Optimization in Data Science: Formulations

Stephen Wright (UW-Madison)

MoCaO, July, 2022

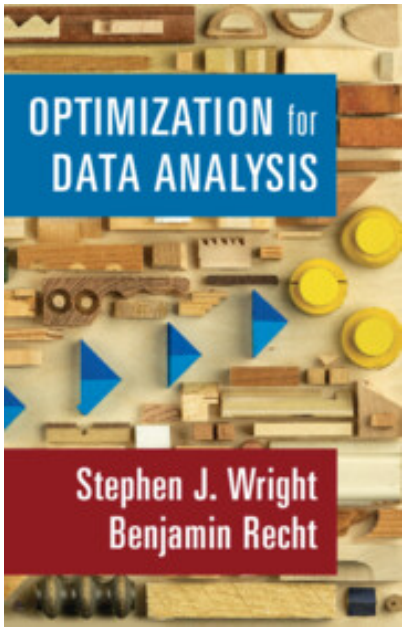
Outline (3 lectures)

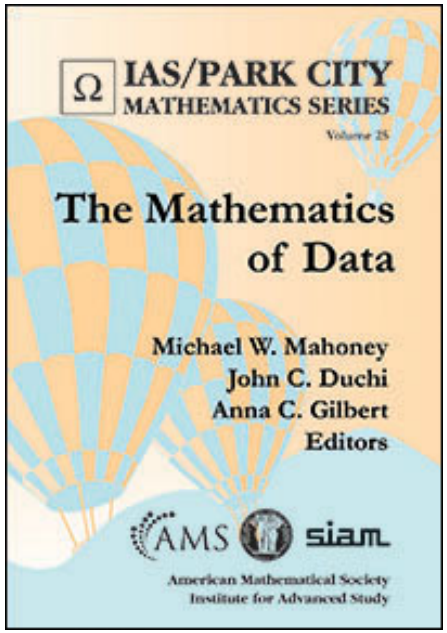
- Introduction / Motivation / Perspective
- FORMULATIONS: Formulating data science problems as optimization problems.
  - 15 applications and one "issue" (nonconvexity)
  - Neural networks
- ALGORITHMS: Optimization algorithms that are useful for data science formulations.
  - First-order methods
  - Prox-gradient
  - Stochastic gradient
  - Coordinate descent
  - Primal-dual methods for min-max problems
  - Augmented Lagrangian and ADMM

NOT exhaustive!

Sources Include...

- Wright, S. J. and Recht, B., *Optimization for Data Analysis*, Cambridge, 2022.
- Wright, S. J., Optimization algorithms for data analysis. in *The Mathematics of Data*, 25, 49 (2018).





Optimization and Data Science

Optimization is being revolutionized by its interactions with machine learning and data science.

- New algorithms + new interest in *old* algorithms;
- Challenging formulations and new paradigms;
- Renewed emphasis on certain topics: convex optimization algorithms, complexity, structured nonsmoothness, nonconvex optimization, stochastic gradient, augmented Lagrangian,
- Large research community now working on the machine learning / optimization spectrum. The optimization / ML interface is a key component of many top conferences (ISMP, SIOPT, NeurIPS, ICML, ICLR COLT, AISTATS, ...) and journals (JMLR, Math Programming, SIOPT, ...).

Data Science

Related To: AI, Machine Learning, Data Analysis, Statistical Inference, Learning Theory, ...

- Extract meaning from data: Learn important features and fundamental structures in the data.
- Use this knowledge to make predictions (inferences) about other, similar data.

Multidisciplinary!

- Foundations in Statistics;
- Computer Science: AI, Machine Learning, Databases, Parallel Systems, Architectures (GPUs);
- Optimization provides tools for modeling / formulation / algorithms;
- Closely tied to applications in some areas e.g. signal (speech / image / natural language) processing, robotics.

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

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**. M can be very large!
- multiple labels e.g. $y_j \in \mathbb{R}^t$ or $y_j \in \{0, 1\}^t$: 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 clusters; each cluster groups objects with similar features.

Fundamental Data Analysis Task: Classical Perspective

Seek a function ϕ that:

- approximately maps a_j to y_j for each j: $\phi(a_j) \approx y_j$, $j = 1, 2, \ldots, m$.
- satisfies additional properties that make it "plausible" for the application, robust to perturbations in the data, generalizable to other data samples from the same distribution.

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

Find an x such that $\phi(a_j; x) \approx y_j$ for $j = 1, 2, \ldots, m$.

Objective function in this problem often built up of m terms, each depending on a single (a_j, y_j) pair, that capture mismatch between predictions and observations for that pair.

(If no labels y_j, ϕ assigns each a_j to a cluster or subspace.)

The process of finding ϕ is called learning or training.

How is ϕ used?

- Prediction: Given new data vector a_k, predict outputs $y_k \leftarrow \phi(a_k; x)$.
- Analysis: ϕ (or the parameter x) reveals structure in the data.
  - Feature selection: reveal the components of vectors a_j that are most important in determining the outputs y_j.
  - Uncovers some hidden structure:
    - ⋆ low-dimensional subspaces that contain the a_j (approx);
    - ⋆ find clusters of a_j's;
    - ⋆ decision tree building 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). But see later!
- Distributional shift.

Continuous Optimization and Data Analysis

Optimization is a major source of algorithms for machine learning and data analysis.

- Optimization Formulations translate statistical principles (e.g. risk, likelihood, significance, generalizability) into measures and functions that can be attacked with an algorithm.
- Optimization Algorithms provide practical means to solve these problems, but the "black-box" approach often doesn't work well. Structure and context are important.
- Duality is valuable in several cases (e.g. kernel learning).
- Nonsmoothness appears often e.g. as a formulation tool to promote generalizability, but often in a highly structured way that can be exploited by algorithms.

ML's Influence on (Continuous) Optimization

The needs of ML and the ML community have influenced optimization in recent years.

ML has a different perspective on some important algorithmic issues:

- Computational complexity and global convergence rates are more interesting.
- Fast local convergence rates are less interesting, possibly because fast convergence often occurs only below the level of accuracy required for minimization of the empirical risk.
- Prefer cheaper, approximate solutions over expensive, accurate solutions (for the same reason).
- Huge problems, both in number of parameters (x) and size of data set defining the problems ((a_j, y_j), $j = 1, 2, \ldots, m$). Leads to a need for
  - Parallel methods ("federated learning")
  - Methods that don't need exact functions or gradients (stochastic gradient).

What's an "Algorithm" in ML and Optimization

In ML, the "algorithm" is the process that maps a training data set S to a predictor. (Sometimes randomized, e.g. stochastic gradient.)

Aim for generalizability: Predicting well on unseen data.

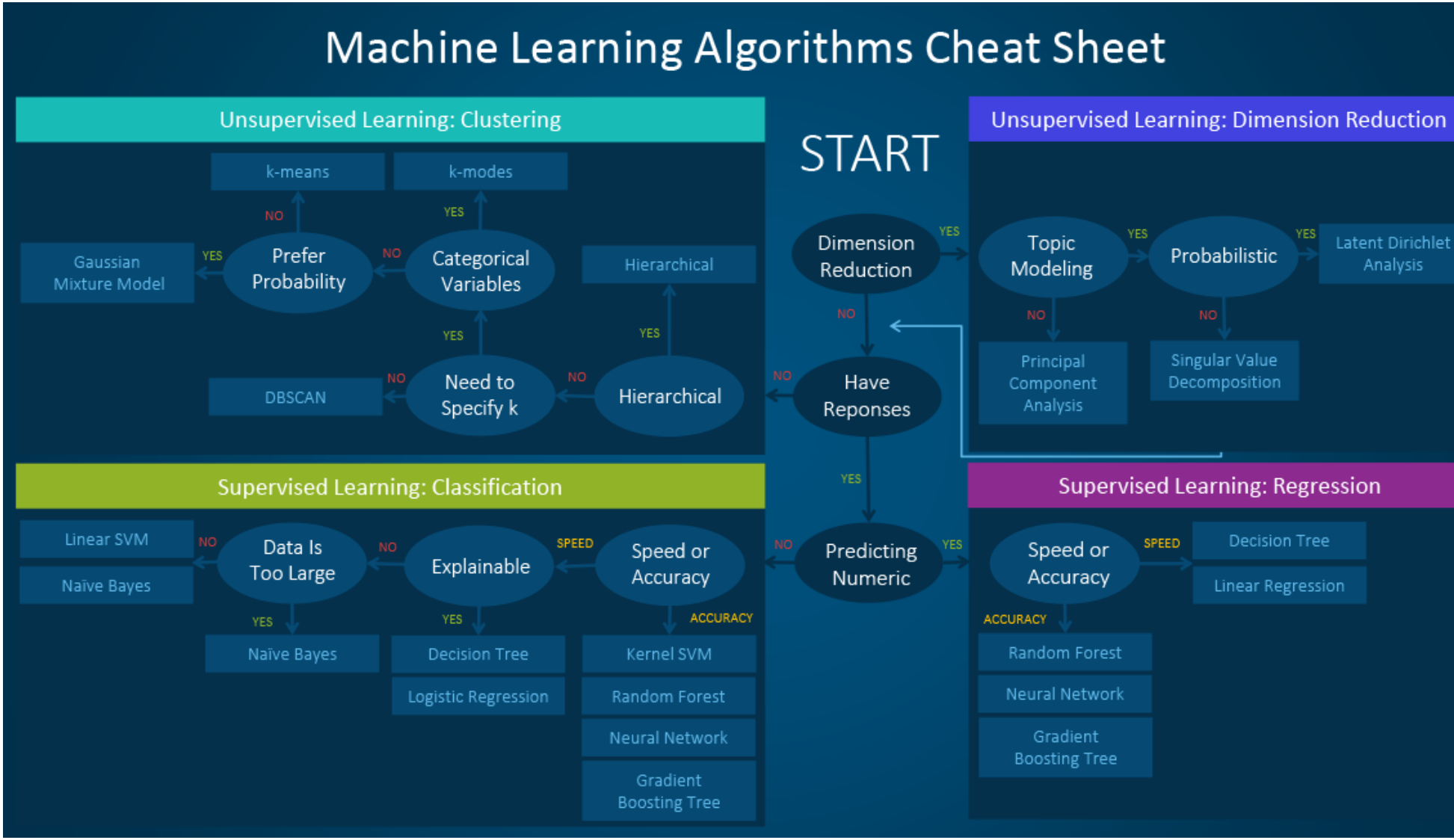
An ML "algorithm" is implemented via optimization in two stages: the optimization formulation and the optimization algorithm.

The responsibility for good generalizability thus falls both on the optimization formulation and the optimization algorithm.

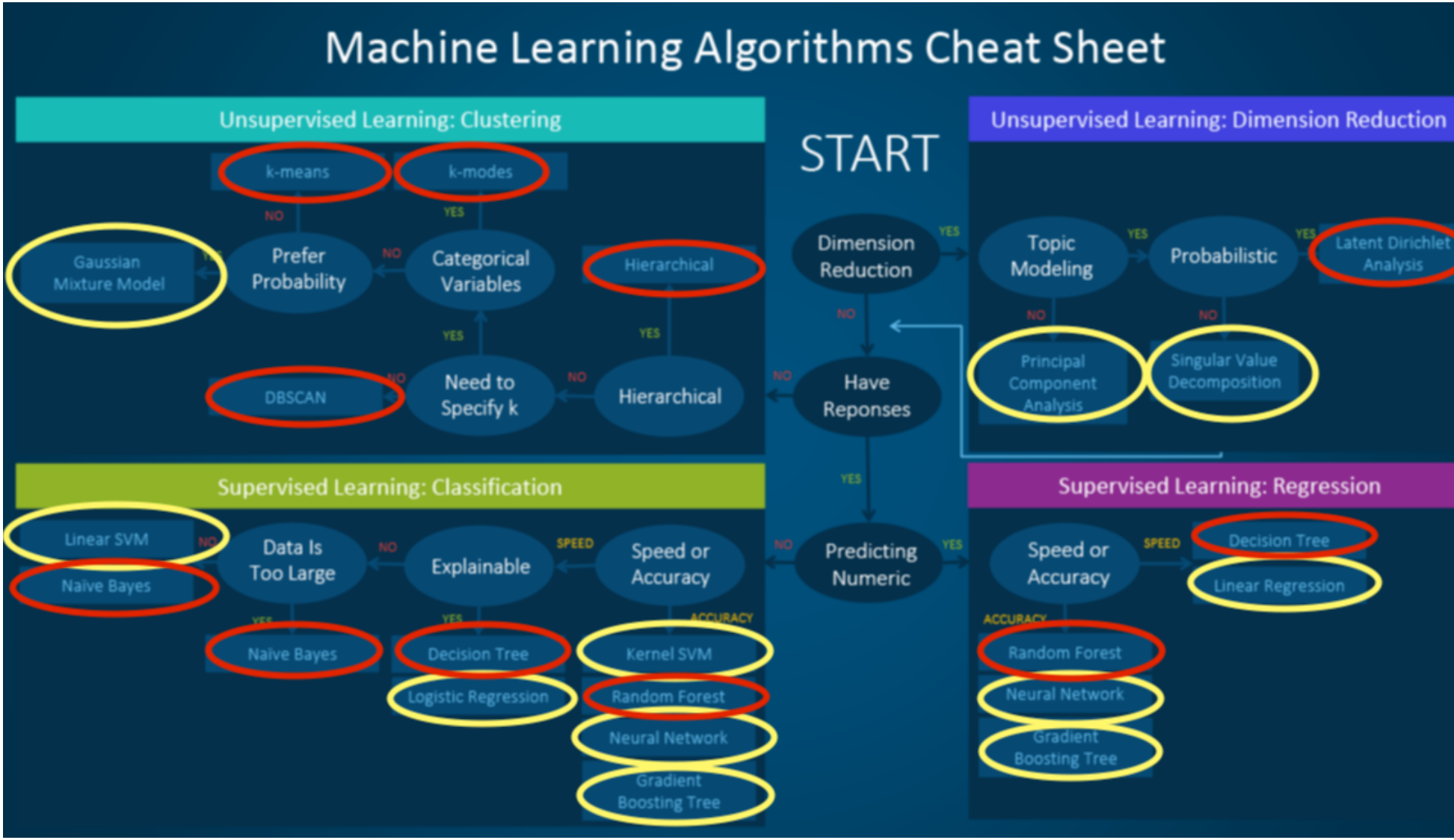
This overloads the optimization algorithm! Traditionally, optimization algorithms were just tasked with finding the solution of the formulation. Nowadays they have to pursue a more nebulous and unfamiliar goal.

New questions arise, e.g.

- Does small-batch SGD give results with better generalizability?
- Is the algorithm finding a "low-norm" or "low-curvature" solution that generalizes better?



[1] https://blogs.sas.com/content/subconsciousmusings/2020/12/09/machine-learning-algorithm-use/



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

Application 1: Least Squares Regression (+ Regularization)

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

[Gauss, 1799], [Legendre, 1805]. Still a popular topic!

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

- 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 ("Lasso") yields sparse 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 feature vectors a_j.

Application 2: 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 3: Matrix Sensing / 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 a "simple" X e.g. low rank. Can add a nuclear-norm (sum-of-singular-values) regularization term $\lambda \|X\|_*$ [Recht et al., 2010].

When the observations $\mathcal{A}_j$ are of individual elements, recovery is still possible when X is low-rank and incoherent (i.e. not too "spiky" and with singular vectors randomly oriented).

Procedures based on trimming + truncated singular value decomposition (for initialization) and projected gradient (for refinement) produce good solutions [Keshavan et al., 2010].

Explicit Low-Rank Parametrization

Nonconvex, but much more useful for large problems:

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

Despite the nonconvexity, near-global minima can be found when $\mathcal{A}_j$ are incoherent. Use appropriate initialization [Candès et al., 2015], [Zheng and Lafferty, 2015] or the observation that all local minima are near-global [Bhojanapalli et al., 2016].

(More on this later.)

Application 4: 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, genomics, ...

Often $r \ll \min(m, n)$, so L and R are "skinny." Low rank.

Several variants, e.g.

- Not all elements of Y are known (matrix completion + bounds);
- Additional structure in L and/or R, e.g. columns of R have disjoint support.

Application 5: Recovering Dependency Networks

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: $X \approx S^{-1}$.

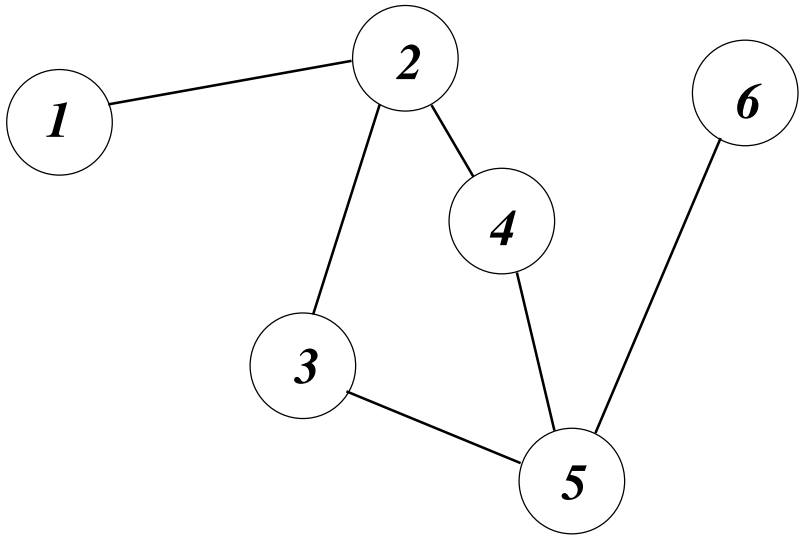
X reveals dependencies between components of Z: $X_{ij} = 0$ if the i and j components of Z are *conditionally independent*, i.e. don't influence each other directly (but may still be correlated due to a chain of dependencies involving other components).

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].

Reveals Network Structure. Example with $p = 6$.



$$X = \begin{bmatrix} * & * & 0 & 0 & 0 & 0 \\ * & * & * & * & 0 & 0 \\ 0 & * & * & 0 & * & 0 \\ 0 & * & 0 & * & * & 0 \\ 0 & 0 & * & * & * & * \\ 0 & 0 & 0 & 0 & * & * \end{bmatrix}$$

Application 6: 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:

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

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

Adjust the parameter R to obtain desired sparsity. Can be generalized for first r principal components: use $\langle I, M \rangle = r$.

Sparse PCA: Explicit Parametrization

Seek an approximation to the leading r-dimensional eigenspace with space basis vectors.

Explicit low-rank formulation is

$$\max_{F \in \mathbb{R}^{n \times r}} \langle S, FF^T \rangle \quad \text{s.t. } \|F\|_2 \leq 1, \ \|F\|_{2,1} \leq \bar{R},$$

where $\|F\|_{2,1} := \sum_{i=1}^{n} \|F_{i\cdot}\|_2$ to promote sparsity
[Chen and Wainwright, 2015, Chi et al., 2019].

Application 7: Sparse + Low-Rank (Robust PCA)

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 8: Subspace Identification

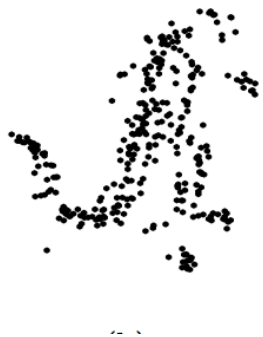
Given vectors $a_j \in R^n$ with missing entries, find a subspace of $\mathbb{R}^n$ such that all "completed" vectors a_j lie approximately in this subspace.

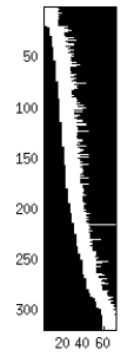
If $\Omega_j \subset \{1, 2, \ldots, n\}$ is the set of observed elements in a_j, seek $X \in \mathbb{R}^{n \times d}$ such that

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

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 9: 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

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

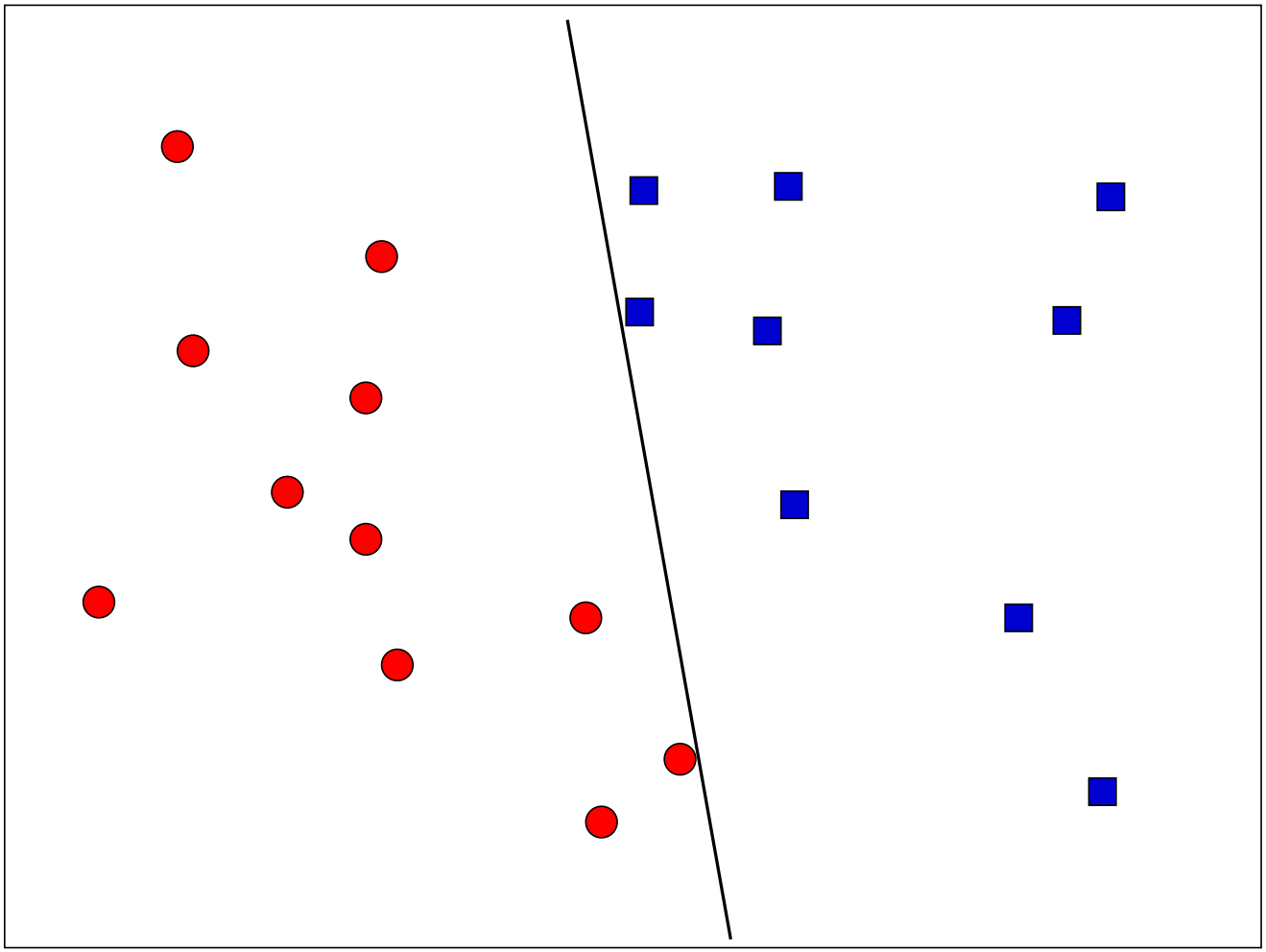
The mapping is $\phi(a_j; x) = \text{sign}(a_j^T x - \beta)$.

In the objective, the jth loss term is zero when $\phi(a_j; x) = y_j$, positive otherwise. A popular one is hinge loss:

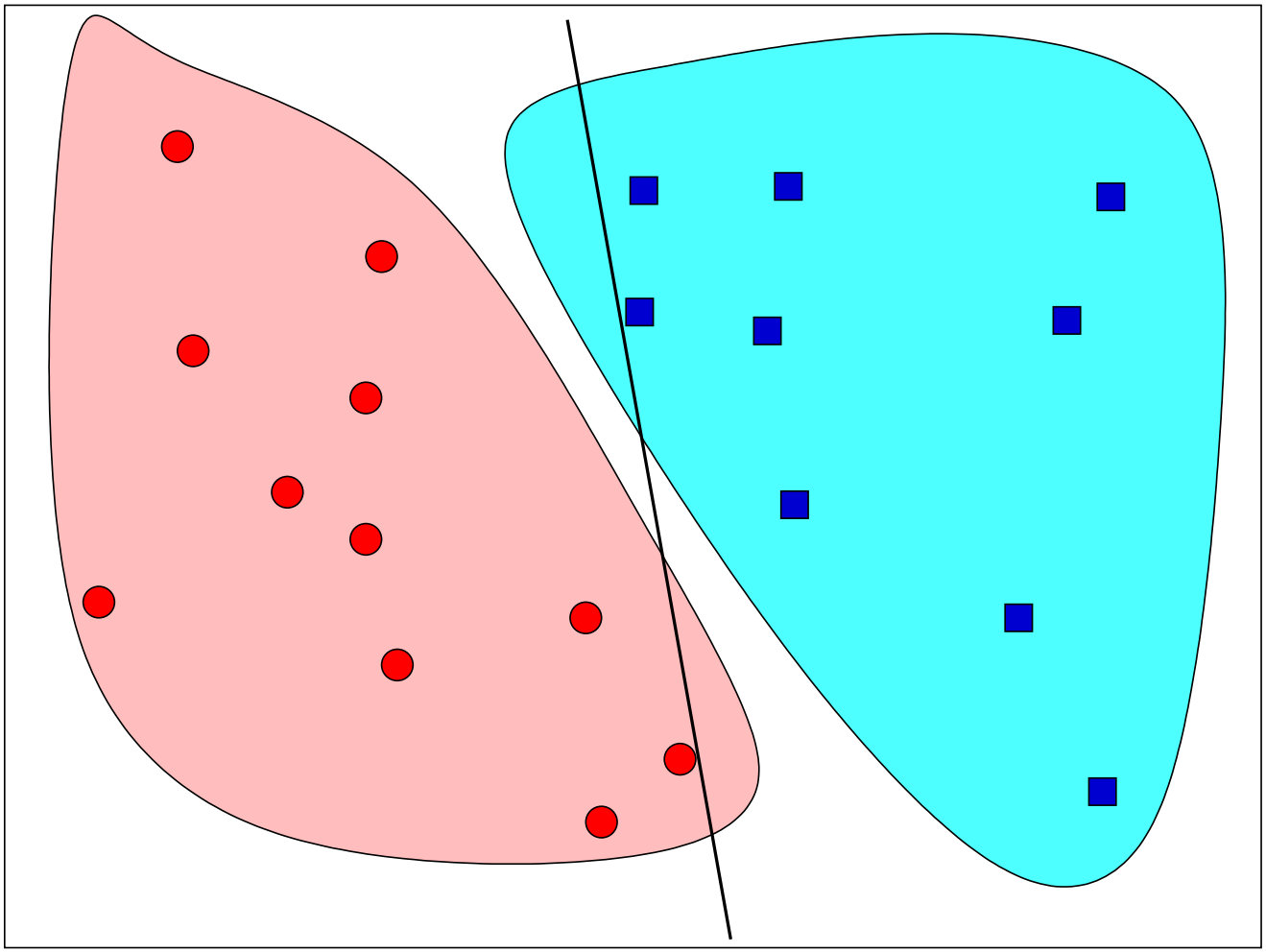
$$H(x, \beta) = \frac{1}{m} \sum_{j=1}^{m} \max(1 - y_j(a_j^T x - \beta), 0).$$

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

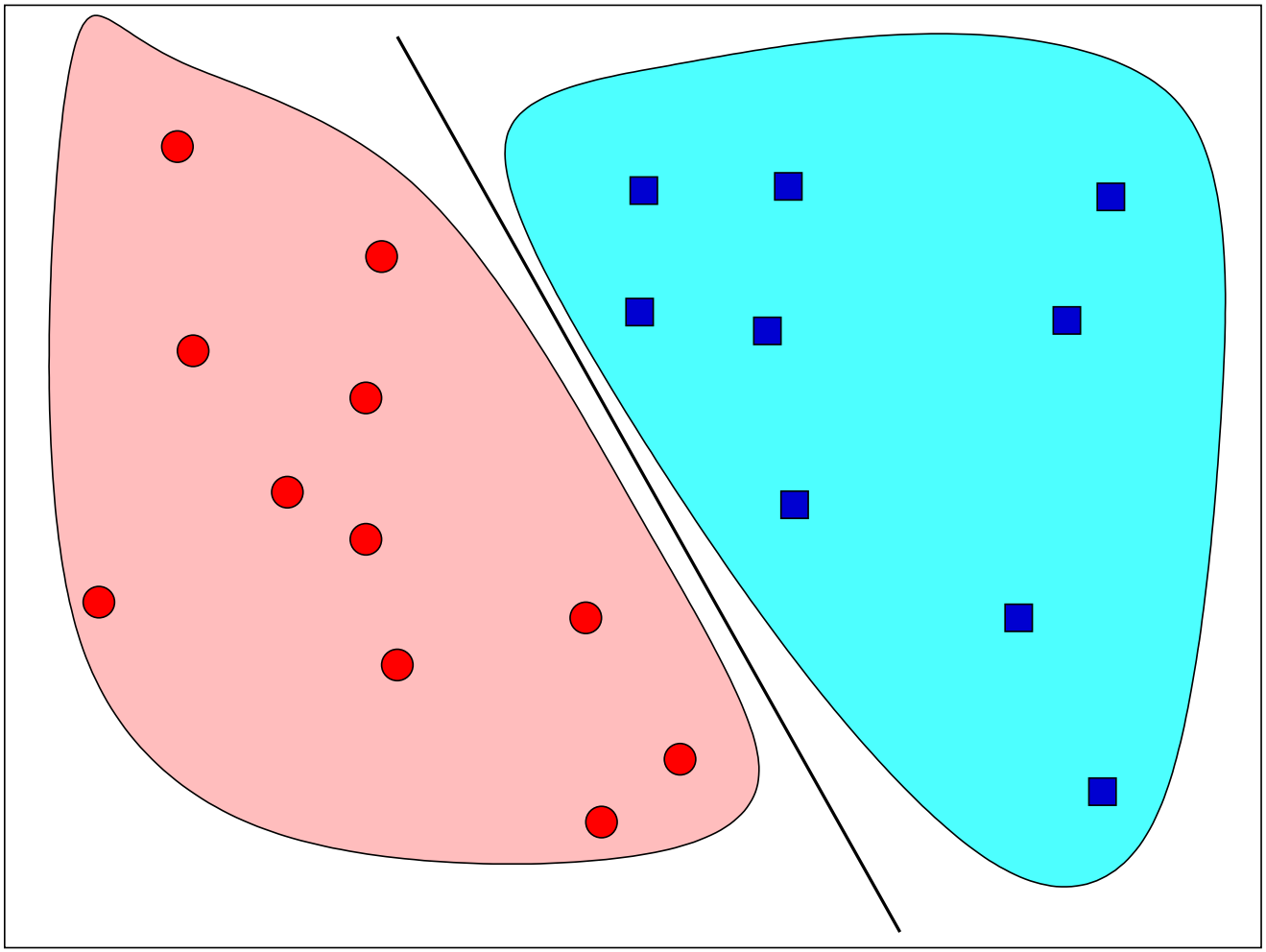
Regularize for Generalizability



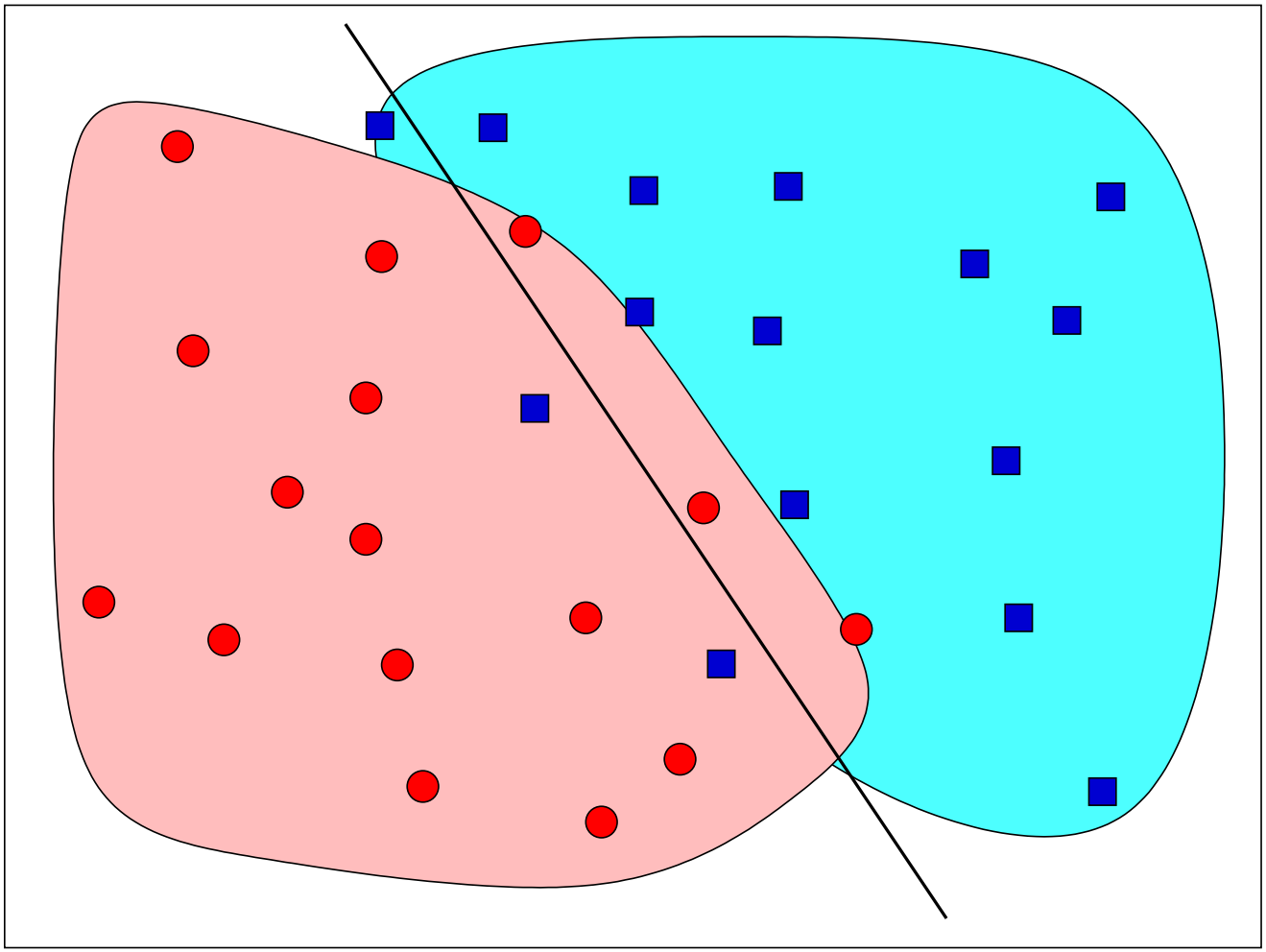
Regularize for Generalizability



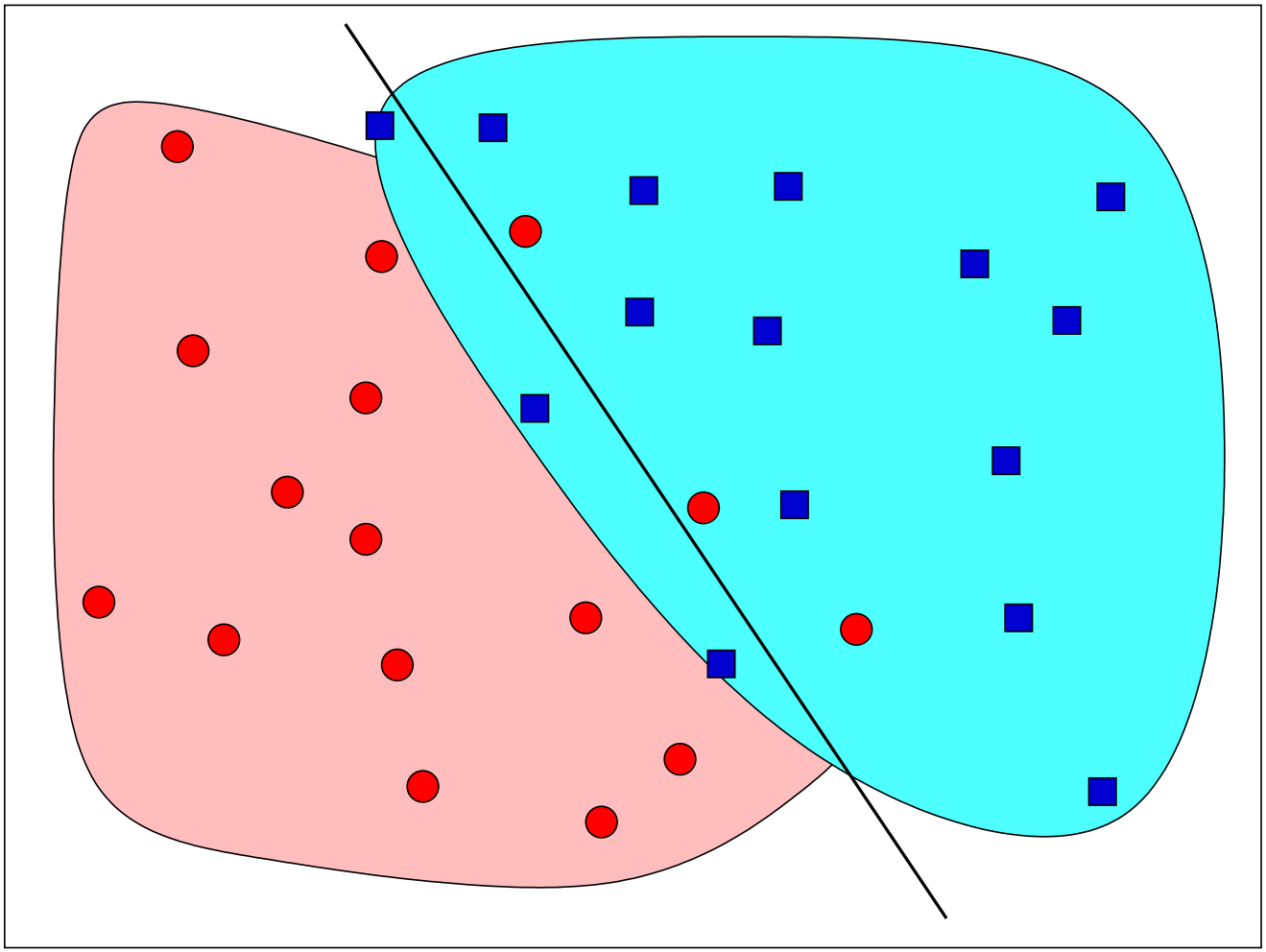
Regularize for Generalizability



Regularize for Generalizability



Regularize for Generalizability



Application 10: Kernel SVM

To enhance "separability" of the data, apply a nonlinear transformation $a_j \to \psi(a_j)$ ("lifting") and do linear classification on $(\psi(a_j), y_j)$:

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

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 11: Multiclass Logistic Regression

Given M classes $y_j \in \{1, 2, \ldots, M\}$ for all j, seek *odds functions* $p_i(a_j)$ that give the probability of a_j belonging to class $i = 1, 2, \ldots, M$:

$$p_i(a) = \frac{\exp(a^T x_i)}{\sum_{l=1}^{M} \exp(a^T x_l)} \quad (\text{"softmax" over the } a^T x_i).$$

Seek x_i so that

$$a_j^T x_{y_j} \gg a_j^T x_l \text{ for all } l \neq y_j$$

so that $p_i(a_j) \approx 1$ when $l = i$, while $p_l(a_j) \approx 0$ for all $l \neq i$.

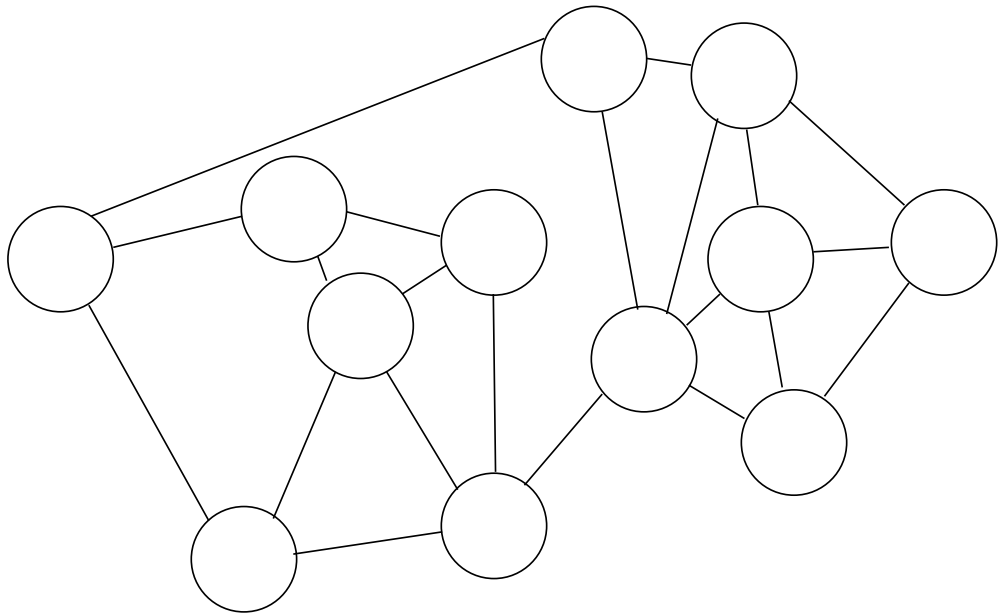
Maximize an a posteriori log-likelihood function:

$$\mathcal{L}(x_1, \ldots, x_M) = \frac{1}{m} \sum_{j=1}^{m} \log \left(\frac{\exp(a_j^T x_{y_j})}{\sum_{l=1}^{M} \exp(a_j^T x_l)} \right), \quad \text{which is convex!}$$

Often forms the final layer of a neural network. Here the a_j are feature vectors for item j, as transformed by the rest of the NN.

Application 12: 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.

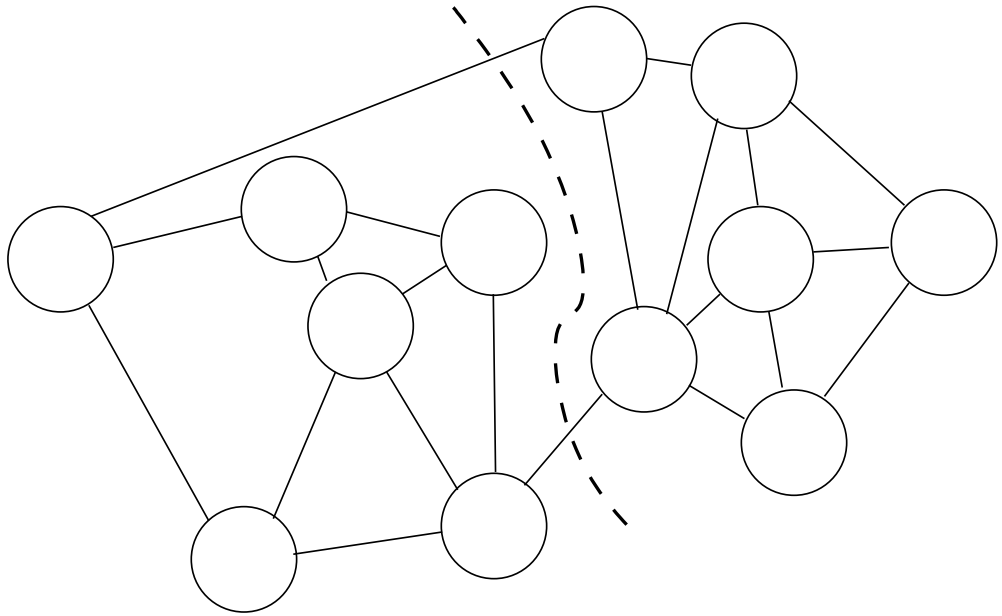


Probability p of being connected to a node *within* your community, and q of being connected to a node *outside* your community, with $0 < q < p < 1$.

Matrix optimization problem: relaxation of max-log-likelihood formulation.

Application 12: 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.



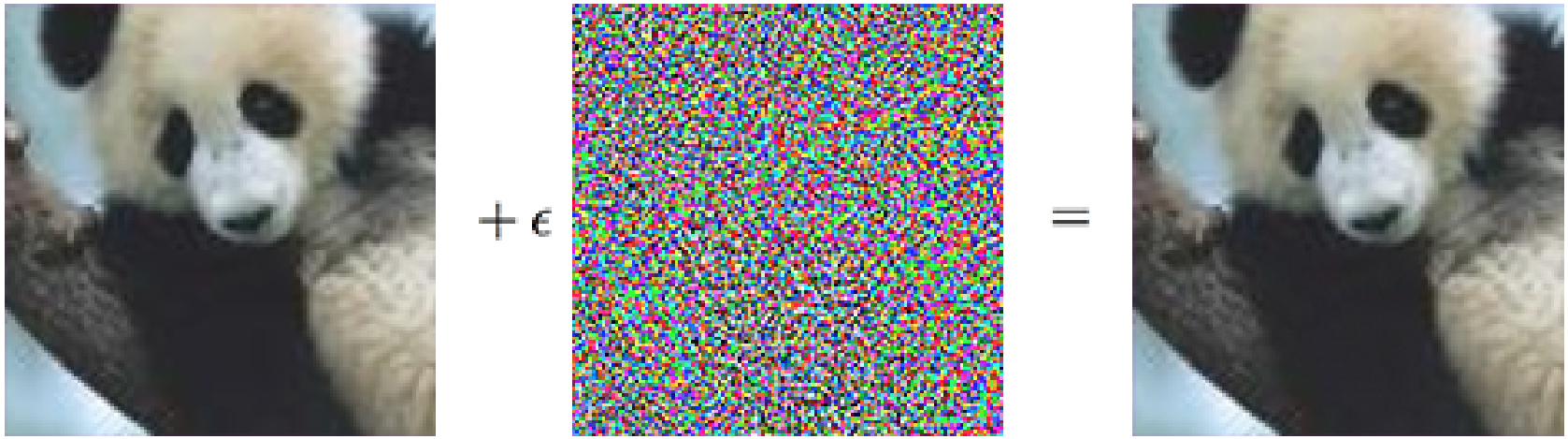
Probability p of being connected to a node *within* your community, and q of being connected to a node *outside* your community, with $0 < q < p < 1$.

Matrix optimization problem: relaxation of max-log-likelihood formulation.

Application 13: 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.



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 prediction function ϕ, obtain "optimal adversarial perturbation" v:

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

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 $h(a_j, y_j; x)$ 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,

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

(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!

Robust Training by Adding Data

A variant on the min-max strategy is to generate perturbations v_j explicitly, add items of the form $(a_j + v_j, y_j)$ to the training set, and re-train. If the v_j are chosen to be nearly optimal adverse perturbations, this might be effective.

Successive rounds of training could be "warm-started," and we might be able to manage the size of the augmented data set by removing points $(a_j + v_j, y_j)$ from earlier rounds that are no longer relevant.

Sparsity and Stability Make Verification Easier

Networks can be trained in a way that makes robustness to perturbations easier to check [Xiao et al., 2018].

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 2018:
`https://adversarial-ml-tutorial.org/`

Application 14: Distributionally Robust Learning

Another perspective on robustness. Take the training data (a_j, y_j), $j = 1, 2, \ldots, m$ to be an empirical approximation $\mathbb{P}_m$ to the true, unknown underlying distribution of the data $\mathbb{P}$.

We don't know $\mathbb{P}$, but we assume that it is close to $\mathbb{P}_m$ according to some metric of distributions $d(\cdot, \cdot)$ (e.g. Wasserstein metric, divergence).

"Traditional" training (minimize loss over training data):

$$\min_x \; E_{(a,y)\sim\mathbb{P}_m} \, h(a, y; x) = \frac{1}{m} \sum_{j=1}^{m} h(x; a_j, y_j).$$

"True" training (not implementable, since we don't know $\mathbb{P}$):

$$\min_x \; E_{(a,y)\sim\mathbb{P}} \, h(a, y; x).$$

"Distributionally robust" training: minimize worst-case loss over "ball" of data distributions centered at $\mathbb{P}_m$:

$$\min_x \max_{\mathbb{Q}: d(\mathbb{Q},\mathbb{P}_m)\leq\epsilon} \; E_{(a,y)\sim\mathbb{Q}} \, h(a, y; x).$$

Formulating Distributionally Robust Learning

In some cases, this problem can be formulated as a solvable optimization problem.

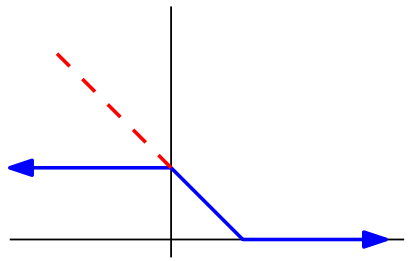
Example [Ho-Nguyen and Wright, 2022]: When the model is linear and $h(a, y; x)$ is discontinuous "zero-one loss":

$$h(a, y; x) = \begin{cases} 1 & \text{if } y(a^T x) < 0 \\ 0 & \text{if } y(a^T x) \geq 0 \end{cases}$$

then the DRO problem with Wasserstein 2-metric is

$$\min_x \epsilon \|x\|_2 + \frac{1}{m} \sum_{j=1}^{m} L_R(y_j(a_j^T x)),$$

where L_R is the ramp-loss function: $L_R(r) = 1$ for $r < 0$; $L_R(r) = 1 - r$ for $r \in [0, 1]$, $L_R(r) = 0$ for $r > 1$:



In fact, [Song et al., 2021a] show that when h is any convex function of $y(a^T x)$, the DRO problem using both Wasserstein 2-metric and a alertdivergence metric can be formulated as a generalized linear program:

$$\min_x \ c^T x + r(x) \ \text{ subject to } Ax = b, \ x \in \mathcal{X},$$

where $\mathcal{X}$ is a convex set that is easy to project onto while $r(x)$ is a convex function for which the "prox" operation is easy to compute. (For example, $r(x)$ could be a separable convex function $\sum_{i=1}^n r_i(x_i)$.)

Primal-dual algorithms for solving the generalized LP are described in [Song et al., 2021a]; see also [Song et al., 2021b], [Alacaoglu et al., 2022].

Application 15: Convex-Concave Min-Max

Several of the preceding applications are special cases of a convex-concave min-max problem:

$$\min_x \max_y L(x,y) := y^T Ax - \sum_{j=1}^m h_j^*(y_j) + g(x),$$

- $x \in \mathbb{R}^n$, $y \in \mathbb{R}^m$, A is an $m \times n$ matrix;
- each $h_j^* : \mathbb{R} \to \mathbb{R}$ is a conjugate of a convex scalar, that is

$$h_j^*(z) = \sup_t [tx - h_j(t)]$$

  function.
- $g : \mathbb{R}^n \to \mathbb{R}$ is convex.

By performing the max w.r.t. y explicitly, we obtain

$$\min_x h(Ax) + g(x),$$

where $h : \mathbb{R}^m \to \mathbb{R}$ has the separable form $h(z) = \sum_{j=1}^m h_j(z_j)$.

Convex-Concave Min-Max

Empirical Risk Minimization (ERM) problems have the latter form, including

- Least squares: $h_i(t) = \frac{1}{2}t^2$;
- ℓ_1 regression: $h_i(t) = |t|$;
- Hinge loss: $h_i(t) = \max(t, 0)$ (used in SVM, neural nets);
- Regularization: Tikhonov $g(x) = \lambda\|x\|_2^2$, $\quad \ell_1$: $g(x) = \lambda\|x\|_1$;
- TV regularization: $h_i(t) = \|t\|_2$, $A_i \in \mathbb{R}^{2\times d}$;
- logistic regression, least absolute deviation, ...

Also DRO with linear models (see above).

The min-max form of the problem can be useful for algorithms — see later!

Issue: Benign Nonconvexity

Nonconvexity arises often in ML, but still useful solutions (even global minima) are often easy to find:

- matrix and tensor problems with explicit low-rank parametrizations (not convex relaxations);
- phase retrieval;
- neural networks (NNs);
- AC power flow;
- (many other settings).

Ju Sun's excellent page: `https://sunju.org/research/nonconvex/`

Several structures and properties promote benign nonconvexity, e.g.

- All local minima are global minima;
- All saddle points are *strict* saddle points, so are easy to escape from (e.g. by detecting negative curvature in the Hessian);
- Smart initialization schemes place x^0 in a neighborhood of a global minimizer.

Example: Matrix Sensing

Matrix Sensing (Application 3) can be benignly nonconvex.

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

- When symmetric X has low rank r, write $X = ZZ^T$ where $Z \in \mathbb{R}^{n \times r}$.
- $\mathcal{A}_j(X) = \langle A_j, ZZ^T \rangle$ for some symmetric $A_j \in \mathbb{R}^{n \times n}$.
- Assume that the A_j satisfy a restricted isometry property (RIP):

$$(1 - \delta_q)\|X\|_F^2 \leq \frac{1}{m} \sum_{j=1}^{m} \langle A_j, X \rangle^2 \leq (1 + \delta_q)\|X\|_F^2,$$

for all X with rank at most q and some $\delta_q \in (0, 1)$.

Formulation is thus

$$\min_Z h(Z) := \frac{1}{2m} \sum_{j=1}^{m} (\langle A_j, ZZ^T \rangle - y_j)^2.$$

Example: Matrix Sensing

If the properties above hold with $q = 2r$ and $\delta_{2r} \in (0, .1]$, then

- All local minima of h are global;
- All stationary points of h that are not strict have negative curvature in $\nabla^2 h(Z)$.

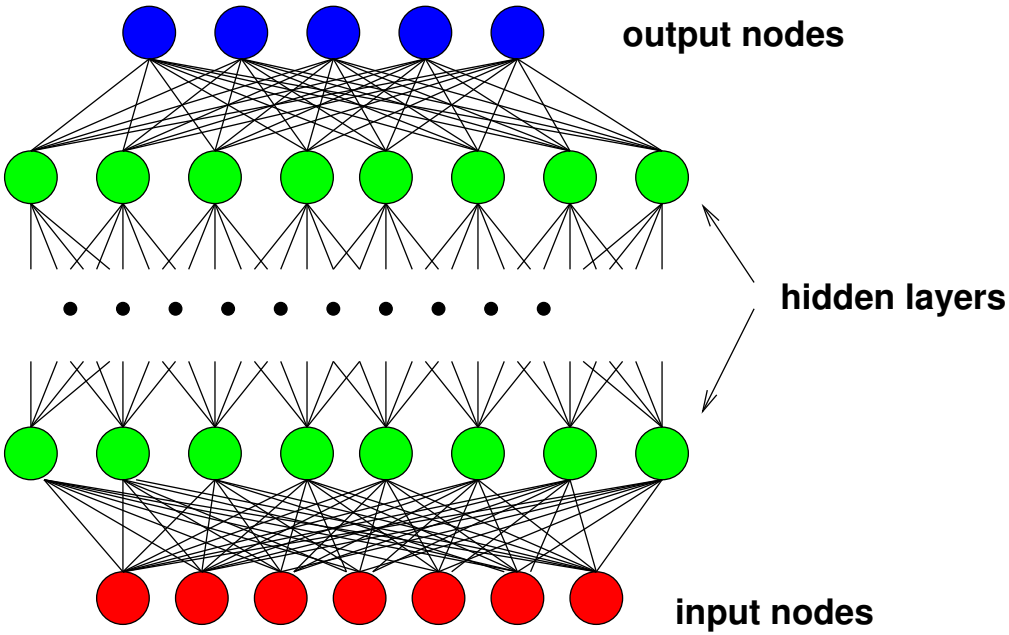
[Bhojanapalli et al., 2016]

Smart initialization: The matrix

$$Y := \frac{1}{m} \sum_{j=1}^{m} y_j A_j$$

is close to the solution if RIP properties are satisfied for $q = 2r$. Steepest descent on h can converge from such a starting point.

Neural Networks



Inputs are the vectors $a_j = a_j^0$. The M output nodes are compared with label y_j to form the loss function, e.g. softmax or linear combination.

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

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

where a^ℓ is vector of outputs from layer ℓ, (W^ℓ, b^ℓ) are *parameters* or *weights* in the network.

Neural Networks

The element-wise function σ makes transformations to scalar input. Nowadays, it's usually the ReLU / hinge function:

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

NN architectures are engineered to the application (speech processing, image 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.
- "Long-short-term memory" (LSTM): for sequenced data (speech, text).

(The network above is fully connected - a "convolutional NN" or CNN.)

Training NNs

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 usual form:

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

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 (components of x) is often vastly greater than the number of data points — overparametrization.

Nonlinear, Nonconvex, usually Nonsmooth.

Many software packages available for training: Caffe, PyTorch, TensorFlow, ... Many run on GPUs.

NN Training: Stochastic Gradient

NNs 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

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

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

- Choice of batch B_k (large or small) and step size α_k greatly affect performance of training algorithms. Schemes for choosing these terms (hyper-parameter optimization) are a major issue.
- Momentum terms sometimes help convergence: 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.

Calculating Gradients: "Backpropagation"

The structure of the function represented by the NN allows gradients $\nabla_x h(a_j, y_j; x)$ to be computed efficiently by using a form of the chain rule, or reverse-mode automatic differentiation.

The technique was already well known in other contexts (e.g. as "adjoints" in data assimilation for dynamic models).

Each loss function $h(a_j, y_j; x)$ in the summation has the "progressive" form

$$h(x) = \phi(\phi_l(x_l, \phi_{l-1}(x_{l-1}, \phi_{l-2}(x_{l-2} \dots (x_2, \phi_2(x_2, \phi_1(x_1)) \dots),$$

where

$$\phi_1 : \mathbb{R}^{n_1} \to \mathbb{R}^{m_1}, \quad \phi_i : \mathbb{R}^{n_i} \times \mathbb{R}^{m_{i-1}} \to \mathbb{R}^{m_i} \ (i = 2, \dots, l), \quad \text{and } \phi : \mathbb{R}^{m_l} \to \mathbb{R}.$$

Each x_i represents the weights introduced at layer i of the NN. The output ϕ_i of layer i depends on x_i as well as the output of the previous layer ϕ_{i-1}. The weights (variables) are introduced progressively by layers.

Calculating Gradients: "Backpropagation"

$$
\begin{aligned}
\nabla_{x_l} h(x) = &\quad (\nabla_{x_l}\phi_l)\,(\nabla_{\phi_l}\phi) \\
\nabla_{x_{l-1}} h(x) = &\quad \left(\nabla_{x_{l-1}}\phi_{l-1}\right)\left(\nabla_{\phi_{l-1}}\phi_l\right)(\nabla_{\phi_l}\phi) \\
\nabla_{x_{l-2}} h(x) = &\quad \left(\nabla_{x_{l-2}}\phi_{l-2}\right)\left(\nabla_{\phi_{l-2}}\phi_{l-1}\right)\left(\nabla_{\phi_{l-1}}\phi_l\right)(\nabla_{\phi_l}\phi) \\
\nabla_{x_{l-3}} h(x) = &\quad \left(\nabla_{x_{l-3}}\phi_{l-3}\right)\left(\nabla_{\phi_{l-3}}\phi_{l-2}\right)\left(\nabla_{\phi_{l-2}}\phi_{l-1}\right)\left(\nabla_{\phi_{l-1}}\phi_l\right)(\nabla_{\phi_l}\phi) \\
&\quad \vdots
\end{aligned}
$$

The matrices $\nabla_{\phi_{l-1}}\phi_l, \nabla_{\phi_{l-2}}\phi_{l-1}, \nabla_{\phi_{l-3}}\phi_{l-2}, \ldots$ appear in multiple terms.

- Can evaluate and store these terms during the progressive evaluation of $\phi_1, \phi_2, \ldots, \phi_l, \phi = h$.
- Then perform a backward recurrence to compute $\nabla_{x_l} h(x), \nabla_{x_{l-1}} h(x), \ldots, \nabla_{x_1} h(x)$.

Similar principles apply to more complex NN structures.

Overparametrized NN

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

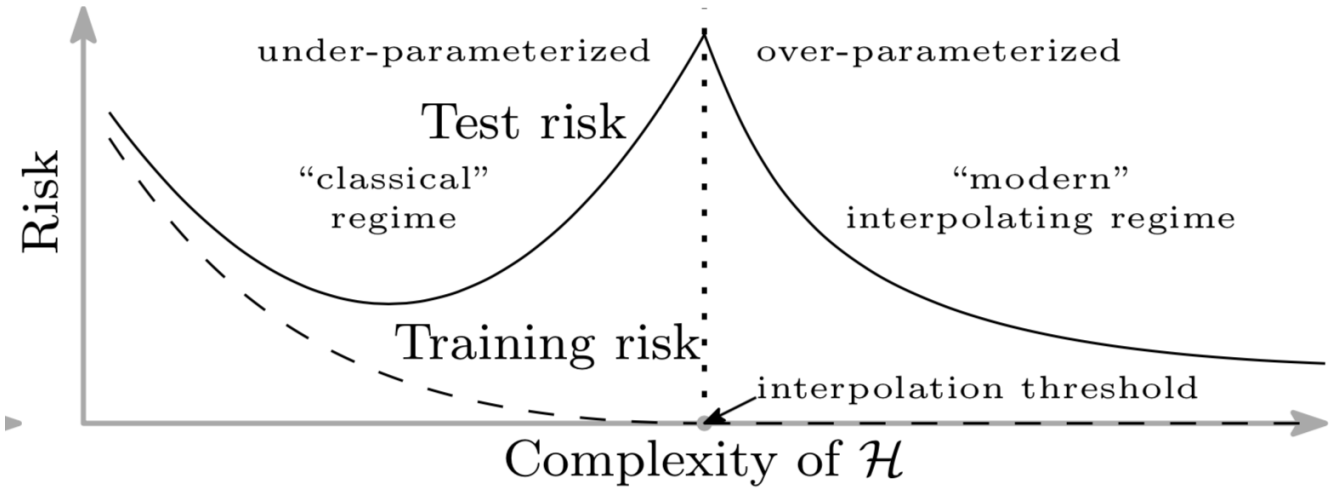
Training such networks can often achieve zero loss a.k.a. interpolation, that is, all items in the training data set are correctly classified. (In our earlier notation: $y_j = \phi(a_j)$, $j = 1, 2, \ldots, m$.)

Two big questions arise.

1. Isn't this overfitting? Apparently not: such models often generalize well, making good predictions on non-training data, flouting conventional wisdom.
2. Why is stochastic gradient (SGD) reliably finding the global minimum of the nonsmooth, nonlinear, nonconvex training problem?

Double Descent

[Belkin et al., 2018, Belkin et al., 2019, Belkin, 2021] show that as the number of parameters in the model is increased beyond the number needed to interpolate training data, performance on test data improves again: double-descent curve.



(b) "double descent" risk curve

The solution is not uniquely defined in the overparametrized regime. We need to be careful about which solution we converge to.

Discussion

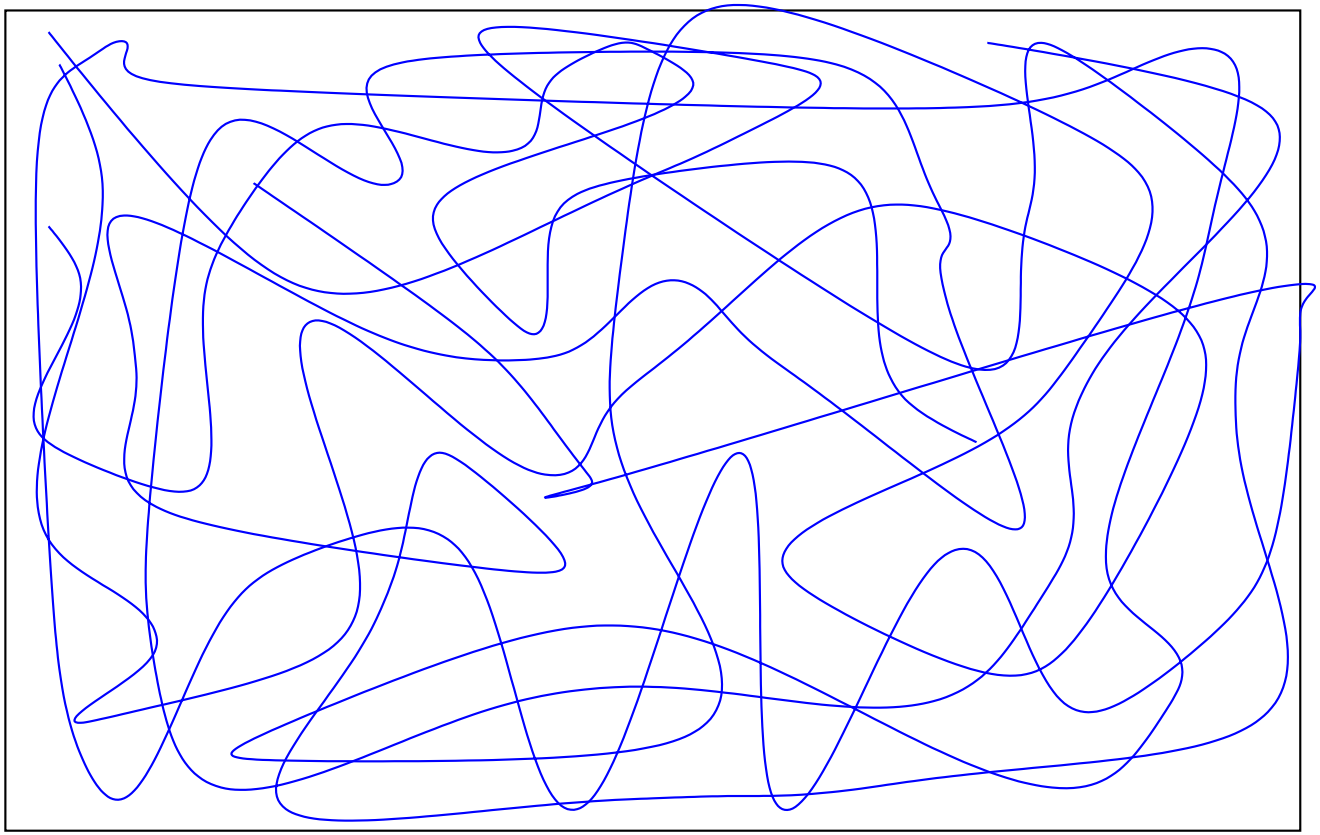
- As extra parameters are added to the model of ϕ, there is an entire manifold of interpolating solutions — among which are a small fraction that are "regular" in some sense, and generalize well.
- Classical theory (due to Vapnik, for example) fails to explain good generalization - because it gives results that apply to *all* interpolating solutions, not just the nice ones.
- Standard algorithms (e.g. gradient-based) are apparently good at finding such nice solutions — they have an "implicit regularization" or "inductive bias" property.
  - Typically, they don't even have to move far from an initial point to find an interpolating solution.

Finding Zero-Loss / Interpolating Solutions

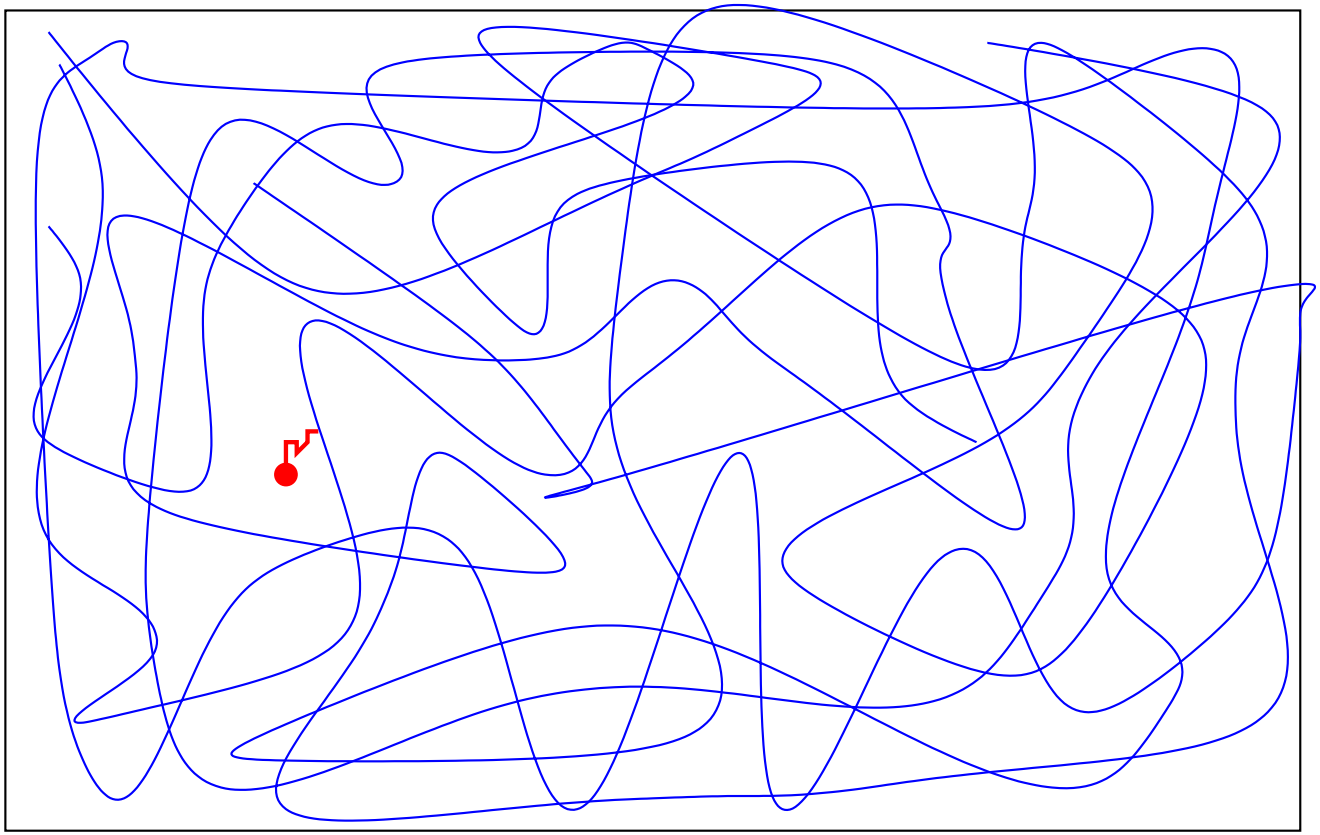
Thus, in an overparametrized NN, we observe:

- As the dimension grows, the "manifold" of solutions grows to fill the space.
- A random initial point for the weights will be close to the solution manifold — closer as the dimension grows.
- Don't need to change many ReLU activations to get from the initial point to the solution, so the nonsmoothness of ReLU may not be important.
- Gradient descent (or stochastic gradient descent with big enough batches) will move from the initial point to the solution.

Solution manifold fills the space as number of parameters grows. Any random starting point is close to a solution (and a gradient method will find it).



Solution manifold fills the space as number of parameters grows. Any random starting point is close to a solution (and a gradient method will find it).



Transition to Linearity: Neuro-Tangent Kernels

Suppose $\phi(a_j; x)$ defines the output of an NN with input $a_j \in \mathbb{R}^d$ and weights $x \in \mathbb{R}^n$. Training yields x such that $\phi(a_j; x) \approx y_j$.

Consider a general class of NN, with

- final layer with large width p, and
- output $\phi(a_j; x)$ is a linear function of the p final-layer outputs.

Define $\nabla_x \phi(a; x)$ to be a feature map:

$$\psi_x(a) := \nabla_x \phi(a; x) \in \mathbb{R}^n,$$

and kernel function $K_{(a,z)}(x) := \langle \nabla_x \phi(a; x), \nabla_x \phi(z; x) \rangle = \langle \psi_x(a), \psi_x(z) \rangle$.

Under these conditions, $\psi_x(a)$ (and hence $K_{(a,z)}(x)$) are almost independent of x, for x near a reference / starting point x_0. Thus ϕ is almost linear in x:

$$\phi(a; x) \approx \phi(a; x_0) + \langle \psi_{x_0}(a), x - x_0 \rangle.$$

Why is the kernel $K_{(a,z)}$ interesting?

The function K is familiar in the context of kernel learning / kernel regression. It appears for example when we solve the least-distance interpolation problem for the training data around some given point x_0:

$$\min_x \ \tfrac{1}{2}\|x - x_0\|_2^2 \ \text{ s.t. } \ \phi(a_j; x) + \langle \psi_{x_0}(a_j), x - x_0 \rangle = y_j, \ \ j = 1, 2, \ldots, m.$$

Optimality conditions: there are multipliers $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_m)$ such that

$$x - x_0 = \sum_{j=1}^{m} \lambda_j \psi_{x_0}(a_j), \quad \langle \psi_{x_0}(a_j), x - x_0 \rangle = y_j - \phi(a_j; x_0)) =: r_j, \ \ j = 1, 2, \ldots,$$

Defining the $n \times n$ kernel *matrix* based on the training data:

$$K_{ij} := K_{a_i, a_j}(x_0) = \langle \psi_{x_0}(a_i), \psi_{x_0}(a_j) \rangle, \quad i, j = 1, 2, \ldots, m,$$

we have $\lambda = K^{-1} r$, from which the optimal weights x can be recovered.

Usefulness of (approximate) linearity

This suggests why linearity of f is important — it vastly simplifies the interpolation / optimization process.

The whole setup reduces to Kernel Learning (see Application 10) where the data item a_j is lifted to $\psi_{x_0}(a_j) = \nabla_x \phi(a_j; x)$.

Such problems reduce to being "almost quadratic" — provided we stay in the region of near-linearity, near x_0.

Can show that particular NNs have this property:

- wide final layer
- output f is a linear combination of the final layer outputs.

See [Belkin, 2021].

Recent Perspectives

Investigations continue into the related issues of finding zero-loss solutions and generalizability.

Many consider simplified, impractical networks (e.g. single-hidden-layer, very wide). Recent example: [Ding et al., 2022] consider a wide, deep ResNet, proving global convergence results via mean-field and ODE analysis of the continuous limits.

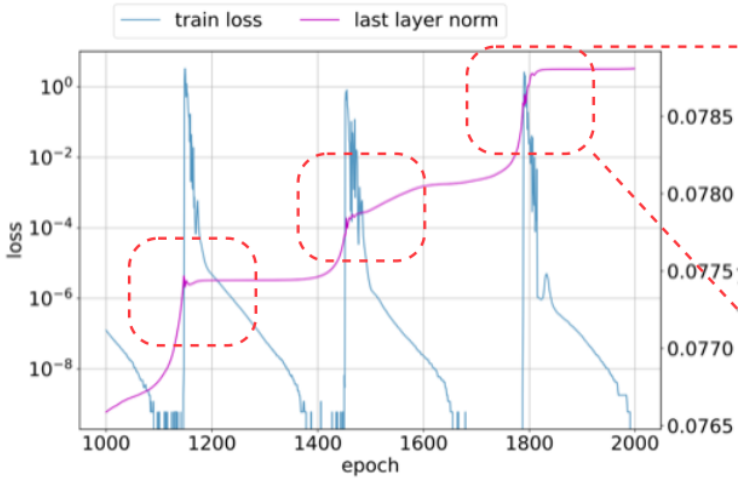
Catapult [Zhu et al., 2022]: Uses a quadratic approximation to the NN loss (rather than the linear NTK approximation) to explain a phenomenon seen in training with large steplengths.

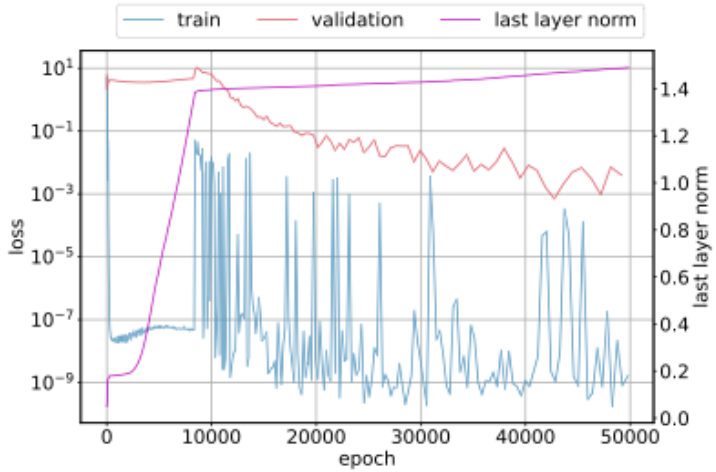
- Loss sometimes increases sharply, then peaks and decreases rapidly.
- Observed in many settings, can be explained convincingly using quadratic approx.

Recent Perspectives

Other phenomenon related to "catapult" are grokking and slingshot [Thilak et al., 2022].

- Observed for adaptive stochastic gradient methods (like Adam).
- Loss converges to *almost* zero, but occasionally spikes.
- Spikes cause norm of last layer of NN weights to jump, cause generalization (as measured by validation error) to continue improving long after the training loss has converged nearly to zero.

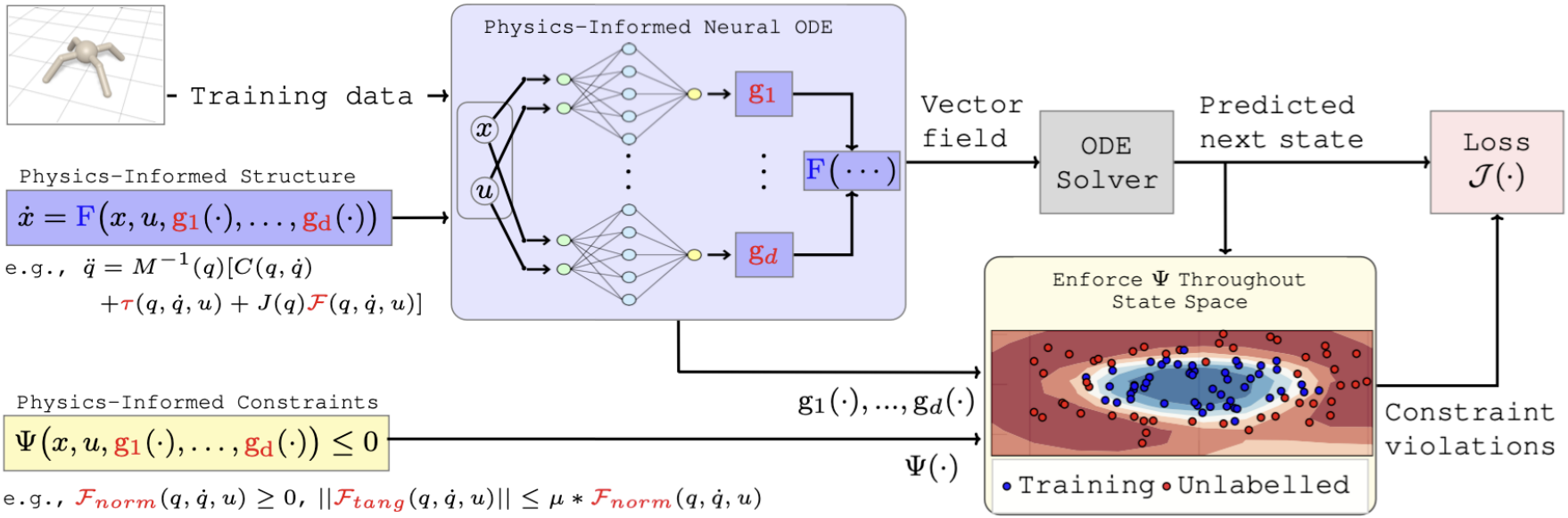




NNs with Output Constraints

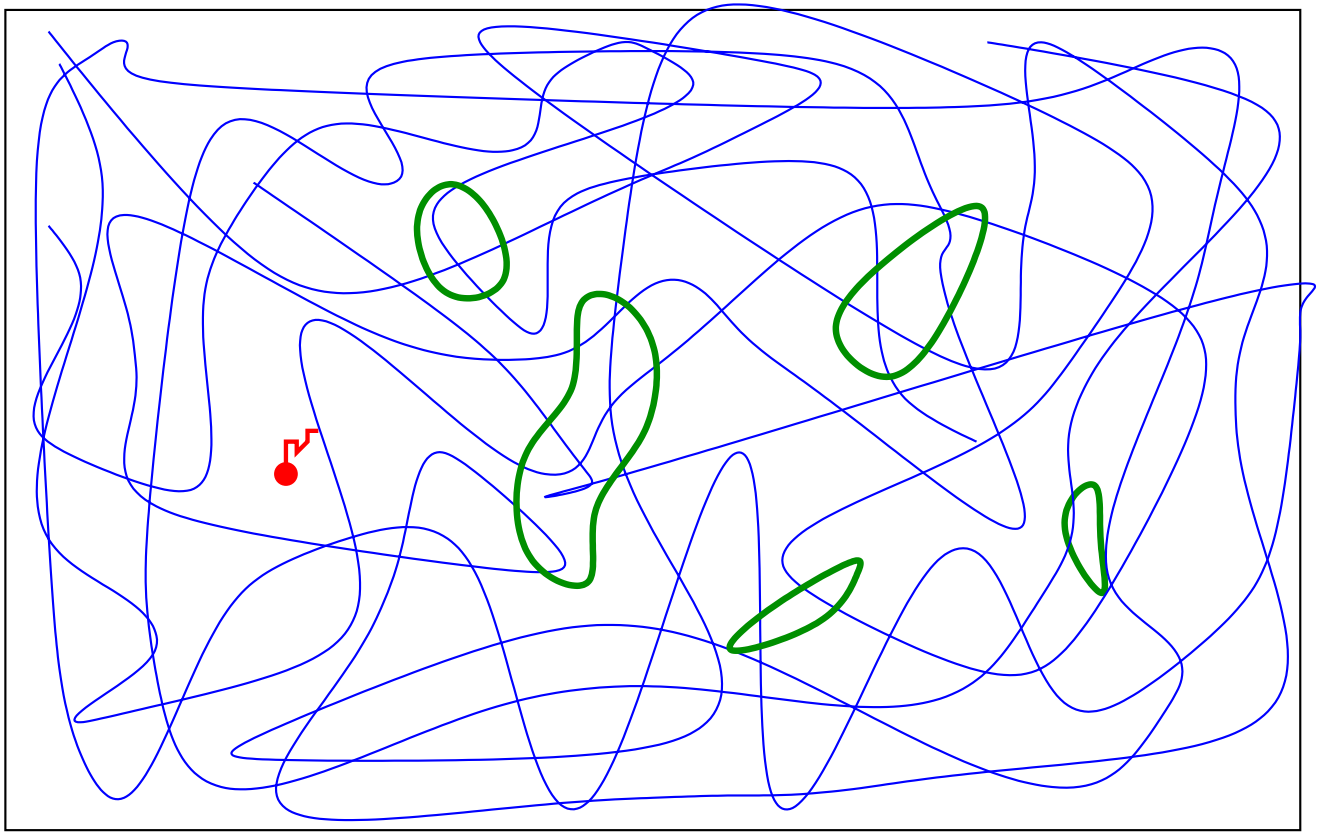
Much recent work on Physics-Informed NNs (PINNs) in which physical modeling is combined with NNs that "fill gaps" in the models.

Often the output of these NN components must be constrained to make sense for the model. Example from [Djeumou et al., 2022]:



For constrained problems, feasible solution may be distant from a random starting point, so the usual logic of overparametrization + random starting does not apply.

Algorithms can "get lost" traversing the parameter space between initial point and feasible solution.



Algorithms for Constrained NNs

What algorithms work for Constrained NN?

- This area is in its infancy & is an active research topic.
- Naive methods (e.g. quadratic penalty) don't work - possibly for the reasons shown above. Augmented Lagrangian has had some success (e.g. [Lu et al., 2021, Djeumou et al., 2022]).
- But there are many other algorithms for constrained optimization that might be relevant:
  - Exact penalty,
  - Primal-dual interior-point,
  - SQP.
- One question is: Can these methods adapt well to NN problems, where exact first-derivative information is not available. We only have "stochastic gradients."

Formulations: Summary

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

The usefulness of the optimization perspective continues to grow. See for example the use of robust optimization tools in formulating adversarial ML and distributionally robust learning.

Usually not enough to use off-the-shelf optimization software to solve the optimization formulations. The algorithms need to be customized to problem structure, context, and size.

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


Alacaoglu, A., Cevher, V., and Wright, S. J. (2022).
On the complexity of a practical primal-dual coordinate method.


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.


Belkin, M. (2021).
Fit without fear: The remarkable mathematical phenomenon of deep learning through the prism of interpolation.
Technical Report arXiv:2105.14368, UC San Diego.
To appear in *Acta Numerica*.


Belkin, M., Hsu, D., Ma, S., and Mandal, S. (2018).
Reconciling modern machine learning and the bias-variance trade-off.
Technical Report arXiv:1812.11118, The Ohio State University.

References II


Belkin, M., Hsu, D., and Xu, J. (2019).
Two models of double descent for weak features.
Technical Report arXiv:1903.07571, The Ohio State University.


Bhojanapalli, S., Neyshabur, B., and Srebro, N. (2016).
Global optimality of local search for low-rank matrix recovery.
Technical Report arXiv:1605.07221, Toyota Technological Institute.


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., Li, X., and Soltanolkotabi, M. (2015).
Phase retrieval via a Wirtinger flow.
IEEE Transactions on Information Theory, 61:1985–2007.


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 III


Chen, Y. and Wainwright, M. J. (2015).
Fast low-rank estimation by projected gradent descent: General statistical and algorithmic guarantees.
Technical Report arXiv:1509.03025, University of California-Berkeley.


Chi, Y., Lu, Y. M., and Chen, Y. (2019).
Nonconvex optimization meets low-rank matrix factorization: An overview.
IEEE Transactions in Signal Processing, 67:5239–5269.


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.

References IV


Ding, Z., Chen, S., Li, Q., and Wright, S. J. (2022).
Overparameterization of deep resnet: Zero loss and mean-field analysis.
Journal of Machine Learning Research, 23:1–65.


Djeumou, F., Neary, C., Goubault, E., Putot, S., and Topcu, U. (2022).
Neural networks for physics-informed architectures and constraints for dynamical systems modeling.
Proceedings of Machine Learning Research, 144:1–14.


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


Ho-Nguyen, N. and Wright, S. J. (2022).
Adversarial classification via distributional robustness with Wasserstein ambiguity.
Mathematical Programming, Series B, pages 1–37.


Keshavan, R. H., Montanari, A., and Oh, S. (2010).
Matrix completion from a few entries.
IEEE Transactions on Information Theory, 56(6):2980–2998.


Lu, L., Pestourie, R., Yao, W., Wang, Z., Verdugo, F., and Johnson, S. G. (2021).
Physics-informed neural networks with hard constraints for inverse design.
SIAM Journal on Scientific Computing, 43(6):B1105–B1132.

References V


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.


Song, C., Lin, C. Y., Wright, S. J., and Diakonikolas, J. (2021a).
oordinate linear variance reduction for generalized linear programming.
arXiv.


Song, C., Wright, S. J., and Diakonikolas, J. (2021b).
Variance reduction via primal-dual accelerated dual averaging for nonsmooth convex finite-sums.
In *International Conference on Machine Learning*, pages 9824–9834. PMLR.


Thilak, V., Littwin, E., Zhai, S., Saremi, O., Paiss, R., and Susskind, J. (2022).
The slingshot mechanism: An empirical study of adaptive optimizers and the grokking phenomenon.
arXiv.


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 VI

Zheng, Q. and Lafferty, J. (2015).
A convergent gradient descent algorithm for rank minimization and semidefinite programming from random linear measurements.
Technical Report arXiv:1506.06081, Statistics Department, University of Chicago.

Zhu, L., Liu, C., Radhakrishnan, A., and Belkin, M. (2022).
Quadratic models for understanding neural network dynamics.
arXiv.