$ $\rightarrow$ pooling $\rightarrow$ classifier $\rightarrow$ object category
- Phone classification pipeline:
  waveform $\rightarrow$ MFCC/PLP $\rightarrow$ classifier $\rightarrow$ phoneme label
- Preprocessing for regression/classification:
  image pixels $\rightarrow$ PCA/LDA $\rightarrow$ classifier $\rightarrow$ output/label
- Deep net: $\mathbf{x} \rightarrow \{\sigma(\mathbf{w}_i^T\mathbf{x} + a_i)\} \rightarrow \{\sigma(\mathbf{w}_j^T\{\sigma(\mathbf{w}_i^T\mathbf{x} + a_i)\}) + b_j\} \rightarrow \cdots \rightarrow \mathbf{y}$



Nested systems

Mathematically, they construct a (deeply) nested, parametric mapping from inputs to outputs:

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

- Each layer (processing stage) has its own trainable parameters (weights) $\mathbf{W}_k$.
- Each layer performs some (nonlinear, nondifferentiable) processing on its input, extracting ever more sophisticated features from it
  (ex.: pixels $\rightarrow$ edges $\rightarrow$ parts $\rightarrow \cdots$)
- Often inspired by biological brain processing
  (e.g. retina $\rightarrow$ LGN $\rightarrow$ V1 $\rightarrow \cdots$)
- The ideal performance is when the parameters at all layers are jointly optimised towards the overall goal (e.g. classification error). This work is about how to do this easily and efficiently.

Shallow vs deep (nested) systems

Shallow systems: 0 to 1 hidden layer between input and output.

- Often convex problem: linear function, linear SVM, LASSO, etc.

  . . . Or "forced" to be convex: $\mathbf{f}(\mathbf{x}) = \sum_{m=1}^{M} \mathbf{w}_m \phi_m(\mathbf{x})$:

  - RBF network: fix nonlinear basis functions ϕ_m (e.g. k-means), then fix linear weights $\mathbf{w}_m$.
  - SVM: basis functions (support vectors) result from a QP.
- Practically useful:
  - Linear function: robust (particularly with high-dim data, small samples).
  - Nonlinear function: very accurate if using many BFs (wide hidden layer).
- Easy to train: no local optima; no need for nonlinear optimisation

  linear system, LP/QP, eigenproblem, etc.

Shallow vs deep (nested) systems (cont.)

Deep (nested) systems: at least one hidden layer:

- Examples: deep nets; “wrapper” regression/classification; CV/speech pipelines.
- Nearly always nonconvex

  The composition of functions is nonconvex in general.
- Practically useful: powerful nonlinear function

  Depending on the number of layers and of hidden units/BFs.
- May be better than shallow systems for some problems.
- Difficult to train: local optima; requires nonlinear optimisation, or suboptimal approach.

How does one train a nested system?

Training nested systems: backpropagated gradient

- Apply the chain rule, layer by layer, to obtain a gradient wrt all the parameters.

  Ex.: $\frac{\partial}{\partial \mathbf{g}}(\mathbf{g}(\mathbf{F}(\cdot))) = \mathbf{g}'(\mathbf{F}(\cdot))$, $\frac{\partial}{\partial \mathbf{F}}(\mathbf{g}(\mathbf{F}(\cdot))) = \mathbf{g}'(\mathbf{F}(\cdot))\,\mathbf{F}'(\cdot)$.

  Then feed to nonlinear optimiser.

  Gradient descent, CG, L-BFGS, Levenberg-Marquardt, Newton, etc.

- Major breakthrough in the 80s with neural nets.

  It allowed to train multilayer perceptrons from data.

- Disadvantages:
  - requires differentiable layers

    in order to apply the chain rule
  - the gradient is cumbersome to compute, code and debug
  - requires nonlinear optimisation
  - vanishing gradients $\Rightarrow$ ill-conditioning $\Rightarrow$ slow progress even with second-order methods

    This gets worse the more layers we have.

Training nested systems: layerwise, “filter”

- Fix each layer sequentially (in some way).
- Fast and easy, but suboptimal.

  The resulting parameters are not a minimum of the joint objective function.

  Sometimes the results are not very good.
- Sometimes used to initialise the parameters and refine the model with backpropagation (“fine tuning”).

Examples:

- Deep nets:
  - Unsupervised pretraining (Hinton & Salakhutdinov 2006)
  - Supervised greedy layerwise training (Bengio et al. 2007)
- RBF networks: the centres of the first (nonlinear) layer’s basis functions are set in an unsupervised way

  k-means, random subset

Training nested systems: layerwise, “filter” (cont.)

“Filter” vs “wrapper” approaches: consider a nested mapping $\mathbf{g}(\mathbf{F}(\mathbf{x}))$ (e.g. $\mathbf{F}$ reduces dimension, $\mathbf{g}$ classifies). How to train $\mathbf{F}$ and $\mathbf{g}$?

Filter approach:

- Greedy sequential training:
  1. Train $\mathbf{F}$ (the “filter”):
     - Unsupervised: use only the input data $\{\mathbf{x}_n\}$

       PCA, k-means, etc.
     - Supervised: use the input and output data $\{(\mathbf{x}_n, \mathbf{y}_n)\}$

       LDA, sliced inverse regression, etc.
  2. Fix $\mathbf{F}$, train $\mathbf{g}$: fit a classifier with inputs $\{\mathbf{F}(\mathbf{x}_n)\}$ and labels $\{\mathbf{y}_n\}$.
- Very popular; $\mathbf{F}$ is often a fixed “preprocessing” stage.
- Works well if using a good objective function for $\mathbf{F}$.
- . . . But is still suboptimal: the preprocessing may not be the best possible for classification.

Training nested systems: layerwise, “filter” (cont.)

Wrapper approach:

- Train $\mathbf{F}$ and g jointly to minimise the classification error.
  This is what we would like to do.
- Optimal: the preprocessing is the best possible for classification.
- Even if local optima exist, initialising it from the “filter” result will give a better model.
- Rarely done in practice.
- Disadvantage: same problems as with backpropagation.
  Requires a chain rule gradient, difficult to compute, nonlinear optimisation, slow.

Training nested systems: model selection

Finally, we also have to select the best architecture:

- Number of units or basis functions in each layer of a deep net; number of filterbanks in a speech front-end processing; etc.
- Requires a combinatorial search, training models for each hyperparameter choice and picking the best

  according to a model selection criterion, cross-validation, etc.
- In practice, this is approximated using expert know-how:
  - Train only a few models, pick the best from there.
  - Fix the parameters of some layers irrespective of the rest of the pipeline.

Very costly in runtime, in effort and expertise required, and leads to suboptimal solutions.

Summary

Nested systems:

- Ubiquitous way to construct nonlinear trainable functions
- Powerful
- Intuitive
- Difficult to train:
  - Layerwise: easy but suboptimal
  - Backpropagation: optimal but slow, difficult to implement, needs differentiable layers.

The method of auxiliary coordinates (MAC)

- A general strategy to train all parameters of a nested system.
- Enjoys the benefits of layerwise training (fast, easy steps) but with optimality guarantees.
- Embarrassingly parallel iterations.
- Not an algorithm but a meta-algorithm (like EM).
- Basic idea:
  1. Turn the nested problem into a constrained optimisation problem by introducing new parameters to be optimised over (the auxiliary coordinates).
  2. Optimise the constrained problem with a penalty method.
  3. Optimise the penalty objective function with alternating optimisation.

  Result: alternate “layerwise training” steps with “coordination” steps.

The nested objective function

Consider for simplicity:

- a single hidden layer: $\mathbf{x} \to \mathbf{F}(\mathbf{x}) \to \mathbf{g}(\mathbf{F}(\mathbf{x}))$
- a least-squares regression for inputs $\{\mathbf{x}_n\}_{n=1}^N$ and outputs $\{\mathbf{y}_n\}_{n=1}^N$:

$$\min E_{\text{nested}}(\mathbf{F}, \mathbf{g}) = \frac{1}{2} \sum_{n=1}^{N} \|\mathbf{y}_n - \mathbf{g}(\mathbf{F}(\mathbf{x}_n))\|^2$$

$\mathbf{F}$, $\mathbf{g}$ have their own parameters (weights).
We want to find a local minimum of E_{nested}.

The MAC-constrained problem

Transform the problem into a constrained one in an augmented space:

$$\min E(\mathbf{F}, \mathbf{g}, \mathbf{Z}) = \frac{1}{2} \sum_{n=1}^{N} \|\mathbf{y}_n - \mathbf{g}(\mathbf{z}_n)\|^2$$

$$\text{s.t. } \mathbf{z}_n = \mathbf{F}(\mathbf{x}_n) \quad n = 1, \dots, N.$$

- For each data point, we turn the subexpression $\mathbf{F}(\mathbf{x}_n)$ into an equality constraint associated with a new parameter $\mathbf{z}_n$ (the auxiliary coordinates).

  Thus, a constrained problem with N equality constraints and new parameters $\mathbf{Z} = (\mathbf{z}_1, \dots, \mathbf{z}_N)$.
- We optimise over $(\mathbf{F}, \mathbf{g})$ and $\mathbf{Z}$ jointly.
- Equivalent to the nested problem.

The MAC quadratic-penalty function

We solve the constrained problem with the quadratic-penalty method: we minimise the following while driving the penalty parameter $\mu \to \infty$:

$$\min E_Q(\mathbf{F}, \mathbf{g}, \mathbf{Z}; \mu) = \frac{1}{2} \sum_{n=1}^{N} \|\mathbf{y}_n - \mathbf{g}(\mathbf{z}_n)\|^2 + \frac{\mu}{2} \sum_{n=1}^{N} \underbrace{\|\mathbf{z}_n - \mathbf{F}(\mathbf{x}_n)\|^2}_{\substack{\text{constraints as} \\ \text{quadratic penalties}}}$$

We can also use the augmented Lagrangian method instead:

$$\min E_{\mathcal{L}}(\mathbf{F}, \mathbf{g}, \mathbf{Z}, \mathbf{\Lambda}; \mu) = \frac{1}{2} \sum_{n=1}^{N} \|\mathbf{y}_n - \mathbf{g}(\mathbf{z}_n)\|^2 + \sum_{n=1}^{N} \boldsymbol{\lambda}_n^T (\mathbf{z}_n - \mathbf{F}(\mathbf{x}_n)) + \frac{\mu}{2} \sum_{n=1}^{N} \|\mathbf{z}_n - \mathbf{F}(\mathbf{x}_n)\|^2$$

For simplicity, we focus on the quadratic-penalty method.

What have we achieved?

- Net effect: unfold the nested objective into shallow additive terms connected by the auxiliary coordinates:

$$E_{\text{nested}}(\mathbf{F}, \mathbf{g}) = \frac{1}{2} \sum_{n=1}^{N} \|\mathbf{y}_n - \mathbf{g}(\mathbf{F}(\mathbf{x}_n))\|^2 \Longrightarrow$$

$$E_Q(\mathbf{F}, \mathbf{g}, \mathbf{Z}; \mu) = \frac{1}{2} \sum_{n=1}^{N} \|\mathbf{y}_n - \mathbf{g}(\mathbf{z}_n)\|^2 + \frac{\mu}{2} \sum_{n=1}^{N} \|\mathbf{z}_n - \mathbf{F}(\mathbf{x}_n)\|^2$$

- All terms equally scaled, but uncoupled.

  Vanishing gradients less problematic.

  Derivatives required are simpler: no backpropagated gradients, sometimes no gradients at all.
- Optimising E_{nested} follows a convoluted trajectory in $(\mathbf{F}, \mathbf{g})$ space.
- Optimising E_Q can take shortcuts by jumping across $\mathbf{Z}$ space.

  This corresponds to letting the layers mismatch during the optimisation.

Alternating optimisation of the MAC/QP objective

$(\mathbf{F}, \mathbf{g})$ step, for $\mathbf{Z}$ fixed:

$$\min_{\mathbf{g}} \frac{1}{2} \sum_{n=1}^{N} \|\mathbf{y}_n - \mathbf{g}(\mathbf{z}_n)\|^2 \qquad \min_{\mathbf{F}} \frac{1}{2} \sum_{n=1}^{N} \|\mathbf{z}_n - \mathbf{F}(\mathbf{x}_n)\|^2$$

- Layerwise training: each layer is trained independently (not sequentially):
  - fit $\mathbf{g}$ to $\{(\mathbf{z}_n, \mathbf{y}_n)\}_{n=1}^N$ (gradient needed: $\mathbf{g}'(\cdot)$)
  - fit $\mathbf{F}$ to $\{(\mathbf{x}_n, \mathbf{z}_n)\}_{n=1}^N$ (gradient needed: $\mathbf{F}'(\cdot)$)
- Usually simple fit, even convex.
- Can be done by using existing algorithms for shallow models

  linear, logistic regression, SVM, RBF network, k-means, decision tree, etc.

  Does not require backpropagated gradients.

Alternating optimisation of the MAC/QP objective (cont.)

$\mathbf{Z}$ step, for $(\mathbf{F}, \mathbf{g})$ fixed:

$$\min_{\mathbf{z}_n} \frac{1}{2} \|\mathbf{y}_n - \mathbf{g}(\mathbf{z}_n)\|^2 + \frac{\mu}{2} \|\mathbf{z}_n - \mathbf{F}(\mathbf{x}_n)\|^2 \qquad n = 1, \dots, N$$

- The auxiliary coordinates are trained independently for each point.
  N small problems (of size $|\mathbf{z}_n|$) instead of one large problem (of size $N\,|\mathbf{z}_n|$).
- They "coordinate" the layers.
- Has the form of a proximal operator.
  $\min_{\mathbf{z}} f(\mathbf{z}) + \frac{\mu}{2} \|\mathbf{z} - \mathbf{u}\|^2$
- The solution has a geometric flavour ("projection").
- Often closed-form (depending on the model).

Alternating optimisation of the MAC/QP objective (cont.)

MAC/QP is a "coordination-minimisation" (CM) algorithm:

- M step: minimise (train) layers
- C step: coordinate layers

The coordination step is crucial: it ensures we converge to a minimum of the nested function (which layerwise training by itself does not do).

MAC/QP is different from pure alternating optimisation over layers:

$$\min E_{\text{nested}}(\mathbf{F}, \mathbf{g}) = \frac{1}{2} \sum_{n=1}^{N} \|\mathbf{y}_n - \mathbf{g}(\mathbf{F}(\mathbf{x}_n))\|^2$$

- Over $\mathbf{g}$ for fixed $\mathbf{F}$: fit $\mathbf{g}$ to $\{(\mathbf{F}(\mathbf{x}_n), \mathbf{y}_n)\}_{n=1}^{N}$ (needs $\mathbf{g}'(\cdot)$)
- Over $\mathbf{F}$ for fixed $\mathbf{g}$: needs backprop. gradients over $\mathbf{F}$ ($\mathbf{g}'(\mathbf{F}(\cdot))\,\mathbf{F}'(\cdot)$)

Pure alternating optimisation $\neq$ "layerwise training".

MAC in general (K layers)

The nested objective function:

$$E_{\text{nested}}(\mathbf{W}) = \frac{1}{2}\sum_{n=1}^{N} \|\mathbf{y}_n - \mathbf{f}(\mathbf{x}_n;\mathbf{W})\|^2 \qquad \mathbf{f}(\mathbf{x};\mathbf{W}) = \mathbf{f}_{K+1}(\dots \mathbf{f}_2(\mathbf{f}_1(\mathbf{x};\mathbf{W}_1);\mathbf{W}_2)\dots;\mathbf{W}_{K+1})$$

The MAC-constrained problem:

$$E(\mathbf{W},\mathbf{Z}) = \frac{1}{2}\sum_{n=1}^{N} \left\|\mathbf{y}_n - \mathbf{f}_{K+1}(\mathbf{z}_{K,n};\mathbf{W}_{K+1})\right\|^2 \text{ s.t. } \left\{\begin{array}{l}\mathbf{z}_{K,n} = \mathbf{f}_K(\mathbf{z}_{K-1,n};\mathbf{W}_K)\\ \dots \\ \mathbf{z}_{1,n} = \mathbf{f}_1(\mathbf{x}_n;\mathbf{W}_1)\end{array}\right\} n = 1,\dots,N.$$

The MAC quadratic-penalty function:

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

Alternating optimisation:

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

MAC also applies with various loss functions, full/sparse layer connectivity, constraints, etc.

MAC in general (K layers): convergence guarantees

Theorem 1: the nested problem and the MAC-constrained problem are equivalent in the sense that their minimisers, maximisers and saddle points are in a one-to-one correspondence. The KKT conditions for both problems are equivalent.

Theorem 2: given a positive increasing sequence $(\mu_k) \to \infty$, a nonnegative sequence $(\tau_k) \to 0$, and a starting point $(\mathbf{W}^0, \mathbf{Z}^0)$, suppose the QP method finds an approximate minimizer $(\mathbf{W}^k, \mathbf{Z}^k)$ of $E_Q(\mathbf{W}^k, \mathbf{Z}^k; \mu_k)$ that satisfies $\left\|\nabla_{\mathbf{W},\mathbf{Z}} E_Q(\mathbf{W}^k, \mathbf{Z}^k; \mu_k)\right\| \leq \tau_k$ for $k = 1, 2, \ldots$ Then, $\lim_{k\to\infty} (\mathbf{W}^k, \mathbf{Z}^k) = (\mathbf{W}^*, \mathbf{Z}^*)$, which is a KKT point for the nested problem, and its Lagrange multiplier vector has elements $\boldsymbol{\lambda}_n^* = \lim_{k\to\infty} -\mu_k \left(\mathbf{Z}_n^k - \mathbf{F}(\mathbf{Z}_n^k, \mathbf{W}^k; \mathbf{x}_n)\right),\ n = 1, \ldots, N$.

That is, MAC/QP defines a continuous path $(\mathbf{W}^*(\mu), \mathbf{Z}^*(\mu))$ that converges to a local minimum of the constrained problem and thus to a local minimum of the nested problem. In practice, we follow this path loosely.

MAC in general (K layers): the design pattern

How to train your system using auxiliary coordinates:

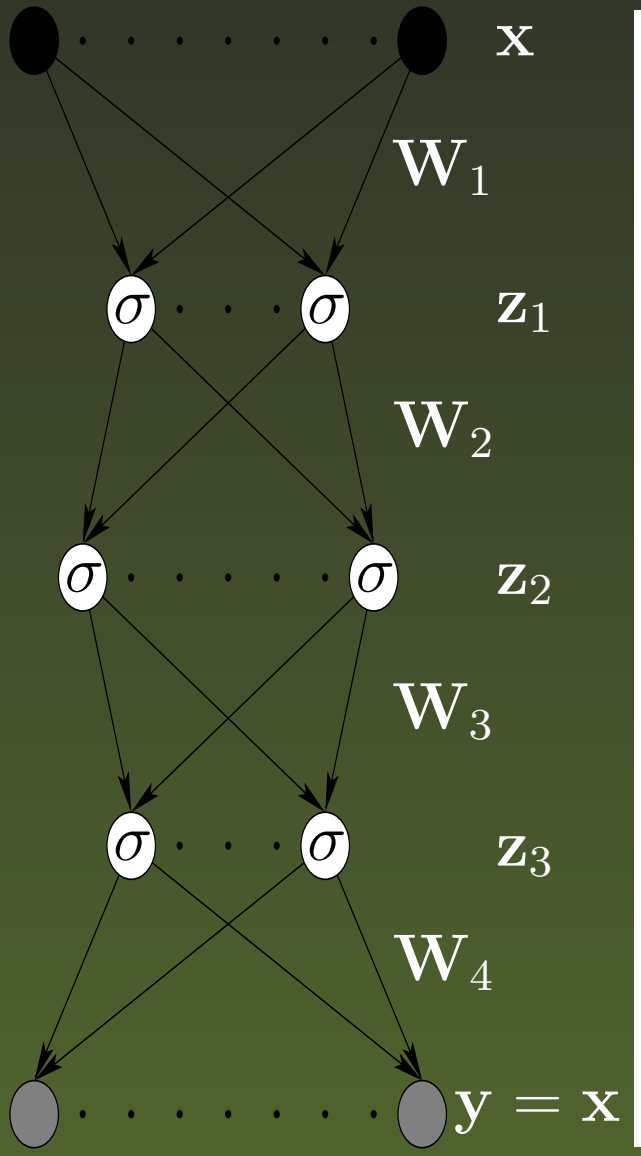
1. Write your nested objective function $E_{\text{nested}}(\mathbf{W})$.
2. Identify subexpressions and turn them into auxiliary coordinates with equality constraints.
3. Apply quadratic-penalty or augmented Lagrangian method.
4. Do alternating optimisation:
   - $\mathbf{W}$ step: reuse a single-layer training algorithm, typically
   - $\mathbf{Z}$ step: needs to be solved specially for your problem

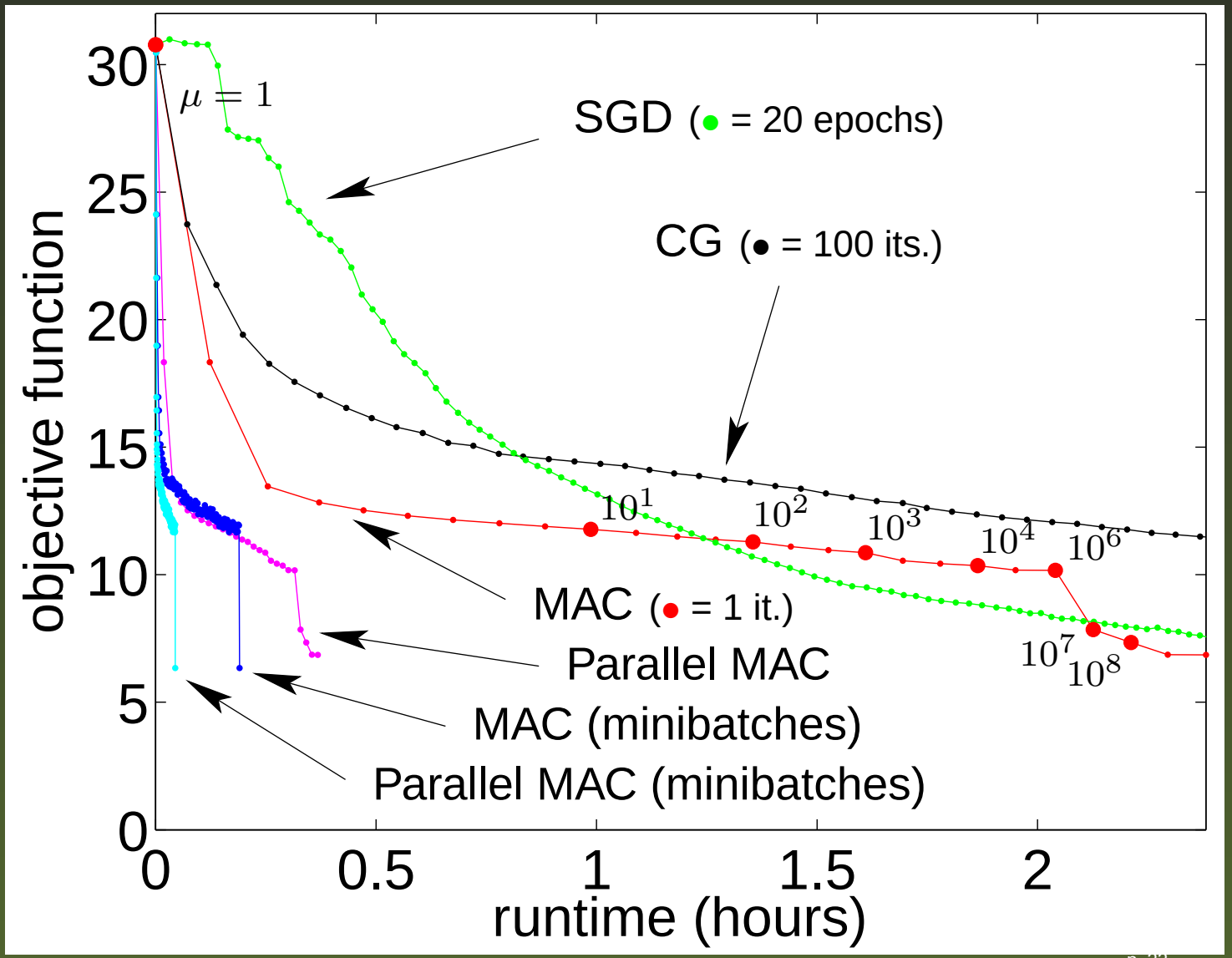
     proximal operator; for many important cases closed-form or simple to optimise.

Similar to deriving an EM algorithm: define your probability model, write the log-likelihood objective function, identify hidden variables, write the complete-data log-likelihood, obtain E and M steps, solve them.

Experiments: deep sigmoidal autoencoder

USPS handwritten digits, 256–300–100–20–100–300–256 autoencoder ($K = 5$ logistic layers), auxiliary coordinates at each hidden layer, random initial weights. $\mathbf{W}$ and $\mathbf{Z}$ steps use Gauss-Newton.





Experiments: deep sigmoidal autoencoder (cont.)

Typical behaviour in practice:

- Very large error decrease at the beginning, causing large changes to the parameters at all layers

  unlike backpropagation-based methods.
- Eventually slows down, slow convergence

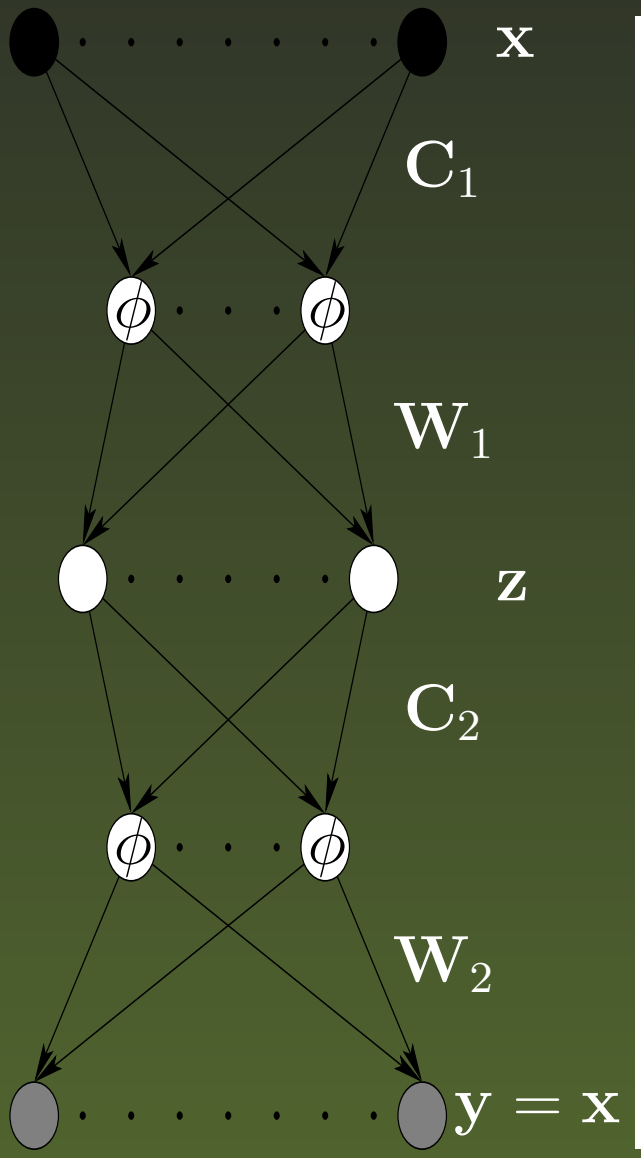
  typical of alternating optimisation algorithms.
- “Pretty good net pretty fast”.
- Competitive with state-of-the-art nonlinear optimisers, particularly with many nonlinear layers.

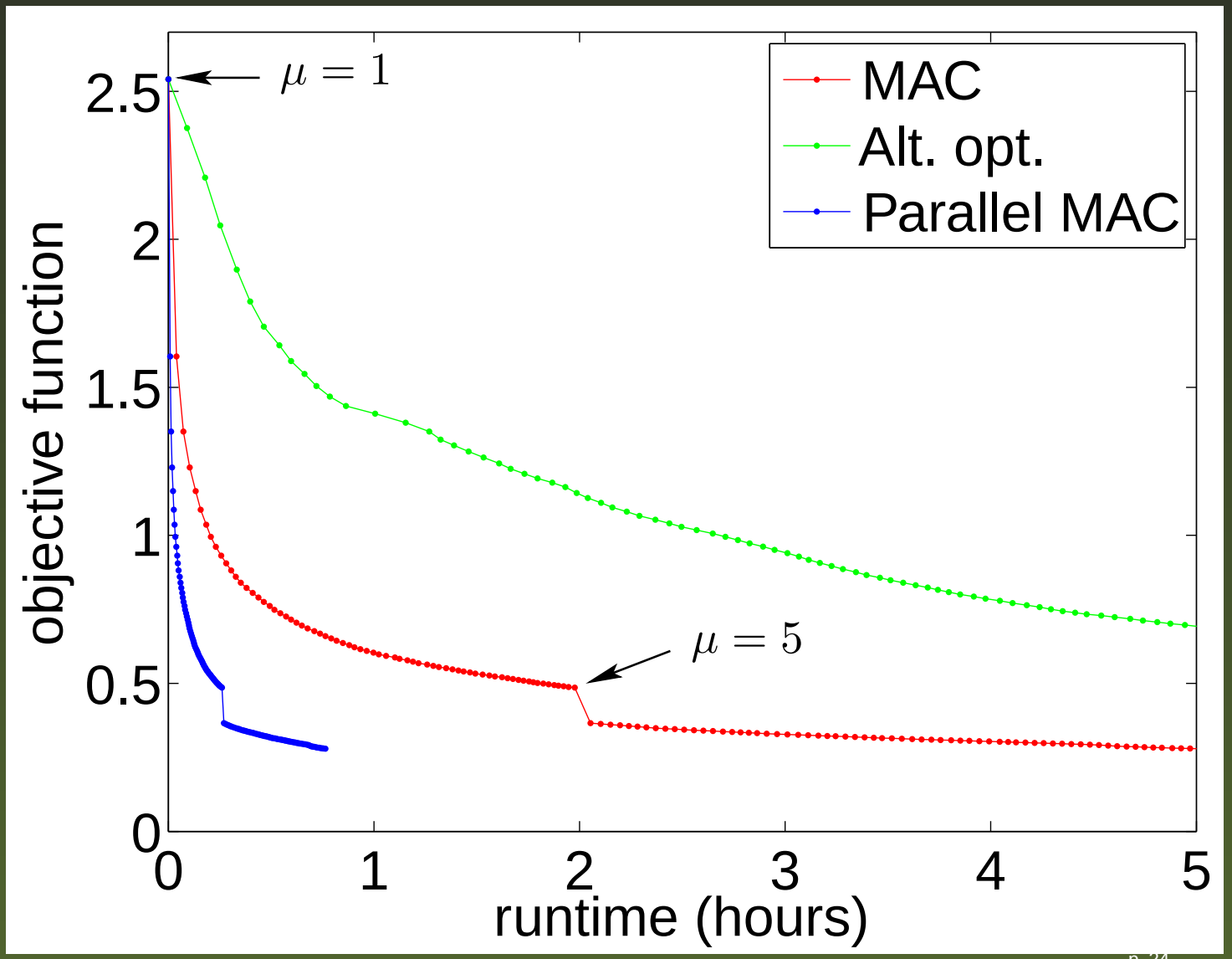
Note: the MAC iterations can be done much faster (see later):

- With better optimisation
- With parallel processing

Experiments: RBF autoencoder

COIL object images, 1024–1368–2–1368–1024 autoencoder ($K = 3$ hidden layers), auxiliary coordinates in bottleneck layer only, initial $\mathbf{Z}$. $\mathbf{W}$ step uses k-means ($\mathbf{C}_k$) + linsys ($\mathbf{W}_k$). $\mathbf{Z}$ step uses Gauss-Newton.





Practicalities

Schedule of the penalty parameter μ:

- Theory: $\mu \to \infty$ for convergence.
- Practice: stop with finite μ.
- Keeping $\mu = 1$ gives quite good results.
- How fast to increase μ depends on the problem.
- We increase μ when the error in a validation set increases.

The postprocessing step:

- After the algorithm stops, we satisfy the constraints by:
  - Setting $\mathbf{z}_{kn} = \mathbf{f}_k(\mathbf{z}_{k-1,n}; \mathbf{W}_k)$, $k = 1, \ldots, K$, $n = 1, \ldots, N$
    That is, project on the feasible set by forward propagation..
  - Keeping 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 $(\mathbf{f}_K(\ldots(\mathbf{f}_1(\mathbf{X}))), \mathbf{Y})$. This provably reduces the error.

Practicalities (cont.)

Choice of optimisation algorithm for the steps:

- $\mathbf{W}$ step: typically, reuse existing single-layer algorithm

  Linear: linsys; SVM: QP; RBF net: k-means + linsys; etc.

  Large datasets: use stochastic updates w/ data minibatches (SGD).
- $\mathbf{Z}$ step: often closed-form, otherwise:
  - Small number of parameters in $\mathbf{z}_n$: Gauss-Newton

    The GN matrix is always positive definite because of the $\|\mathbf{z} - \cdot\|^2$ terms.
  - Large number of parameters in $\mathbf{z}_n$: CG, Newton-CG, L-BFGS...

Standard optimisation and linear algebra techniques apply:

- Inexact steps.
- Warm starts.
- Caching factorisations.

Cleverly used, they can make the $\mathbf{W}$ and $\mathbf{Z}$ steps very fast.

Practicalities (cont.)

Defining the auxiliary coordinates:

- With neural nets, we can introduce them before the nonlinearity: $z = \sigma(\mathbf{w}^T\mathbf{x} + b)$ vs $z = \mathbf{w}^T\mathbf{x} + b$ (linear $\mathbf{W}$ step).
- No need to introduce auxiliary coordinates at each layer.

  Spectrum between fully nested (no auxiliary coordinates, pure backpropagation) and fully unnested (auxiliary coordinates at each layer, no chain rule).
- Can even redefine $\mathbf{Z}$ over the optimisation.

The best strategy will depend on the dataset dimensionality and size, and on the model.

Related work: dimensionality reduction

- Given high-dim data $\mathbf{y}_1, \ldots, \mathbf{y}_N \in \mathbb{R}^D$, we want to project to latent coordinates $\mathbf{z}_1, \ldots, \mathbf{z}_N \in \mathbb{R}^L$ with $L \ll D$.
- Optimise reconstruction error over the reconstruction mapping $\mathbf{f}: \mathbf{z} \to \mathbf{y}$ and the latent coordinates $\mathbf{Z}$:

$$\min_{\mathbf{f},\mathbf{Z}} \sum_{n=1}^{N} \|\mathbf{y}_n - \mathbf{f}(\mathbf{z}_n)\|^2$$

where $\mathbf{f}$ can be linear (least-squares factor analysis; Young 1941, Whittle 1952...) or nonlinear: spline (Leblanc & Tibshirani 1994), single-layer neural net (Tan & Mavrovouniotis 1995), RBF net (Smola et al. 2001), kernel regression (Meinicke et al. 2005), Gaussian process (GPLVM; Lawrence 2005), etc.

- Problem: nearby $\mathbf{z}$s map to nearby $\mathbf{y}$s, but not necessarily vice versa. This can produce a poor representation in latent space.
- This can be solved by introducing the "inverse" mapping $\mathbf{F}: \mathbf{y} \to \mathbf{z}$.

Related work: dimensionality reduction (cont.)

- "Dimensionality reduction by unsupervised regression" (Carreira-Perpinan & Lu, 2008, 2010):

$$\min_{\mathbf{f},\mathbf{F},\mathbf{Z}} \sum_{n=1}^{N} \|\mathbf{y}_n - \mathbf{f}(\mathbf{z}_n)\|^2 + \|\mathbf{z}_n - \mathbf{F}(\mathbf{y}_n)\|^2$$

  - Learns both mappings: reconstruction $\mathbf{f}$ and projection $\mathbf{F}$, together with the latent coordinates $\mathbf{Z}$.
  - Now nearby $\mathbf{y}$'s also map to nearby $\mathbf{x}$'s

    $\mathbf{f}$ and $\mathbf{F}$ become approximate inverses of each other on the data manifold.
  - Special case of MAC/QP to solve the autoencoder problem:

$$\min_{\mathbf{f},\mathbf{F}} \sum_{n=1}^{N} \|\mathbf{y}_n - \mathbf{f}(\mathbf{F}(\mathbf{x}_n))\|^2$$

    but with $\mu = 1$ (so biased solution).

Related work: learning good internal representations

Updating weights and hidden unit activations in neural nets:

- Idea originates in 1980s, focused on (single-layer) neural nets.
  Grossman et al. 1988, Saad & Marom 1990, Krogh et al. 1990, Rohwer 1990, Olshausen & Field 1996, Ma et al. 1997, Castillo et al. 2006, Ranzato et al. 2007, Kavukcuoglu et al. 2008, Baldi & Sadowski 2012, etc.
- Learning good internal representations was seen as important as learning good weights.
- Desirable activation values were explicitly generated in different ways: ad-hoc objective function (e.g. to make them sparse), sampling, etc.
- The weights and activations were updated in alternating fashion.
- The generation of activation values wasn't directly related to the nested objective function, so the algorithm doesn't converge to a minimum of the latter.

Related work: ADMM and EM

Alternating direction method of multipliers (ADMM):

- Optimisation algorithm for constrained problems with separability.
- Alternates steps on the augmented Lagrangian over the primal and dual variables.
- Often used in consensus problems:

$$\min_{\mathbf{x}} \sum_{n=1}^{N} f_n(\mathbf{x}) \Leftrightarrow \min_{\mathbf{x}_1,\ldots,\mathbf{x}_N,\mathbf{z}} \sum_{n=1}^{N} f_n(\mathbf{x}_n) \text{ s.t. } \mathbf{x}_n = \mathbf{z},\ n = 1,\ldots,N \Leftrightarrow$$

$$\min \mathcal{L}(\mathbf{X}, \mathbf{z}, \mathbf{\Lambda}) = \sum_{n=1}^{N} \left(f_n(\mathbf{x}_n) + \boldsymbol{\lambda}_n^T(\mathbf{x}_n - \mathbf{z}) + \frac{\mu}{2} \|\mathbf{x}_n - \mathbf{z}\|^2 \right)$$

The aug. Lag. $\mathcal{L}$ is minimised alternatingly over $\mathbf{X}$, $\mathbf{z}$ and $\mathbf{\Lambda}$.

- Can be applied to the MAC-constrained problem as well.

Related work: ADMM and EM (cont.)

Expectation-maximisation (EM) algorithm:

- Trains probability models by maximum likelihood.
- Can be seen as:
  - bound optimisation
  - alternating optimisation over the posterior probabilities (E step) and model parameters (M step).

    The posterior probabilities “coordinate” the individual models.

ADMM, EM and MAC/QP have the following properties:

- The specific algorithm is very easy to develop in many cases; intuitive steps where simple models are fit
- Convergence guarantees
- Large initial steps, eventually slower convergence
- Innate parallelism

Model selection “on the fly”

- Model selection criteria (AIC, BIC, MDL, etc.) separate over layers:

  $$\overline{E}(\mathbf{W}) = E_{\text{nested}}(\mathbf{W}) + C(\mathbf{W}) = \text{nested-error} + \text{model-cost}$$

  $$C(\mathbf{W}) \propto \text{total \# parameters} = |\mathbf{W}_1| + \cdots + |\mathbf{W}_K|$$

- Traditionally, a grid search (with M values per layer) means testing an exponential number of nested models, M^K.
- In MAC, the cost $C(\mathbf{W})$ separates over layers in the $\mathbf{W}$ step, so each layer can do model selection independently of the others, testing a polynomial number of shallow models, MK.

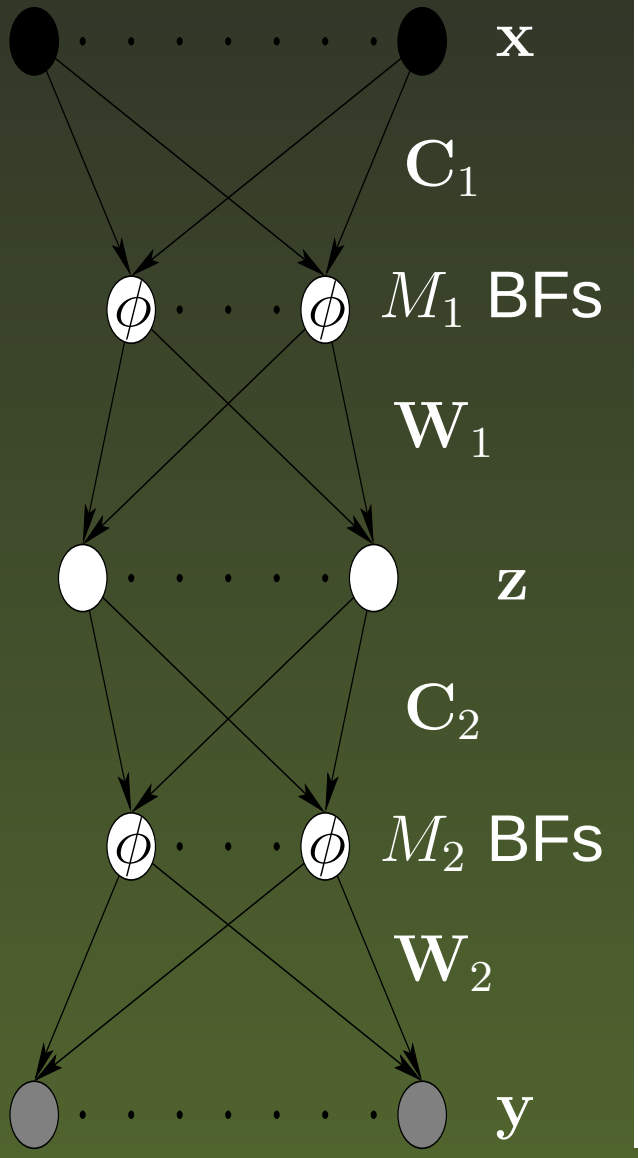
  This still provably minimises the overall objective $\overline{E}(\mathbf{W})$.
- Instead of a criterion, we can do cross-validation in each layer.
- In practice, no need to do model selection at each $\mathbf{W}$ step.

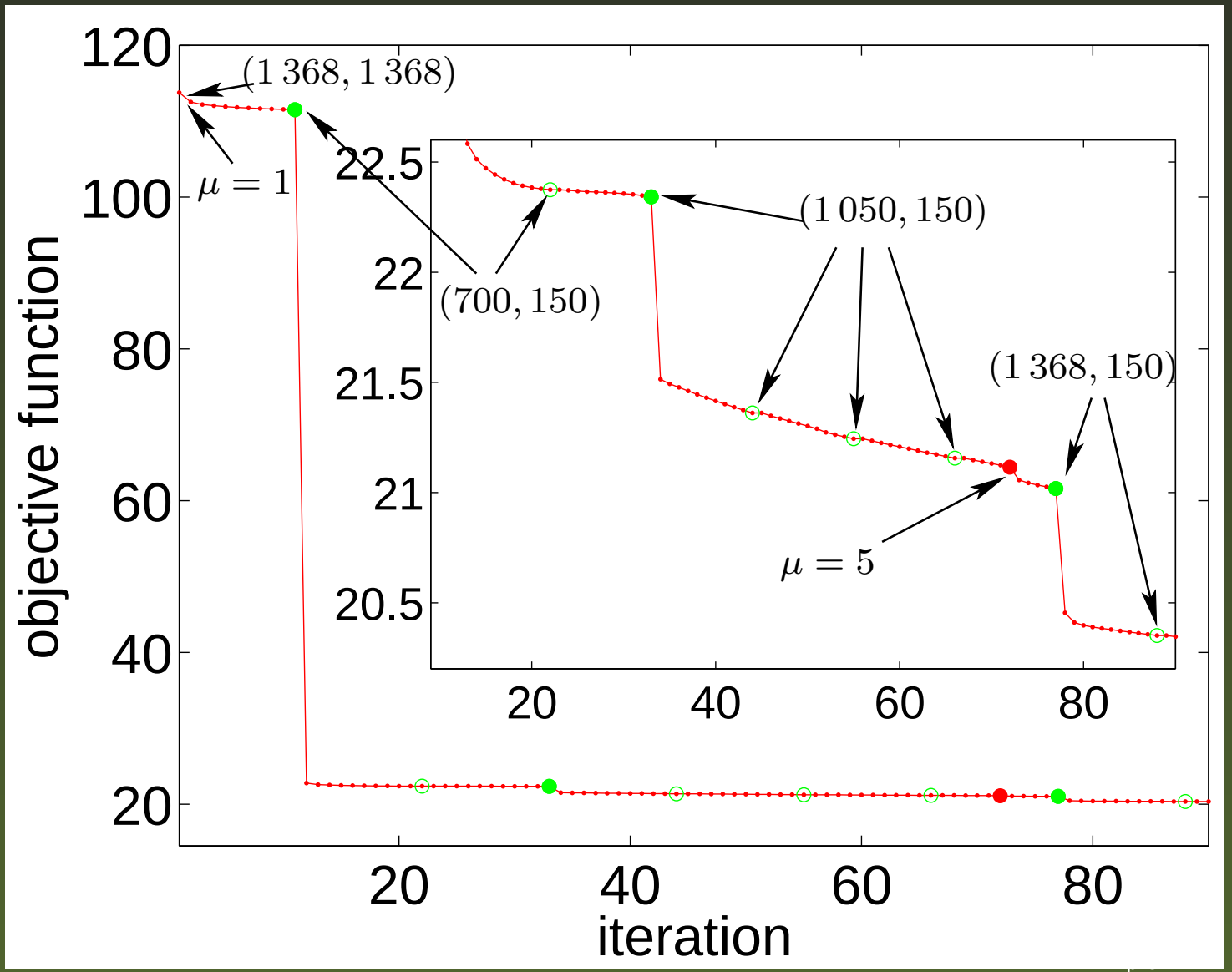
  The algorithm usually settles in a region of good architectures early during the optimisation, with small and infrequent changes thereafter.

MAC searches over the parameter space of the architecture and over the space of architectures itself, in polynomial time, iteratively.

Experiments: RBF autoencoder (model selection)

COIL object images, 1024–M_1–2–M_2–1024 autoencoder ($K = 3$ hidden layers), AIC model selection over (M_1, M_2) in $\{150, \ldots, 1368\}$ (50 values $\Rightarrow 50^2$ possible models).





Distributed optimisation

MAC/QP is embarrassingly parallel:

- $\mathbf{W}$ step:
  - all layers separate ($K+1$ independent subproblems)
  - often, all units within each layer separate $\Rightarrow$
    one independent subproblem for each unit's input weight vector
  - the model selection steps also separate

    test each model independently
- $\mathbf{Z}$ step: all points separate (N independent subproblems)

Enormous potential for parallel implementation:

- Unlike other machine learning or optimisation algorithms, where subproblems are not independent (e.g. SGD).
- Suitable for large-scale data.

Distributed optimisation: example

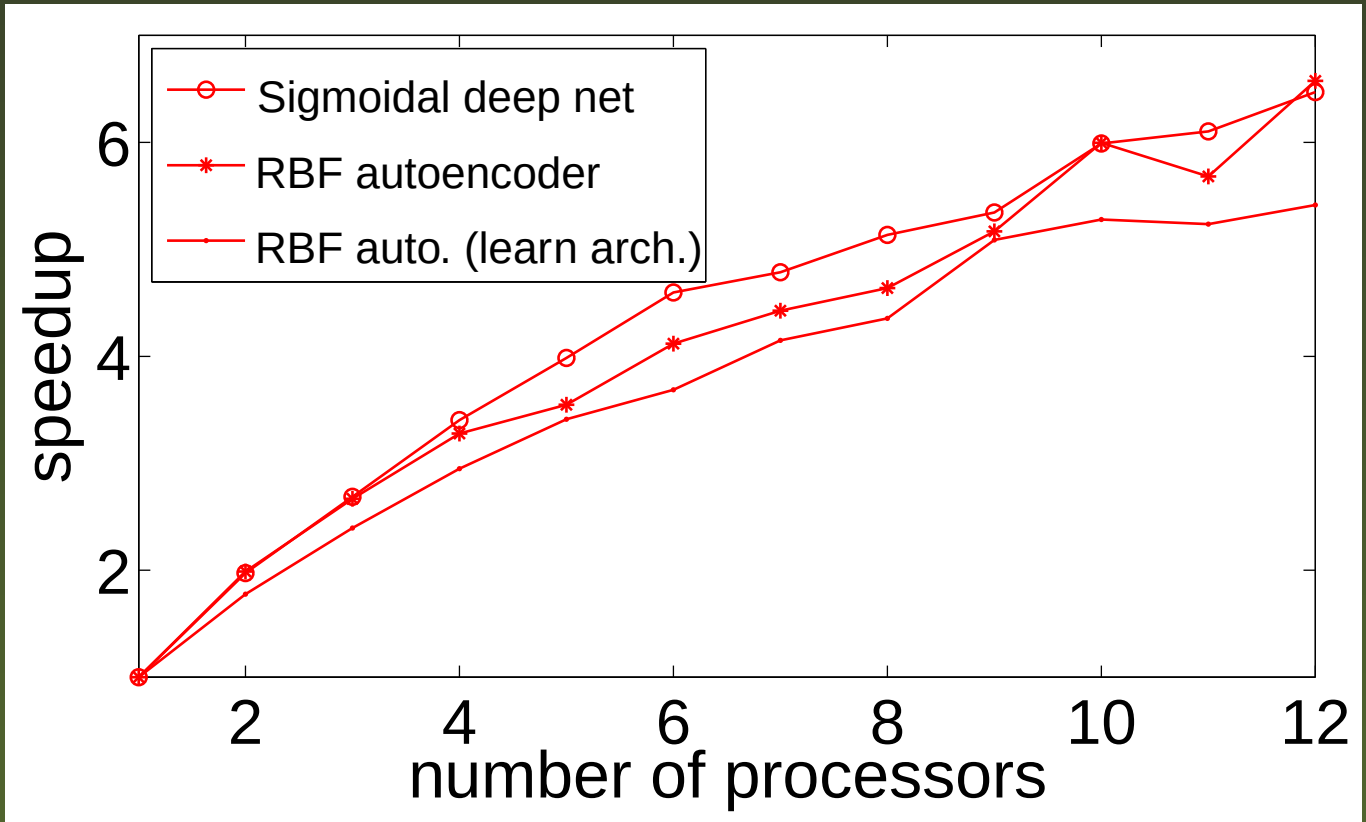
- Shared-memory multiprocessor model using the Matlab Parallel Processing Toolbox: change `for` to `parfor` in the $\mathbf{W}$ and $\mathbf{Z}$ loops.

  So Matlab sends each iteration to a different server.

- Near-linear speedups as a function of the number of processors

  even though the Matlab Parallel Processing Toolbox is quite inefficient.

- Other options: MPI on a distributed architecture, etc.



Conclusion: the method of auxiliary coordinates (MAC)

- Jointly optimises a nested function over all its parameters.
- Restructures the nested problem into a sequence of iterations with independent subproblems; a coordination-minimisation algorithm:
  - M step: minimise (train) layers
  - C step: coordinate layers
- Advantages:
  - Easy to develop, reuses existing algorithms for shallow models
  - Convergent
  - Efficient
  - Embarrassingly parallel
  - Can work with nondifferentiable or discrete layers
  - Can do model selection “on the fly”
- Widely applicable in machine learning, computer vision, speech, NLP, etc.

A long-term goal

Develop a software tool where:

- A non-expert user builds a nested system by connecting individual modules from a library, LEGO-like:

  linear, SVM, RBF net, logistic regression, feature selector. . .

- The tool automatically:
  - Selects the best way to apply MAC

    choice of auxiliary coordinates, choice of optimisation algorithms, etc.
  - Reuses training algorithms from a library
  - Maps the overall algorithm to a target distributed architecture
  - Generates runtime code.

Papers about this work

Main reference (`http://faculty.ucmerced.edu/mcarreira-perpinan`):

- Miguel Á. Carreira-Perpiñán and Weiran Wang: “Distributed optimization of deeply nested systems”. `http://arxiv.org/abs/1212.5921`.

Extensions or related work:

- Weiran Wang and Miguel Á. Carreira-Perpiñán: “The role of dimensionality reduction in classification”. `http://arxiv.org/abs/XXXX.XXXX`.
- Weiran Wang and Miguel Á. Carreira-Perpiñán: “Nonlinear low-dimensional regression using auxiliary coordinates”. AISTATS 2012.
- Miguel Á. Carreira-Perpiñán and Zhengdong Lu: “Parametric dimensionality reduction by unsupervised regression”. CVPR 2010.
- Miguel Á. Carreira-Perpiñán and Zhengdong Lu: “Dimensionality reduction by unsupervised regression”. CVPR 2008.

Work partially supported by NSF CAREER award IIS–0754089 and a Google Faculty Research Award.