Distributed Optimization of Deeply Nested Systems

Miguel Á. Carreira-Perpiñán **Weiran Wang**
Electrical Engineering and Computer Science, School of Engineering, University of California, Merced

Abstract

Intelligent processing of complex signals such as images is often performed by a hierarchy of nonlinear processing layers, such as a deep net or an object recognition cascade. Joint estimation of the parameters of all the layers is a difficult nonconvex optimization. We describe a general strategy to learn the parameters and, to some extent, the architecture of nested systems, which we call the *method of auxiliary coordinates (MAC)*. This replaces the original problem involving a deeply nested function with a constrained problem involving a different function in an augmented space without nesting. The constrained problem may be solved with penalty-based methods using alternating optimization over the parameters and the auxiliary coordinates. MAC has provable convergence, is easy to implement reusing existing algorithms for single layers, can be parallelized trivially and massively, applies even when parameter derivatives are not available or not desirable, can perform some model selection on the fly, and is competitive with state-of-the-art nonlinear optimizers even in the serial computation setting, often providing reasonable models within a few iterations.

The continued increase in recent years in data availability and processing power has enabled the development and practical applicability of ever more powerful models in statistical machine learning, for example to recognize faces or speech, or to translate natural language. However, physical limitations in serial computation suggest that scalable processing will require algorithms that can be massively parallelized, so they can profit from the thousands of inexpensive processors available in cloud computing. We focus on hierarchical, or *nested*, processing architectures. As a particular but important example, consider deep neural nets (fig. 1), which were originally inspired by biological systems such as the visual and auditory cortex in the mammalian brain (Serre et al., 2007), and which have been proven very successful at learning sophisticated tasks, such as recognizing faces or speech, when trained on data.

A typical neural net defines a hierarchical, feedforward, parametric mapping from inputs to outputs. The parameters (*weights*) are learned given a dataset by numerically minimizing an objective function. The outputs of the hidden units at each layer are obtained by transforming the previous layer's outputs by a linear operation with the layer's weights followed by a nonlinear elementwise mapping (e.g. sigmoid). Deep, nonlinear neural nets are universal approximators, that is, they can approximate any target mapping (from a wide class) to arbitrary accuracy given enough units (Bishop, 2006), and can have more representation power than shallow nets (Bengio and LeCun, 2007). The hidden units may encode hierarchical, distributed features that are useful to deal with complex sensory data. For example, when trained on images, deep nets can learn low-level features such as edges and T-junctions and high-level features such as parts decompositions. Other examples of hierarchical processing systems, sometimes consisting of heterogeneous layers such as a deep net followed by an SVM, are cascades for object recognition and scene understanding in computer vision (Serre et al., 2007) or for phoneme classification in speech processing (Saon and Chien, 2012), wrapper approaches to classification or regression (e.g. based on dimensionality reduction; Wang and Carreira-Perpiñán, 2012), or kinematic chains in robotics. These and other architectures share a fundamental design principle: *mathematically, they construct a deeply nested mapping from inputs to outputs*.

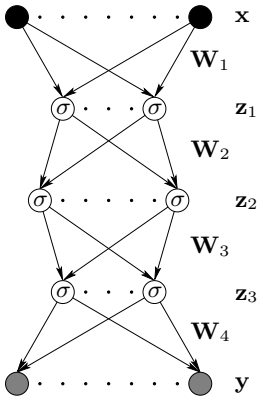

Figure 1: Net with $K = 3$ hidden layers ($\mathbf{W}_k$: weights, $\mathbf{z}_k$: auxiliary coordinates).

The ideal performance of a nested system arises when all

Appearing in Proceedings of the 17^{th} International Conference on Artificial Intelligence and Statistics (AISTATS) 2014, Reykjavik, Iceland. JMLR: W&CP volume 33.

the parameters at all layers are jointly trained to minimize an objective function for the desired task, such as classification error (indeed, there is evidence that plasticity and learning probably occurs at all stages of the ventral stream of primate visual cortex; Serre et al., 2007). However, this is challenging because *nesting (i.e., function composition) produces inherently nonconvex functions*. Joint training is usually done by recursively computing the gradient with respect to each parameter using the chain rule, as in the backpropagation algorithm (Rumelhart et al., 1986). One can then simply update the parameters with a small step in the negative gradient direction as in gradient descent and stochastic gradient descent (SGD), or feed the gradient to a nonlinear optimization method that will compute a better search direction, possibly using second-order information, such as conjugate gradients or L-BFGS (Orr and Müller, 1998). This process is repeated until a convergence criterion is satisfied. Backprop in any of these variants suffers from the problem of vanishing gradients (Rögnvaldsson, 1994; Erhan et al., 2009), where the gradients for lower layers are much smaller than those for higher layers, which causes ill-conditioning of the objective function and leads to tiny steps, slowly zigzagging down a curved valley, and a very slow convergence. This problem worsens with the depth of the net and led researchers in the 1990s to give up in practice with nets beyond around two hidden layers (with special architectures such as convolutional nets (LeCun et al., 1998) being an exception) until recently, when improved initialization strategies (Hinton and Salakhutdinov, 2006; Bengio et al., 2007) and much faster computers—but not really any improvement in the optimization algorithms themselves—have renewed interest in deep architectures. Besides, backprop does not parallelize over layers (and, with nonconvex problems, is hard to parallelize over minibatches if using SGD), *is only applicable if the mappings are differentiable with respect to the parameters*, and needs careful tuning of learning rates. In summary, after decades of research in neural net optimization, simple backprop-based algorithms such as stochastic gradient descent remain the state-of-the-art, particularly when combined with good initialization strategies (Orr and Müller, 1998; Hinton and Salakhutdinov, 2006). In addition, selecting the best architecture, for example the number of units in each layer of a deep net, or the number of filterbanks in a speech front-end processing, requires a *combinatorial* search. In practice, this is approximated with a manual trial-and-error procedure that is very costly in effort and expertise required, and leads to suboptimal solutions (where often the parameters of each layer are set irrespective of the rest of the cascade).

We describe a general optimization strategy for deeply nested functions that we call *method of auxiliary coordinates (MAC)*, which partly alleviates the vanishing gradients problem, has embarrassing parallelization, reuses existing algorithms (possibly not gradient-based) that optimize single layers or individual units, and does some model selection on the fly. Section 1 describes MAC, section 2 describes related work and section 3 gives experimental results that illustrate the different advantages of MAC.

1 The Method of Auxiliary Coordinates (MAC)

1.1 The Nested Objective Function

For definiteness, we describe the approach for a deep net such as that of fig. 1. Later sections will show other settings. Consider a regression problem of mapping inputs $\mathbf{x}$ to outputs $\mathbf{y}$ (both high-dimensional) with a deep net $\mathbf{f}(\mathbf{x})$ given a dataset of N pairs $(\mathbf{x}_n, \mathbf{y}_n)$. A typical objective function to learn a deep net with K hidden layers has the form (to simplify notation, we ignore bias parameters):

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

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

where each layer function has the form $\mathbf{f}_k(\mathbf{x}; \mathbf{W}_k) = \sigma(\mathbf{W}_k\mathbf{x})$, i.e., a linear mapping followed by a squashing nonlinearity ($\sigma(t)$ applies a scalar function, such as the sigmoid $1/(1+e^{-t})$, elementwise to a vector argument, with output in $[0, 1]$). Our method applies to loss functions other than squared error (e.g. cross-entropy for classification), with fully or sparsely connected layers each with a different number of hidden units, with weights shared across layers, and with regularization terms on the weights $\mathbf{W}_k$. The basic issue is the deep nesting of the mapping $\mathbf{f}$. The traditional way to minimize (1) is by computing the gradient over all weights of the net using backpropagation and feeding it to a nonlinear optimization method. However, because of the multiple squashing nonlinear layers, the effect on the output of modifying a weight in layer 1 is far smaller than for layer K. This poor scaling means the gradient magnitudes are far smaller for the weights at the innermost levels, which causes the Hessian of E_1 to be ill-conditioned; this worsens with the number of layers (Rögnvaldsson, 1994; Erhan et al., 2009). As a result, most methods take tiny steps, slowly zigzagging down a curved valley, and take a huge time to converge in practice. Also, the chain rule requires differentiable layers wrt $\mathbf{W}$.

1.2 The Method of Auxiliary Coordinates (MAC)

We introduce one auxiliary variable per data point and per hidden unit and define the following equality-constrained optimization problem:

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

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

Each $\mathbf{z}_{k,n}$ can be seen as the coordinates of $\mathbf{x}_n$ in an intermediate feature space, or as the hidden unit activations for $\mathbf{x}_n$. Intuitively, by eliminating $\mathbf{Z}$ we see this is equivalent to the nested problem (1); we can prove under very general assumptions that both problems have exactly the same minimizers. Problem (2) seems more complicated (more variables and constraints), but each of its terms (objective and constraints) involve only a small subset of parameters and no nested functions. Below we show this reduces the ill-conditioning caused by the nesting, and partially decouples many variables, affording an efficient and distributed optimization, and model selection within each layer.

1.3 MAC with Quadratic-Penalty (QP) Optimization

Problem (2) may be solved with several constrained optimization methods. To show the advantages of MAC in the simplest way, we use the quadratic-penalty (QP) method (Nocedal and Wright, 2006). Using the augmented Lagrangian is also possible. We optimize the following function over $(\mathbf{W}, \mathbf{Z})$ for fixed $\mu > 0$ and drive $\mu \to \infty$:

$$E_Q(\mathbf{W}, \mathbf{Z}; \mu) = \frac{1}{2} \sum_{n=1}^{N} \|\mathbf{y}_n - \mathbf{f}_{K+1}(\mathbf{z}_{K,n}; \mathbf{W}_{K+1})\|^2 + \frac{\mu}{2} \sum_{n=1}^{N} \sum_{k=1}^{K} \|\mathbf{z}_{k,n} - \mathbf{f}_k(\mathbf{z}_{k-1,n}; \mathbf{W}_k)\|^2. \quad (3)$$

This defines a continuous path $(\mathbf{W}^*(\mu), \mathbf{Z}^*(\mu))$ which, under mild assumptions, converges to a minimum of the constrained problem (2), and thus to a minimum of the original problem (1) (Carreira-Perpiñán and Wang, 2012). In practice, we follow this path loosely.

The QP objective function can be seen as breaking the functional dependences in the nested mapping $\mathbf{f}$ and unfolding it over layers. Every squared term involves only a shallow mapping; all variables $(\mathbf{W}, \mathbf{Z})$ are equally scaled, which improves the conditioning of the problem; and the derivatives required are simpler: we require no backpropagated gradients over $\mathbf{W}$, and sometimes no gradients over $\mathbf{W}_k$ at all (see later). We now apply alternating optimization of the QP objective over $\mathbf{Z}$ and $\mathbf{W}$:

W-step Minimizing over $\mathbf{W}$ for fixed $\mathbf{Z}$ results in a separate minimization over the weights of each hidden unit—each a single-layer, single-unit problem that can be solved with existing algorithms. Specifically, for unit (k, h), for $k = 1, \dots, K+1$ (where we define $\mathbf{z}_{K+1,n} = \mathbf{y}_n$) and $h = 1, \dots, H_k$ (assuming there are H_k units in layer k), we have a nonlinear, least-squares regression of the form $\min_{\mathbf{w}_{kh}} \sum_{n=1}^{N} (z_{kh,n} - f_{kh}(\mathbf{z}_{k-1,n}; \mathbf{w}_{kh}))^2$, where $\mathbf{w}_{kh}$ is the weight vector (hth row of $\mathbf{W}_k$) that feeds into the hth output unit of layer k, and $z_{kh,n}$ the corresponding scalar target for point $\mathbf{x}_n$.

Z-step Minimizing over $\mathbf{Z}$ for fixed $\mathbf{W}$ separates over the coordinates $\mathbf{z}_n$ for each data point $n = 1, \dots, N$ (omitting the subindex n and weights):

$$\min_{\mathbf{z}} \frac{1}{2} \|\mathbf{y} - \mathbf{f}_{K+1}(\mathbf{z}_K)\|^2 + \frac{\mu}{2} \sum_{k=1}^{K} \|\mathbf{z}_k - \mathbf{f}_k(\mathbf{z}_{k-1})\|^2$$

and can be solved using the derivatives w.r.t. $\mathbf{z}$ of the single-layer functions $\mathbf{f}_1, \dots, \mathbf{f}_{K+1}$.

Thus, the $\mathbf{W}$-step results in many independent, single-layer single-unit problems that can be solved with existing algorithms, without extra programming cost. The $\mathbf{Z}$-step is new, however it always has the same form of a "generalized" proximal operator (Rockafellar, 1976). MAC reduces a complex, highly-coupled problem—training a deep net—to a sequence of simple, uncoupled problems (the $\mathbf{W}$-step) which are coordinated through the auxiliary variables (the $\mathbf{Z}$-step). For a large net with a large dataset, this affords an enormous potential for parallel, distributed computation. And, because each $\mathbf{W}$- or $\mathbf{Z}$-step operates over very large, decoupled blocks of variables, the decrease in the QP objective function is large in each iteration, unlike the tiny decreases achieved in the nested function. These large steps are effectively shortcuts through $(\mathbf{W}, \mathbf{Z})$-space, instead of tiny steps along a curved valley in $\mathbf{W}$-space.

Stopping Criterion and Penalty Parameter Exactly optimizing $E_Q(\mathbf{W}, \mathbf{Z}; \mu)$ for each μ follows the minima path strictly but is unnecessary, and one usually performs an inexact, faster optimization. Unlike in a general QP problem, in our case we have an accurate way to know when we should exit the optimization for a given μ. Since our real goal is to minimize the nested error $E_1(\mathbf{W})$, we exit when its value increases or decreases less than a set tolerance in relative terms. Further, as is common in neural net training, we use the validation error (i.e., $E_1(\mathbf{W})$ measured on a validation set). This means we follow the path $(\mathbf{W}^*(\mu), \mathbf{Z}^*(\mu))$ not strictly but only inasmuch as we approach a nested minimum, and our approach can be seen as a sophisticated way of taking a descent step in $E_1(\mathbf{W})$ but derived from $E_Q(\mathbf{W}, \mathbf{Z}; \mu)$. Using this stopping criterion maintains our theoretical convergence guarantees, because the path still ends in a minimum of E_1 and we drive $\mu \to \infty$. How to set μ depends on the problem, as with penalty methods in general. A practical strategy is to set $\mu = 1$, since we have observed this already gives good models in many problems, and then increase μ as needed.

The Postprocessing Step Once we have finished optimizing the MAC formulation with the QP method, we can apply a fast post-processing step that both reduces the objective function, achieves feasibility and eliminates the auxiliary coordinates. We simply satisfy the constraints by setting $\mathbf{z}_{kn} = \mathbf{f}_k(\mathbf{z}_{k-1,n}; \mathbf{W}_k)$, $k = 1, \dots, K$, $n = 1, \dots, N$, and keep all the weights the same except for the last layer, where we set $\mathbf{W}_{K+1}$ by fitting $\mathbf{f}_{K+1}$ to the dataset $(\mathbf{f}_K(\dots(\mathbf{f}_1(\mathbf{X}))), \mathbf{Y})$. One can prove the resulting weights reduce or leave unchanged the value of $E_1(\mathbf{W})$.

Jointly Learning All the Parameters in Heterogeneous Architectures Although the previous description was focused on neural nets, where each layer has the same form, MAC applies more generally to "heterogeneous" architectures. Here, each layer or function $\mathbf{f}_k$ may be different and perform a particular type of processing, such as a SVM, logistic regressor or sparse feature extractor. Typically, a specialized training algorithm exists to train such functions on their own, possibly not based on derivatives over the parameters (so that backprop is not applicable or not convenient for the entire nested system). For example, a quantization layer of an object recognition cascade, or the nonlinear layer of a radial basis function (RBF) network, often use a k-means training to estimate the parameters. Simply reusing this existing training algorithm as the $\mathbf{W}$-step for that layer allows MAC to learn jointly the parameters of the entire system with minimal programming effort, something that is not easy or not possible with other methods.

1.4 Model Selection

A final advantage of MAC is that it enables an efficient search not just over the parameter values of a given architecture, but (to some extent) *over the architectures themselves*. Traditional model selection usually involves obtaining optimal parameters (by running an already costly numerical optimization) for each possible architecture, and then evaluating each architecture based on a criterion such as cross-validation or a Bayesian Information Criterion (BIC), and picking the best (Hastie et al., 2009). This discrete-continuous optimization involves training an exponential number of models, so in practice one settles with a suboptimal search (e.g. fixing by hand part of the architecture based on an expert's judgment, or selecting parts separately and then combining them, or doing a grid or random search; Bergstra and Bengio, 2012). With MAC, model selection may be achieved "on the fly" by having the $\mathbf{W}$-step do model selection separately for each layer, and then letting the $\mathbf{Z}$-step coordinate the layers in the usual way. Specifically, consider a model selection criterion of the form $E_1(\mathbf{W}) + C(\mathbf{W})$, where E_1 is the nested objective function (1) and $C(\mathbf{W}) = C_1(\mathbf{W}_1) + \cdots + C_K(\mathbf{W}_K)$ is additive over the layers of the net. This is satisfied by many criteria, such as BIC, AIC or minimum description length (Hastie et al., 2009), in which $C(\mathbf{W})$ is essentially proportional to the number of free parameters. Cross-validation can also be considered. While optimizing $E_1(\mathbf{W}) + C(\mathbf{W})$ directly involves testing M^K deep nets if we have M choices for each layer, with MAC the $\mathbf{W}$-step separates over layers, and requires testing only MK single-layer nets at each iteration. While these model selection tests are still costly, they may be run in parallel, and they need not be run at each iteration. That is, we may alternate between running multiple iterations that optimize $\mathbf{W}$ for a given architecture, and running a model-selection iteration, and we still guarantee a monotonic decrease of $E_Q(\mathbf{W}) + C(\mathbf{W})$. In practice, we observe that a near-optimal model is often found in early iterations. Thus, the ability of MAC to decouple optimizations reduces a search of an exponential number of complex problems to an iterated search of a linear number of simple problems (of course, the model selection search in MAC is still local).

2 Discussion and Related Work

Rather than an algorithm, *the method of auxiliary coordinates is a mathematical device to design optimization algorithms suited for any specific nested architecture, that are provably convergent, highly parallelizable and reuse existing algorithms for non-nested (or shallow) architectures*. The key idea is the judicious elimination of subexpressions in a nested function via equality constraints. The architecture need not be strictly feedforward (e.g. recurrent nets). The designer need not introduce auxiliary coordinates at every layer: there is a spectrum between no auxiliary coordinates (full nesting), through hybrids that use some auxiliary coordinates and some semi-deep nets (which use the chain rule), to every single hidden unit having an auxiliary coordinate. An auxiliary coordinate may replace any subexpression of the nested function (e.g. the input to a hidden unit, rather than its output, leading to a quadratic $\mathbf{W}$-step). It is even possible to redefine $\mathbf{Z}$ at any time during the optimization. Other methods for constrained optimization may be used (e.g. the augmented Lagrangian rather than the quadratic-penalty method). Depending on the characteristics of the problem, the $\mathbf{W}$- and $\mathbf{Z}$-steps may be solved with any of a number of nonlinear optimization methods, from gradient descent to Newton's method, and using standard techniques such as warm starts, caching factorizations, inexact steps, stochastic updates using data minibatches, etc. In this respect, MAC is similar to other "metaalgorithms" such as expectation-maximization (EM) algorithms (Bishop, 2006) and alternating-direction method of multipliers (Boyd et al., 2011), which have become ubiquitous in statistics, machine learning and optimization. Indeed, the result of MAC is a "coordination-minimization" (CM) algorithm, that alternates fitting independent layers (M) with coordinating the layers (C).

Related Work We believe we are the first to propose the MAC formulation in full generality for nested function learning as a provably equivalent, constrained problem that is to be optimized jointly in the space of parameters and auxiliary coordinates using quadratic-penalty, augmented Lagrangian or other methods. However, there exist several lines of work related to it, and MAC/QP can be seen as giving a principled setting that justifies previous heuristic but effective approaches, and opening the door for new, principled ways of training deep nets and other nested systems.

Updating the activations of hidden units separately from the weights of a neural net has been done in the past. (1) Early work in neural nets considered learning good internal representations as important as learning good weights

(Grossman et al., 1988; Saad and Marom, 1990; Krogh et al., 1990; Rohwer, 1990). In fact, backpropagation was presented as a method to construct good internal representations that represent important features of the task domain (Rumelhart et al., 1986). Thus, while several papers proposed objective functions of both the weights and the activations, these were not intended to solve the nested problem or to achieve distributed optimization, but to help learn good representations. Typically, these algorithms did not converge to a solution of the nested problem and were developed for a single-hidden-layer net in very small problems. More recent variations have similar problems (Ma et al., 1997; Castillo et al., 2006). (2) More recent work in learning sparse features with overcomplete dictionaries use an explicit penalty (e.g. L_1) over the features, but this has only been considered for a single layer (the one that extracts the features) and again does not minimize the nested problem (Olshausen and Field, 1996; Ranzato et al., 2007; Kavukcuoglu et al., 2008). Some work for a single hidden layer net mentions the possibility of recovering backpropagation in a limit (Krogh et al., 1990; Kavukcuoglu et al., 2008), but this is not used to construct an algorithm that converges to a nested problem optimum. Recent works in deep net learning, such as pretraining (Hinton and Salakhutdinov, 2006), greedy layerwise training (Bengio et al., 2007) or deep stacking Deng et al. (2012), do a single pass over the net from the input to the output layer, fixing the weights of each layer sequentially, but without optimizing a joint objective of all weights. While these heuristics can be used to achieve good initial weights, they do not converge to a minimum of the nested problem. (3) Auxiliary variables have been used before in statistics and machine learning, from early work in factor and homogeneity analysis (Gifi, 1990), to learn dimensionality reduction mappings given a dataset of high-dimensional points $\mathbf{x}_1, \dots, \mathbf{x}_N$. Here, one takes the latent coordinates $\mathbf{z}_n$ of each data point $\mathbf{x}_n$ as parameters to be estimated together with the reconstruction mapping $\mathbf{f}$ that maps latent points to data space and minimize a least-squares error function $\sum_{n=1}^{N} \|\mathbf{x}_n - \mathbf{f}(\mathbf{z}_n)\|^2$, often by alternating over $\mathbf{f}$ and $\mathbf{Z}$. Various nonlinear versions of this approach exist where $\mathbf{f}$ is a single-layer neural net (Tan and Mavrovouniotis, 1995), Gaussian process (Lawrence, 2005) and others. However, particularly with nonparametric functions, the error can be driven to zero by separating infinitely apart the $\mathbf{Z}$, and so these methods need ad-hoc terms on $\mathbf{Z}$ to prevent this. The dimensionality reduction by unsupervised regression approach of Carreira-Perpiñán and Lu (2008, 2010, 2011) (generalized to supervised dimensionality reduction in Wang and Carreira-Perpiñán, 2012) solves this by optimizing instead $\sum_{n=1}^{N} \|\mathbf{z}_n - \mathbf{F}(\mathbf{x}_n)\|^2 + \|\mathbf{x}_n - \mathbf{f}(\mathbf{z}_n)\|^2$ jointly over $\mathbf{Z}$, $\mathbf{f}$ and the projection mapping $\mathbf{F}$. This can be seen as a truncated version of our quadratic-penalty approach, where μ is kept constant, and limited to a single-hidden-layer net. The resulting estimate for the nested mapping $\mathbf{f}(\mathbf{F}(\mathbf{x}))$ does not minimize the nested error.

In summary, these works typically sought to have explicit control on the internal representations or features but did not solve the nested problem (1). None of them realize the possibility of using heterogeneous architectures with layer-specific algorithms, or of learning the architecture itself by minimizing a model selection criterion that separates in the $\mathbf{W}$-step. In MAC, the auxiliary coordinates are purely a mathematical construct to solve a well-defined, general nested optimization problem, with embarrassing parallelism suitable for distributed computation, and is not necessarily related to learning good hidden representations.

Finally, the MAC formulation is similar in spirit to problem transformations used with the alternating direction method of multipliers (ADMM) (Boyd et al., 2011). Specifically, in consensus problems one splits an existing variable that appears in multiple, additive terms of the objective function (which then decouple) rather than a functional nesting, for example $\min_{\mathbf{x}} f(\mathbf{x}) + g(\mathbf{x})$ becomes $\min_{\mathbf{x},\mathbf{y}} f(\mathbf{x}) + g(\mathbf{y})$ s.t. $\mathbf{x} = \mathbf{y}$, or $\mathbf{x}$ is split into non-negative and non-positive parts. In contrast, MAC introduces new variables to break the nesting. ADMM is known to be very simple, effective and parallelizable, and able to achieve a pretty good estimate pretty fast, thanks to the decoupling introduced and the ability to use existing optimizers for the subproblems that arise. MAC also gives these characteristics to problems involving function nesting.

3 Experiments

We have successfully tested MAC with a variety of models and datasets. Here we report a subset of experiments which illustrate the basic properties of MAC: how it can learn a homogeneous architecture (deep sigmoidal autoencoder), a heterogeneous architecture (RBF autoencoder combining k-means- and gradient-based steps), and the heterogeneous architecture itself. In all cases, we show the speedup achieved with a parallel implementation of MAC as well. All experiments were run in the same computer using a single processor except the parallel MAC ones.

The $\mathbf{W}$- and $\mathbf{Z}$-steps decouple into subproblems for each hidden unit $\mathbf{w}_{kh}$ and auxiliary vector $\mathbf{z}_n$, respectively. In this paper (except for the MAC minibatch experiment) we solve each of the subproblems (a nonlinear least-squares fit) with a Gauss-Newton approach (Nocedal and Wright, 2006), which essentially approximates the Hessian by linearizing the objective function, solves a linear system to get a search direction, does a line search (we use backtracking with initial step size 1), and iterates. In practice 1–2 iterations converge with high tolerance.

Our parallel implementation of MAC/QP is extremely simple at present, yet it achieves large speedups (about $6\times$ faster if using 12 processors), which are nearly linear as a function of the number of processors for all experiments,

as shown in fig. 5. We used the Matlab Parallel Processing Toolbox. The programming effort is insignificant: all we do is replace the "for" loop over weight vectors (in the **W**-step) or over auxiliary coordinates (in the **Z**-step) with a "parfor" loop. Matlab then sends each iteration of the loop to a different processor. We ran this in a shared-memory multiprocessor machine using up to 12 processors (a limit imposed by our Matlab license). While simple, the Matlab Parallel Processing Toolbox is quite inefficient. Larger speedups may be achievable with other parallel computation models such as MPI in C, and using a distributed architecture (so that cache and other overheads are reduced).

3.1 Homogeneous training: deep sigmoidal autoencoder

We use the USPS dataset, containing 16×16 grayscale images of handwritten digits, i.e., 256D vectors with values in $[0, 1]$. We use $N = 5\,000$ images for training and $5\,000$ for validation, both randomly selected equally over all digits. We train a deep autoencoder architecture 256–300–100–20–100–300–256, for a total of over $200\,000$ weights, with all $K = 5$ hidden layers being logistic sigmoid units and the output layer being linear, trained to minimize the squared reconstruction error. The initial weights are uniformly sampled from $[-1/\sqrt{f_k}, 1/\sqrt{f_k}]$ for layers $k = 1, \ldots, K$, resp., where f_k is the input dimension (fan-in) to each layer (Orr and Müller, 1998). We used MAC/QP introducing auxiliary coordinates for each hidden unit at each layer. For each value of μ, we optimize $E_Q(\mathbf{W}, \mathbf{Z}; \mu)$, exiting when the value of the nested error $E_1(\mathbf{W})$ on a validation set increases or decreases less than a tolerance of 10^{-2} in relative terms. We then increase μ to 10μ.

We compare with two classical backprop-based techniques, stochastic gradient descent and conjugate gradients. Stochastic gradient descent (SGD) has several parameters that should be carefully set by the user to ensure that convergence occurs, and that convergence is as fast as possible. We did a grid search for the minibatch size in $\{1, 10, 20, 50, 100, 200, 500, 1000, 5000\}$ and learning rate in $\{1, 10, 100, 1000\} \times 10^{-7}$. We found minibatches of 20 samples and a learning rate of 10^{-6} were best. We randomly permute the training set at the beginning of each epoch. For nonlinear conjugate gradients (CG), we used the Polak-Ribière version, widely regarded as the best (Nocedal and Wright, 2006). We use Carl Rasmussen's implementation `minimize.m`, which uses a line search based on cubic interpolation that is more sophisticated than backtracking, and allows steps longer than 1. We found a minibatch of N (i.e., batch mode) worked best, and restarts every 100 steps. While it is possible to fine-tune the performance of SGD, CG and MAC even further, our results seem representative of each method's behavior.

Figure 2 plots the mean squared training error for the nested objective function (1) vs run time for the different algorithms. The validation error follows closely the training error and is not plotted. The learning curve of the parallel version of MAC (using 12 processors) is also shown in blue. SGD and CG need many iterations to decrease the error, but each MAC/QP iteration achieves a large decrease, particularly at the beginning, so that it can reach a pretty good network pretty fast. While MAC/QP's serial performance is already remarkable, its parallel implementation achieves a linear speedup on the number of processors (fig. 5).

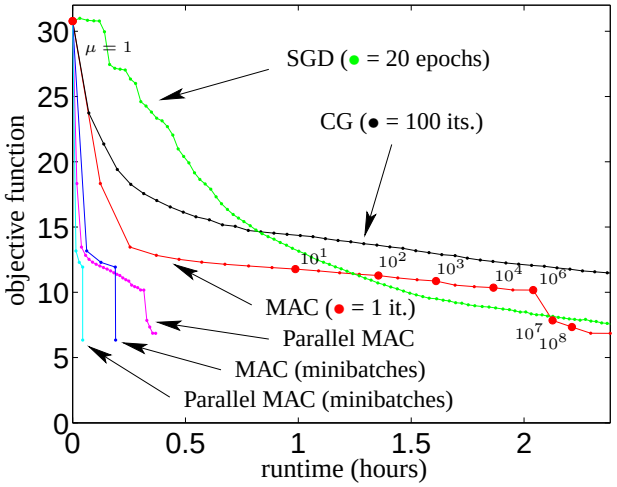

Figure 2: Deep autoencoder with USPS handwritten digit images: nested function error (1) for each algorithm, with markers shown every iteration (MAC) or every epoch (MAC minibatches), every 100 iterations (CG) or every 20 epochs (SGD); one epoch is one pass over the training set. For MAC/QP, we incremented the penalty parameter μ at the beginning of each iteration having a red solid marker.

Finally, we illustrate how MAC can benefit from using minibatches (as in SGD) and inexact steps. We modified the **W** and **Z** steps as follows. Using minibatches of size 100, we run the **Z** step on a minibatch and then run the **W** step on that minibatch (using 5 iterations of CG). After that, we switch to the next minibatch. The blue curve shows the result, where each marker denotes the error after one epoch. The first epoch reduces the error to that of several epochs of batch MAC. We run 3 epochs and then applied a postprocessing step, reaching even a better error than with the batch MAC. The cyan curve shows the parallel MAC minibatch. Thus, SGD should not be viewed as a different, competing approach to MAC, but as an optimization technique that can be used within the MAC steps.

3.2 Heterogeneous Training: RBF Autoencoder

We use the COIL–20 image dataset (Nene et al., 1996), containing rotation sequences of 20 different objects every 5 degrees (72 images per object), each a 32×32 grayscale image with pixel intensity in $[0, 1]$. Thus, the data contain 20 closed, nonlinear 1D manifolds in a $1\,024$–dimensional space. We pick half of the images from objects 1 (duck) and 4 (cat) as validation set, which leaves a training set of $N = 1\,368$ images.

We train an autoencoder architecture to minimize the squared reconstruction error, defined not to use gradients during training, as follows. The bottleneck layer of low-dimensional codes has only 2 units. Both the encoder and the decoder are radial basis function (RBF) networks, each having a single hidden layer. The first one (encoder) has the form $\mathbf{z} = \mathbf{f}_2(\mathbf{f}_1(\mathbf{x};\mathbf{W}_1);\mathbf{W}_2) = \mathbf{W}_2\,\mathbf{f}_1(\mathbf{x};\mathbf{W}_1)$, where the vector $\mathbf{f}_1(\mathbf{x};\mathbf{W}_1)$ has $M_1 = 1\,368$ elements (basis functions) of the form $\exp\left(-\|(\mathbf{x}-\mathbf{w}_{1i})/\sigma_1\|^2\right)$, $i = 1,\dots,M_1$, with $\sigma_1 = 4$, and maps an image $\mathbf{x}$ to a 2D space $\mathbf{z}$. The second one (decoder) has the form $\mathbf{x}' = \mathbf{f}_4(\mathbf{f}_3(\mathbf{z};\mathbf{W}_3);\mathbf{W}_4) = \mathbf{W}_4\,\mathbf{f}_3(\mathbf{z};\mathbf{W}_3)$, where the vector $\mathbf{f}_3(\mathbf{z};\mathbf{W}_3)$ has $M_3 = 1\,368$ elements (basis functions) of the form $\exp\left(-\|(\mathbf{x}-\mathbf{w}_{3i})/\sigma_3\|^2\right)$, $i = 1,\dots,M_3$, with $\sigma_3 = 0.5$, and maps a 2D point $\mathbf{z}$ to a $1\,024$D image. Thus, the complete autoencoder is the concatenation of the two Gaussian RBF networks, it has $K = 3$ hidden layers with sizes 1024–1368–2–1368–1024, and a total of almost 3 million weights. As is usual with RBF networks, we applied a quadratic regularization to the linear-layer weights with a small value ($\lambda_2 = \lambda_4 = 10^{-3}$). The nested problem is then to minimize the following objective function, which is a least-squares error plus a quadratic regularization on the linear-layer weights:

$$E_1(\mathbf{W}) = \frac{1}{2}\sum_{n=1}^{N}\|\mathbf{y}_n - \mathbf{f}(\mathbf{x}_n;\mathbf{W})\|^2 + \lambda_2\|\mathbf{W}_2\|_F^2 + \lambda_4\|\mathbf{W}_4\|_F^2$$
$$\mathbf{f}(\mathbf{x};\mathbf{W}) = \mathbf{f}_4(\mathbf{f}_3(\mathbf{f}_2(\mathbf{f}_1(\mathbf{x};\mathbf{W}_1);\mathbf{W}_2);\mathbf{W}_3);\mathbf{W}_4). \quad (4)$$

In practice, RBF networks are trained in two stages (Bishop, 2006). Consider the encoder, for example. First, one trains the centers $\mathbf{W}_1$ using a clustering algorithm applied to the inputs $\{\mathbf{x}_n\}_{n=1}^N$, typically k-means or (when the number of centers is large) simply by fixing them to be a random subset of the inputs. Second, having determined $\mathbf{W}_1$, one obtains $\mathbf{W}_2$ from a linear least-squares problem, by solving a linear system. The reason why this is preferred to a fully nonlinear optimization over centers $\mathbf{W}_1$ and weights $\mathbf{W}_2$ is that it achieves near-optimal nets with a simple, noniterative procedure. This type of two-stage noniterative strategy to obtain nonlinear networks is widely applied beyond RBF networks, for example with support vector machines, kernel PCA, sliced inverse regression and others (Hastie et al., 2009). We wish to capitalize on this attractive property to train deep autoencoders constructed by concatenating RBF networks. However, backprop-based algorithms are incompatible with this two-stage training procedure, since it does not use derivatives to optimize over the centers. This leads us to the two following optimization methods: an alternating optimization approach, and MAC.

We define the MAC-constrained problem as follows. We introduce auxiliary coordinates only at the coding layer (rather than at all $K = 3$ hidden layers). This allows the $\mathbf{W}$-step to become the desired k-means plus linear system training for the encoder and decoder separately. It requires no programming effort; we simply call an existing, k-means-based RBF training algorithm for each of the encoder and decoder separately. We start with $\mu = 1$ and increase it to $\mu = 5$ after 70 iterations.

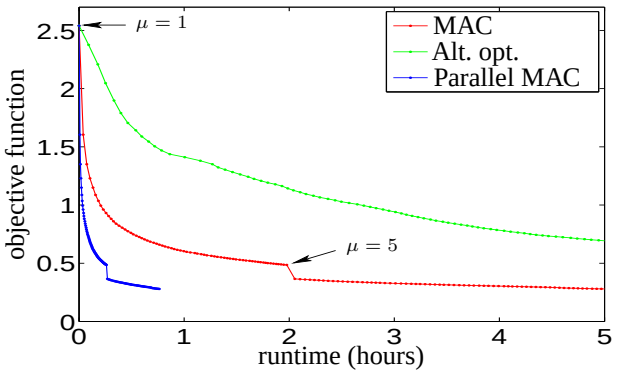

Figure 3: RBF autoencoder with COIL–20 images: nested function error (4) for each algorithm, with markers shown every iteration. Other details as in fig. 2.

We can use an alternating optimization approach directly on the nested problem by alternating the following two steps: (1) A step where we fix $(\mathbf{W}_1,\mathbf{W}_2)$ and train $(\mathbf{W}_3,\mathbf{W}_4)$, by applying k-means to $\mathbf{W}_3$ and a linear system to $\mathbf{W}_4$. This step is identical to the $\mathbf{W}$-step in MAC over $(\mathbf{W}_1,\mathbf{W}_2)$. (2) A step where we fix $(\mathbf{W}_3,\mathbf{W}_4)$ and train $(\mathbf{W}_1,\mathbf{W}_2)$, by applying k-means to $\mathbf{W}_1$ and a nonlinear optimization to $\mathbf{W}_2$ (we use nonlinear conjugate gradients with 10 steps). This is because $\mathbf{W}_2$ no longer appears linearly in the objective function, but is nonlinearly embedded as the argument of the decoder. This step is significantly slower than the $\mathbf{W}$-step in MAC over $(\mathbf{W}_3,\mathbf{W}_4)$. Since we use as many centers as data points ($M_1 = M_3 = N$), the k-means step simplifies (also for MAC) to setting each basis function center to an input point.

In this experiment, instead of using random initial weights, we obtained initial values for the $\mathbf{Z}$ coordinates by running a nonlinear embedding method based on the $N \times N$ Gaussian similarity values between every pair of COIL images.

Fig. 3 shows the nested function error (4) per data point. As before, MAC/QP achieves a large error decrease in a few iterations. Alternating optimization is much slower. The parallel implementation of MAC achieves a large speedup.

3.3 Learning the Architecture: RBF Autoencoder

Now we jointly learn the architecture of the RBF encoder and decoder by trying 50 different values for the number of basis functions in each (a search space of $50^2 = 2\,500$ architectures). We define the following objective function over architectures and their weights:

$$\bar{E}(\mathbf{W}) = E_1(\mathbf{W}) + C(\mathbf{W}) \quad (5)$$

where $E_1(\mathbf{W})$ is the nested error from eq. (4) (including regularization terms), and $C(\mathbf{W}) = C(\mathbf{W}_1) + \cdots + C(\mathbf{W}_4)$ is the model selection term. We use the AIC criterion (Hastie et al., 2009), defined as $\bar{E}(\boldsymbol{\Theta}) = \text{SSE}(\boldsymbol{\Theta}) +$

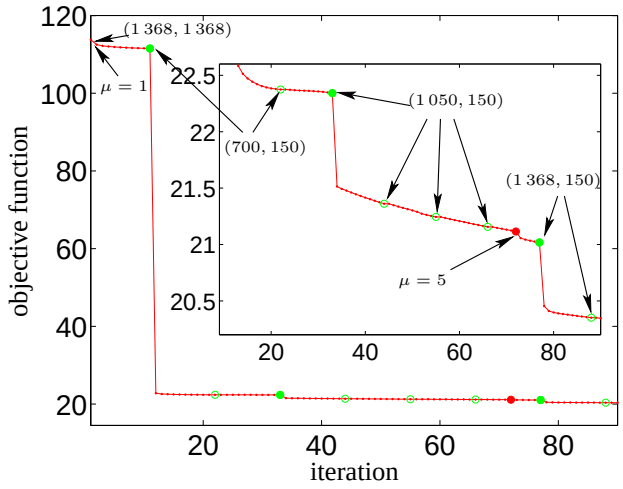

Figure 4: Learning the architecture of the RBF autoencoder of fig. 3 using MAC. We show the total error $E_1(\mathbf{W}) + C(\mathbf{W})$ (the nested function error (1) plus the model cost) per point. Model selection steps are run every 10 iterations, shown with green markers (solid if the architecture changes and empty if it does not change). Other details as in fig. 3.

$C(\boldsymbol{\Theta})$ (times a constant $\frac{1}{N}$, which we omit). SSE($\boldsymbol{\Theta}$) is the sum of squared errors achieved with a model having parameters $\boldsymbol{\Theta}$ in the training set. And $C(\boldsymbol{\Theta}) = 2\epsilon^2 |\boldsymbol{\Theta}|$, where ϵ^2 is the mean squared error achieved by a low-bias model (typically estimated by training a model with many parameters) and $|\boldsymbol{\Theta}|$ is the number of free parameters in the model. In our case, this equals $|\mathbf{W}| = (D+L)(M_1+M_3)$ (centers and linear weights), where M_1 and M_3 are the numbers of centers for the encoder and decoder, resp. (first and third hidden layers), and $D = 1\,024$ and $L = 2$ are the input and output dimension of the encoder, resp. (equivalently, the output and input dimension of the decoder).

We choose each of the numbers of centers M_1 and M_3 from a discrete set consisting of the 50 equispaced values in the range 150 to 1 368 (a total of $50^2 = 2\,500$ different architectures). We estimated $\epsilon^2 = 0.05$ from the result of the RBF autoencoder of section 3.2, which had a large number of parameters and thus a low bias. As in that section, the centers of each network are constrained to be equal to a subset of the input points (chosen at random). We set $\sigma_1 = 4$, $\sigma_3 = 2.5$ and $\lambda_1 = \lambda_2 = 10^{-3}$. We start the MAC/QP optimization from the most complex model, having $M_1 = M_3 = 1\,368$ centers (i.e., the model of the previous section). While every iteration optimizes the MAC/QP objective function (3) over $(\mathbf{W}, \mathbf{Z})$, we run a model selection step only every 10 iterations. This selects separately for each net the best M_k value and potentially changes the size of $\mathbf{W}$. Thus, every 11th iteration is a model selection step, which may or may not change the architecture.

Figure 4 shows the total error $\bar{E}(\mathbf{W}) = E_1(\mathbf{W}) + C(\mathbf{W})$ of eq. (5) (the nested function error plus the model cost). Model selection steps are indicated with green markers (solid if the architecture did change and empty if it did not change), annotated with the resulting value of (M_1, M_3). The first change of architecture moves to a far smaller model $(M_1, M_3) = (700, 150)$, achieving an enormous decrease in objective. This is explained by the strong penalty that AIC imposes on the number of parameters, favoring simpler models. Then, this is followed by a few minor changes of architecture interleaved with a continuous optimization of its weights. The final architecture has $(M_1, M_3) = (1\,368, 150)$, for a total of 1.5 million weights. While this architecture incurs a larger training error than that of the previous section, it uses a much simpler model and has a lower value for the overall objective function of eq. (5). Because, early during the optimization, MAC/QP settles on an architecture that is quite smaller than the one used in fig. 3, the result is in fact achieved in even less time. And, again, the parallel implementation is trivial and achieves an approximately linear speedup on the number of processors (fig. 5).

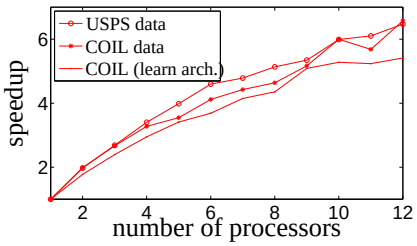

Figure 5: Parallel processing speedup of MAC/QP vs the number of processors for the experiments of figures 2–4.

4 Conclusion

MAC drastically facilitates, in runtime and human effort, the practical design and estimation of nonconvex, nested problems by jointly optimizing over all parameters, reusing existing algorithms (possibly not based on gradients), searching automatically over architectures and affording massively parallel computation, while provably converging to a solution of the nested problem. It could replace or complement backprop-based algorithms in learning nested systems both in the serial and parallel settings. An important application may be the joint, automatic tuning of all stages of a complex processing such as those in computer vision and speech, in a distributed cloud computing environment. A more ambitious goal would be to allow non-experts to construct and deploy complex nested systems by simply combining existing modules (such as neural nets, RBFs or SVMs) in a LEGO-like way and having a "compiler" automatically derive a MAC formulation and map it onto a target multiprocessor architecture. MAC also opens many questions, such as the optimal way to introduce auxiliary coordinates in a given problem, the choice of specific algorithms to optimize the $\mathbf{W}$- and $\mathbf{Z}$-steps, and extensions to deep belief nets, recurrent nets and other models.

Acknowledgments

Work funded in part by NSF CAREER award IIS–0754089.

References

Y. Bengio and Y. LeCun. Scaling learning algorithms toward AI. In L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, editors, *Large-Scale Kernel Machines*, Neural Information Processing Series, pages 321–360. MIT Press, 2007.

Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. Greedy layer-wise training of deep networks. In B. Schölkopf, J. Platt, and T. Hofmann, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 19, pages 153–160. MIT Press, Cambridge, MA, 2007.

J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *J. Machine Learning Research*, 13:281–305, 2012.

C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer Series in Information Science and Statistics. Springer-Verlag, Berlin, 2006.

S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2011.

M. Á. Carreira-Perpiñán and Z. Lu. Dimensionality reduction by unsupervised regression. In *Proc. of the 2008 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'08)*, Anchorage, AK, June 23–28 2008.

M. Á. Carreira-Perpiñán and Z. Lu. Parametric dimensionality reduction by unsupervised regression. In *Proc. of the 2010 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'10)*, pages 1895–1902, San Francisco, CA, June 13–18 2010.

M. Á. Carreira-Perpiñán and Z. Lu. Manifold learning and missing data recovery through unsupervised regression. In *Proc. of the 12th IEEE Int. Conf. Data Mining (ICDM 2011)*, pages 1014–1019, Vancouver, BC, Dec. 11–14 2011.

M. Á. Carreira-Perpiñán and W. Wang. Distributed optimization of deeply nested systems. Unpublished manuscript, arXiv:1212.5921, Dec. 24 2012.

E. Castillo, B. Guijarro-Berdiñas, O. Fontenla-Romero, and A. Alonso-Betanzos. A very fast learning method for neural networks based on sensitivity analysis. *J. Machine Learning Research*, 7:1159–1182, July 2006.

L. Deng, D. Yu, and J. Platt. Scalable stacking and learning for building deep architectures. In *Proc. of the IEEE Int. Conf. Acoustics, Speech and Sig. Proc. (ICASSP'12)*, pages 2133–2136, Kyoto, Japan, Mar. 25–30 2012.

D. Erhan, P. A. Manzagol, Y. Bengio, S. Bengio, and P. Vincent. The difficulty of training deep architectures and the effect of unsupervised pre-training. In *Proc. of the 12th Int. Workshop on Artificial Intelligence and Statistics (AISTATS 2009)*, pages 153–160, Clearwater Beach, FL, Mar. 21–24 2009.

A. Gifi. *Nonlinear Multivariate Analysis*. John Wiley & Sons, 1990.

T. Grossman, R. Meir, and E. Domany. Learning by choice of internal representations. *Complex Systems*, 2(5):555–575, 1988.

T. J. Hastie, R. J. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning—Data Mining, Inference and Prediction*. Springer Series in Statistics. Springer-Verlag, second edition, 2009.

G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313 (5786):504–507, July 28 2006.

K. Kavukcuoglu, M. Ranzato, and Y. LeCun. Fast inference in sparse coding algorithms with applications to object recognition. Technical Report CBLL–TR–2008–12–01, Dept. of Computer Science, New York University, Dec. 4 2008.

A. Krogh, C. J. Thorbergsson, and J. A. Hertz. A cost function for internal representations. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems (NIPS)*, volume 2, pages 733–740. Morgan Kaufmann, San Mateo, CA, 1990.

N. Lawrence. Probabilistic non-linear principal component analysis with Gaussian process latent variable models. *J. Machine Learning Research*, 6:1783–1816, Nov. 2005.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11):2278–2324, Nov. 1998.

S. Ma, C. Ji, and J. Farmer. An efficient EM-based training algorithm for feedforward neural networks. *Neural Networks*, 10(2):243–256, Mar. 1997.

S. A. Nene, S. K. Nayar, and H. Murase. Columbia object image library (COIL-20). Technical Report CUCS–005–96, Dept. of Computer Science, Columbia University, Feb. 1996.

J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer-Verlag, New York, second edition, 2006.

B. A. Olshausen and D. J. Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381(6583):607–609, June 13 1996.

G. B. Orr and K.-R. Müller, editors. *Neural Networks: Tricks of the Trade*, volume 1524 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1998.

M. Ranzato, C. Poultney, S. Chopra, and Y. LeCun. Efficient learning of sparse representations with an energy-based model. In B. Schölkopf, J. Platt, and T. Hofmann,

editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 19, pages 1137–1144. MIT Press, Cambridge, MA, 2007.

R. T. Rockafellar. Monotone operators and the proximal point algorithm. *SIAM J. Control and Optim.*, 14(5): 877–898, 1976.

T. Rögnvaldsson. On Langevin updating in multilayer perceptrons. *Neural Computation*, 6(5):916–926, Sept. 1994.

R. Rohwer. The 'moving targets' training algorithm. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems (NIPS)*, volume 2, pages 558–565. Morgan Kaufmann, San Mateo, CA, 1990.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

D. Saad and E. Marom. Learning by choice of internal representations: An energy minimization approach. *Complex Systems*, 4(1):107–118, 1990.

G. Saon and J.-T. Chien. Large-vocabulary continuous speech recognition systems: A look at some recent advances. *IEEE Signal Processing Magazine*, 29(6):18–33, Nov. 2012.

T. Serre, L. Wolf, S. Bileschi, M. Riesenhuber, and T. Poggio. Robust object recognition with cortex-like mechanisms. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 29(3):411–426, Mar. 2007.

S. Tan and M. L. Mavrovouniotis. Reducing data dimensionality through optimizing neural network inputs. *AIChE Journal*, 41(6):1471–1479, June 1995.

W. Wang and M. Á. Carreira-Perpiñán. Nonlinear low-dimensional regression using auxiliary coordinates. In N. Lawrence and M. Girolami, editors, *Proc. of the 15th Int. Workshop on Artificial Intelligence and Statistics (AISTATS 2012)*, pages 1295–1304, La Palma, Canary Islands, Spain, Apr. 21–23 2012.