Distributed Optimization of Deeply Nested Systems

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

Electrical Engineering and Computer Science

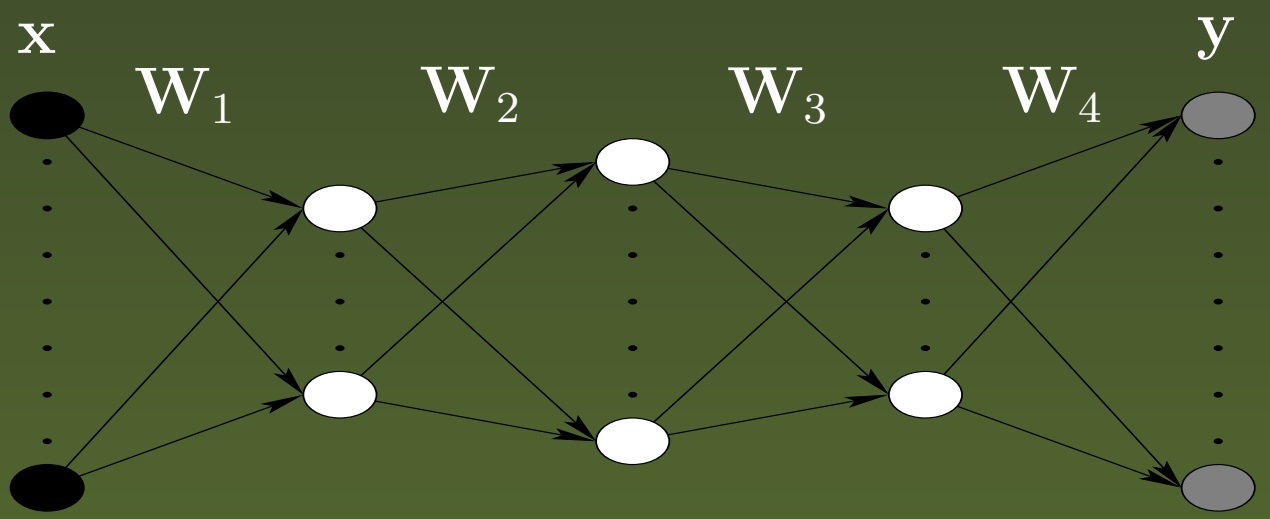
University of California, Merced

http://eecs.ucmerced.edu

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 $\rightarrow \cdots$)
- Nearly always nonconvex; difficult to train.
  The composition of functions is nonconvex in general.
- 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.

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.

- Disadvantages:
  - requires differentiable layers
  - 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
  - difficult to parallelise.

Training nested systems: layerwise, “filter”

- Fix each layer sequentially from the input to the output.
  Fast and easy, but suboptimal.
- Sometimes used to initialise the parameters and refine the model with backpropagation (“fine tuning”).

Examples:

- Pretraining approaches for deep nets, RBF networks, etc.
  (Hinton & Salakhutdinov 2006, Bengio et al. 2007)
- Filter (vs wrapper) approach: consider a nested mapping $\mathbf{g}(\mathbf{F}(\mathbf{x}))$ (e.g. $\mathbf{F}$ reduces dimension, $\mathbf{g}$ classifies):
  1. First train $\mathbf{F}$ (the “filter”), e.g. with PCA on the input data $\{\mathbf{x}_n\}$.
  2. Then, fit a classifier $\mathbf{g}$ with inputs $\{\mathbf{F}(\mathbf{x}_n)\}$ and labels $\{\mathbf{y}_n\}$.

  $\mathbf{F}$ can be seen as a fixed “preprocessing” or “feature extraction” stage. But it may not be the best possible preprocessing for classification.
  Wrapper approach: optimise $\mathbf{F}$ and $\mathbf{g}$ jointly to minimise the classification error. More difficult, less frequently done.

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.

The method of auxiliary coordinates (MAC)

- A general strategy to train all parameters of a nested system. Not an algorithm but a meta-algorithm (like EM).
- Enjoys the benefits of layerwise training (fast, easy steps that reuse existing algorithms for shallow systems) but with optimality guarantees.
- Embarrassingly parallel iterations.
- 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 penalised 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 (ADMM) 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:

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

- Layerwise training: each layer is trained independently (not sequentially):
  - Fit $\mathbf{F}$ to $\{(\mathbf{x}_n, \mathbf{z}_n)\}_{n=1}^N$ (gradient needed: $\mathbf{F}'(\cdot)$).
  - Fit $\mathbf{g}$ to $\{(\mathbf{z}_n, \mathbf{y}_n)\}_{n=1}^N$ (gradient needed: $\mathbf{g}'(\cdot)$).

  This looks like a filter approach with intermediate "features" $\{\mathbf{z}_n\}_{n=1}^N$.
- 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 $|\mathbf{z}|$) instead of one large problem (of size $N\,|\mathbf{z}|$).
- They "coordinate" the layers.
- The $\mathbf{Z}$ step 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 in general

MAC applies very generally:

- K layers: $\mathbf{f}(\mathbf{x}; \mathbf{W}) = \mathbf{f}_{K+1}(\dots \mathbf{f}_2(\mathbf{f}_1(\mathbf{x}; \mathbf{W}_1); \mathbf{W}_2) \dots; \mathbf{W}_{K+1})$.
- Various loss functions, full/sparse layer connectivity, constraints...
- The resulting optimisation algorithm depends on:
  - the type of layers $\mathbf{f}_k$ used

    linear, logistic regression, SVM, RBF network, decision tree...
  - how/where we introduce the auxiliary coordinates

    at no/some/all layers; before/after the nonlinearity; can redefine $\mathbf{Z}$ during the optimisation
  - how we optimise each step

    choice of method; inexact steps, warm start, caching factorisations, stochastic updates...

Convergence guarantee: under mild assumptions, 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: the design pattern

How to train your system using auxiliary coordinates:

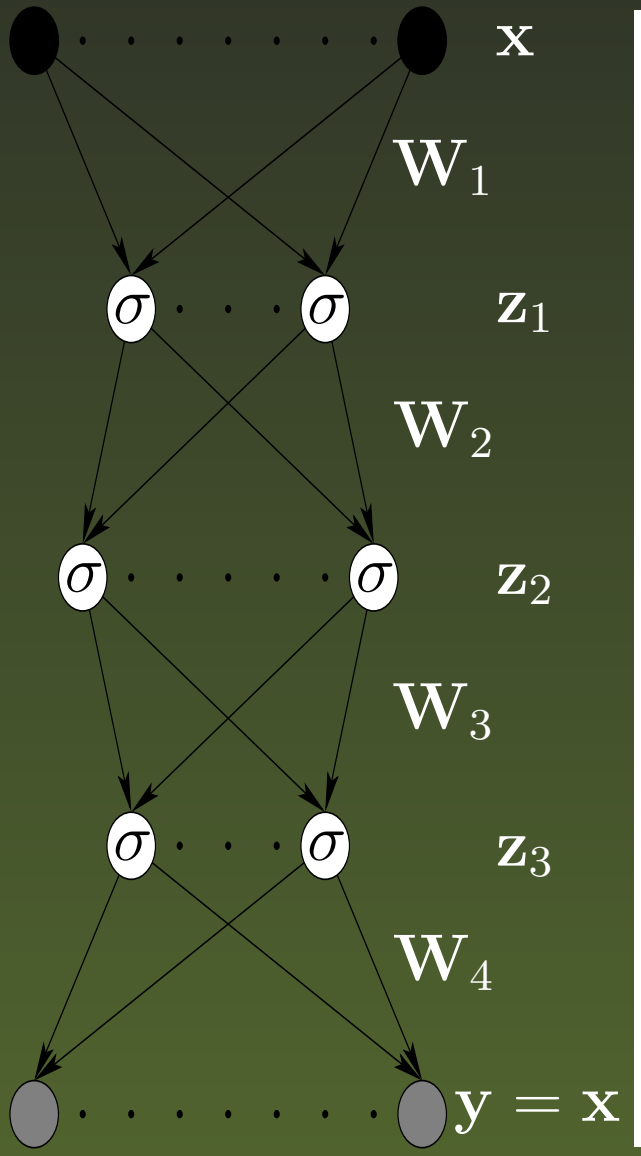
1. Write your nested objective function $E_{\text{nested}}(\mathbf{W})$.
2. Identify subexpressions and turn them into auxiliary coordinates with equality constraints.
3. Apply quadratic penalty or augmented Lagrangian method.
4. Do alternating optimisation:
   - $\mathbf{W}$ step: reuse a single-layer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

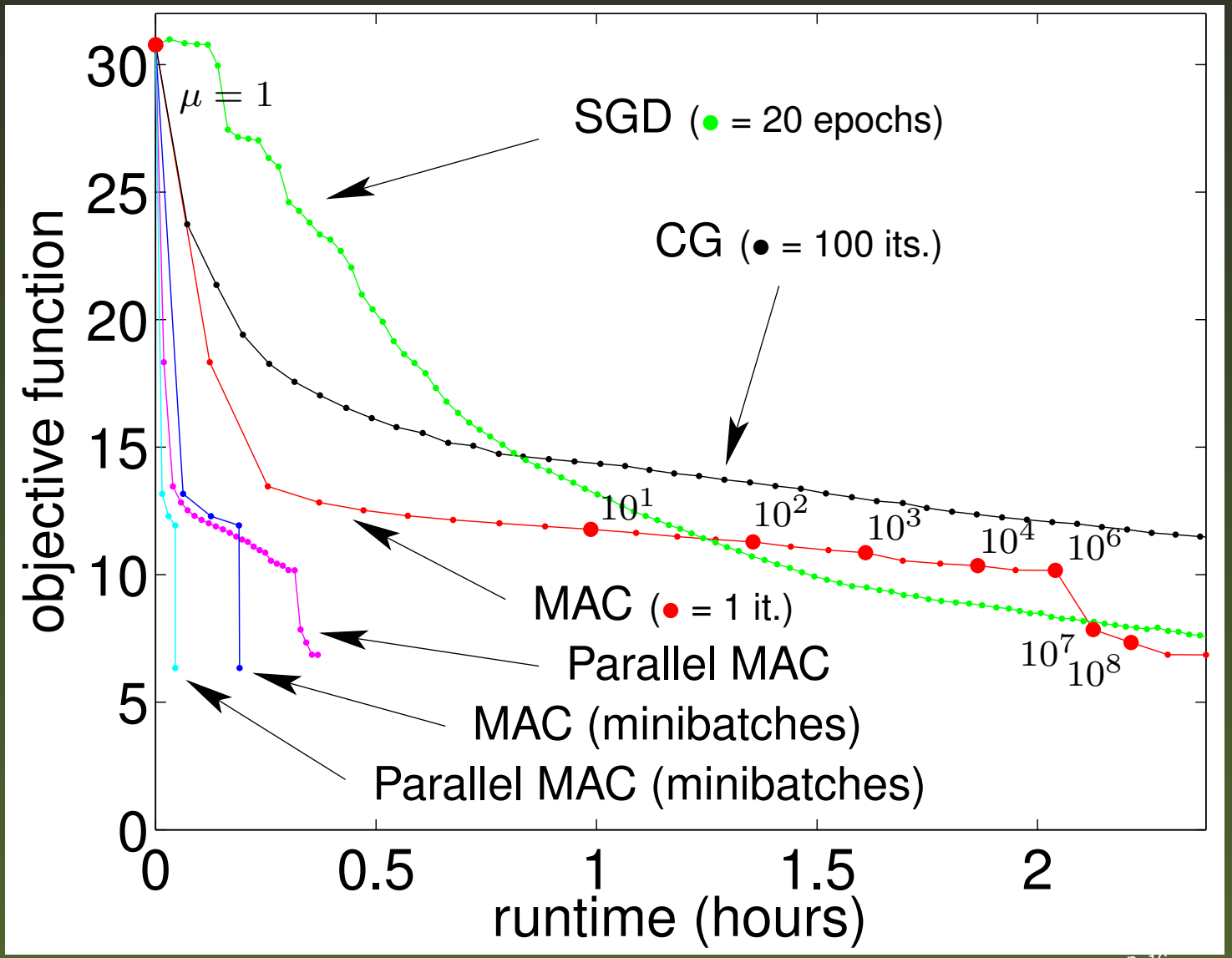
     proximal operator; for many important cases closed-form or simple to optimise.

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.

Experiment: 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.





Another example: low-dimensional SVM

W. Wang and M. Á. Carreira-Perpiñán: "The role of dimensionality reduction in classification", AAAI 2014.

"Wrapper" approach to learn a nonlinear classifier $y = \mathbf{g}(\mathbf{F}(\mathbf{x}))$ where $\mathbf{F}$ is a nonlinear mapping that reduces dimension and $\mathbf{g}$ a linear SVM:

$$\min_{\mathbf{F},\mathbf{g},\boldsymbol{\xi}} \lambda R(\mathbf{F}) + \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{n=1}^{N}\xi_n$$

$$\text{s.t. } \left\{y_n(\mathbf{w}^T\mathbf{F}(\mathbf{x}_n) + b) \geq 1 - \xi_n,\ \xi_n \geq 0\right\}_{n=1}^{N}.$$

MAC/QP results in the following iteration:

- $\mathbf{W}$ step: train, in parallel and reusing existing algorithms:
  - A nonlinear regression $\mathbf{F}$ on $\{(\mathbf{x}_n, \mathbf{z}_n)\}_{n=1}^{N}$.
  - A linear SVM classifier $\mathbf{g}$ on $\{(\mathbf{z}_n, \mathbf{y}_n)\}_{n=1}^{N}$.
- $\mathbf{Z}$ step: comes out in closed form; for each data point $n = 1, \ldots, N$, set $\mathbf{z}_n = \mathbf{F}(\mathbf{x}_n) + \gamma_n y_n \mathbf{w}$ for a certain $\gamma_n \in \mathbb{R}$.

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 M^K of nested models.
- 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 MK of shallow models.

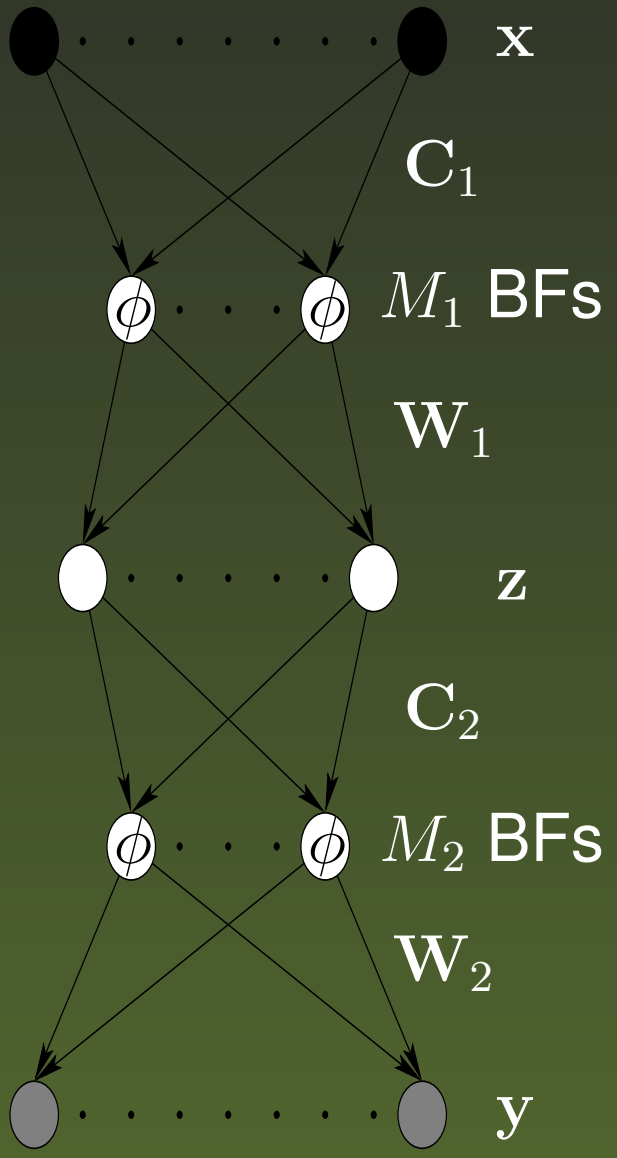
  This still provably minimises the overall objective $\overline{E}(\mathbf{W})$.
- Instead of a criterion, we can do cross-validation in each layer.
- In practice, no need to do model selection at each $\mathbf{W}$ step.

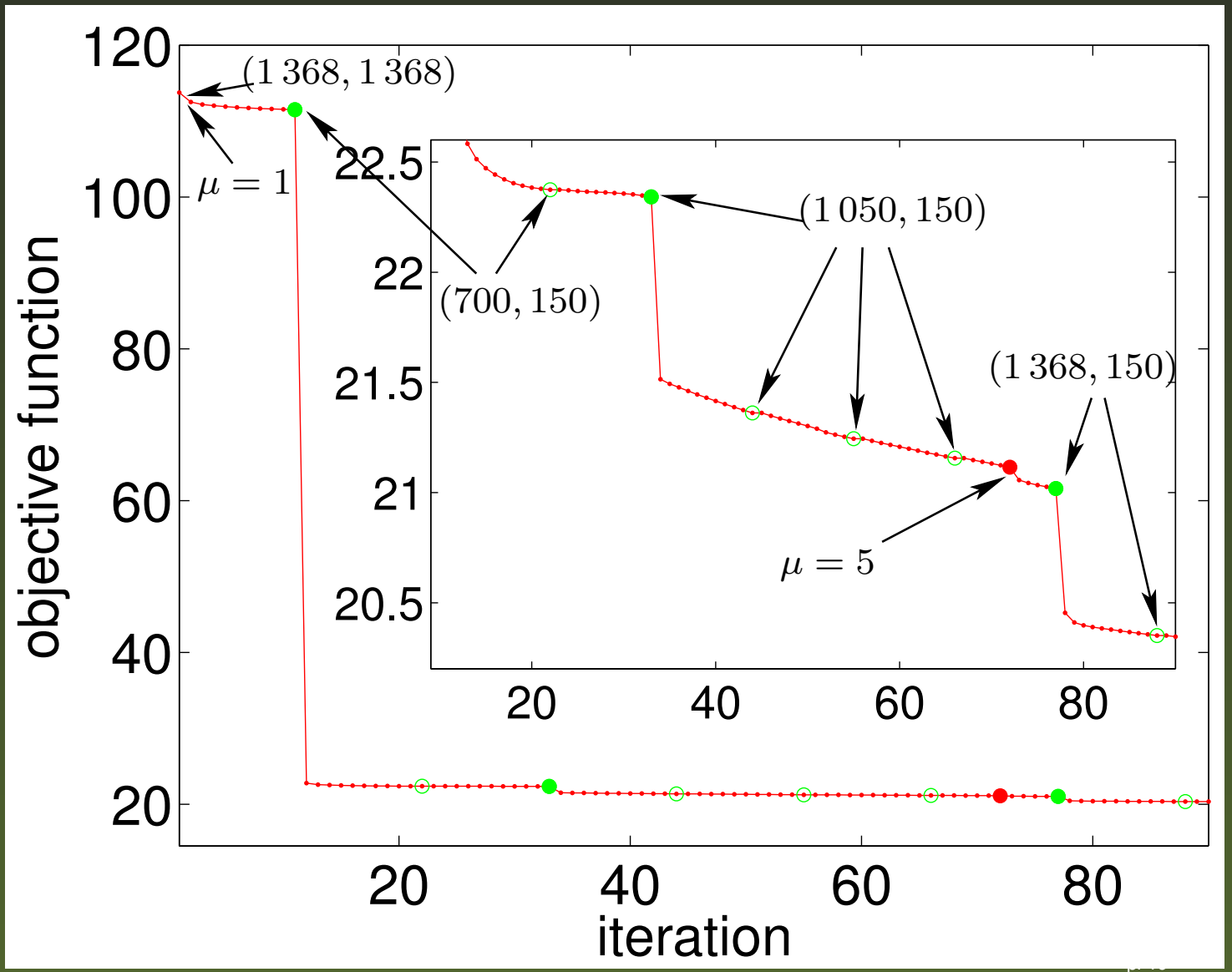
  The algorithm usually settles in a region of good architectures early during the optimisation, with small and infrequent changes thereafter.

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

Experiment: 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 with MAC

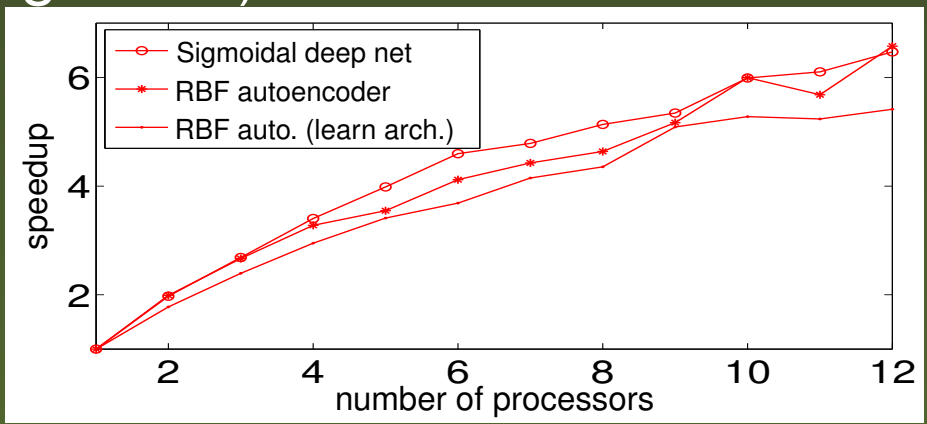
MAC/QP is embarrassingly parallel:

- $\mathbf{W}$ 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.

Shared-memory multiprocessor model, Matlab Parallel Processing Toolbox:



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
  - 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.

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