Optimization in Data Analysis: Some Recent Developments

Stephen Wright (UW-Madison)

VWCO, Vienna, December 2018

Outline

- Context: Data Analysis, Machine Learning (ML), Data Science
- Formulating 13 ML applications as continuous optimization
- Deep Neural Nets (DNN)
- 5 recent issues in ML and optimization:
  1. The "Adam" variant of stochastic gradient.
  2. Discrete optimization for training DNNs.
  3. Adversarial ML
  4. Overparametrization in DNNs.
  5. Reinforcement learning and control

Data Science

Related Terms: AI, Data Analysis, Machine Learning, Statistical Inference, Data Mining.

- Extract meaning from data: Understand statistical properties, learn important features and fundamental structures in the data.
- Use this knowledge to make predictions about other, similar data.

Highly multidisciplinary area!

- Foundations in Statistics;
- Computer Science: AI, Machine Learning, Databases, Parallel Systems, Architectures (GPUs);
- Optimization provides a toolkit of modeling/formulation and algorithmic techniques.

Modeling and domain-specific knowledge is vital: "80% of data analysis is spent on the process of cleaning and preparing the data."
[Dasu and Johnson, 2003].

The Age of Data

New "Data Science Centers" at many institutions, new degree programs (e.g. Undergrad Majors and MS in Data Science), new funding initiatives

- Huge amounts of data are collected, routinely and continuously.
  - Consumer and citizen data: phone calls and text, social media apps, email, surveillance cameras, web activity, online shopping, repositories of survey data and texts,...
  - Scientific data: particle colliders, satellites, biological / genomic, astronomical.
- Powerful computers and new specialized architectures make it possible to handle larger data sets and analyze them more thoroughly.
- Methodological innovations in some areas. e.g. Deep Learning.
  - Speech recognition
  - AlphaGo: Reinforcement Learning for Go
  - Image recognition

Typical Setup

After cleaning and formatting, obtain a data set of m objects:

- Vectors of features: a_j, $j = 1, 2, \ldots, m$.
- Outcome / observation / label y_j for each feature vector.

The outcomes y_j could be:

- a real number: **regression**
- a label indicating that a_j lies in one of M classes (for $M \geq 2$): **classification**
- multiple labels: classify a_j according to multiple criteria.
- no labels (y_j is null):
  - **subspace identification:** Locate low-dimensional subspaces that approximately contain the (high-dimensional) vectors a_j;
  - **clustering**: Partition the a_j into a few clusters.

  (Structure may reveal which features in the a_j are important / distinctive, or enable predictions to be made about new vectors a.)

Fundamental Data Analysis Task

Seek a function ϕ that:

- approximately maps a_j to y_j for each j: $\phi(a_j) \approx y_j$, $j = 1, 2, \ldots, m$.
- (if no labels y_j, or if some labels are missing, ϕ assigns each a_j to a cluster or subspace.)
- satisfies additional properties that make it "plausible" for the application, robust to perturbations in the data, generalizable to other data samples.

Can usually define ϕ in terms of some parameter vector x — thus identification of ϕ becomes a data-fitting problem: Find the best x.

Objective function in this problem often built up of m terms that capture mismatch between predictions and observations for data item (a_j, y_j).

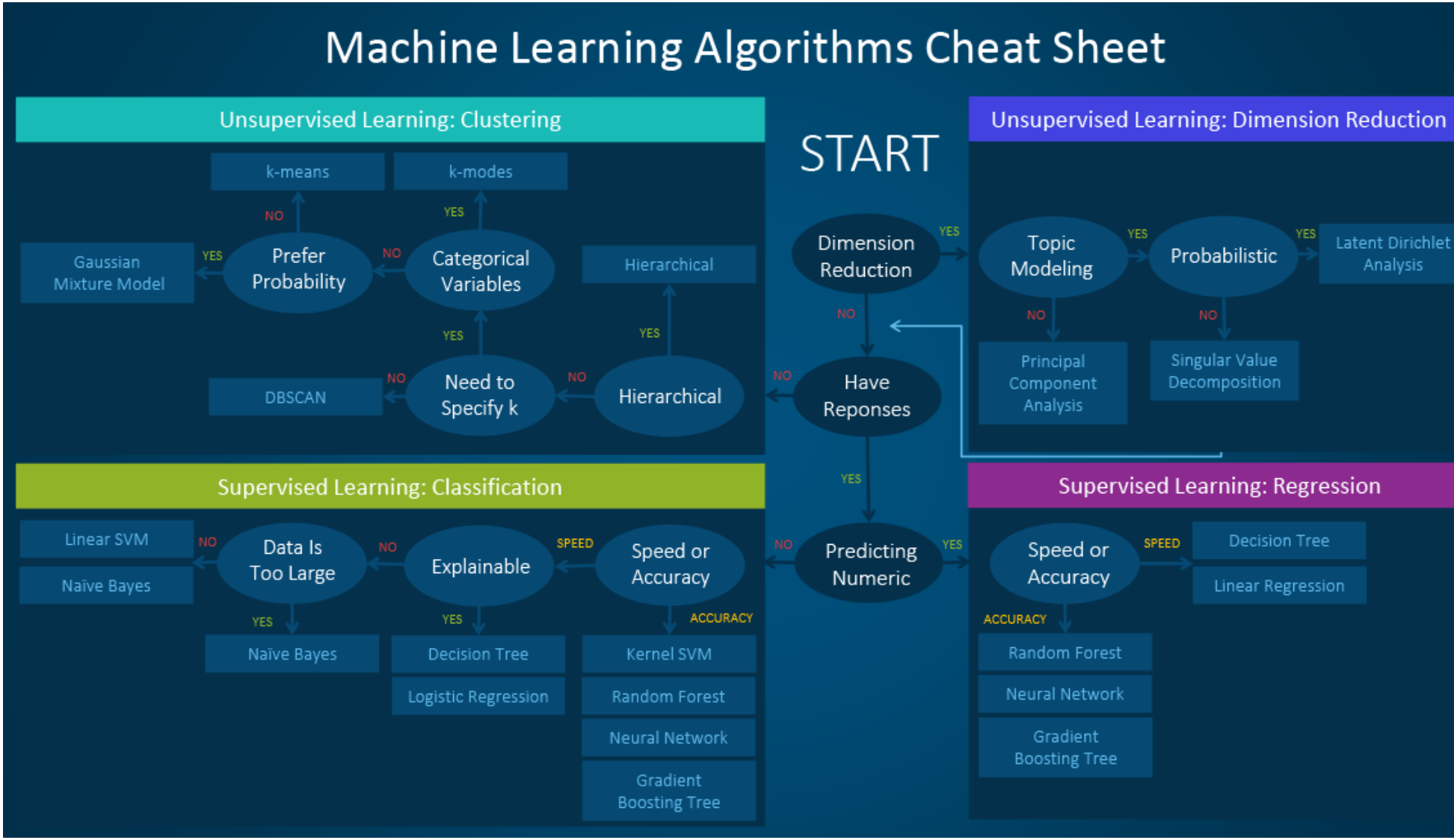
The process of finding ϕ is called learning or training.

What's the use of the mapping ϕ?

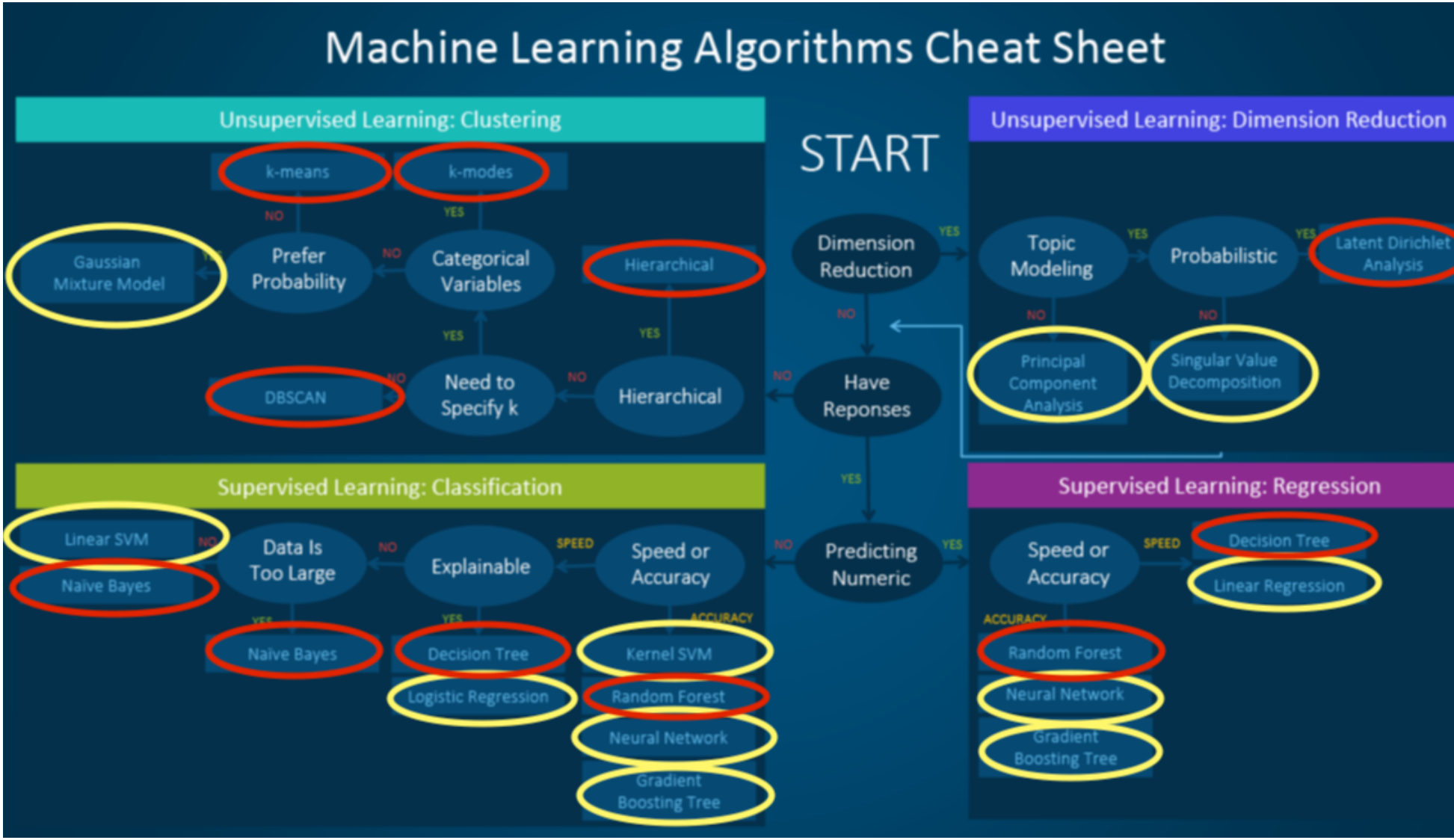
- Prediction: Given new data vectors a_k, predict outputs $y_k \leftarrow \phi(a_k)$.
- Analysis: ϕ — especially the parameter x that defines it — reveals structure in the data. Examples:
  - Feature selection: reveal the components of vectors a_j that are most important in determining the outputs y_j.
  - Uncovers some hidden structure, e.g.
    - ⋆ low-dimensional subspaces that contain the a_j (approx);
    - ⋆ find clusters of a_j's;
    - ⋆ find a decision tree that builds intuition about how y_j depends on a_j.

Many possible complications:

- Noise or errors in a_j and y_j;
- Missing data: elements of a_j and/or y_j;
- Overfitting: ϕ exactly fits the set of training data (a_j, y_j) but predicts poorly on "out-of-sample" data (a_k, y_k).



[1] https://blogs.sas.com/content/subconsciousmusings/2017/04/12/machine-learning-algorithm-use/



There's a lot of continuous optimization here (yellow)!

Application I: (Linear) Least Squares

$$\min_x f(x) := \frac{1}{2}\sum_{j=1}^{m}(a_j^T x - y_j)^2 = \frac{1}{2}\|Ax - y\|_2^2.$$

[Gauss, 1799], [Legendre, 1805]; see [Stigler, 1981].

Here the function mapping data to output is linear: $\phi(a_j) = a_j^T x$.

- ℓ_2 regularization reduces sensitivity of the solution x to noise in y.

$$\min_x \frac{1}{2}\|Ax - y\|_2^2 + \lambda\|x\|_2^2.$$

- ℓ_1 regularization yields solutions x with few nonzeros:

$$\min_x \frac{1}{2}\|Ax - y\|_2^2 + \lambda\|x\|_1.$$

Feature selection: Nonzero locations in x indicate important components of a_j.

Application II: Robust Linear Regression

Least squares assumes Gaussian errors in y_j. When error distributions are otherwise, or contain "outliers," need different formulations.

Use statistics to write down a likelihood function for x given y, then find the maximum likelihood estimate — optimization!

$$\min_x \sum_{j=1}^{m} \ell(a_j^T x - y_j) + \lambda R(x)$$

where ℓ is loss function and R is regularizer.

Can lead to logistic regression (see later), which is convex. But some models lead to nonconvexity in the loss function and/or regularizer term.

- Tukey biweight: $\ell(\theta) = \theta^2/(1+\theta^2)$. Behaves like least squares for θ close to 0, but asymptotes at 1. Outliers don't affect solution much.
- Nonconvex separable regularizers R such as SCAD and MCP behave like $\|\cdot\|_1$ at zero, but flatten out for larger x.

Application III: Matrix Completion

Regression over a structured matrix: Observe a matrix X by probing it with linear operators $\mathcal{A}_j(X)$, giving observations y_j, $j = 1, 2, \ldots, m$.

$$\min_X \frac{1}{2m} \sum_{j=1}^{m} (\mathcal{A}_j(X) - y_j)^2 = \frac{1}{2m} \|\mathcal{A}(X) - y\|_2^2.$$

Each $\mathcal{A}_j$ may observe a single element of X, or a linear combination of elements. Can be represented as a matrix A_j, so that $\mathcal{A}_j(X) = \langle A_j, X \rangle$.

Seek the "simplest" X that satisfies the observations, e.g. low rank.

- Add a nuclear-norm (sum-of-singular-values) regularization term $\lambda \|X\|_*$ for some $\lambda > 0$ [Recht et al., 2010]
- Explicit low-rank parametrization (nonconvex):

$$\min_{L,R} \frac{1}{2m} \sum_{j=1}^{m} (\mathcal{A}_j(LR^T) - y_j)^2.$$

Application IV: Nonnegative Matrix Factorization

Given $m \times n$ matrix Y, seek factors L $(m \times r)$ and R $(n \times r)$ that are element-wise positive, such that $LR^T \approx Y$.

$$\min_{L,R} \frac{1}{2}\|LR^T - Y\|_F^2 \text{ subject to } L \geq 0,\ R \geq 0.$$

Applications in computer vision, document clustering, chemometrics, ...

Could combine with matrix completion, when not all elements of Y are known, if it makes sense on the application to have nonnegative factors.

If positivity constraint were not present, could solve this in closed form with an SVD, since Y is observed completely.

Application V: Sparse Inverse Covariance

Let $Z \in \mathbb{R}^p$ be a (vector) random variable with zero mean. Let $z_1, z_2, \ldots, z_N$ be samples of Z. Sample covariance matrix (estimates covariance between components of Z):

$$S := \frac{1}{N-1} \sum_{\ell=1}^{N} z_\ell z_\ell^T.$$

Seek a sparse inverse covariance matrix: $X \approx S^{-1}$.

X reveals dependencies between components of Z: $X_{ij} = 0$ if the i and j components of Z are *conditionally independent*, i.e. don't influence each other directly.

Obtain X from the regularized formulation:

$$\min_X \langle S, X \rangle - \log\det(X) + \lambda \|X\|_1, \quad \text{where } \|X\|_1 = \textstyle\sum_{i,j} |X_{ij}|.$$

[d'Aspremont et al., 2008, Friedman et al., 2008].

Application VI: Sparse Principal Components (PCA)

Seek sparse approximations to the leading eigenvectors of the sample covariance matrix S.

For the leading sparse principal component, solve

$$\max_{v \in \mathbb{R}^n} v^T S v = \langle S, vv^T \rangle \quad \text{s.t. } \|v\|_2 = 1,\ \|v\|_0 \leq k,$$

for some given $k \in \{1, 2, \ldots, n\}$. Convex relaxation replaces vv^T by an $n \times n$ positive semidefinite proxy M:

$$\max_{M \in S\mathbb{R}^{n \times n}} \langle S, M \rangle \quad \text{s.t. } M \succeq 0,\ \langle I, M \rangle = 1,\ \|M\|_1 \leq R,$$

where $|\cdot|_1$ is the sum of absolute values [d'Aspremont et al., 2007].

Adjust the parameter R to obtain desired sparsity.

Application VII: Sparse + Low-Rank

Given $Y \in \mathbb{R}^{m \times n}$, seek low-rank M and sparse S such that $M + S \approx Y$.

- Robust PCA: Sparse S represents "outlier" observations.
- Foreground-Background separation in video processing.
  - Each column of Y is one frame of video, each row is a single pixel evolving in time.
  - Low-rank part M represents background, sparse part S represents foreground.

Convex formulation [Candès et al., 2011, Chandrasekaran et al., 2011]:

$$\min_{M,S} \|M\|_* + \lambda \|S\|_1 \quad \text{s.t. } Y = M + S.$$

Compact formulation (nonconvex): Variables $L \in \mathbb{R}^{n \times r}$, $R \in \mathbb{R}^{m \times r}$, $S \in \mathbb{R}^{m \times n}$ sparse.

$$\min_{L,R,S} \frac{1}{2} \|LR^T + S - Y\|_F^2 + \lambda \|S\|_1$$

Application VIII: Subspace Identification

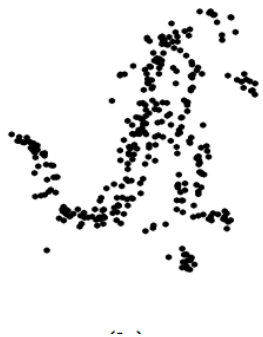
Given vectors $a_j \in R^n$ with missing entries, find a subspace of $\mathbb{R}^n$ such that all "completed" vectors a_j lie approximately in this subspace.

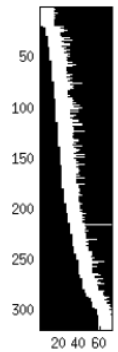
If $\Omega_j \subset \{1, 2, \ldots, n\}$ is the set of observed elements in a_j, seek $X \in \mathbb{R}^{n \times d}$ such that

$$[a_j - X s_j]_{\Omega_j} \approx 0,$$

for some $s_j \in \mathbb{R}^d$ and all $j = 1, 2, \ldots$.
[Balzano et al., 2010, Balzano and Wright, 2014].

Application: Structure from motion. Reconstruct opaque object from planar projections of surface reference points.





Application IX: Linear Support Vector Machines

Each item of data belongs to one of two classes: $y_j = +1$ and $y_j = -1$.

Seek (x, β) such that

$$\begin{aligned} a_j^T x - \beta &\geq 1 \quad \text{when } y_j = +1; \\ a_j^T x - \beta &\leq -1 \quad \text{when } y_j = -1. \end{aligned}$$

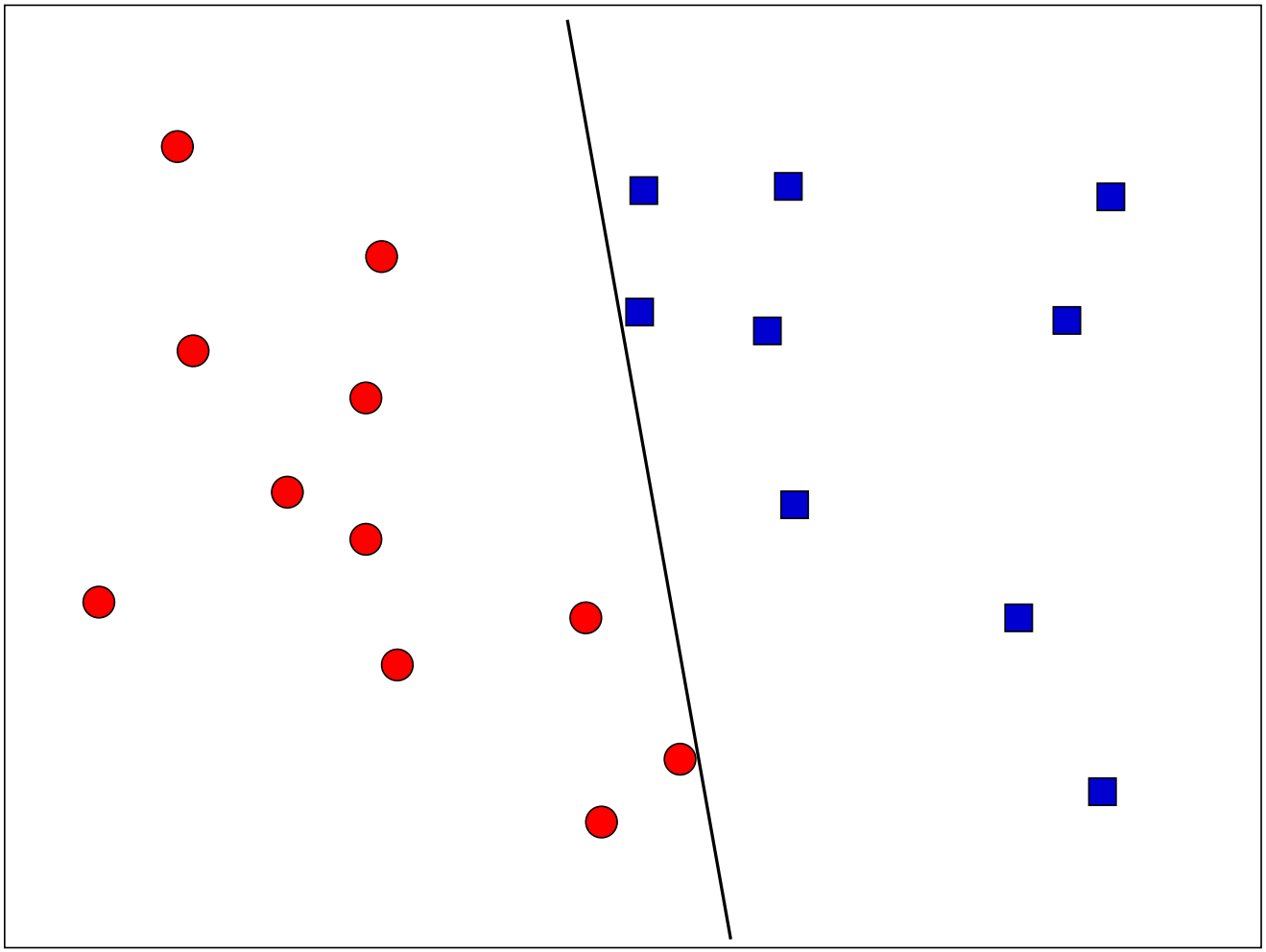
The mapping is $\phi(a_j) = \text{sign}(a_j^T x - \beta)$.

Design an objective so that the jth loss term is zero when $\phi(a_j) = y_j$, positive otherwise. A popular one is hinge loss:

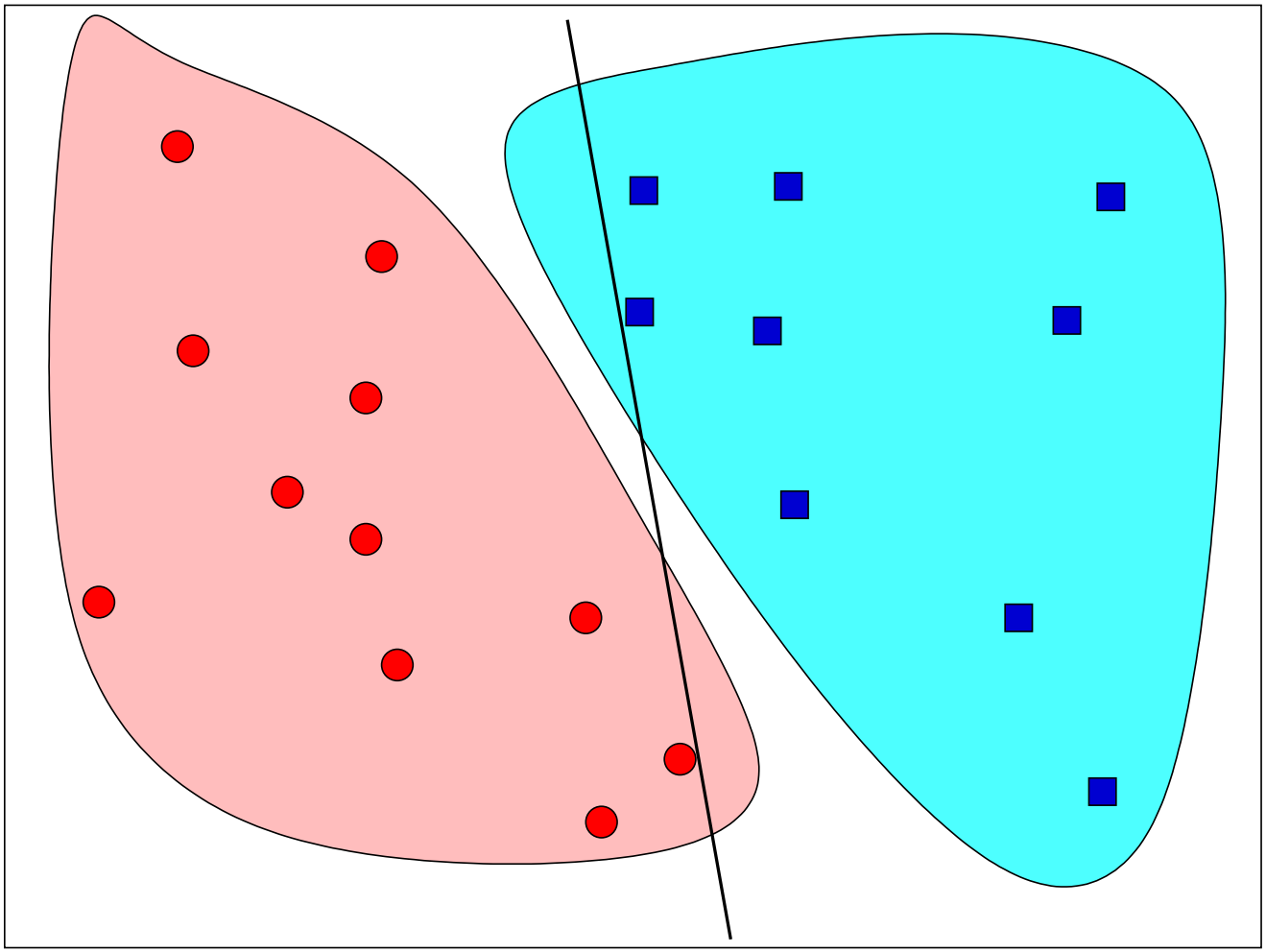
$$H(x, \beta) = \frac{1}{m} \sum_{j=1}^{m} \max(1 - y_j(a_j^T x - \beta), 0).$$

Add a regularization term $(\lambda/2)\|x\|_2^2$ for some $\lambda > 0$ to maximize the margin between the classes.

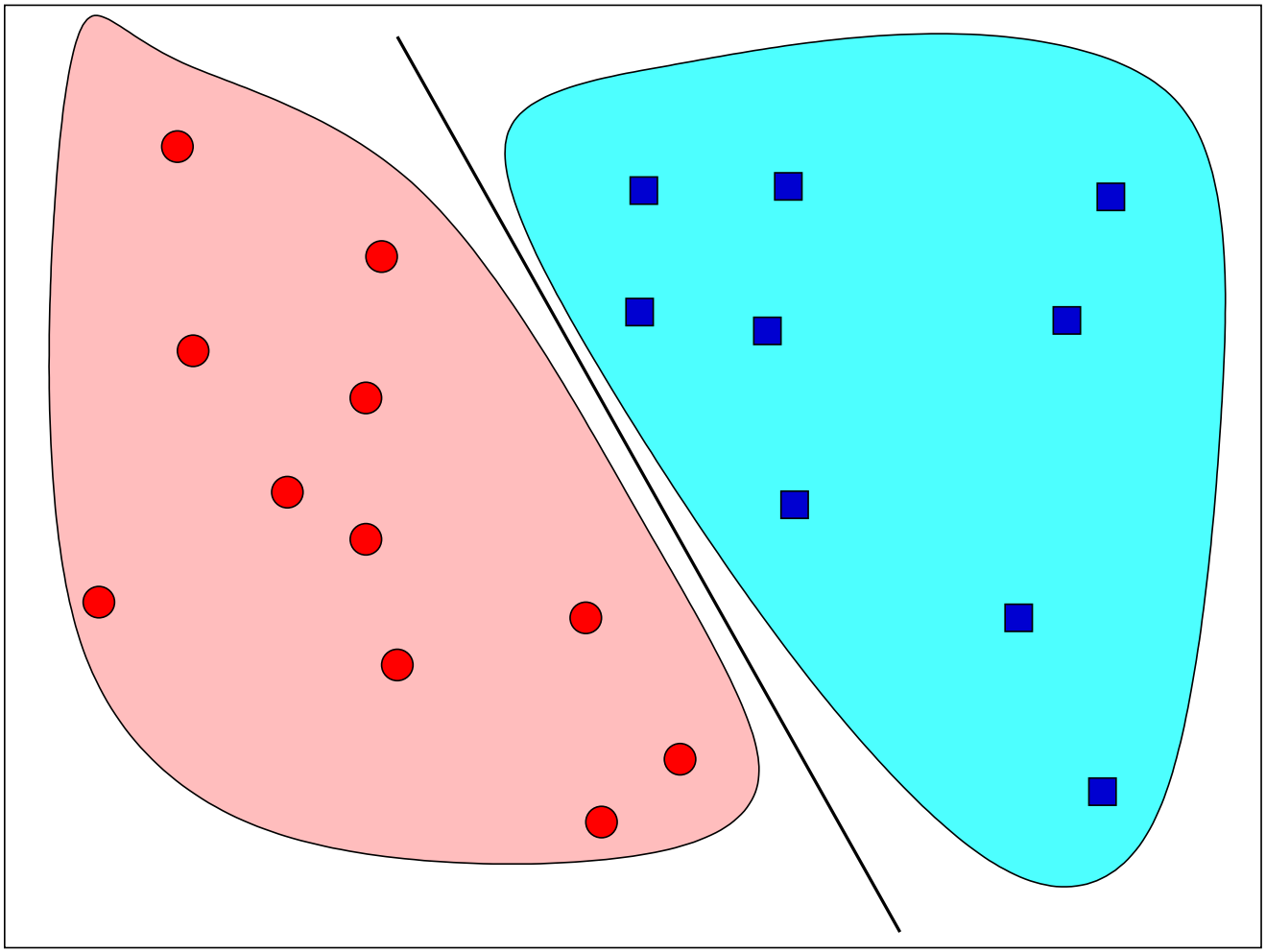
Regularize for Generalizability



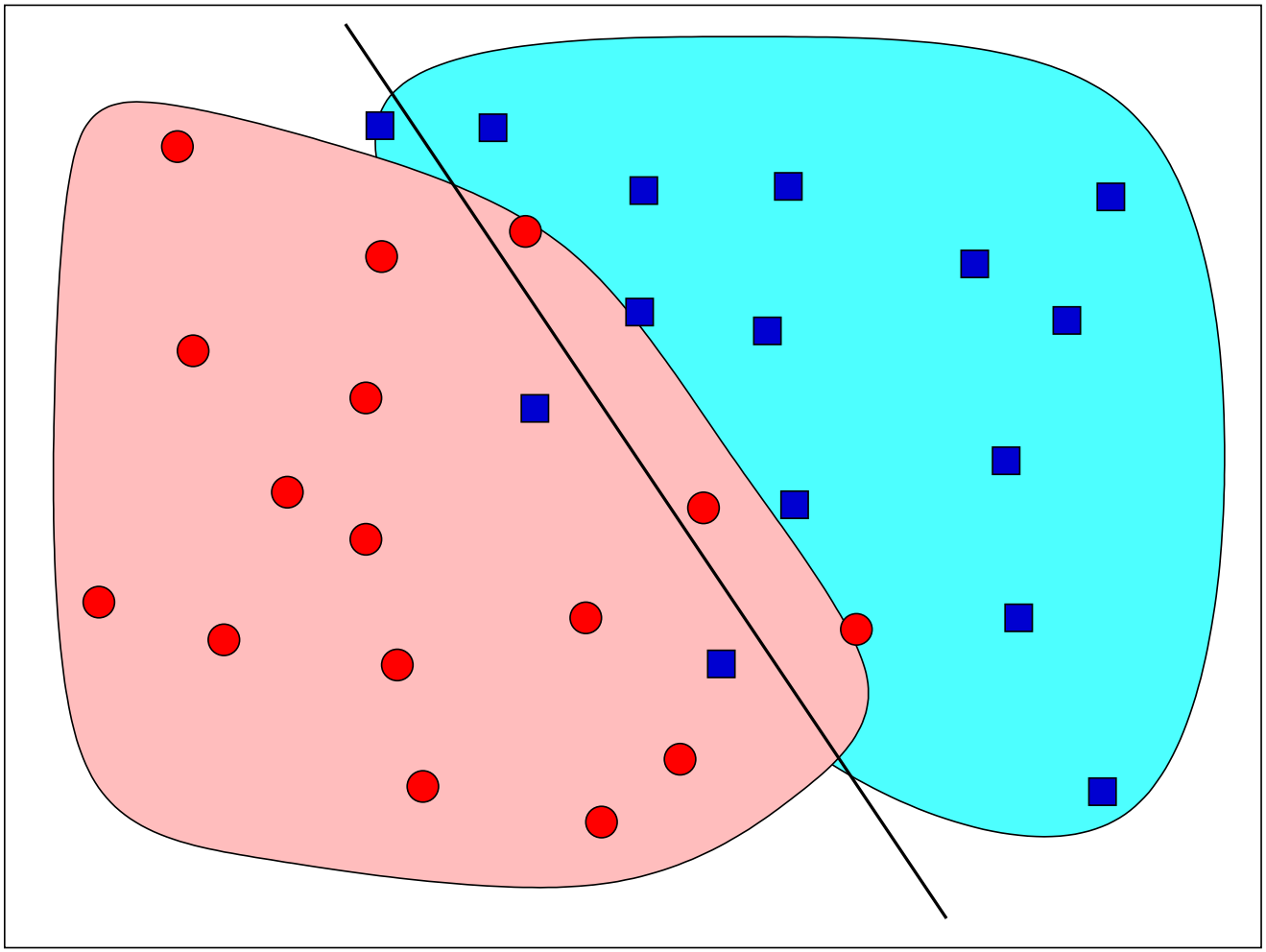
Regularize for Generalizability



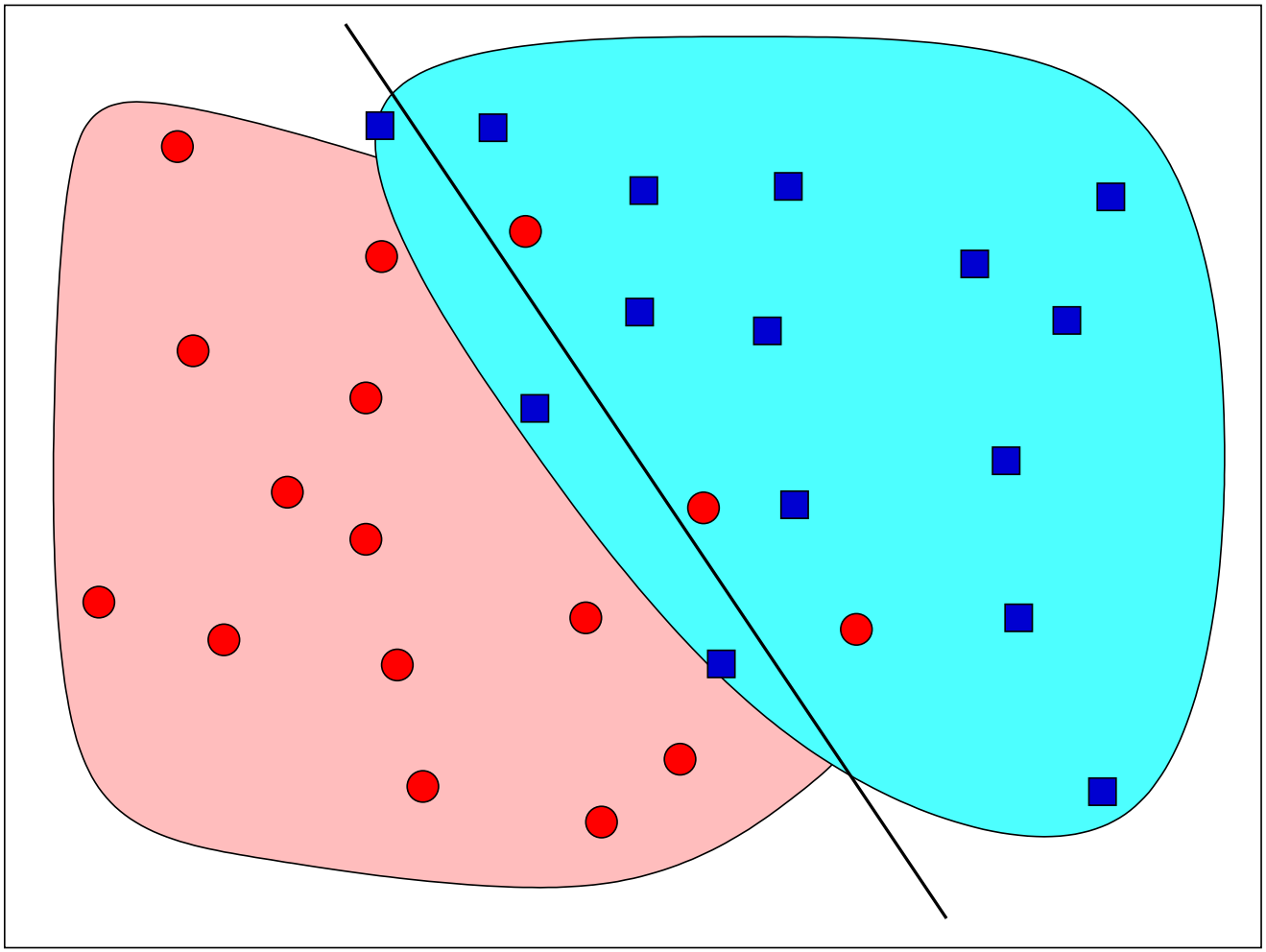
Regularize for Generalizability



Regularize for Generalizability



Regularize for Generalizability



Application X: Kernel SVM

Data a_j, $j = 1, 2, \ldots, m$ may not be *separable* neatly into two classes $y_j = +1$ and $y_j = -1$. Apply a nonlinear transformation $a_j \to \psi(a_j)$ ("lifting") and do linear classification on $(\psi(a_j), y_j)$: Find (x, β) such that

$$\min_{x,\beta} \frac{1}{m} \sum_{j=1}^{m} \max(1 - y_j(\psi(a_j)^T x - \beta), 0) + \frac{1}{2}\lambda \|x\|_2^2.$$

Can avoid defining ψ explicitly by using instead the dual:

$$\min_{\alpha \in \mathbb{R}^m} \frac{1}{2}\alpha^T Q \alpha - e^T \alpha \;\text{ s.t. }\; 0 \le \alpha \le (1/\lambda)e, \;\; y^T \alpha = 0.$$

where $Q_{k\ell} = y_k y_\ell \psi(a_k)^T \psi(a_\ell)$, $y = (y_1, y_2, \ldots, y_m)^T$, $e = (1, 1, \ldots, 1)^T$.

No need to choose $\psi(\cdot)$ explicitly. Instead choose a kernel K, such that

$$K(a_k, a_\ell) \sim \psi(a_k)^T \psi(a_\ell).$$

[Boser et al., 1992, Cortes and Vapnik, 1995]. "Kernel trick."

Application XI: Logistic Regression

Binary logistic regression is similar to binary SVM, except that we seek a function p that gives odds of data vector a being in class 1 or class -1, rather than making a simple prediction.

Seek odds function p parametrized by $x \in \mathbb{R}^n$:

$$p(a; x) := (1 + e^{a^T x})^{-1}.$$

Choose x so that $p(a_j; x) \approx 1$ when $y_j = 1$ and $p(a_j; x) \approx 0$ when $y_j = -1$.

$$\mathcal{L}(x) = -\frac{1}{m}\left[\sum_{y_j=-1} \log(1 - p(a_j; x)) + \sum_{y_j=1} \log p(a_j; x)\right]$$

Can sparsify by including $\lambda\|x\|_1$ in the objective. Generalizes to multiple classes (more than 2): softmax. e.g. identify images, or phonemes in speech. This usually forms the "final layer" of a neural net.

Application XII: Atomic-Norm Regularization

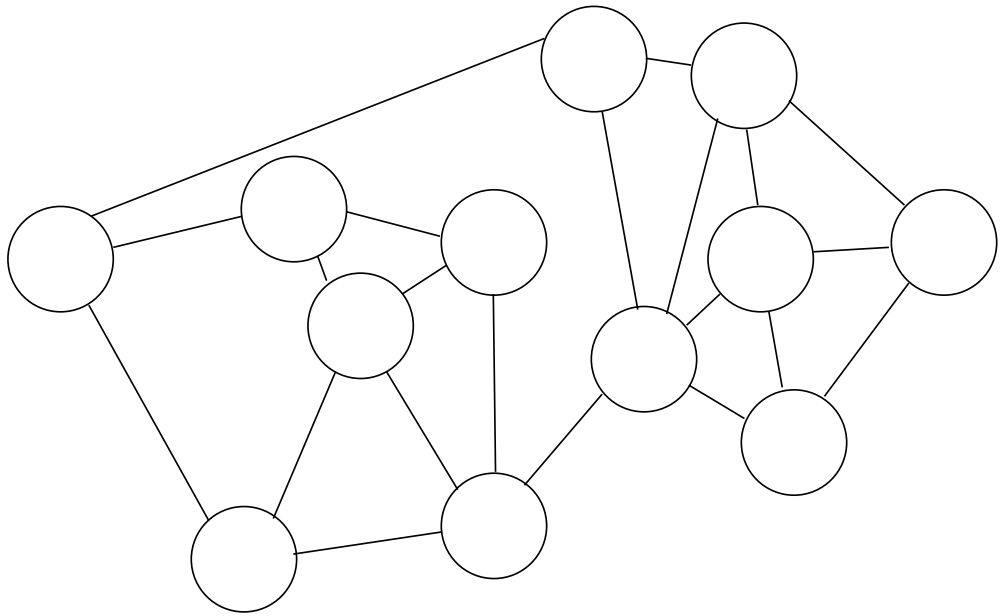
Seek an approx minimizer of $f(x)$ such that x combines a small number of fundamental elements — atoms. Define the atomic norm of x via its "minimal" representation in terms of the set $\mathcal{A}$ of atoms (possibly infinite):

$$\|x\|_{\mathcal{A}} := \inf \left\{ \sum_{a \in \mathcal{A}} c_a \ : \ x = \sum_{a \in \mathcal{A}} c_a a, \ c_a \geq 0 \right\}.$$

- Compressed Sensing: $x \in \mathbb{R}^n$: atoms are $\pm e_i$, where $e_i = (0, 0, \ldots, 0, 1, 0, \ldots, 0)^T$; then $\|x\|_{\mathcal{A}} = \|x\|_1$.
- Low-rank Matrix Problems: $x \in \mathbb{R}^{m \times n}$: atoms are the rank-one matrices (infinite); then $\|\cdot\|_{\mathcal{A}}$ is the nuclear norm.
- Signal processing with few frequencies: $x(t) = \sum_{j=1}^{k} c_j \exp(2\pi i f_j t)$: signal with k frequencies. Atoms are $a_f := \exp(2\pi i f t)$ for any f.
- Image processing using wavelets: Atoms are subtrees of wavelet coefficients.

Application XIII: Community Detection in Graphs

Given an undirected graph, find "communities" (subsets of nodes) such that nodes inside a given community are more likely to be connected to each other than to nodes outside that community.

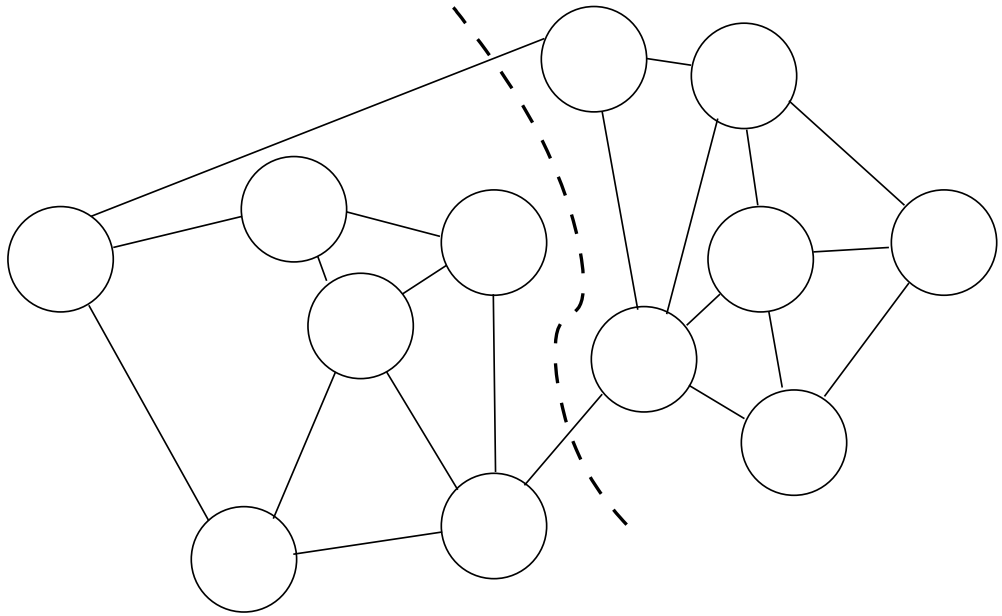


Probability $p \in (0, 1)$ of being connected to a node *within* your community, and $q \in (0, p)$ of being connected to a node *outside* your community.

Leads to a matrix optimization problem: relaxation of a maximum-log-likelihood formulation.

Application XIII: Community Detection in Graphs

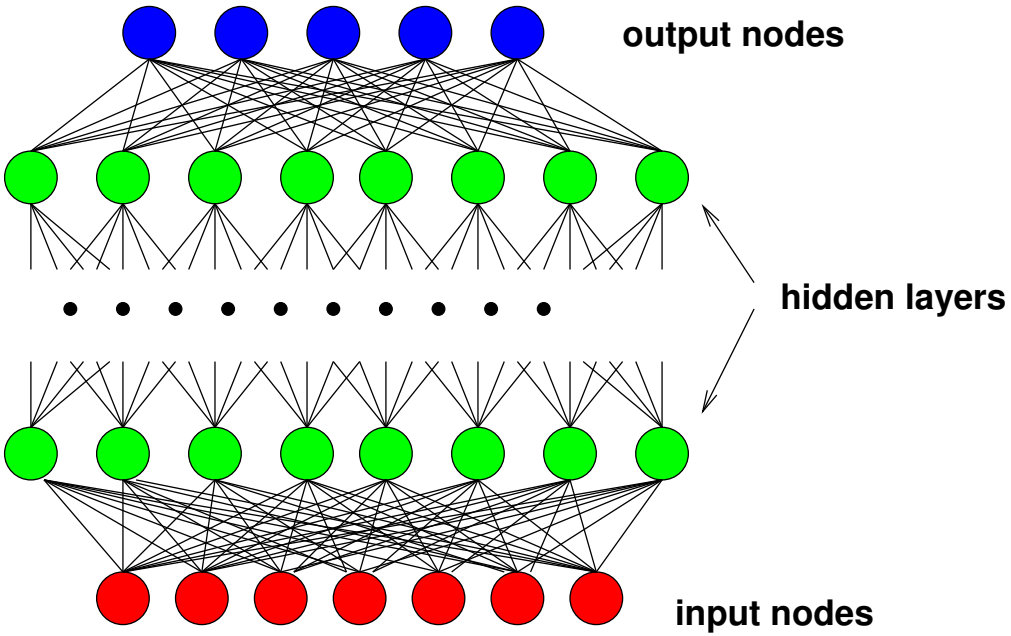
Given an undirected graph, find "communities" (subsets of nodes) such that nodes inside a given community are more likely to be connected to each other than to nodes outside that community.



Probability $p \in (0, 1)$ of being connected to a node *within* your community, and $q \in (0, p)$ of being connected to a node *outside* your community.

Leads to a matrix optimization problem: relaxation of a maximum-log-likelihood formulation.

Deep Learning



Inputs are the vectors a_j, outputs are odds of a_j belonging to each class (as in multiclass logistic regression).

At each layer, inputs are converted to outputs by a linear transformation composed with an element-wise function:

$$a^{\ell+1} = \sigma(W^\ell a^\ell + b^\ell),$$

where a^ℓ is node values at layer ℓ, (W^ℓ, b^ℓ) are *parameters* in the network, σ is the element-wise function.

Deep Learning

The element-wise function σ makes transformations to scalar input. Nowadays, the ReLU / hinge function is almost always used:

$$t \to \max(t, 0).$$

DNN architectures are engineered to the application (speech processing, object recognition, ...).

- local aggregation of inputs: pooling;
- restricted connectivity + constraints on weights (elements of W^ℓ matrices): convolutions.
- connections that skip a layer: ResNet. Each layer fits the "residual" of the fit from the layer below.

Training Deep Learning Networks

The network contains many parameters — (W^ℓ, b^ℓ), $\ell = 1, 2, \ldots, L$ in the notation above — that must be selected by training on the data (a_j, y_j), $j = 1, 2, \ldots, m$. Objective has the form:

$$\frac{1}{m} \sum_{j=1}^{m} h(x; a_j, y_j)$$

where $x = (W^1, b^1, W^2, b^2, \ldots)$ are the parameters in the model and h measures the mismatch between observed output y_j and the outputs produced by the model (as in multiclass logistic regression).

Number of parameters (elements in x) is often vastly greater than the number of data points — overparametrization.

Nonlinear, Nonconvex, usually Nonsmooth.

Many software packages available for training: Caffe, PyTorch, Tensor Flow, Theano,... Many run on GPUs.

DNN Training: Stochastic Gradient

DNNs are trained almost exclusively with some variant of stochastic gradient (SGD). Steps have the form $x_{k+1} \leftarrow x_k - \alpha_k g_k$, where

$$g_k := \frac{1}{B_k} \sum_{j \in B_k} \nabla_x h(x^k; a_j, y_j),$$

and $B_k \subset \{1, 2, \ldots, m\}$ is a randomly sampled "batch."

- Choice of B_k (large or small) and its effect on algorithm speed and quality of outcome is still being actively investigated!
- Choice of α_k is called "hyper-parameter optimization" - still investigated intensely.
- Momentum terms sometimes help: add $\beta_k(x_k - x_{k-1})$.

Results are evaluated not by success in reducing the objective, but by prediction performance on a test set similar to the training set.

Issue 1: Adam!

In early 2015 the paper [Kimgma and Ba, 2015] appeared, describing a modification of SGD which applies a diagonal scaling to the update g_k.

The scaling is computed from a weighted average of components of g_k, and their squares, from previous iterations.

There is convergence theory but it is unclear.

It has become a juggernaut, with over 16,000 citations (as of yesterday). Apparently it has some practical relevance.

Issue 2: DNN Training via MIP!

[Fischetti and Jo, 2017] recently proposed to use MIP to train DNNs with ReLU activations. For each item of data $a = a^0$ have variables:

- a_j^k: Output of neuron j in layer k (must be nonnegative).
- s_j^k: In the event that $a_j^k = 0$, this captures the negative part of the input to neuron j in layer k.
- z_j^k: Binary variable, set to 1 if the input to neuron j in layer k is negative, and 0 otherwise.

The operation of layer k is captured in the following formulae:

$$\sum_{i=1}^{n_{k-1}} W_{ij}^{k-1} a_i^{k-1} + b_j^{k-1} = a_j^k - s_j^k,$$

$$a_j^k, s_j^k \geq 0, \quad z_j^k \in \{0,1\},$$

$$z_j^k = 1 \rightarrow a_j^k \leq 0,$$

$$z_j^k = 0 \rightarrow s_j^k \leq 0.$$

One set of variables for each data item a_j^0. Only works for small problems!

Issue 3: Overparametrization in Neural Networks

The total number of weights (W^ℓ, g^ℓ), $\ell = 1, 2, \ldots, L$ exceeds the number of data items, sometimes by factors of $10 - 100$.

Training such networks can often achieve "zero loss," that is, all items in the training data set are correctly classified. Two big questions arise.

1. Isn't this overfitting? Yet such models often predict well on non-training data, flouting conventional wisdom. Mystery!
2. Why is Stochastic Gradient (SGD) reliably finding the global minimum of the nonsmooth, nonlinear, nonconvex training problem?

We have only started to get some intuition on these issues. See for example [Li and Liang, 2018], and several papers in the recent NeurIPS.

Overparametrization

One key to understanding overparametrization might be stable activations of the ReLU neurons in the hidden layers:
For each class, the input t to each neuron tends to remain either negative or positive throughout training. They always output either 0 (when input is $t < 0$) or t (when input is $t > 0$).

That is, the DNN behaves like a set of *overlapping linear networks* — one for each class. (Linear network = sequence of matrix multiplications: $\bar{W}^L \bar{W}^{L-1} \ldots \bar{W}^1 a_j$. No nonsmoothness!

Solutions are not uniquely defined; there are many that achieve zero loss. Which one we find depends on how the weights are initialized.

Overparametrization

Thus, in an overparametrized network, we suspect that:

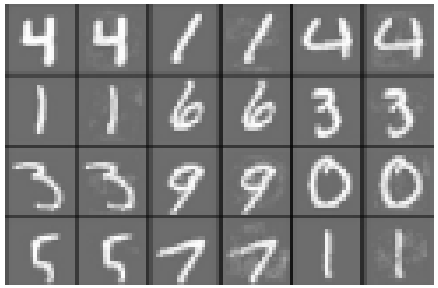
- There are many solutions. As the dimension grows, the "manifold" of solutions grows to fill the space.
- A randomly chosen initial point for the weights will be close to the solution manifold — closer as the dimension grows.
- You don't have to change many ReLU activations to get from the initial point to the solution, so nonsmoothness may not be important.
- Gradient descent (or stochastic gradient descent with big enough batches) will get us from the initial point to the solution efficiently.

This remains an active area of investigation, with important consequences for the understanding the effectiveness of DNNs!

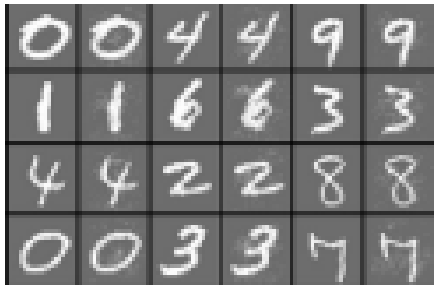
Issue 4: Adversarial Machine Learning

Easy to fool DNN classifiers with a carefully chosen attack!

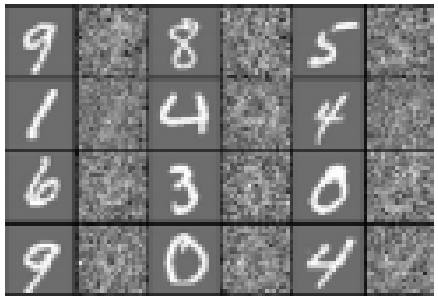
(Szegedy et al, Dec 2013): MNIST with carefully chosen perturbations. NN misclassifies, even though the "correct" answer is visually obvious.



(a) Even columns: adversarial examples for a linear (softmax) classifier (stddev=0.06)



(b) Even columns: adversarial examples for a 200-200-10 sigmoid network (stddev=0.063)



(c) Randomly distorted samples by Gaussian noise with stddev=1. Accuracy: 51%.

Note that even a large random perturbation is usually OK! But a small, carefully crafted perturbation causes misclassification.

Adversarial ML: The Issues

1. Can we generate efficiently the "carefully chosen perturbations" that break the classifier?
   - Various optimization formulations have been proposed, depending on the type of classification.
2. Can we train the network to be robust to perturbations of a certain size?
   - Can use robust optimization techniques (expensive) or selectively generate perturbed data examples and re-train.
3. Can we verify that a given network will continue to give the same classification when we perturb a given training example x by any perturbation of a given size $\epsilon > 0$?
   - Can use MIP, but very expensive even for small networks and data sets (e.g. MNIST, CIFAR).

Finding Adversarial Perturbations

Given a set of parameters x (e.g. weights in a NN) a data pair (a_j, y_j), and a prediction function ϕ, the "optimal adversarial perturbation" is obtained by solving

$$\min_v \|v\| \quad \text{s.t.} \quad \phi(a_j + v) \neq y_j.$$

Generally, this is a hard problem. But one special case is easy.

Suppose that we have two classes ± 1, and that

$$\phi(a) := \text{sign} J(a; x),$$

for some smooth function J. Thus, the optimal adversarial perturbation will be the smallest v such that $J(a_j + v; x) = 0$. Since J is smooth, we can write

$$J(a_j + v; x) \approx J(a_j; x) + v^T \nabla_a J(a_j; x),$$

so the approximate solution is

$$v = -J(a_j; x) \frac{\nabla_a J(a_j; x)}{\|\nabla_a J(a_j; x)\|^2}.$$

Training for Robustness

Instead of incurring a loss $h(x; a_j, y_j)$ for parameters x and data item (a_j, y_j) as above, define the loss to be the worst possible loss for all a within a ball of radius ϵ centered at a_j. That is,

$$\max_{v_j : \|v_j\| \leq \epsilon} h(x; a_j + v_j, y_j).$$

(The norm could be $\|\cdot\|_2$ or $\|\cdot\|_\infty$, or something else.)

Training becomes the following min-max problem:

$$\min_x \frac{1}{m} \sum_{j=1}^{m} \max_{v_j : \|v_j\| \leq \epsilon} h(x; a_j + v_j, y_j).$$

The inner "max" problems can at least be solved in parallel, approximately, and sometimes in closed form.

Subproblems yield a generalized gradient w.r.t. x. We can use this to implement a first-order method for the outer loop. Expensive in general!

Sparsity and Stability Make Verification Easier

Networks can be trained in a way that makes them easier to check [Xiao et al., 2018]. Verification is slow because a perturbation in a_j may cause activations to change, moving across the kink in the function $\max(t, 0)$.

Verification is easier when

- Sparsity: weight matrices W^ℓ are sparse — a lot of missing arcs in the NN, so fewer ReLU neurons to check. Promote sparsity via regularizers $\|W^\ell\|_1$.
- Promoting ReLU stability during training: choose weights W^ℓ, $\ell = 1, 2, \ldots, L$ so that fewer examples a_j change activation. Use interval arithmetic + regularization.

Tutorial by Z. Kolter and A. Madry presented at NeurIPS last week: `https://adversarial-ml-tutorial.org/`

Issue 5: Reinforcement Learning vs Control

Reinforcement Learning (RL): kind of ML with high-profile recent success (e.g. AlphaGo, Backgammon.)

Suited to a situation in which there is a set of states and a set of possible actions (moves). The following recurs indefinitely:

- At state s_k, Take an action / make a move a_k;
- Incur a cost $g(s_k, a_k)$ and go to a new state s_{k+1}, which depends on (s_k, a_k).

Aim for a policy for choosing a_k given s_k that minimizes cost incurred over the long run (e.g. average cost per move).

Generally don't know cost g or the mapping $(s_k, a_k) \to s_{k+1}$ in advance; must be learned from experience.

RL usually seeks the policy directly.

- Implements a policy for a certain number of steps and observes costs;
- Can use gradient or derivative-free techniques to find better policies.

Control Perspective on RL

The setup is very similar to control problems, especially optimal control and model-predictive control (MPC), for which efficient optimization methods are available.

But these methods require knowledge of costs g and transition dynamics $(s_k, a_k) \to s_{k+1}$! They must be preceded by a system identification process that learns these quantities.

[Recht, 2018] studies the simplest variant (quadratic cost, linear dynamics) and concludes that control strategies are superior to RL.

See illuminating discussions from Bertsekas at CDC on 12/16/18:
`http://web.mit.edu/dimitrib/www/Slides_RL_and_Optimal_Control.pdf`

Summary

Optimization provides powerful frameworks for formulating and solving problems in data analysis and machine learning.

Usually not enough to formulate these problems and use off-the-shelf optimization software to solve them. The algorithms need to be customized to the problem structure and context.

Research in this area has exploded over the past decade and is still going strong, with a many unanswered questions — many of them in deep learning.

References I


Balzano, L., Nowak, R., and Recht, B. (2010).
Online identification and tracking of subspaces from highly incomplete information.
In *48th Annual Allerton Conference on Communication, Control, and Computing*, pages 704–711.
http://arxiv.org/abs/1006.4046.


Balzano, L. and Wright, S. J. (2014).
Local convergence of an algorithm for subspace identification from partial data.
Foundations of Computational Mathematics, 14:1–36.
DOI: 10.1007/s10208-014-9227-7.


Boser, B. E., Guyon, I. M., and Vapnik, V. N. (1992).
A training algorithm for optimal margin classifiers.
In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pages 144–152.


Candès, E. J., Li, X., Ma, Y., and Wright, J. (2011).
Robust principal component analysis?
Journal of the ACM, 58.3:11.


Chandrasekaran, V., Sanghavi, S., Parrilo, P. A., and Willsky, A. S. (2011).
Rank-sparsity incoherence for matrix decomposition.
SIAM Journal on Optimization, 21(2):572–596.

References II


Cortes, C. and Vapnik, V. N. (1995).
Support-vector networks.
Machine Learning, 20:273–297.


d'Aspremont, A., Banerjee, O., and El Ghaoui, L. (2008).
First-order methods for sparse covariance selection.
SIAM Journal on Matrix Analysis and Applications, 30:56–66.


d'Aspremont, A., El Ghaoui, L., Jordan, M. I., and Lanckriet, G. (2007).
A direct formulation for sparse PCA using semidefinte programming.
SIAM Review, 49(3):434–448.


Dasu, T. and Johnson, T. (2003).
Exploratory Data Mining and Data Cleaning.
John Wiley & Sons.


Fischetti, M. and Jo, J. (2017).
Deep neural networks as $0-1$ mixed integer linear programs: A feasibility study.
Technical Report arXiv:1712.06174, DEI, University of Padova.


Friedman, J., Hastie, T., and Tibshirani, R. (2008).
Sparse inverse covariance estimation with the graphical lasso.
Biostatistics, 9(3):432–441.

References III

Kimgma, D. P. and Ba, J. (2015).
Adam: A method for stochastic optimization.
In *Proceedings of International Conference on Learning Representations*.

Li, Y. and Liang, Y. (2018).
Learning overparametrized neural networks via stochastic gradient descent on structured data.
Technical Report arXiv:1808.01204, University of Wisconsin-Madison.

Recht, B. (2018).
A tour of reinforcement learning: The view from continuous control.
Technical Report arXiv:1806.09460, University of California-Berkeley.

Recht, B., Fazel, M., and Parrilo, P. (2010).
Guaranteed minimum-rank solutions to linear matrix equations via nuclear norm minimization.
SIAM Review, 52(3):471–501.

Stigler, S. M. (1981).
Gauss and the invention of least squares.
Annals of Statistics, 9(3):465–474.

Xiao, K. Y., Tjeng, V., Shaflullah, N. M., and Madry, A. (2018).
Training for faster adversarial robustness verification via inducing ReLU stability.
Technical Report arXiv:1809.03008, MIT.

References IV