Data Assimilation from the Big Data Perspective

Stephen Wright

University of Wisconsin-Madison

January 2016

Parametric / Nonparametric Models

Machine learning is a booming field. It's closely related to statistical inference, learning from data, data analysis, "big data."

In many contexts, models are "nonparametric". Overwhelming amounts of data take the place of careful modeling based on physical principles or other prior knowledge of the system.

Data assimilation is based on perhaps the ultimate parametric model, a highly successful merging of domain knowledge (meteorology, oceanography), physical modeling, statistical principles, and scientific computing. Still, it might be worth considering some perspectives from learning.

This talk will touch on several currently popular areas of research in machine learning with possible relevance to data assimilation and forecasting.

Data Assimilation

Unknown: Initial state $x_0 := \tilde{x}$.

Dynamics: $x_i = \mathcal{M}(x_{i-1})$, $i = 1, 2, \ldots, N$;

Objective: $J(\tilde{x}, x_0, x_1, \ldots, x_N)$:

$$\frac{1}{2}(\tilde{x} - x^b)^T B^{-1}(\tilde{x} - x^b) + \frac{1}{2}\sum_{i=0}^{N}(\mathcal{H}(x_i) - y_i)^T R_i^{-1}(\mathcal{H}_i(x_i) - y_i).$$

x^b = background / prior; B, R_i, $i = 0, 1, 2, \ldots, N$: covariances.

Derived from maximum-likelihood expression, assuming Gaussian (correlated) noise in measurements with known covariances, and a prior with known covariance.

Lots of assumptions here, but it works well in practice.

Data Assimilation

Unknown: Initial state $x_0 := \tilde{x}$.

Dynamics: $x_i = \mathcal{M}(x_{i-1})$, $i = 1, 2, \ldots, N$;

Objective: $J(\tilde{x}, x_0, x_1, \ldots, x_N)$:

$$\frac{1}{2}(\tilde{x} - x^b)^T B^{-1}(\tilde{x} - x^b) + \frac{1}{2}\sum_{i=0}^{N}(\mathcal{H}(x_i) - y_i)^T R_i^{-1}(\mathcal{H}_i(x_i) - y_i).$$

x^b = background / prior; B, R_i, $i = 0, 1, 2, \ldots, N$: covariances.

Derived from maximum-likelihood expression, assuming Gaussian (correlated) noise in measurements with known covariances, and a prior with known covariance.

Lots of assumptions here, but it works well in practice.

You all know this!

What is Data Analysis?

- Extract meaning from data: Understand statistical properties, important features, fundamental structures in the data. Inference, Data Analysis.
- Use this knowledge to make predictions about other, similar data. Machine Learning.
- Data Mining encompasses both.

... but the usage of these terms is not standard.

Foundations in Statistics, then Computer Science: AI, Machine Learning, Databases, Parallel Systems. Also Optimization.

For a recent Statistics perspective on past, present, and future, see David Donoho: *50 Years of Data Science*, September 2015.

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).

Typical Setup in Machine Learning

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

- Vectors a_j, $j = 1, 2, \ldots, m$ whose elements are "features."
- One outcome / observation / label y_j for each feature vector ($j = 1, 2, \ldots, m$).

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.
- null: We might simply know that all a_j lie approximately in the same subspace, and aim to find that subspace OR we may suspect that the a_j are grouped into several distinct clusters.

A Fundamental Task

Seek a function ϕ that:

- approximately maps a_j to y_j for each j: $\phi(a_j) \approx y_j$ for $j = 1, 2, \ldots, m$.
- satisfies some additional properties — generalizability, simplicity, structure — that make it "plausible" in the context of the application and robust to small changes in the data.

Can usually define ϕ in terms of some parameter vector x. Then the problem of finding ϕ becomes a data-fitting problem.

- Objective function is built up of m loss terms that capture the mismatch between predictions and observations for each data item.

Extra regularization functions or constraints — regularized or constrained data-fitting — ensure that ϕ is not "overfit" to the training data.

The process of finding ϕ is often called learning or training.

Why?

What's the use of ϕ?

Analysis: ϕ — especially the parameter x that defines it — reveals structure in the data. Examples:

- which components (features) in vectors a_j are most important in determining the outputs y_j.
- quantify how the output depend on the features.
- show how each a_j can be expressed in terms of a small number of basis vectors. (X defines these basis vectors.)

Prediction: Given a new data vector a_k, predict the output y_k to be $\phi(a_k)$.

Complications

- a_j and y_j can contain noise or errors. Would like ϕ (and x) to be robust to these errors. Thus, generalization / regularization.
- Missing Data: Vectors a_j may be missing elements (but may still contain useful information).
- Some or all y_j may be missing or null, as in semi-supervised or unsupervised learning.
  - We may be able to assign appropriate values for y_j after examining the training set of a_j, e.g. clustering.
- Online learning: Data (a_j, y_j) is arriving in a stream rather than all known up-front.

Examples of ϕ: Least Squares

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

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

- Could add an intercept: $\phi(a_j) = a_j^T x - \beta$: Parameter is (x, β).
- Add ℓ_2 regularization (Tikhonov) to reduce sensitivity of the solution x to noise in y.

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

- Could add ℓ_1 regularization to obtain solution x with few nonzeros:

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

- Nonconvex separable regularizers (SCAD, MCP) have nice statistical properties — but lead to nonconvex optimization formulations.

Examples of ϕ: 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

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

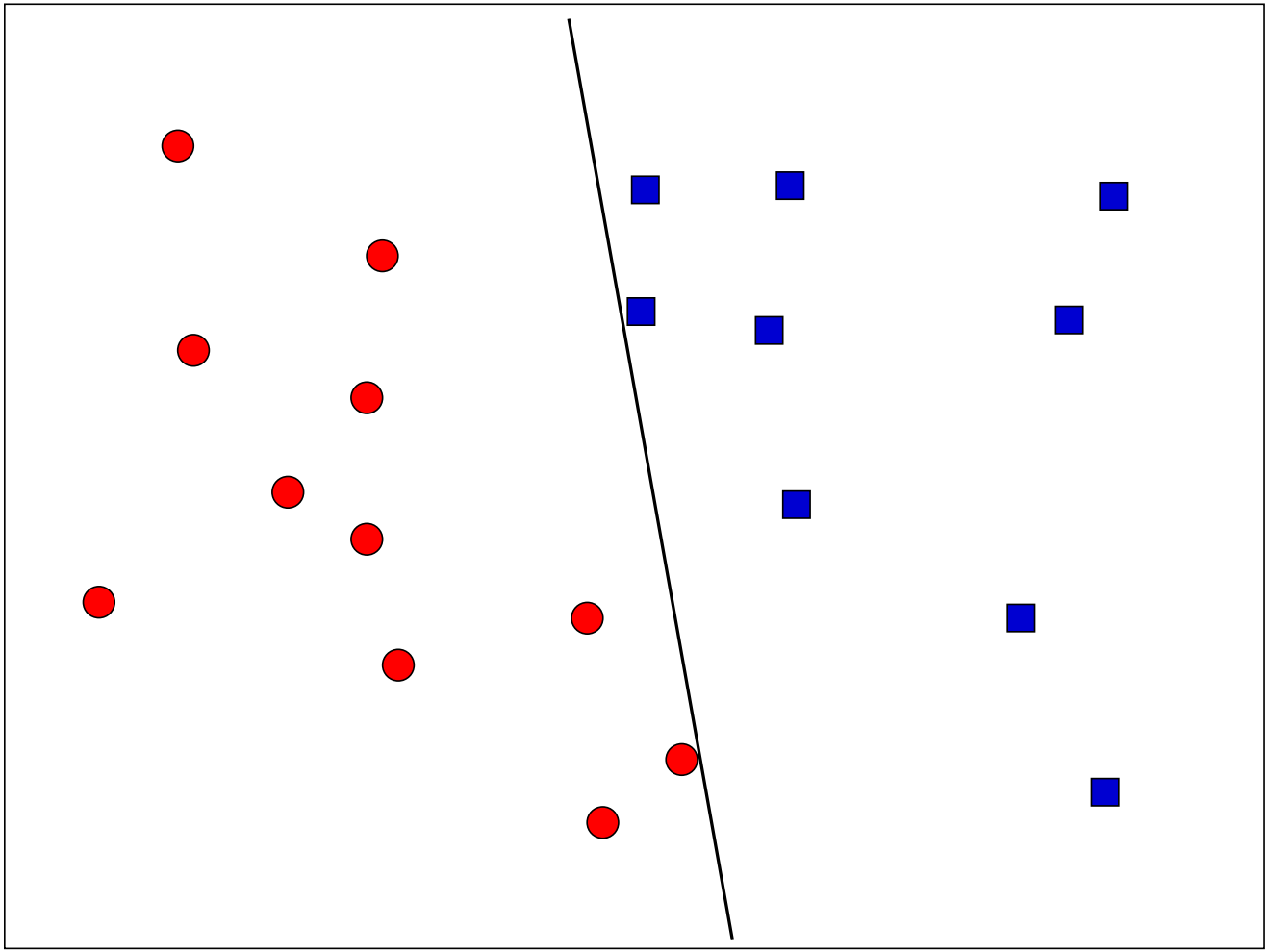
The mapping is $\phi(a_j) = \text{sign}(a_j^T x - \beta)$.

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:

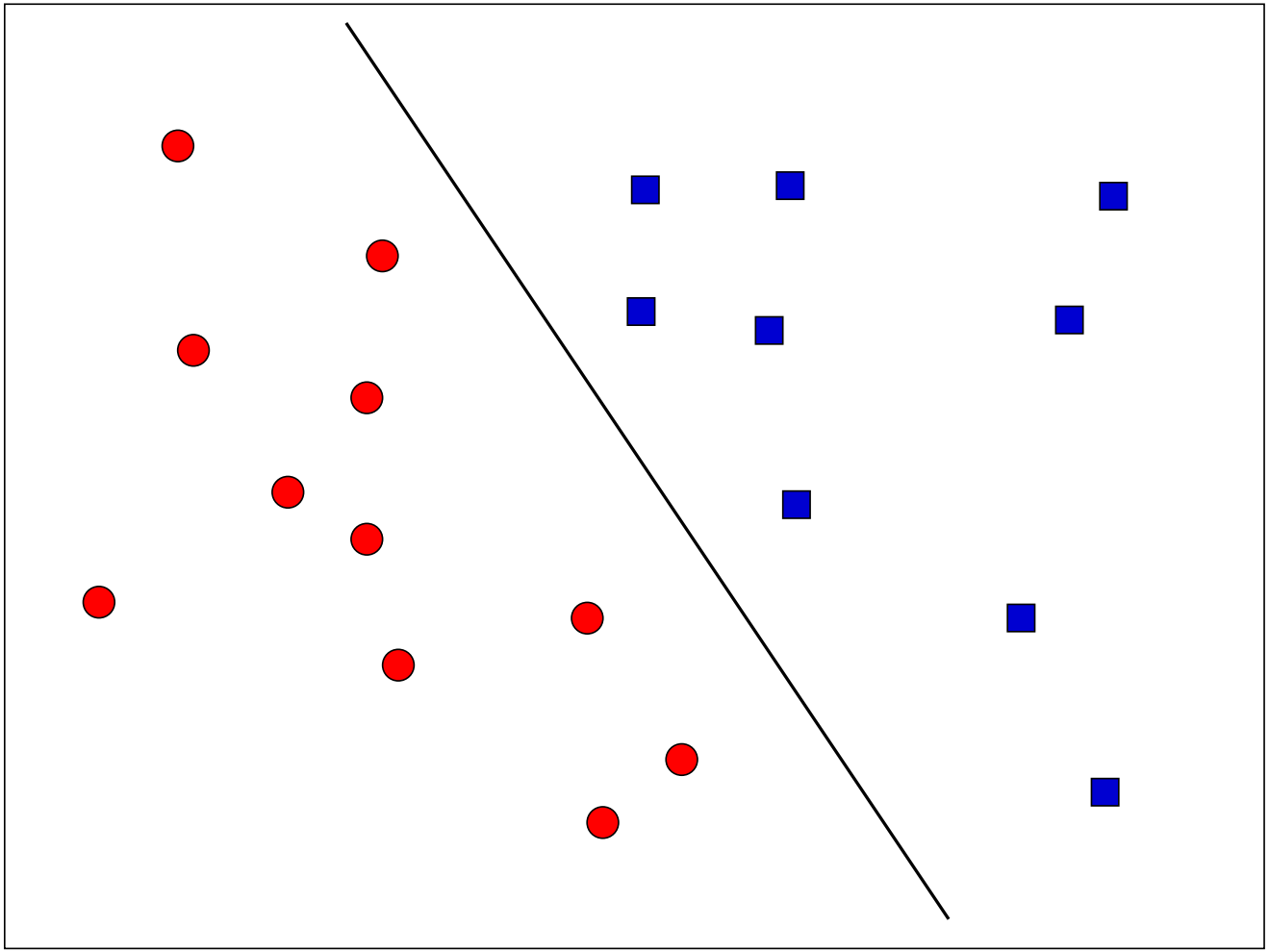
$$\frac{1}{m} \sum_{j=1}^{m} \max(1 - y_j(a_j^T x - \beta), 0).$$

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

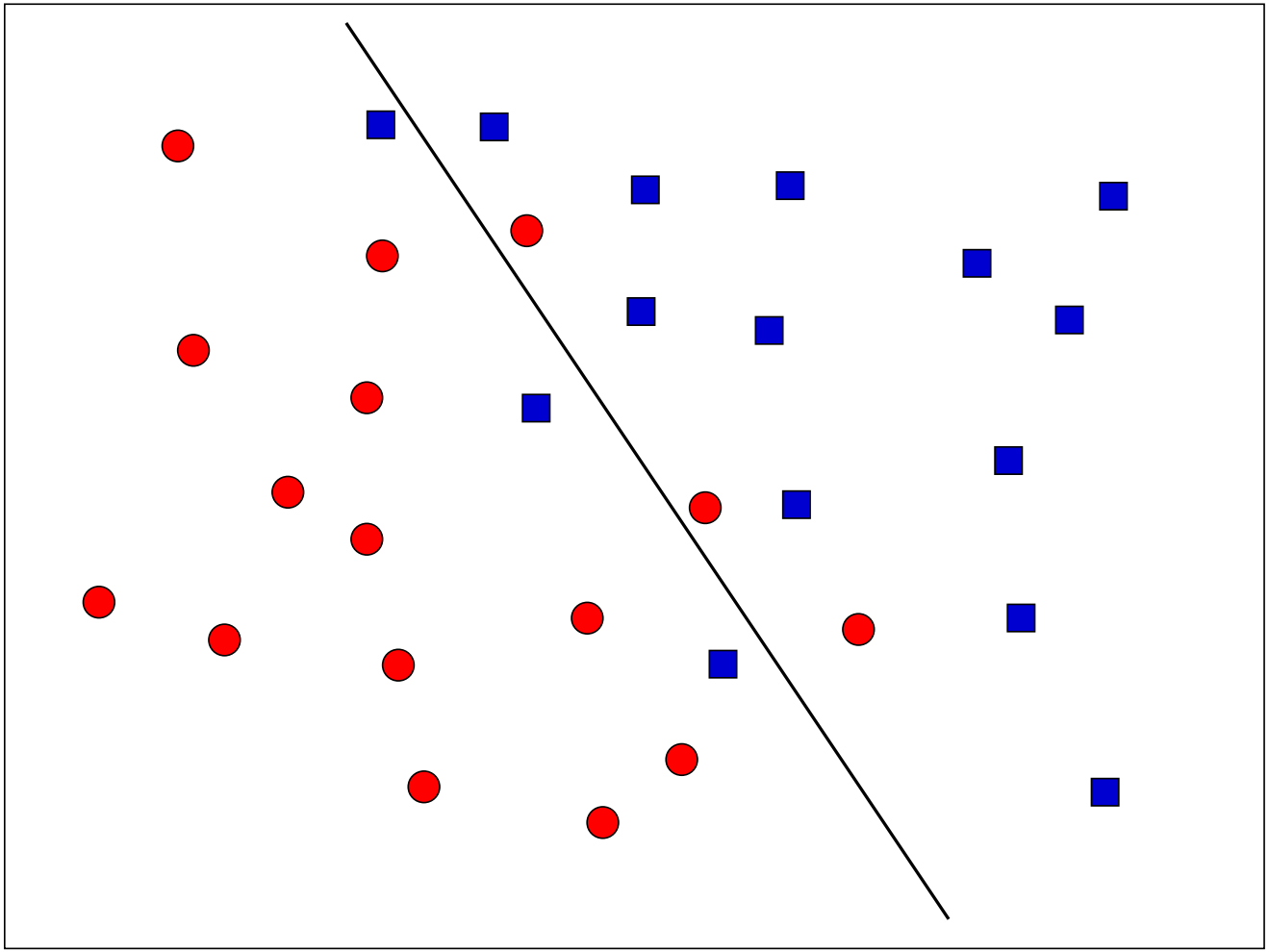
Regularize for Plausibility (Generalizability)



Regularize for Plausibility (Generalizability)



Regularize for Plausibility (Generalizability)



Examples of ϕ: Nonlinear Least Squares

... with regularization:

$$\min_x f(x) := \frac{1}{2m} \sum_{j=1}^{m} h(x; a_j) + \lambda \Omega(x).$$

where $h(x; a_j) = (\phi(a_j; x) - y_j)^2$ is the squared discrepancy between predicted and observed measurements, and Ω is a regularizer.

So, when we include covariances, we're back to data assimilation.

It fits the framework of learning problems, but for a model ϕ that's more physics-based than is typical in "big data" applications.

Loss Functions

The loss function $h(x; a_j)$ in data analysis is sometimes based on statistical principles, and sometimes a matter of art.

Loss functions in classification problems are not obvious — the obvious one (misclassification) is discontinuous, so it is usually replaced with a convex surrogate.

For regression, least-squares objectives make sense when noise is Gaussian and model error is neglible.

Other, more robust estimators have been tried.

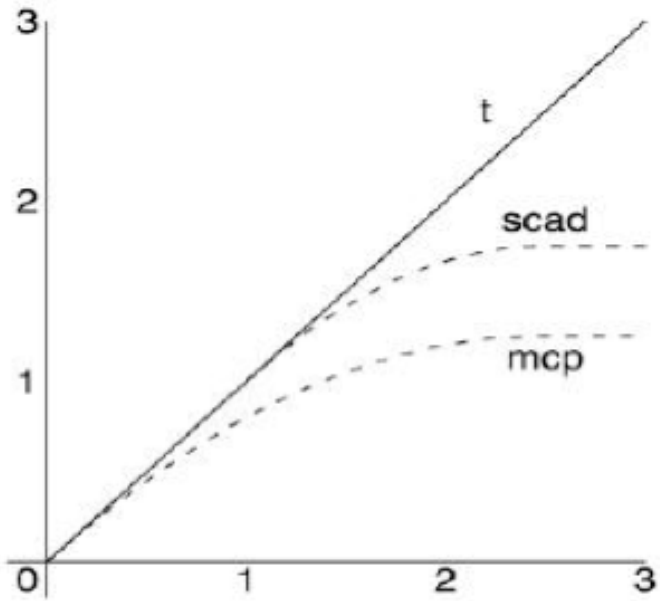
- Are any of these relevant to data assimilation?
- How are outliers among the measurements currently dealt with in data assimilation?

Robust Loss Functions

- ℓ_1: $h(x; a_j) = |\phi(x; a_j) - y_j|$.
- Huber:

$$h(x; a_j) = \begin{cases} (\phi(x; a_j) - y_j)^2 & \text{if } |\phi(x; a_j) - y_j| < T; \\ T|\phi(x; a_j) - y_j| & \text{otherwise.} \end{cases}$$

- Nonconvex: MCP, SCAD. Lead to unbiased estimators.



Regularization

- In machine learning, the function to be minimized is an empirical approximation of some underlying reality:

$$\frac{1}{m}\sum_{j=1}^{m} h(x; a_j) \approx \mathbb{E}_a h(x; a),$$

  where the expectation is taken over the a space. (The true expectation is of course not available.)
- Exact minimization of the empirical function would cause overfitting to the particular data vectors $a_1, a_2, \ldots, a_m$.
- Use regularization to make the computed x^* a better approximation to the minimizer of $\mathbb{E}_a h(x; a)$.

Regularization improves likelihood of fitting unseen data, and generalizability. Also reduces sensitivity to noise in observations (that is, error in $h(\cdot; a_j)$).

Regularization

Regularization can be motivated via Bayesian statistics.

Likelihood (observation model): $p(y|x) = \frac{1}{Z_l} \exp\big(-h(x,y)\big)$

Prior: $p(x) = \frac{1}{Z_p} \exp\big(-\lambda\psi(x)\big)$

Posterior: $p(x|y) = \frac{p(y|x)\ p(x)}{p(y)}$

Log-posterior: $\log\ p(x|y) = K(y) - h(x,y) - \lambda\psi(x)$

$\widehat{x}$ is a maximum a posteriori (MAP) estimate.

$\psi(x)$ is the regularizer, λ is the regularization parameter.

Familiar Regularizers

- $\psi(x) = \|x\|^2$: Used in least-squares to reduce sensitivity to noise.
- $\psi(x) = \|x\|_1$. Has become extremely popular in compressed sensing, feature selection, machine learning, many application areas where sparse solutions are required: x with relatively few nonzeros. Control the number of nonzeros by manipulating regularization parameter λ.
- group-sparse: $\sum_{j=1}^{m} \|x_{[j]}\|_2$ or $\sum_{j=1}^{m} \|x_{[j]}\|_\infty$, where each $x_{[j]}$ is a subvector of x. Turns the subvectors "on" and "off."
- In matrix optimization, the nuclear norm $\|X\|_*$, defined to be the sum of singular values, tends to induce low rank in X.
- Total variation norm in image processing tends to produce more natural-looking images, with sharp edges separatiing fields of constant color/intensity.

Many others. The "trick" is to define the regularizer to match the structure you seek, which is a kind of "prior."

Randomized Algorithms

Stochastic gradient has become a workhorse algorithm in machine learning. It's designed for problems of the form

$$\min_x \mathbb{E}_a h(x;a) \quad \text{or} \quad \min_x h(x) := \frac{1}{m}\sum_{j=1}^{m} h(x;a_j).$$

We focus on the latter.

Iteration k:

- Choose $j_k \in \{1,2,\ldots,m\}$ uniformly at random;
- $x^{k+1} \leftarrow x^k - \alpha_k \nabla_x h(x^k; a_{j_k})$;

Uses $\nabla_x h(x^k; a_{j_k})$ as a proxy for $\nabla h(x^k)$.

Useful in machine learning because $\nabla_x h(x^k; a_{j_k})$ can be evaluated by looking at just one item of data, whereas $\nabla h(x)$ requires us to scroll through the entire data set — much of it probably redundant.

Incremental Gradient Algorithms

Given $h(x) := \frac{1}{m}\sum_{j=1}^{m} h(x; a_j)$, an incremental gradient algorithm is like stochastic gradient, except that it does not select j_k uniformly at random but rather scrolls through them in order:

$$1, 2, 3, \ldots, m, 1, 2, \ldots, m, 1, 2, \ldots .$$

Different steplength schemes produce different convergence results. Standard conditions:

$$\sum_{j=1}^{\infty} \alpha_j = \infty, \quad \sum_{j=1}^{\infty} \alpha_j^2 < \infty.$$

Randomized Algorithms in Data Assimilation?

Is there any place for randomized algorithms in data assimilation?

- Is Ensemble Kalman Filter a kind of incremental algorithm, incorporating one measurement at a time into the objective?
- Is there any gain to be had in performing SQP iterations in 4DVAR with a different, random selection of measurement data?
  - How does the number of measurements factor into the cost of an SQP iteration?
  - Have other schemes been tried e.g. performing one SQP iteration with relatively few measurements, then gradually incorporating more measurements on subsequent iterations?

Inverse Covariance Estimation

Suppose we have a radom variable $Y \in \mathbb{R}^n$, where $Y \sim \mathcal{N}(\mu, C)$ for some (unknown) covariance C.

Given N samples of Y, say $y_1, y_2, \ldots, y_N$), we can estimate C by the sample covariance $S := \sum_{i=1}^{N} (y_i - \mu)(y_i - \mu)^T/(N-1)$

We can formulate the problem of finding the inverse of S as follows:

$$\max_{X \succ 0} \; f(X) := \log\det(X) - \langle X, S \rangle.$$

This is because $\nabla_X f(X) = X^{-1} - S$, so $X = S^{-1}$ at the solution.

Zeros in X reveal conditional independencies between components of Y:

$$P_{ij} = 0 \quad \Leftrightarrow \quad Y_i \perp\!\!\!\perp Y_j | \{Y_k, \; k \neq i, j\}$$

...exploited to infer (in)dependencies among Gaussian variables. Widely used in computational biology and neuroscience, social network analysis, ...

Sparse Inverse Covariance

We can include a regularization term in the objective to encourage sparse solutions:

$$\max_{X \succ 0} \log\det(X) - \langle X, S\rangle - \lambda\|X\|_1,$$

where $\|X\|_1$ is the "element-wise" norm of the symmetric positive definite $n \times n$ matrix X.

This can be solved with a coordinate descent scheme; see Scheinberg and Ma (2012).

Each step: choose a single entry of X, indexed by (i, j) (and its symmetric counterpart (j, i)) and find scalar θ that maximizes the objective for $X + \theta e_i e_j^T + \theta e_j e_i^T$.

Surprisingly, we can find θ exactly! Need some linear algebra tricks, facts about matrix logs and determinants, and simple calculus.

Suppose we know $W = X^{-1}$. Then can show

$$\det(X + \theta e_i e_j^T + \theta e_j e_i^T) = (\det X)(1 + 2W_{ij}\theta + \theta^2(W_{ij}^2 - W_{ii}W_{jj})).$$

The problem of maximizing along θ is equivalent to

$$\max_{\theta} \log(1 + 2W_{ij}\theta + \theta^2(W_{ij}^2 - W_{ii}W_{jj})) - 2S_{ij}\theta - 2\lambda|X_{ij} + \theta|.$$

Using u to denote an element in the subdifferential of $\partial|\cdot|$ at $X_{ij} + \theta$, we set the derivative of the subproblem to zero:

$$2\frac{W_{ij} + \theta(W_{ij}^2 - W_{ii}W_{jj})}{1 + 2W_{ij}\theta + \theta^2(W_{ij}^2 - W_{ii}W_{jj})} - 2S_{ij} - 2\lambda u.$$

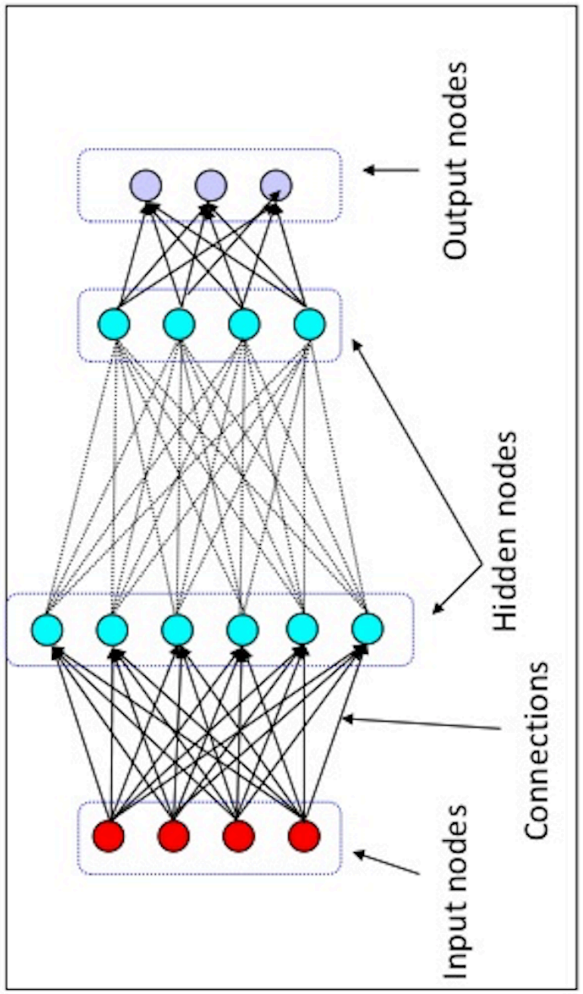
The θ and u that satisfy this expression can be obtained by following the same procedure as for the prox-operator for $\|x\|_1$.

- Try setting $u = +1$ and solve a quadratic for θ. If $X_{ij} + \theta \geq 0$, we are done.
- Try setting $u = -1$ and solve a quadratic for θ. If $X_{ij} + \theta \leq 0$, we are done.
- Otherwise set $\theta = -X_{ij}$; we must have $u \in [-1, 1]$ for this value of θ.

Finally, update $W = X^{-1}$ to reflect the rank-2 change to X using Sherman-Morrison-Woodbury.

Requires $O(n^2)$ steps per iteration.

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:

$$a^{l+1} = \sigma(W^l a^l + h^l),$$

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

Deep Learning

The element-wise function σ makes simple transformations to each element:

- Logistic function: $t \to 1/(1+e^{-t})$;
- Hinge: $t \to \max(t, 0)$;
- Bernoulli: $t \to 1$ with probability $1/(1+e^{-t})$ and $t \to 0$ otherwise (inspired by neuron behavior).

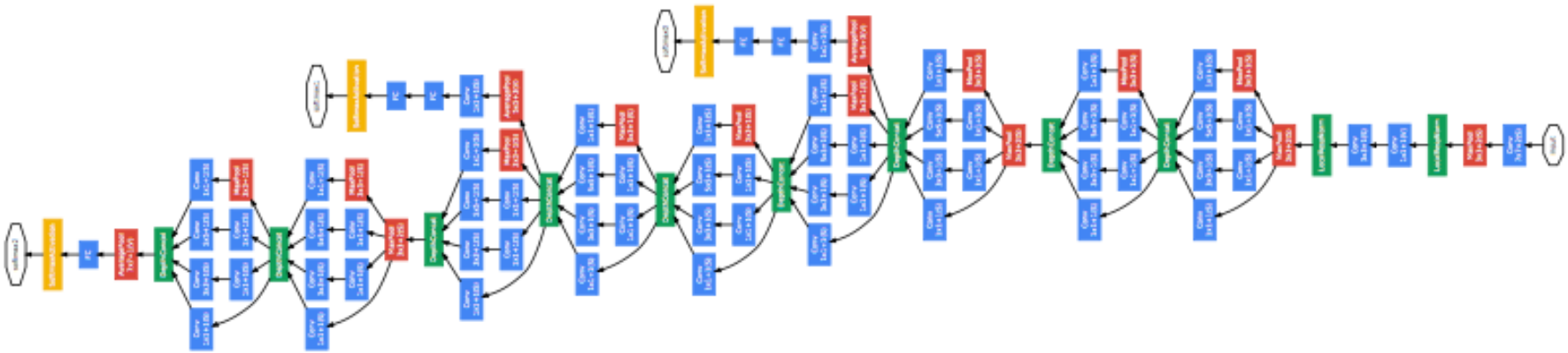
The example depicted shows a completely connected network — but more typically networks are engineered to the application (speech processing, object recognition, ...).

- local aggregation of inputs: pooling;
- restricted connectivity + constraints on weights (elements of W^l matrices): convolutions.

These models are nominally nonparametric, but they have tons of parameters!

GoogLeNet

Visual object recognition: Google's State of the Art network from 2014 Szegedy et al. (2015).



Training Deep Learning Networks

The network contains many parameters — (W^l, h^l), $l = 1, 2, \ldots, L$ 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 $x = (W^1, h^1, W^2, h^2, \ldots)$ 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).

Nonlinear, Nonconvex. It's also random in some cases (but then we can work with expectation).

Composition of many simple functions.

Algorithms for Deep Learning

Stochastic gradient (SG) is the workhorse method. At iteration k:

- Choose training example j_k uniformly at random;
- Set

$$x^{k+1} \leftarrow x^k - \nabla_x h(x^k; a_{j_k}, y_{j_k}).$$

Adjoint procedures used to evaluate $\nabla_x h$.

Many enhancements:

- Regularization.
- Knockout.
- Minibatch.
- Reduced variance: SAG, SVRG, SAGA. Essentially hybrids of SG and steepest descent.

References I

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

Scheinberg, K. and Ma, S. (2012). Optimization methods for sparse inverse covariance selection. In Sra, S., Nowozin, S., and Wright, S. J., editors, *Optimization for Machine Learning*, Neural Information Processing Series. MIT Press.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *CVPR*.