Optimising nested functions using auxiliary coordinates

Miguel Á. Carreira-Perpiñán

Electrical Engineering and Computer Science

University of California, Merced

`http://faculty.ucmerced.edu/mcarreira-perpinan`

work with **Mehdi Alizadeh**, **Ramin Raziperchikolaei**, **Max Vladymyrov** and **Weiran Wang**

Outline

- Nested (deep) and shallow systems
- Training nested systems: chain-rule gradient; greedy layerwise
- The method of auxiliary coordinates (MAC)
  - Design pattern
  - Convergence guarantees
  - Practicalities
  - Related work
  - Model selection “on the fly”
  - Distributed optimisation with MAC: ParMAC
- A gallery of nested models trainable with MAC
  1. Learning low-dimensional features for classification

     low-dimensional SVM, low-dimensional logistic regression
  2. Parametric nonlinear embeddings
  3. Binary hashing for fast image search

     binary autoencoder, affinity-based objective function

Nested (hierarchical) systems: examples

Common in computer vision, speech processing, machine learning…

- Object recognition pipeline:

  image pixels → SIFT/HoG → $\begin{array}{c}k\text{-means}\\ \text{sparse coding}\end{array}$ → pooling → classifier → object category

- Phone classification pipeline:

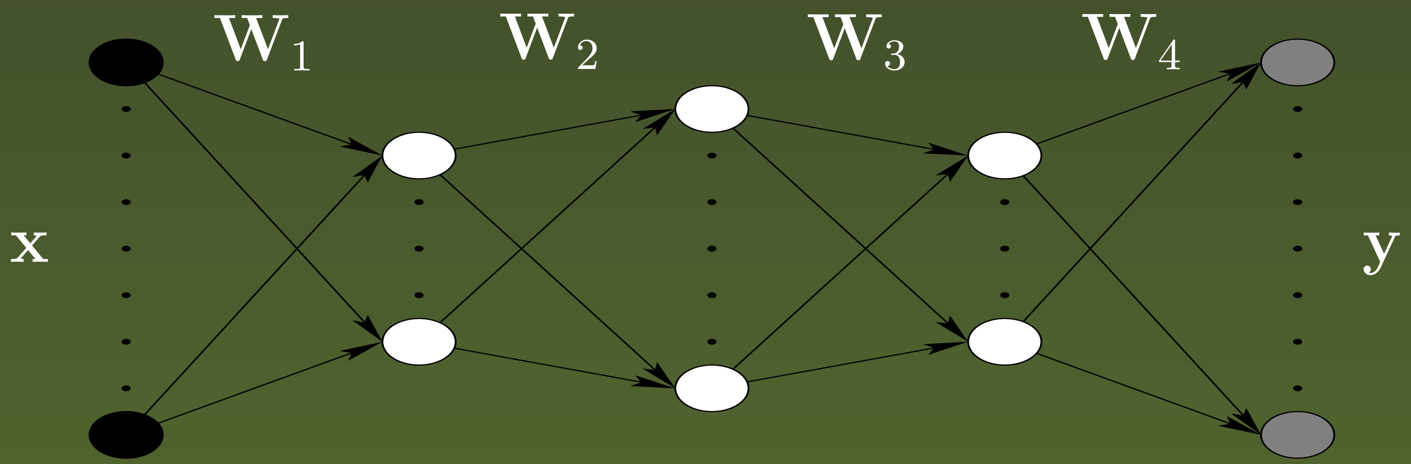
  waveform → MFCC/PLP → classifier → phoneme label

- Preprocessing for regression/classification/search/etc.:

  image pixels → PCA/LDA → classifier → output/label

  image pixels → projection → thresholding → search → nearest neighbour index

- 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:

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

- 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

    This may be avoided with automatic differentiation
  - requires nonlinear optimisation
  - vanishing gradients $\Rightarrow$ ill-conditioning $\Rightarrow$ slow progress even with second-order methods

    This gets worse the more layers we have
  - difficult to parallelise.

Training nested systems: layerwise, “filter”

- Fix each layer sequentially from the input to the output (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” or “feature extraction” stage which can be done with existing algorithms.
- 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 $\mathbf{g}$ 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.
- Less frequently 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
- Nonconvex, nonlinear, maybe nondifferentiable so difficult to train:
  - Layerwise: easy but suboptimal
  - Backpropagation: optimal but slow, difficult to implement, needs differentiable layers.

The method of auxiliary coordinates (MAC)

M. Carreira-Perpiñán and W. Wang: *Distributed optimization of deeply nested systems*, AISTATS 2014, arXiv:1212.5921.

- 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 $\{\mathbf{y}_n\}_{n=1}^N$:

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

$\mathbf{F}$, $\mathbf{g}$ 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:

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

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

- 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-penalised function

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

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

We can also use the augmented Lagrangian method (ADMM) instead:

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

which does an extra update of the Lagrange multipliers: $\boldsymbol{\lambda}_n \leftarrow \boldsymbol{\lambda}_n - \mu(\mathbf{z}_n - \mathbf{F}(\mathbf{x}_n)),\ n = 1,\dots,N$.

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:

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

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

- 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-penalised function

$(\mathbf{F},\mathbf{g})$ 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 opt. of the MAC-penalised function (cont.)

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

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

- 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:
  $\min_{\mathbf{z}} f(\mathbf{z}) + \frac{\mu}{2} \left\| \mathbf{z} - \mathbf{u} \right\|^2$
  The solution has a geometric flavour (“projection”).
- Often closed-form (depending on the model).

Alternating opt. of the MAC-penalised function (cont.)

MAC 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 is different from pure alternating optimisation over layers:

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

- 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

Assuming differentiable functions:

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

That is, MAC defines a continuous path $(\mathbf{W}^*(\mu), \mathbf{Z}^*(\mu))$ that converges (as $\mu \rightarrow \infty$) to a local stationary point of the constrained problem and thus to a local stationary point 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:

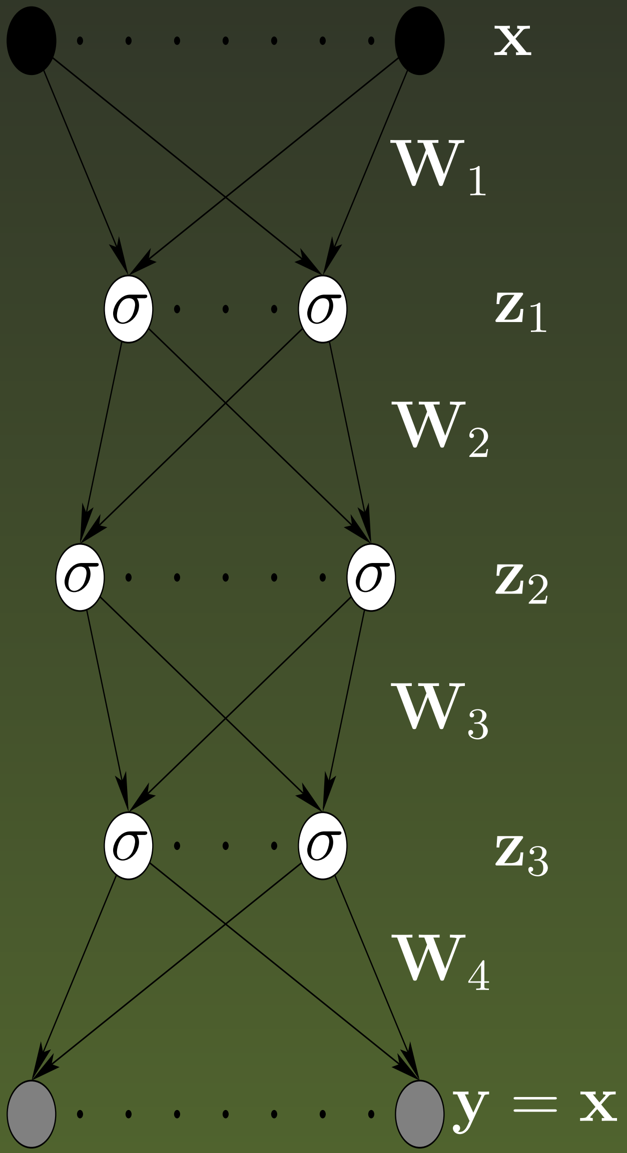
1. Write your nested objective function $E_{\text{nested}}(\mathbf{W})$.
2. Identify subexpressions and turn them into auxiliary coordinates with equality constraints.
3. Apply quadratic penalty or augmented Lagrangian method.
4. Do alternating optimisation of the penalised function:
   - $\mathbf{W}$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

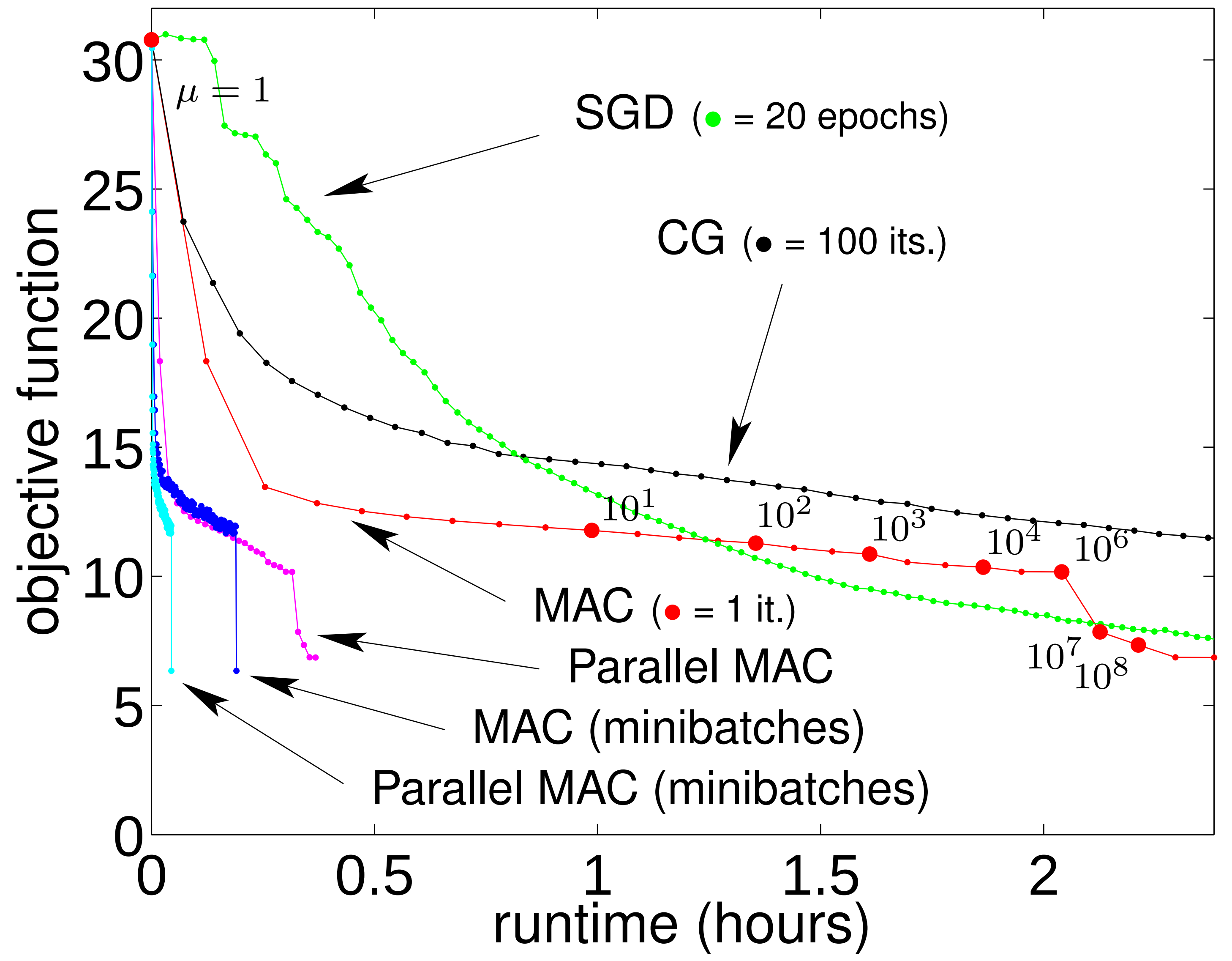
     proximal operator; for many important cases closed-form or simple to optimise.

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.





Experiment: 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

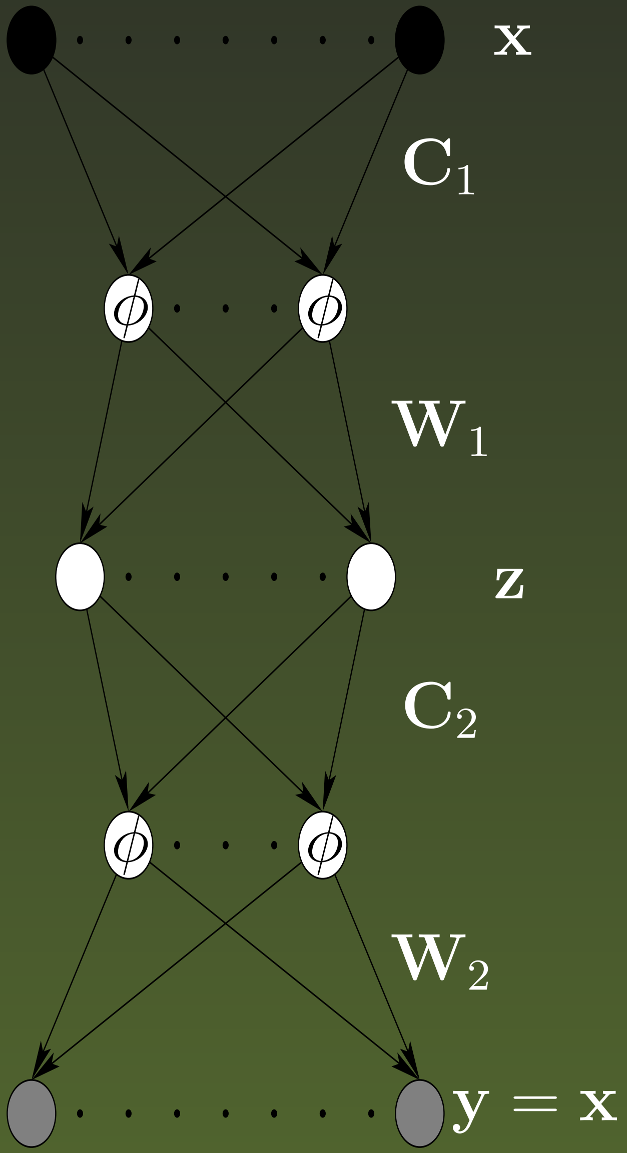
  typical of alternating optimisation algorithms.
- “Pretty good net pretty fast”.
- Competitive with state-of-the-art nonlinear optimisers, particularly with many nonlinear layers.

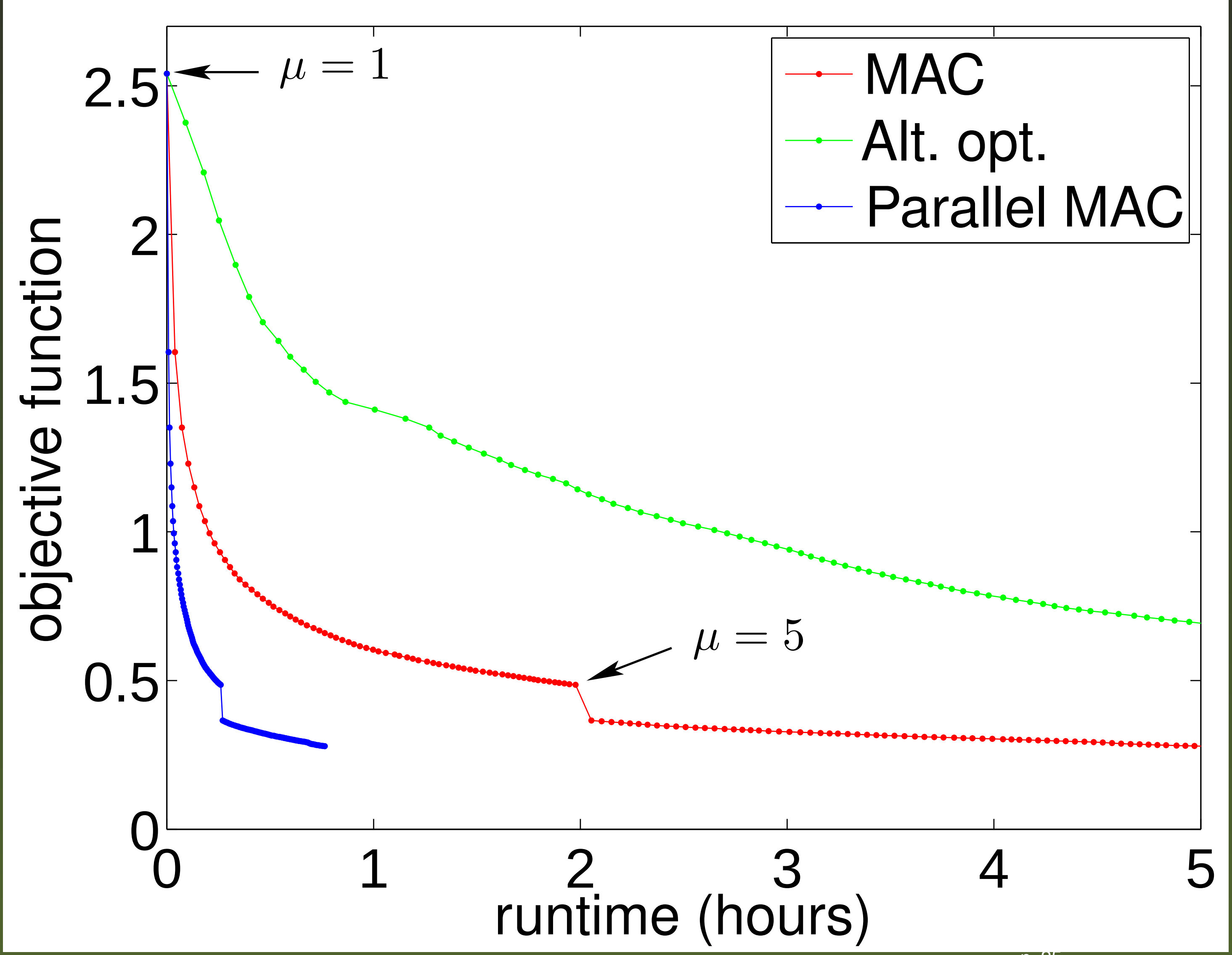
Note: the MAC iterations can be done much faster (see later):

- With better optimisation
- With parallel processing

Experiment: 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 $\mathbf{z}_{kn} = \mathbf{f}_k(\mathbf{z}_{k-1,n}; \mathbf{W}_k)$, $k = 1, \dots, K$, $n = 1, \dots, N$

    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 $(\mathbf{f}_K(\dots(\mathbf{f}_1(\mathbf{X}))), \mathbf{Y})$. 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.

Practicalities (cont.)

In summary, 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. . .

Related work: dimensionality reduction

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

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

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 $\mathbf{F}\colon \mathbf{y} \rightarrow \mathbf{z}$.

Related work: dimensionality reduction (cont.)

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

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

  - 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 to solve the autoencoder problem:

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

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 didn't converge to a minimum of the latter.

Related work: ADMM

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:

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

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

  The aug. Lag. $\mathcal{L}$ is minimised alternatingly over $\mathbf{X}$, $\mathbf{z}$ and $\boldsymbol{\Lambda}$.
- Can be applied to the MAC-constrained problem as well.

Related work: expectation-maximisation (EM) algorithm

- Trains probability models by maximum likelihood.
- Can be seen as bound optimisation (majorisation) or alternating optimisation (of a specially constructed function).
- E step: update the posterior probabilities $\mathbf{p}_1, \dots, \mathbf{p}_N$, in parallel.
  Like MAC's $\mathbf{Z}$ step. The posterior probabilities "coordinate" the individual models.
- M step: update the model parameters, in parallel (the Gaussians for a GM).
  Like MAC's step over the parameters. Each model is trained on its own data and $\mathbf{p}_1, \dots, \mathbf{p}_N$.

EM and MAC 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.
- Parallelism.

Related work: automatic differentiation

- Evaluates numerically the gradient (or some other derivative) of a function at a desired point, by applying the chain rule recursively.

  Assuming the function is differentiable.

- Saves us from computing the gradient analytically and coding it.

- Less efficient than a hand-crafted implementation of a particular gradient (expression simplification, etc.).

- It is not a numerical optimisation algorithm.

  We still need to use that gradient with an algorithm (gradient descent, Levenberg-Marquardt. . .), do a line search, select parameters (momentum rate, damping factor. . .), have a stopping criterion, etc.

- In MAC we don't use the gradient of the objective function because we split it into subproblems. Each of these is often a well-known optimisation problem for which efficient code (possibly optimised for our architecture) exists. MAC calls this as a black box by passing to it the data and auxiliary coordinates and receiving the solution.

  We need not compute/code the gradient, do l.s., select parameters, etc.

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.

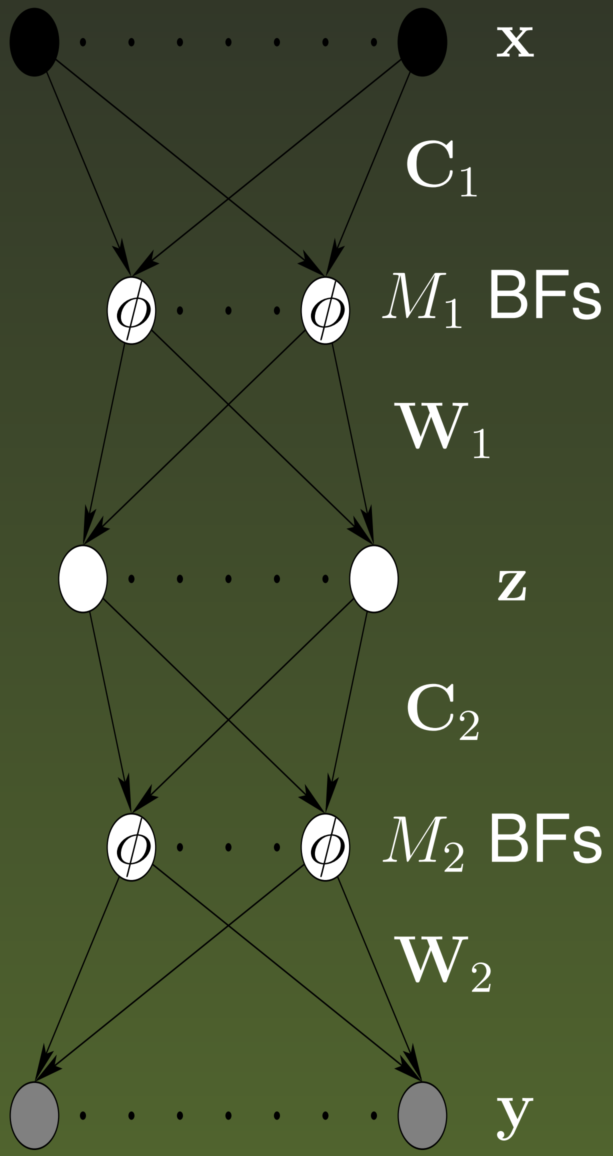
- In practice, no need to do model selection at each $\mathbf{W}$ step.

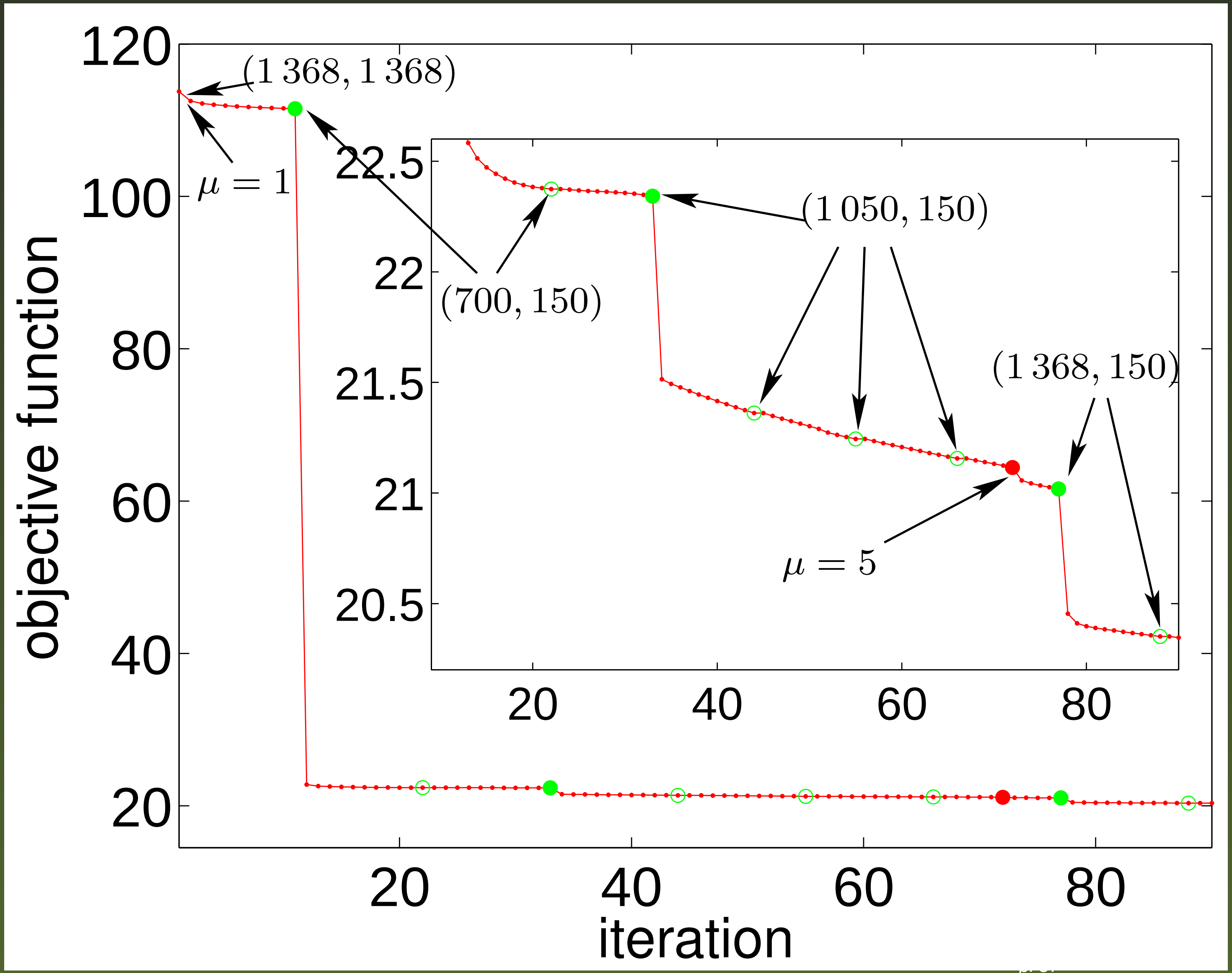
  The algorithm usually settles in a region of good architectures early during the optimisation, with small and infrequent changes thereafter.

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 $\{150, \dots, 1368\}$ (50 values ⇒ 50^2 possible models).





Distributed optimisation with MAC

MAC 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.

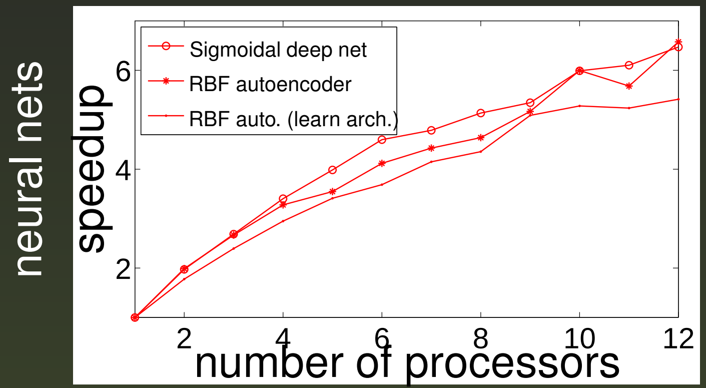
Distributed optimisation with MAC: Matlab

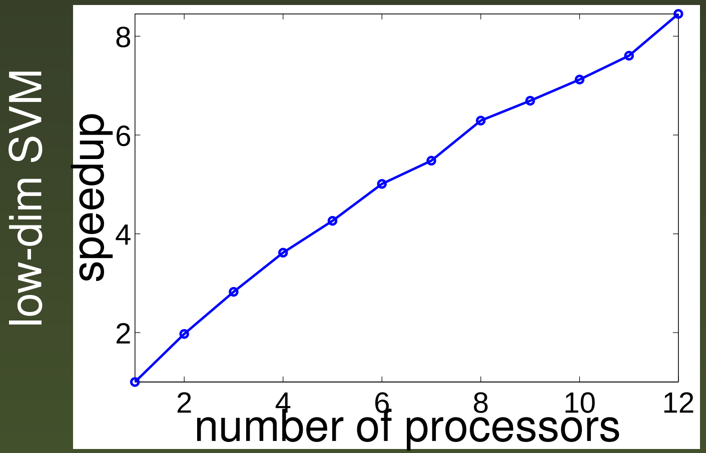
- Shared-memory multiprocessor model using the Matlab Parallel Processing Toolbox: simply change `for` to `parfor` in the $\mathbf{W}$ and $\mathbf{Z}$ loops.

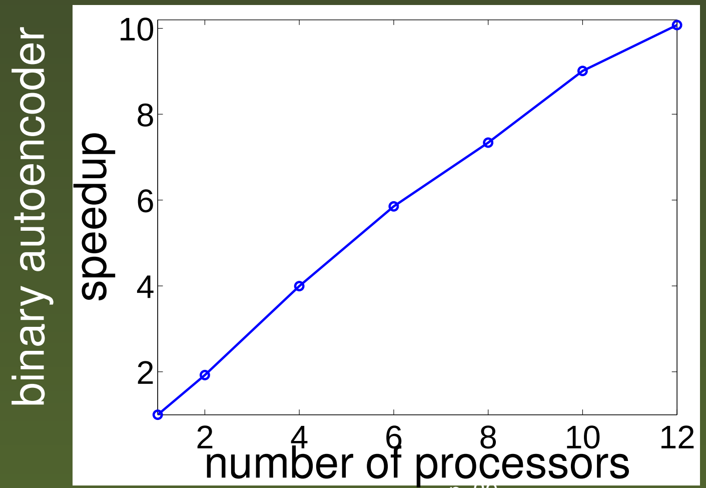
  So Matlab sends each iteration to a different server.

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

  even though the Matlab Parallel Processing Toolbox is quite inefficient.







Distributed optimisation with MAC: ParMAC

M. Carreira-Perpiñán and M. Alizadeh: *ParMAC: distributed optimisation of nested functions, with application to learning binary autoencoders*. arXiv:1605.09114.

- Large dataset distributed over P machines, each storing $\frac{N}{P}$ points.
- Must keep communication low (it is much more costly than computation).
- ParMAC: a parallel, distributed optimisation framework for MAC:
  - $\mathbf{W}$ step: only model parameters are communicated (not data or auxiliary coordinates), following a circular topology, which implicitly implements a stochastic optimisation.
  - $\mathbf{Z}$ step: as usual (no communication).

  Facilitates data shuffling, load balancing, fault tolerance and streaming data processing.
- Convergence guarantees carry over with an SGD-type condition for the $\mathbf{W}$ step.
- We can predict theoretically the parallel speedup $S(P)$. Typically, $S(P) \approx P$ if $P \leq$ # submodels in $\mathbf{W}$ step, i.e., the speedup is nearly perfect if using fewer machines than submodels. This affords very large speedups for large datasets and large models.

Distributed optimisation with MAC: ParMAC (cont.)

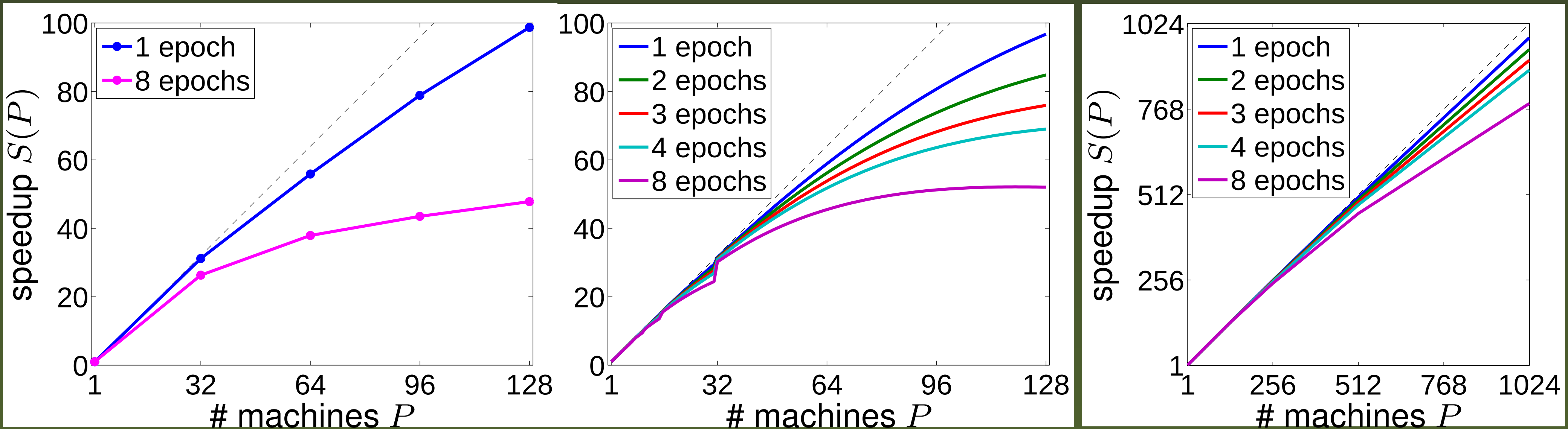
Experiments with a binary autoencoder for binary hashing using a database of images ($D = 128$ SIFT features):

- SIFT-1M: $N = 10^6$ training points, hash code of $L = 16$ bits
- SIFT-1B: $N = 10^8$ training points, hash code of $L = 64$ bits

Implemented in C/C++ using MPI (Message Passing Interface).

expt. ← SIFT-1M → theory

SIFT-1B, theory



A gallery of nested models trainable with MAC

MAC is very widely applicable. Here are a few examples:

1. Learning low-dimensional features for classification:
   - Low-dimensional SVM
   - Low-dimensional logistic regression
2. Parametric nonlinear embeddings:
   - t-SNE/elastic embedding with linear/neural net mapping
3. Binary hashing for fast image search:
   - Binary autoencoder
   - Affinity-based objective function

These demonstrate MAC in various settings:

- various types of regressor or classifier
- various optimisation algorithms for each step
- various penalty functions: quadratic penalty, augmented Lagrangian
- differentiable and nondifferentiable (or even discrete) functions
- auxiliary coordinates in objective or constraints
- $\mathbf{Z}$ step separable or coupled on N auxiliary coordinates

1. Learning low-dimensional features for classification

We want to learn a low-dimensional classifier $y = g(\mathbf{F}(\mathbf{x}))$:

- first we reduce the dimensionality of the input $\mathbf{x}$ with a mapping $\mathbf{F}$

  linear, RBF network, neural net, Gaussian process, etc.
- Then we classify the low-dimensional projection with a classifier g

  linear SVM, logistic regression

We introduce as auxiliary coordinates $\mathbf{z}_n$ the projection of each $\mathbf{x}_n$.
The resulting MAC algorithm alternates:

- train a regressor $\mathbf{F}$ on $(\mathbf{X}, \mathbf{Z})$ and a classifier g on $(\mathbf{Z}, \mathbf{Y})$, in parallel

  done with existing algorithms
- update the N auxiliary coordinates $\mathbf{Z}$, in parallel

What if we want to use a different $\mathbf{F}$ or g?

- $\mathbf{F}$: simply call a different algorithm in $\mathbf{F}$ step

  black box: no need to compute gradients or code anything
- g: call a different algorithm in g step, but the $\mathbf{Z}$ step does change.

Low-dimensional SVM

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

Binary classifier g: linear SVM $g(\mathbf{z}) = \mathbf{w}^T\mathbf{z} + b$.

Objective function over the parameters of g and $\mathbf{F}$ given a dataset $\{(\mathbf{x}_n, y_n)\}_{n=1}^N$ of (inputs,labels):

$$\begin{aligned} \min_{\mathbf{F},\mathbf{g},\boldsymbol{\xi}} \quad & \lambda R(\mathbf{F}) + \frac{1}{2}\left\|\mathbf{w}\right\|^2 + C\sum_{n=1}^N \xi_n \\ \text{s.t.} \quad & y_n(\mathbf{w}^T\mathbf{F}(\mathbf{x}_n) + b) \geq 1 - \xi_n,\ \xi_n \geq 0,\ n = 1,\dots,N. \end{aligned}$$

- If $\mathbf{F} =$ identity then $y = g(\mathbf{F}(\mathbf{x})) = \mathbf{w}^T\mathbf{x} + b$ is a linear SVM and the problem is a convex quadratic program, easy to solve.
- If $\mathbf{F}$ is nonlinear then $y = g(\mathbf{F}(\mathbf{x})) = \mathbf{w}^T\mathbf{F}(\mathbf{x}) + b$ is a nonlinear classifier and the problem is nonconvex, difficult to solve.

Low-dimensional SVM: auxiliary coordinates

The problem is “nested” because of $g(\mathbf{F}(\cdot))$ in the constraints:

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

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

Break the nesting by introducing an auxiliary coordinate $\mathbf{z}_n$ per point:

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

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

The auxiliary coordinates correspond to the low-dimensional projections but are separate parameters.

Low-dimensional SVM: alternating optimisation

Solve this constrained problem with a quadratic-penalty method: optimise for fixed penalty parameter $\mu > 0$ and drive $\mu \to \infty$:

$$\begin{aligned} \min_{\mathbf{F},\mathbf{g},\boldsymbol{\xi},\mathbf{Z}} \quad & \lambda R(\mathbf{F}) + \frac{1}{2}\left\|\mathbf{w}\right\|^2 + C\sum_{n=1}^{N}\xi_n + \frac{\mu}{2}\sum_{n=1}^{N}\left\|\mathbf{z}_n - \mathbf{F}(\mathbf{x}_n)\right\|^2 \\ \text{s.t.} \quad & y_n(\mathbf{w}^T\mathbf{z}_n + b) \geq 1 - \xi_n,\ \xi_n \geq 0,\ n = 1,\dots,N. \end{aligned}$$

Alternating optimisation of the penalty function for fixed μ:

- g step: fit a linear SVM to $\{(\mathbf{z}_n, y_n)\}_{n=1}^N$
  A classifier using as inputs the auxiliary coordinates.
- $\mathbf{F}$ step: fit a nonlinear mapping $\mathbf{F}$ to $\{(\mathbf{x}_n, \mathbf{z}_n)\}_{n=1}^N$
  A regressor using as outputs the auxiliary coordinates.

$\Bigg\}$ in parallel

- $\mathbf{Z}$ step: set $\mathbf{z}_n = \mathbf{F}(\mathbf{x}_n) + \gamma_n y_n \mathbf{w}$, $n = 1,\dots,N$
  A closed-form update for each point's auxiliary coordinate $\mathbf{z}_n$.

$\Big\}$ in parallel

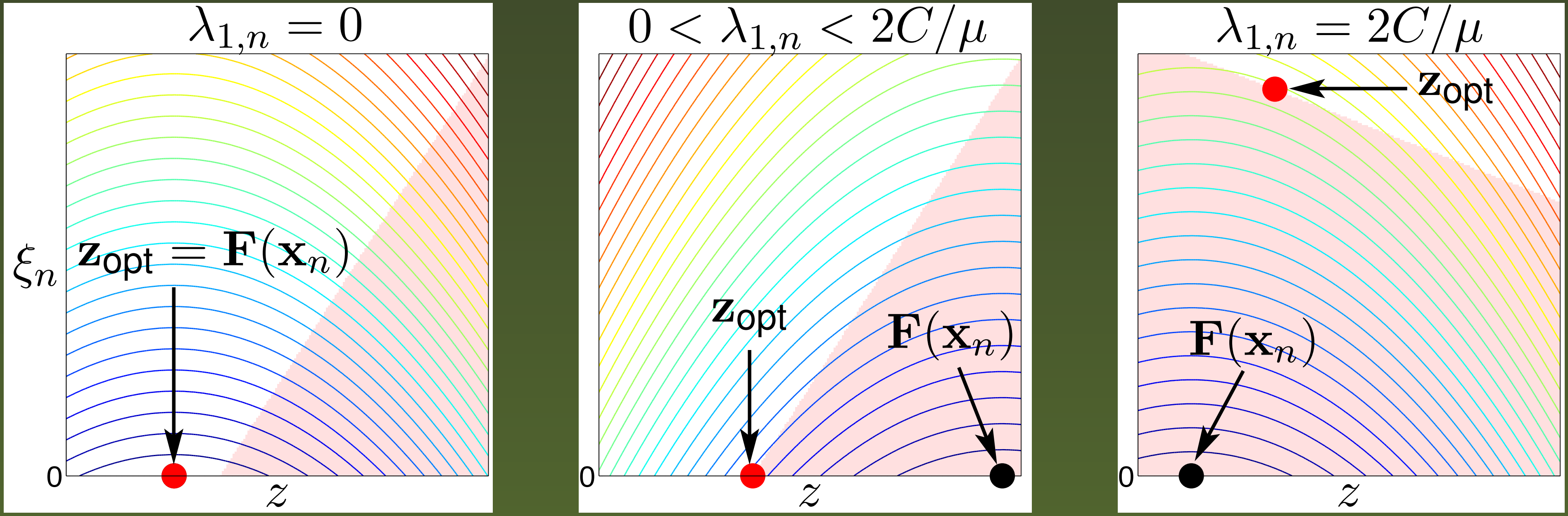
Low-dimensional SVM: solution of the Z step

The $\mathbf{Z}$ step decouples into N independent problems, each a quadratic program on L parameters ($L =$ dimension of latent space):

$$\min_{\mathbf{z}_n \in \mathbb{R}^L, \xi_n \in \mathbb{R}} \left\| \mathbf{z}_n - \mathbf{F}(\mathbf{x}_n) \right\|^2 + \frac{2C}{\mu}\xi_n \quad \text{s.t.} \quad y_n(\mathbf{w}^T\mathbf{z}_n + b) \geq 1 - \xi_n, \ \xi_n \geq 0$$

with closed-form solution $\mathbf{z}_n = \mathbf{F}(\mathbf{x}_n) + \gamma_n y_n \mathbf{w}$, where $\gamma_n = \frac{1}{2}\min(\lambda_{1,n}, 2C/\mu)$ and $\lambda_{1,n} = \frac{2}{\|\mathbf{w}\|^2}\max(0, 1 - y_n(\mathbf{w}^T\mathbf{F}(\mathbf{x}_n) + b))$.

This corresponds to three possible cases. Cost: $\mathcal{O}(L)$.



Low-dimensional SVM: multiclass case

With K classes, we can use the one-versus-all scheme and have K binary SVMs, each of which classifies whether a point belongs to one class or not.

- Objective function: the sum of that of K binary SVMs.
- $\mathbf{F}$ step: a regressor as before.
- g step: a classifier as before, but with K binary SVMs trained in parallel (one per class).
- $\mathbf{Z}$ step: a QP for each point on $L+K$ variables and $2K$ constraints:

$$\min_{\mathbf{z}_n, \{\xi_n^k\}_{k=1}^K} \quad \left\|\mathbf{z}_n - \mathbf{F}(\mathbf{x}_n)\right\|^2 + \sum_{k=1}^K C^k \xi_n^k$$

$$\text{s.t.} \quad y_n^k((\mathbf{w}^k)^T \mathbf{z}_n + b^k) \geq 1 - \xi_n^k,\ \xi_n^k \geq 0, \qquad k = 1, \dots, K.$$

This has no closed-form solution, so we solve it numerically.

Low-dimensional SVM: experiment

Reducing dimension with a RBF network $\mathbf{F}(\mathbf{x}) = \sum_{m=1}^{M} \boldsymbol{\alpha}_m \phi_m(\mathbf{x})$ the classifier has the form of a nonlinear SVM $y = \sum_{m=1}^{M} \mathbf{v}_m \phi_m(\mathbf{x}) + b$.

- We have direct control on the classifier complexity through the # BFs M. Unlike in an SVM, where the number of support vectors M is often very large.
- We can trade off classification error vs runtime:
  - Training: linear on sample size N and # BFs M.
  - Testing: linear on # BFs M.

  Error comparable to an SVM's, but with far fewer basis functions.
- The resulting classifier is competitive with nonlinear SVMs but faster to train and at test time.
- The classification accuracy improves drastically as L increases from 1 and stabilizes at $L \approx K - 1$, by which time the training samples are perfectly separated and the classes form point-like clusters approximately lying on the vertices of a simplex.
- MAC algorithm: large progress in first few iterations. ☞ ☞

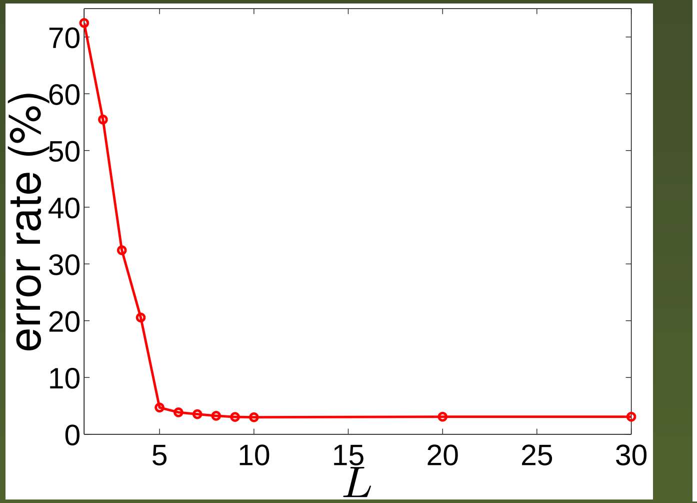
Low-dimensional SVM: experiment (cont.)

10k MNIST handwritten digit images of $28 \times 28 = 784$ dimensions with $K = 10$ classes (one-versus-all).

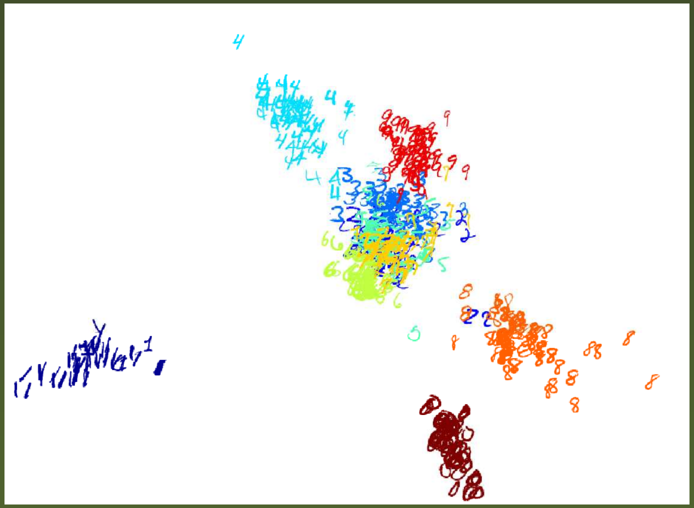
A low-dim SVM ($L = K$ dimensions) does as well as a kernel SVM but using 5.5 times fewer support vectors. The low-dim SVM is also faster to train and parallelizes trivially.

Method	Test error	# BFs
Nearest Neighbour	5.34	10 000
Linear SVM	9.20	–
Gaussian SVM	2.93	13 827
low-dim SVM	2.99	2 500
LDA (9) + Gaussian SVM	10.67	8 740
PCA (5) + Gaussian SVM	24.31	13 638
PCA (10) + Gaussian SVM	7.44	5 894
PCA (40) + Gaussian SVM	2.58	12 549
PCA (40) + low-dim SVM	2.60	2 500

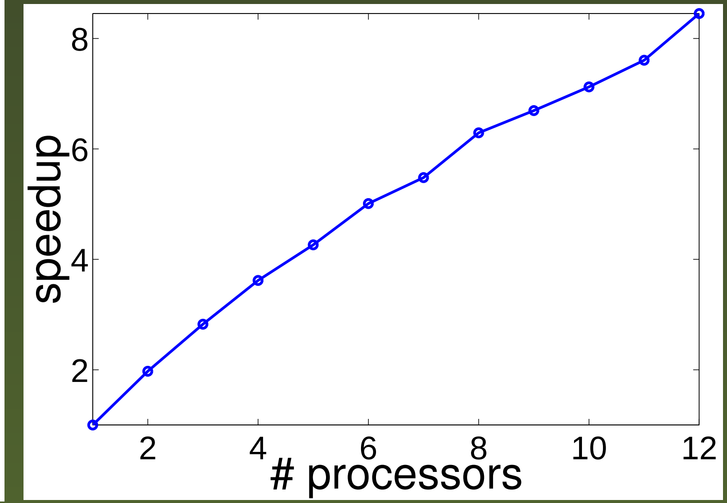
classification error wrt L



L-dim space (+ 2D PCA)



parallel processing speedup



Low-dimensional logistic regression

Binary classifier g: logistic regression $g(\mathbf{z}) = \sigma(\mathbf{w}^T\mathbf{z}) = \frac{1}{1+e^{-\mathbf{w}^T\mathbf{z}}}$.

Original (nested) objective function: cross-entropy + regularisation

$$\min_{\mathbf{F},\mathbf{w}} -\sum_{n=1}^{N} \big(y_n \log(\sigma(\mathbf{w}^T\mathbf{F}(\mathbf{x}_n))) + (1-y_n)\log(\sigma(-\mathbf{w}^T\mathbf{F}(\mathbf{x}_n)))\big) + \lambda_{\mathbf{F}} R(\mathbf{F}) + \lambda_{\mathbf{g}} R(\mathbf{w})$$

MAC-constrained problem:

$$\begin{aligned} \min_{\mathbf{Z},\mathbf{F},\mathbf{w}} \quad & -\sum_{n=1}^{N} \big(y_n \log(\sigma(\mathbf{w}^T\mathbf{z}_n)) + (1-y_n)\log(\sigma(-\mathbf{w}^T\mathbf{z}_n))\big) + \lambda_{\mathbf{F}} R(\mathbf{F}) + \lambda_{\mathbf{g}} R(\mathbf{w}) \\ \text{s.t.} \quad & \mathbf{z}_n = \mathbf{F}(\mathbf{x}_n),\ n = 1,\dots,N. \end{aligned}$$

MAC-penalised objective (using the augmented Lagrangian, with Lagrange multipliers $\boldsymbol{\alpha}_n$):

$$\begin{aligned} \min_{\mathbf{Z},\mathbf{F},\mathbf{w}} \quad & -\sum_{n=1}^{N} \big(y_n \log(\sigma(\mathbf{w}^T\mathbf{z}_n)) + (1-y_n)\log(\sigma(-\mathbf{w}^T\mathbf{z}_n))\big) + \lambda_{\mathbf{F}} R(\mathbf{F}) + \lambda_{\mathbf{g}} R(\mathbf{w}) \\ & -\sum_{n=1}^{N} \boldsymbol{\alpha}_n^T(\mathbf{z}_n - \mathbf{F}(\mathbf{x}_n)) + \frac{\mu}{2}\sum_{n=1}^{N} \|\mathbf{z}_n - \mathbf{F}(\mathbf{x}_n)\|^2 \end{aligned}$$

Low-dimensional logistic regression (cont.)

Alternating optimisation over $(\mathbf{g}, \mathbf{F}, \mathbf{Z})$:

$$\left.\begin{array}{l}\text{❖ } \mathbf{g} \text{ step: fit a logistic regression to } (\mathbf{Z}, \mathbf{Y}).\\ \text{❖ } \mathbf{F} \text{ step: fit a nonlinear mapping to } (\mathbf{X}, \mathbf{Z}).\end{array}\right\} \text{ in parallel}$$

$$\left.\begin{array}{l}\text{❖ } \mathbf{Z} \text{ step: solve for each point } n = 1, \dots, N\\ \quad \text{an } L\text{-dim smooth convex problem:}\end{array}\right\} \text{ in parallel}$$

$$\min_{\mathbf{z}_n \in \mathbb{R}^L} \; -y_n \log(\sigma(\mathbf{w}^T \mathbf{z}_n)) - (1 - y_n) \log(\sigma(-\mathbf{w}^T \mathbf{z}_n)) + \frac{\mu}{2} \|\mathbf{z}_n - \mathbf{F}(\mathbf{x}_n)\|^2 - \boldsymbol{\alpha}_n^T (\mathbf{z}_n - \mathbf{F}(\mathbf{x}_n))$$

- ✦ Solution: a closed-form update $\mathbf{z}_n = \mathbf{F}(\mathbf{x}_n) + \frac{1}{\mu}\boldsymbol{\alpha}_n - \frac{t}{\mu}\mathbf{w}$

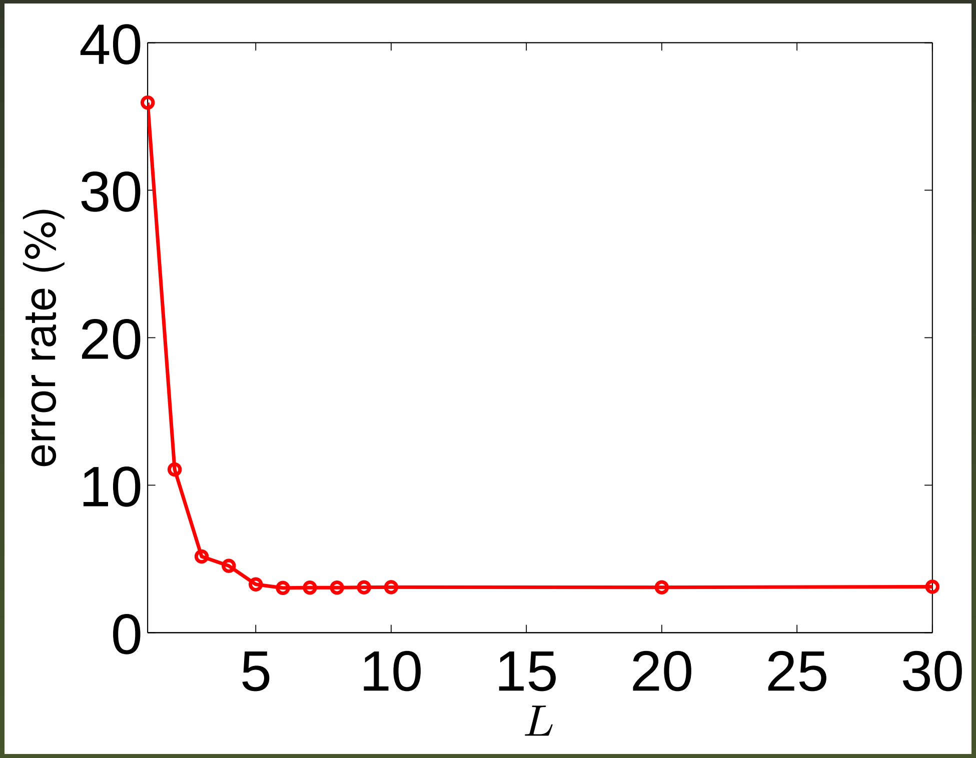
  t is the single solution of the scalar eq. $t = 1 - y_n - \sigma\left(\frac{\|\mathbf{w}\|^2}{\mu} t - \mathbf{w}^T \left(\mathbf{F}(\mathbf{x}_n) + \frac{1}{\mu}\boldsymbol{\alpha}_n\right)\right)$.
- ✦ Also convex for multi-class case with softmax

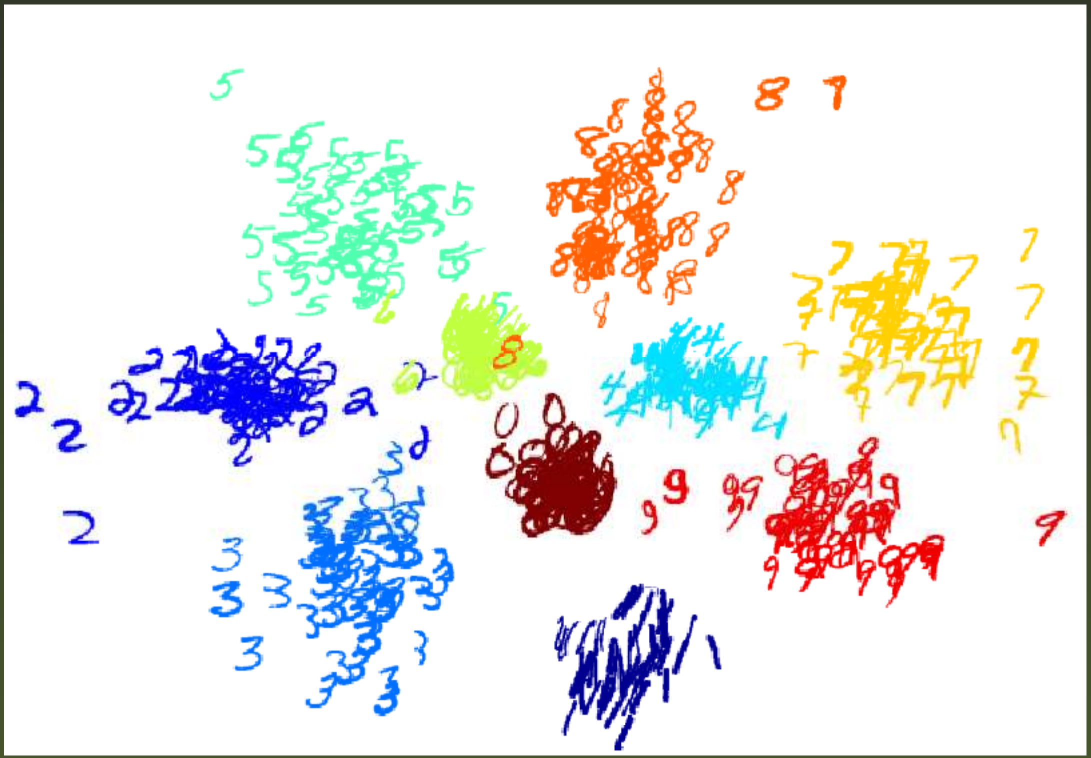
Then, update the Lagrange multipliers:

$$\boldsymbol{\alpha}_n \leftarrow \boldsymbol{\alpha}_n - \mu(\mathbf{z}_n - \mathbf{F}(\mathbf{x}_n)), \; n = 1, \dots, N$$

Low-dimensional logistic regression (cont.)

Low-dimensional logistic regression ($\mathbf{F}$ = RBF network with $2\,500$ BFs) of MNIST digits ($K = 10$ classes)





classification error over L

embedding $\mathbf{F}(\mathbf{X})$ for $L = 2$

The results are similar to those of the low-dimensional SVM.

2. Parametric nonlinear embeddings

M. Carreira-Perpiñán and M. Vladymyrov: *A fast, universal algorithm to learn parametric nonlinear embeddings*, NIPS 2015.

Nonlinear embedding, eg t-SNE or the elastic embedding:

- Finds L-dimensional projections $\mathbf{x}_1, \dots, \mathbf{x}_N$ of N high-dimensional data points $\mathbf{y}_1, \dots, \mathbf{y}_N$ by preserving similarities w_{nm} between pairs of points $(\mathbf{y}_n, \mathbf{y}_m)$

  $$E(\mathbf{X}) = \sum_{n,m=1}^{N} w_{nm} \left\lVert \mathbf{x}_n - \mathbf{x}_m \right\rVert^2 + \lambda \sum_{n,m=1}^{N} \exp\left(- \left\lVert \mathbf{x}_n - \mathbf{x}_m \right\rVert^2\right)$$

  Often used to visualise high-dimensional datasets in 2D.
- Nonlinear, nonconvex, $\mathcal{O}(N^2)$ terms in objective function.
- Efficient algorithms proposed in the last few years, iterations are $\mathcal{O}(N)$, scale to millions of points

  spectral direction, fast multipole methods, Barnes-Hut tree.

2. Parametric nonlinear embeddings (cont.)

Parametric embedding: out-of-sample mapping to project points not in the training set ($\mathbf{F}$ can be linear, a neural net, etc.):

$$P(\mathbf{F}) = \sum_{n,m=1}^{N} w_{nm} \left\|\mathbf{F}(\mathbf{y}_n) - \mathbf{F}(\mathbf{y}_m)\right\|^2 + \lambda \sum_{n,m=1}^{N} \exp\left(- \left\|\mathbf{F}(\mathbf{y}_n) - \mathbf{F}(\mathbf{y}_m)\right\|^2 \right)$$

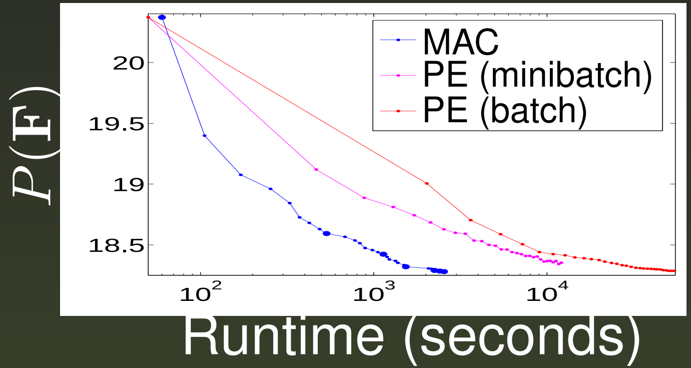
- The parametric embedding objective is a nested function

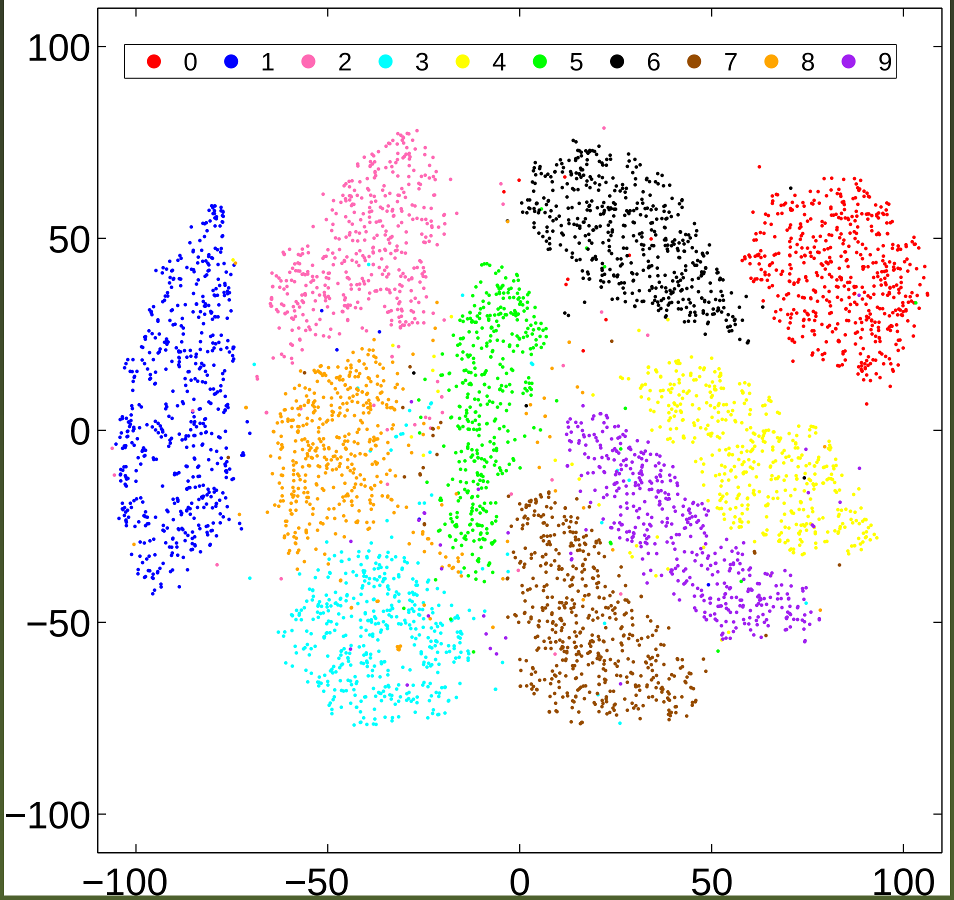
  nonlinear, nonconvex, $\mathcal{O}(N^2)$ terms, very slow gradient-based optimisation.
- Introducing the latent projections $\mathbf{z}_n = \mathbf{F}(\mathbf{x}_n)$ as auxiliary coordinates, the MAC algorithm alternates these two steps:
  - $\mathbf{F}$ step: nonlinear regression on $(\mathbf{Y}, \mathbf{Z})$
  - $\mathbf{Z}$ step: regularised nonlinear embedding $P(\mathbf{Z})$

    reuse above algorithms; does not separate over $\mathbf{z}_1, \dots, \mathbf{z}_N$.
- Much faster than using the chain-rule gradient over $\mathbf{F}$.
- If we want to use a different embedding objective $E(\mathbf{X})$ or projection mapping $\mathbf{F}$, we simply call a different embedding or regression routine in the $\mathbf{Z}$ or $\mathbf{F}$ step, resp., as a black box.

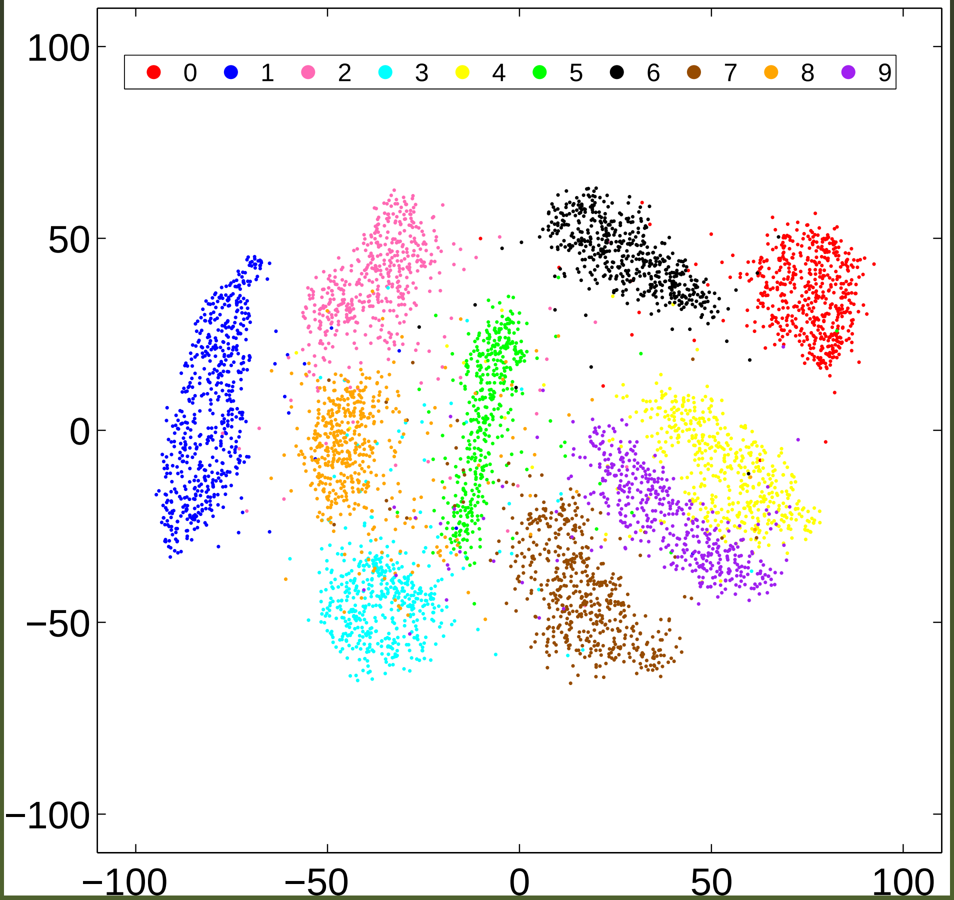
2. Parametric nonlinear embeddings (cont.)

MNIST digits visualised with t-SNE neural net embedding ((28×28)–500–500–2000–2, initialized with pretraining). ☞





initial, free embedding $E(\mathbf{X})$



Final parametric embedding $\mathbf{F}(Y)$

3. Binary hashing for fast image search

Binary hashing: we want to map a high-dimensional vector $\mathbf{x} \in \mathbb{R}^D$ (e.g. an image) to a low-dimensional binary vector $\mathbf{z} = \mathbf{h}(\mathbf{x}) \in \{0,1\}^L$ using a binary hash function $\mathbf{h}$, such that Hamming distances in the binary space approximate distances in the original space.

Ex: $\mathbf{h}(\mathbf{x}) = s(\mathbf{W}\mathbf{x})$ where $s(\cdot)$ is a step function.

- Application: fast search in image databases
  - Distance calculation fast with hardware support

    XOR then count 1s instead of floating-point operations
  - Binary database may fit in (fast) memory

    1M points of $D = 300$ floats take 1.2 GB but only 4 MB with a 32-bit code
- In order to learn the hash function $\mathbf{h}$ from data, we formulate an objective function $E(\mathbf{h})$ that preserves distances in some way.

  Many formulations exist.
- Difficult optimisation because E depends on the output of $\mathbf{h}$, which is binary, so E is piecewise constant, nonsmooth and nonconvex. The problem is usually NP-complete.

3. Binary hashing for fast image search (cont.)

We introduce as auxiliary coordinates $\mathbf{z}_n \in \{0,1\}^L$ the binary code of each $\mathbf{x}_n$. The resulting MAC algorithm alternates:

- training L binary classifiers on $(\mathbf{X}, \mathbf{Z})$, in parallel

  done with existing algorithms, e.g. linear SVM

- updating the N auxiliary coordinates $\mathbf{Z}$

  need to solve this

as $\mu \longrightarrow \infty$ (in fact, we can prove the iterations stop for a *finite* μ).
If we want to use a different hash function $\mathbf{h}$ (linear, neural net. . .), we simply call a different classifier routine in the $\mathbf{h}$ step as a black box.

We consider two generic types of objective function:

- Binary autoencoder: unsupervised

  preserve (Euclidean) distances between original feature vectors

- Affinity-based objective function: supervised

  neighbourhood information provided by the user (e.g. class labels)

Binary autoencoder

M. Carreira-Perpiñán and R. Raziperchikolaei: *Hashing with binary autoencoders*, CVPR 2015.

Original (nested) objective function: reconstruction error:

$$E(\mathbf{h},\mathbf{f}) = \sum_{n=1}^{N} \left\|\mathbf{x}_n - \mathbf{f}(\mathbf{h}(\mathbf{x}_n))\right\|^2 \qquad \begin{cases} \text{encoder } \mathbf{h}\text{: } \mathbb{R}^D \rightarrow \{0,1\}^L & \text{(our hash function)} \\ \text{decoder } \mathbf{f}\text{: } \{0,1\}^L \rightarrow \mathbb{R}^D \end{cases}$$

MAC-constrained problem:

$$\min_{\mathbf{h},\mathbf{f},\mathbf{Z}} \sum_{n=1}^{N} \left\|\mathbf{x}_n - \mathbf{f}(\mathbf{z}_n)\right\|^2 \quad \text{s.t.} \quad \mathbf{z}_n = \mathbf{h}(\mathbf{x}_n),\ \mathbf{z}_n \in \{0,1\}^L,\ n = 1,\dots,N.$$

MAC-penalised objective:

$$\min_{\mathbf{h},\mathbf{f},\mathbf{Z}} \sum_{n=1}^{N} \left(\left\|\mathbf{x}_n - \mathbf{f}(\mathbf{z}_n)\right\|^2 + \mu \left\|\mathbf{z}_n - \mathbf{h}(\mathbf{x}_n)\right\|^2\right) \quad \text{s.t.} \quad \begin{cases} \mathbf{z}_n \in \{0,1\}^L \\ n = 1,\dots,N. \end{cases}$$

Binary autoencoder (cont.)

Alternating optimisation over $(\mathbf{h}, \mathbf{f}, \mathbf{Z})$:

- $\mathbf{h}$ step: fit L binary classifiers to $\{(\mathbf{x}_n, \mathbf{z}_n)\}_{n=1}^N$.
- $\mathbf{f}$ step: fit a regression mapping to $\{(\mathbf{z}_n, y_n)\}_{n=1}^N$.

} in parallel

- $\mathbf{Z}$ step: solve for each point $n = 1, \dots, N$
  a binary optimisation in L variables:

} in parallel

$$\min_{\mathbf{z}_n \in \{0,1\}^L} \left\| \mathbf{x}_n - \mathbf{f}(\mathbf{z}_n) \right\|^2 + \mu \left\| \mathbf{z}_n - \mathbf{h}(\mathbf{x}_n) \right\|^2$$

This is usually NP-complete, but the number of variables is small.

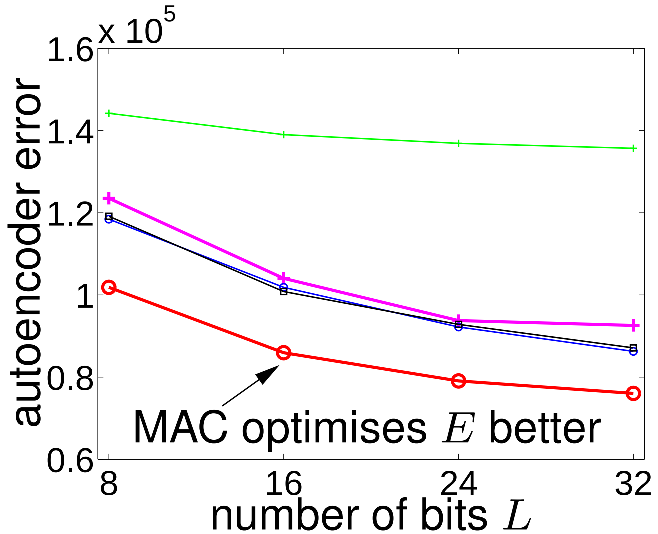
- If $L \lesssim 10$ we can afford an exhaustive search over the 2^L codes.
- Otherwise, we use alternating optimisation over (groups of) bits.

We can use some other tricks to speed this up (incremental computation, good initialisation, etc.).

Binary autoencoder (cont.)

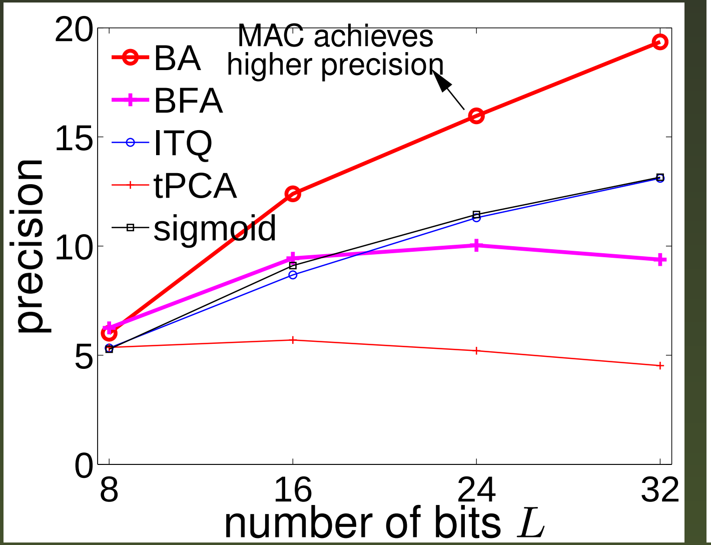
NUS-WIDE-LITE dataset, $N = 27\,807$ training/ $27\,808$ test images, $D = 128$ wavelet features, linear hash function $\mathbf{h}(\mathbf{x}) = s(\mathbf{W}\mathbf{x})$.

reconstruction error wrt L



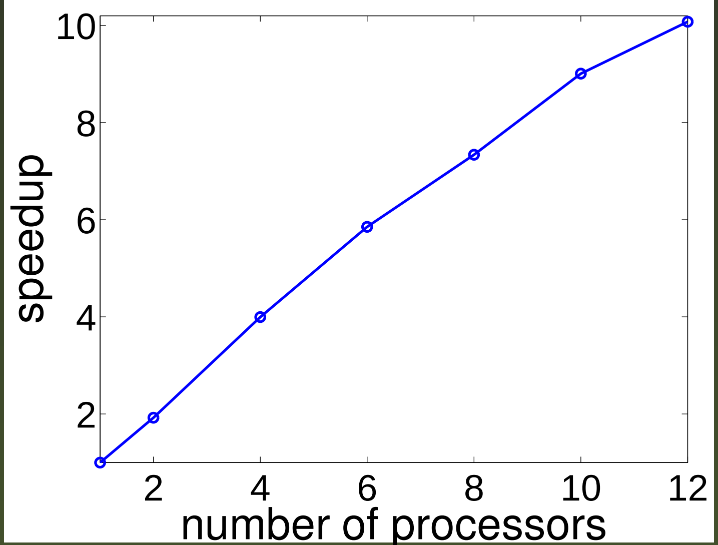
MAC optimises E better

precision in retrieval



MAC achieves higher precision

parallel processing speedup



Optimising the binary autoencoder with MAC achieves both a lower reconstruction error and a better precision/recall in searching than if using other approximate methods to optimise it (e.g. relaxing/truncating, using a sigmoid instead of a step function, etc.).

It also parallelises very well.

Affinity-based objective function

R. Raziperchikolaei and M. Carreira-Perpiñán: *Optimizing affinity-based binary hashing using auxiliary coordinates*, arXiv:1501.05352.

Original (nested) objective function: $\mathcal{L}(\mathbf{h}) = \sum_{n,m=1}^{N} L(\mathbf{h}(\mathbf{x}_n), \mathbf{h}(\mathbf{x}_m); w_{nm})$

where L is a loss function that compares the codes for two images with their affinity w_{nm}.

Many formulations exist, e.g. $\begin{cases} w_{nm} = -1/+1/0 \text{ if similar/dissimilar/don't know} \\ L_{\text{KSH}}(\mathbf{z}_n, \mathbf{z}_m; w_{nm}) = (\mathbf{z}_n^T \mathbf{z}_m - L w_{nm})^2. \end{cases}$

MAC-constrained problem:

$$\min_{\mathbf{h},\mathbf{Z}} \sum_{n,m=1}^{N} L(\mathbf{z}_n, \mathbf{z}_m; w_{nm}) \quad \text{s.t.} \quad \mathbf{z}_n = \mathbf{h}(\mathbf{x}_n),\ \mathbf{z}_n \in \{0,1\}^b,\ n = 1,\dots,N.$$

MAC-penalised objective:

$$\min_{\mathbf{h},\mathbf{Z}} \sum_{n,m=1}^{N} L(\mathbf{z}_n, \mathbf{z}_m; w_{nm}) + \mu \sum_{n=1}^{N} \|\mathbf{z}_n - \mathbf{h}(\mathbf{x}_n)\|^2 \quad \text{s.t.} \quad \begin{cases} \mathbf{z}_n \in \{0,1\}^L \\ n = 1,\dots,N. \end{cases}$$

Affinity-based objective function (cont.)

Alternating optimisation over $(\mathbf{h}, \mathbf{Z})$:

- $\mathbf{h}$ step: fit L binary classifiers to $\{(\mathbf{x}_n, \mathbf{z}_n)\}_{n=1}^N$, in parallel.
- $\mathbf{Z}$ step: minimise the following regularised binary embedding:

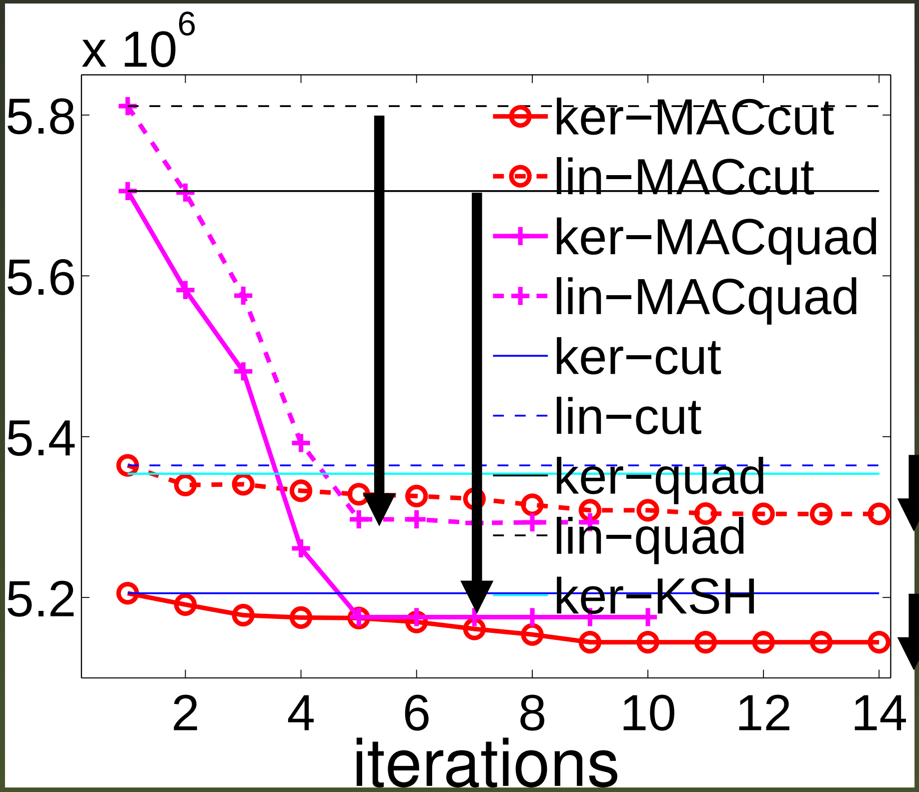
$$\min_{\mathbf{Z}} \sum_{n,m=1}^{N} L(\mathbf{z}_n, \mathbf{z}_m; w_{nm}) + \mu \sum_{n=1}^{N} \|\mathbf{z}_n - \mathbf{h}(\mathbf{x}_n)\|^2 \quad \text{s.t.} \quad \begin{cases} \mathbf{z}_n \in \{0,1\}^L \\ n = 1, \dots, N. \end{cases}$$

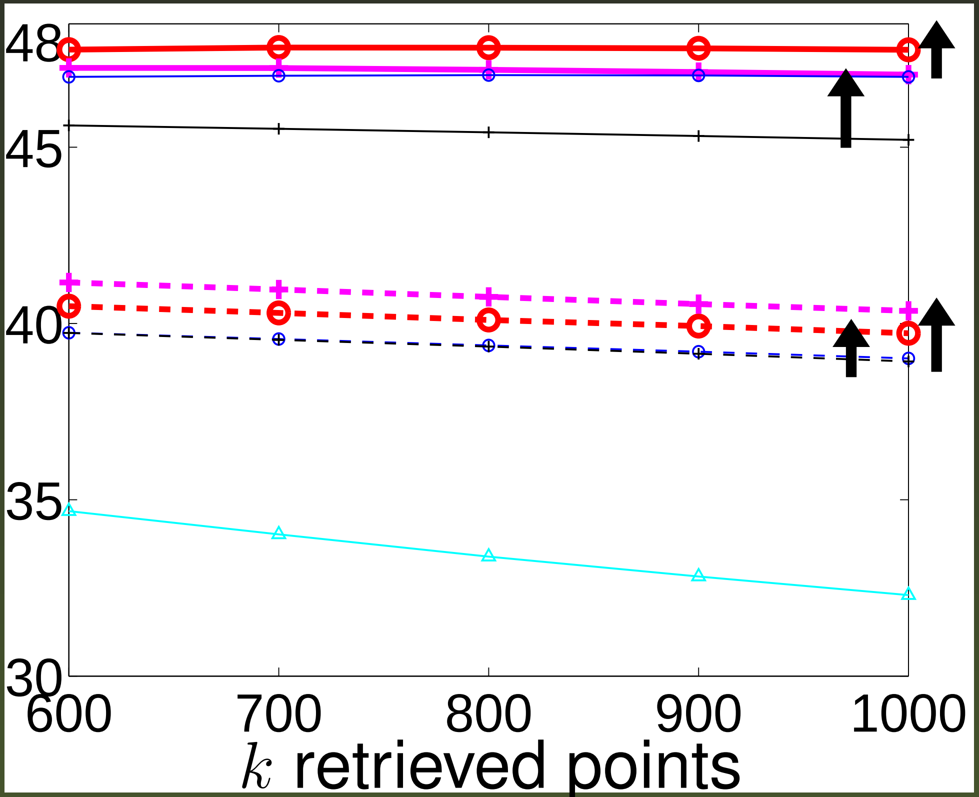
This is a binary optimisation over NL binary variables, usually NP-complete. We reuse existing approximate methods to optimise this.

Alternating optimisation over the lth bit of each code for $l = 1, \dots, L$, with each optimisation over the N binary variables solved approximately by relaxation/truncation or by GraphCut (Lin et al. 2013, 2014).

Affinity-based objective function (cont.)

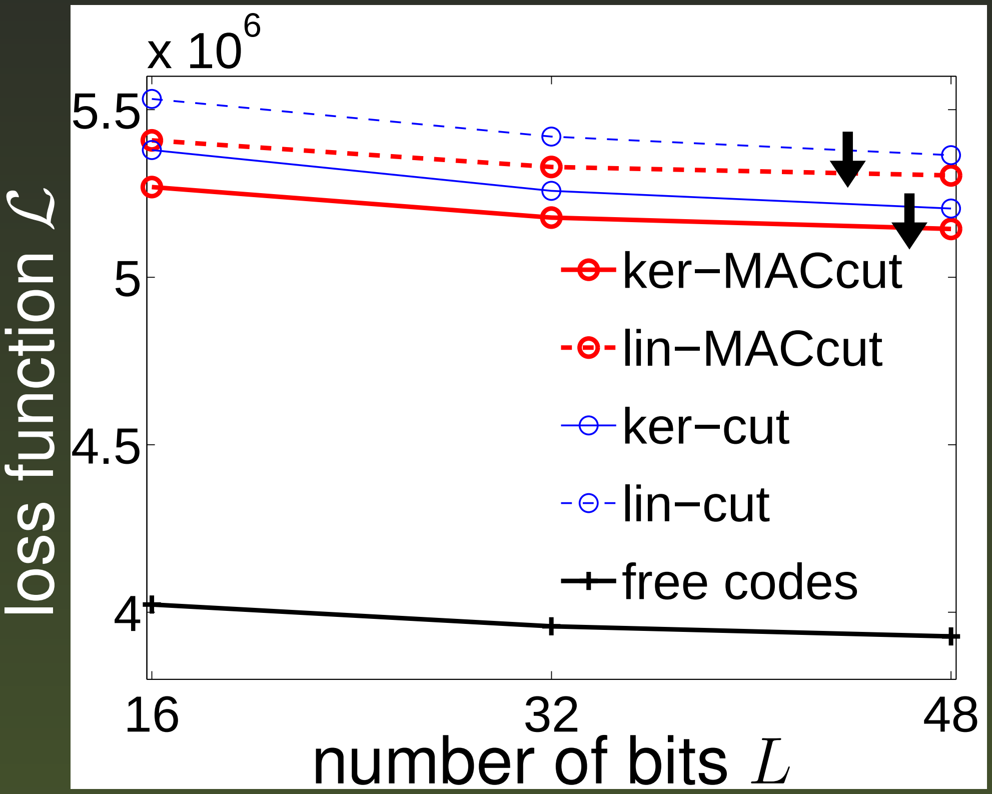
CIFAR dataset, $N = 58\,000$ training/ $2\,000$ test images, $D = 320$ SIFT features, $L = 48$ bits.

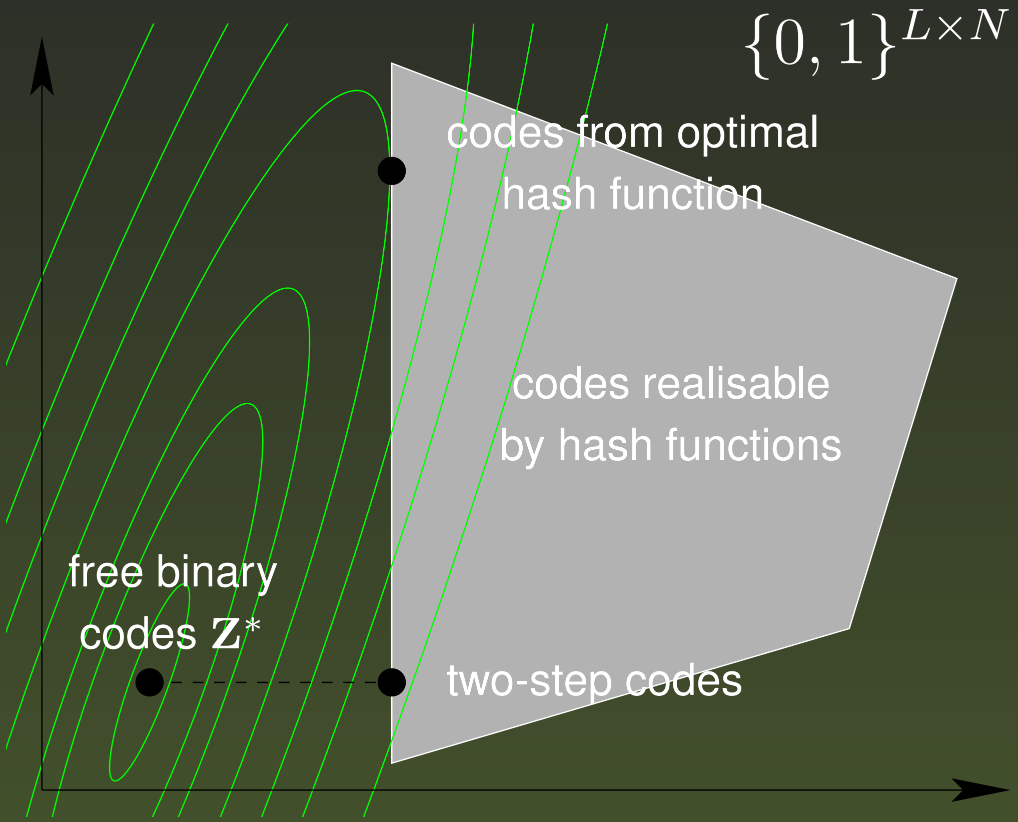




MAC finds hash functions with significantly better objective function value and precision/recall than two-step approaches (which first optimise the binary codes and then fit to them the hash function) and other approximate methods.

Affinity-based objective function (cont.)




$\{0,1\}^{L\times N}$
codes from optimal hash function
codes realisable by hash functions
free binary codes $\mathbf{Z}^*$
two-step codes

The free binary codes $\mathbf{Z}^*$ are an (approximate) minimiser of the loss $\mathcal{L}(\mathbf{Z})$ without any constraint (i.e., disregarding the hash function). They are usually not realisable by any (linear, say) hash function ($\mathbf{h}(\mathbf{X}) \neq \mathbf{Z}\ \forall\mathbf{h}$). Two-step methods project the free codes onto the feasible set of codes realisable by linear hash functions, which is suboptimal. MAC gradually adjusts both the codes and the hash function so they eventually match, which results in a better solution.

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 or code for shallow models as a black box
  - 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.

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.

References and code

http://faculty.ucmerced.edu/mcarreira-perpinan/MAC

Original reference for MAC:

- Miguel Carreira-Perpiñán and Weiran Wang: *Distributed optimization of deeply nested systems*. AISTATS 2014, arXiv:1212.5921.

Tutorials, reviews:

- C-P: *A tutorial on nested system optimisation using the method of auxiliary coordinates*.
- C-P: *A gallery of proximal operators arising in the method of auxiliary coordinates*.

Extensions or related work:

- C-P & Alizadeh: *ParMAC: distributed optimisation of nested functions, with application to learning binary autoencoders*. arXiv:1605.09114.
- C-P & Raziperchikolaei: *Optimizing affinity-based binary hashing using auxiliary coordinates*. arXiv:1501.05352.
- C-P & Vladymyrov: *A fast, universal algorithm to learn parametric nonlinear embeddings*. NIPS 2015.
- C-P & Raziperchikolaei: *Hashing with binary autoencoders*. CVPR 2015, arXiv:1501.00756.
- Wang and C-P: *The role of dimensionality reduction in classification*. AAAI 2014, arXiv:1405.6444.
- Wang and C-P: *Nonlinear low-dimensional regression using auxiliary coordinates*. AISTATS 2012.
- C-P and Lu: *Parametric dimensionality reduction by unsupervised regression*. CVPR 2010.
- C-P and Lu: *Dimensionality reduction by unsupervised regression*. CVPR 2008.

Work partially supported by NSF awards CAREER IIS–0754089, IIS–1423515 and a Google Faculty Research Award.