Linear Models

Numerical Methods for Deep Learning

Lars Ruthotto

Departments of Mathematics and Computer Science, Emory University

Course Overview

Course Overview

- Part 1: Linear Models
  1. Introduction and Applications
  2. Linear Models: Least-Squares and Logistic Regression
- Part 2: Neural Networks
  1. Introduction to Nonlinear Models
  2. Parametric Models, Convolutions
  3. Single Layer Neural Networks
  4. Training Algorithms for Single Layer Neural Networks
  5. Neural Networks and Residual Neural Networks (ResNets)
- Part 3: Neural Networks as Differential Equations
  1. ResNets as ODEs
  2. Residual CNNs and their relation to PDEs

Intro: Machine Learning

Machine Learning in 3 slides

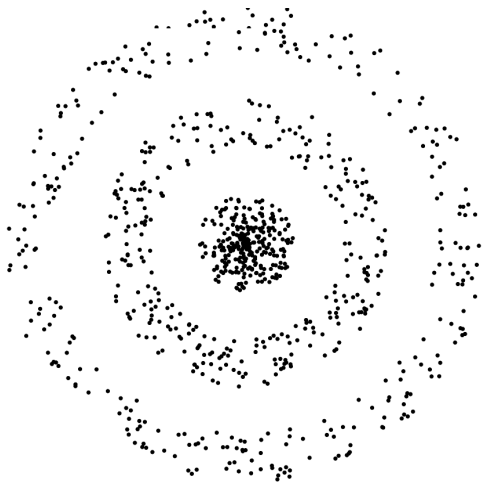
Machine learning (ML) is the scientific study of algorithms and statistical models that computer systems use to perform a specific task without using explicit instructions, relying on patterns and inference instead. (wiki)

Two common tasks in machine learning:

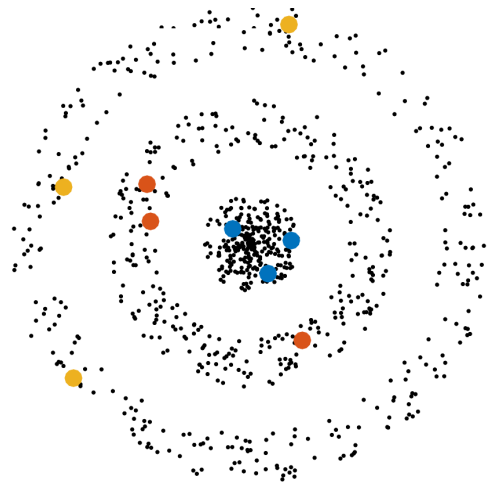
- given data, cluster it and detect patterns in it (unsupervised learning)
- given data and labels, find a functional relation between them (supervised learning)

Machine Learning in 3 slides

unsupervised



semi-supervised



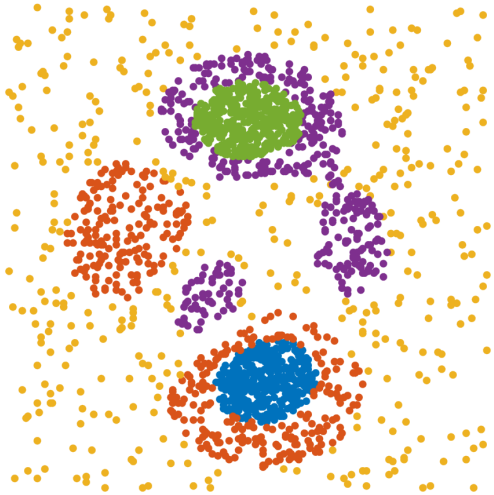
Unsupervised learning - given the data set $\mathbf{Y} = [\mathbf{y}_1, \ldots, \mathbf{y}_n]$ cluster the data into "similar" groups (labels).

- helps find hidden patterns
- often explorative and open-ended

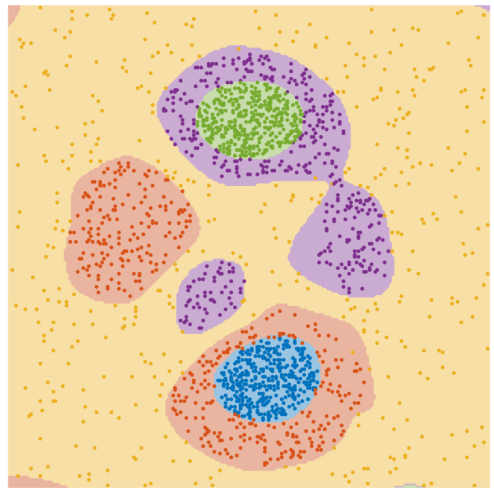
Semisupervised - label the data based on a few examples

Machine Learning in 3 slides

training data



trained model



Supervised learning - given the data set $\mathbf{Y} = [\mathbf{y}_1, \ldots, \mathbf{y}_n] \in \mathcal{Y}$ and their labels $\mathbf{C} = [\mathbf{c}_1, \ldots, \mathbf{c}_n] \in \mathcal{C}$, find the relation $f : \mathcal{Y} \to \mathcal{C}$

- models range in complexity
- older models based on support vector machines (SVM) and kernel methods
- recently, deep neural networks (DNNs) dominate

Learning From Data: The Core of Science

Given inputs and outputs, how to choose f?

Option 1 (Fundamental(?) understanding): For example, Newton's formula

$$x(t) = \frac{1}{2} g t^2,$$

with unknown parameter g.
To estimate g observe falling object

t	x
0	0
1	4.9
2	20.1
3	44.1

Goal: Derive model from theory, calibrate it using data.

Learning From Data: The Core of Science

Given inputs and outputs, how to choose f?

Option 2 (Phenomenological models): For example, Archie's law - what is the electrical resistivity of a rock and how it relates to its porosity, ϕ and saturation, S_w?

$$\rho(\phi, S_w) = a\phi^{n/2} S_w^p$$

a, n, p unknown parameters

Obtaining parameters from observed data and lab experiments on rocks.

Goal: Find model that consistent with fundamental theory, without directly deriving it from theory.

Phenomenological vs. Fundamental

Fundamental laws come from understanding(?) the underlying process. They are **assumed invariant** and can therefore be predictive(?).

Phenomenological models are data-driven. They "work" on some given data. Hard to know what their limitations are.

But ...

- models based on understanding can do poorly - weather, economics ...
- models based on data can sometimes do better
- how do we quantify understanding?

Intro: Deep Learning

Deep Neural Networks: History

- Neural Networks with a particular (deep) architecture
- Exist for a long time (70's and even earlier) [17, 18, 14]
- Recent revolution - computational power and lots of data [2, 16, 13]
- Can perform very well when trained with lots of data
- Applications
  - Image recognition [10, 12, 13], segmentation, natural language processing [3, 5, 11]
- A few recent news articles:
  - Apple Is Bringing the AI Revolution to Your iPhone, WIRED 2016
  - Why Deep Learning Is Suddenly Changing Your Life, FORTUNE 2016
  - Data Scientist: Sexiest Job of the 21st Century, Harvard Business Rev '17

Learning Objective: Demystify Deep Learning

Artificial Intelligence / Machine Learning

The Dark Secret at the Heart of AI

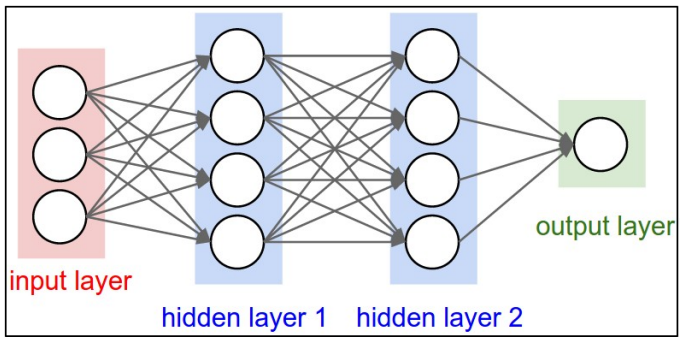
No one really knows how the most advanced algorithms do what they do. That could be a problem.

by **Will Knight** Apr 11, 2017

Learning objectives of this minicourse:

- look under the hood of some deep learning examples
- describe deep learning mathematically (see also [9])
- expose numerical challenges / approaches to improve DL

DNN - A Quick Overview - 1



$$\begin{cases} \mathbf{y}_{l+1} = \sigma(\mathbf{K}_l \mathbf{y}_l + \mathbf{b}_l) \\ \mathbf{y}_{l+1} = \mathbf{y}_l + \sigma(\mathbf{K}_l \mathbf{y}_l + \mathbf{b}_l) \\ \mathbf{y}_{l+1} = \mathbf{y}_l + \sigma\left(\mathbf{L}_l \sigma(\mathbf{K}_l \mathbf{y}_j + \mathbf{b}_l)\right) \\ \qquad \vdots \end{cases}$$

Here:

- $l = 0, 1, 2, \ldots, N$ is the layer
- $\sigma : \mathbb{R} \to \mathbb{R}$ is the activation function
- $\mathbf{y}_0 = \mathbf{y} \in \mathbb{R}^{n_f}$ is the input data (e.g., an image)
- $\mathbf{c} \in \mathbb{R}^{n_c}$ is the output (e.g. class of the image)
- $\mathbf{L}_l, \mathbf{K}_l, \mathbf{b}_l$ are parameters of the model f

DNN - A Quick Overview - 2

Neural networks are data interpolator/classifier when the underlying model is unknown.

A generic way to write it is

$$\mathbf{c} = f(\mathbf{y}, \boldsymbol{\theta}).$$

- the function f is the computational model
- $\mathbf{y} \in \mathbb{R}^{n_f}$ is the input data (e.g., an image)
- $\mathbf{c} \in \mathbb{R}^{n_c}$ is the output (e.g. class of the image)
- $\boldsymbol{\theta} \in \mathbb{R}^{n_p}$ are parameters of the model f

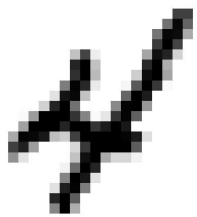
In supervised learning we have examples $\{(\mathbf{y}_j, \mathbf{c}_j) \ : \ j = 1, \ldots, n\}$ and the goal is to estimate or "learn" the parameters $\boldsymbol{\theta}$.

Example: Classification of Hand-written Digits

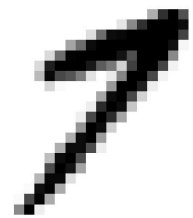
- Let $\mathbf{y}_j \in \mathbb{R}^{n_f}$ and let $\mathbf{c}_j \in \mathbb{R}^{n_c}$.
- The vector $\mathbf{c}$ is the probability of $\mathbf{y}$ belonging to a certain class. Clearly, $0 \leq \mathbf{c}_j \leq 1$ and $\sum_{j=1}^{n_c} \mathbf{c}_j = 1$.

Examples (MNIST):

$\mathbf{y}_1$



$\mathbf{y}_2$



$\mathbf{c}_1 = [0, 0, 0, 0, 1, 0, 0, 0, 0, 0]^\top$ $\mathbf{c}_2 = [0, 0.3, 0, 0, 0, 0, 0, 0.7, 0, 0]^\top$

Example: Classification of Natural Images

Image classification of natural images

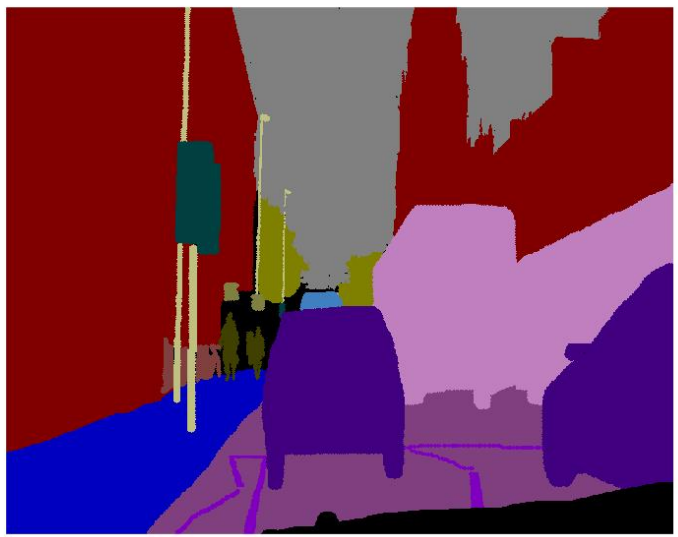
Examples (CIFAR-10):

Example: Semantic Segmentation

- let $\mathbf{y}_j \in \mathbb{R}^n$ be an RGB or grey valued image.
- let the pixels in $\mathbf{c}_j \in \{1, 2, 3, \ldots\}^k$ denote the labels.

$\mathbf{y}$, input image

$\mathbf{c}$, segmentation (labeled image)



Goal: Find map $\mathbf{c} = f(\mathbf{y}, \boldsymbol{\theta})$

Example: Semantic Segmentation

Problem: Given image $\mathbf{y}$ and label $\mathbf{c}$, find a map $f(\cdot, \boldsymbol{\theta})$ such that $\mathbf{c} \approx f(\mathbf{y}, \boldsymbol{\theta})$

First step: Reduce the dimensionality of problem.

- extract features from the image
- classify in the feature space

Reduce the problem of learning from the image to feature detection and classification

Possible features: Color, neighbors, edges ...

Generalization

Suppose that we have examples $\{\mathbf{y}_j, \mathbf{c}_j\}, \quad j = 1, \ldots, n$, a model $f(\mathbf{y}, \boldsymbol{\theta})$ and some optimal parameter $\boldsymbol{\theta}^*$.
Let $\{(\mathbf{y}_j^t, \mathbf{c}_j^t) \ : \ j = 1, \ldots, s\}$ be some test set, that was not used to compute $\boldsymbol{\theta}^*$.

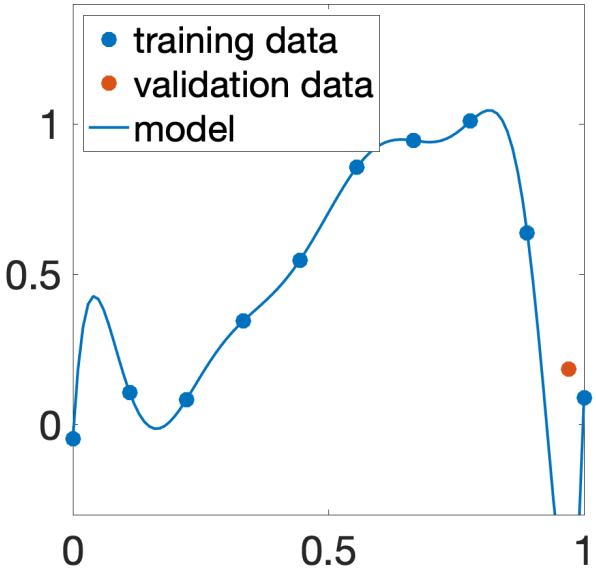
Loosely speaking, if

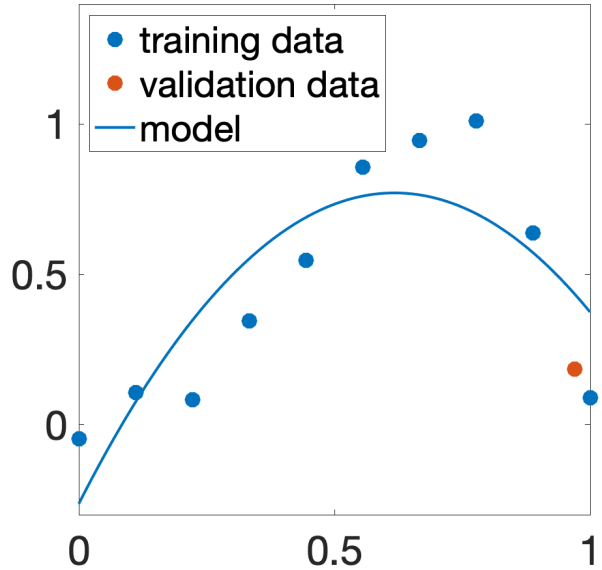
$$\|f(\mathbf{y}_j^t, \boldsymbol{\theta}^*) - \mathbf{c}_j^t\|_p \text{ is small}$$

then the model is predictive - it generalizes well

For phenomenological models, there is no reason why the model should generalize, but in practice it often does.

Generalization





Why would a model generalize poorly?

$$1 \ll \|f(\mathbf{y}_j^t, \boldsymbol{\theta}^*) - \mathbf{c}_j^t\|_p$$

Two common reasons:

1. Our "optimal" $\boldsymbol{\theta}^*$ was optimal for the training but is less so for other data
2. The chosen computational model f is poor (e.g. quadratic model for a nonlinear function).

Linear Models

Supervised Learning Problem

Given examples (inputs)

$$\mathbf{Y} = [\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_n] \in \mathbb{R}^{n_f \times n}$$

and labels (outputs)

$$\mathbf{C} = [\mathbf{c}_1, \mathbf{c}_2, \cdots, \mathbf{c}_n] \in \mathbb{R}^{n_c \times n},$$

find a classification/prediction function $f(\cdot, \boldsymbol{\theta})$, i.e.,

$$f(\mathbf{y}_j, \boldsymbol{\theta}) \approx \mathbf{c}_j, \quad j = 1, \ldots, n.$$

Regression and Least-Squares

Simplest option, a linear model with $\theta = (\mathbf{W}, \mathbf{b})$ and

$$f(\mathbf{Y}, \mathbf{W}, \mathbf{b}) = \mathbf{W}\mathbf{Y} + \mathbf{b}\mathbf{e}_n^\top \approx \mathbf{C}$$

- $\mathbf{W} \in \mathbb{R}^{n_c \times n_f}$ are *weights*
- $\mathbf{b} \in \mathbb{R}^{n_c}$ are *biases*
- $\mathbf{e}_n \in \mathbb{R}^n$ is a vector of ones

Equivalent notation:

$$f(\mathbf{Y}, \mathbf{W}, \mathbf{b}) = \begin{pmatrix} \mathbf{W} & \mathbf{b} \end{pmatrix} \begin{pmatrix} \mathbf{Y} \\ \mathbf{e}_n^\top \end{pmatrix} \approx \mathbf{C}$$

Problem may not have a solution, or may have infinite solutions (when?). Solve through optimization

$$\min_{\mathbf{W}} \frac{1}{2} \|\mathbf{W}\mathbf{Y} - \mathbf{C}\|_F^2$$

(Frobenius norm: $\|\mathbf{A}\|_F^2 = \mathrm{trace}(\mathbf{A}^\top \mathbf{A}) = \sum_{i,j} \mathbf{A}_{i,j}^2$.)

Remark: Relation to Least-Squares

Consider the regression problem

$$\min_{\mathbf{W}} \frac{1}{2}\|\mathbf{W}\mathbf{Y} - \mathbf{C}\|_F^2.$$

It is easy to see that this is equivalent to

$$\min_{\mathbf{W}} \frac{1}{2}\|\mathbf{Y}^\top\mathbf{W}^\top - \mathbf{C}^\top\|_F^2,$$

which can be solved separately for each row in $\mathbf{W}$

$$\mathbf{W}(j,:)^\top = \arg\min_{\mathbf{w}} \frac{1}{2}\|\mathbf{Y}^\top\mathbf{w} - \mathbf{C}(j,:)^\top\|_F^2.$$

Notation: Let $\mathbf{A} = \mathbf{Y}^\top$ and $\mathbf{X} = \mathbf{W}^\top$ (easy to add bias here), we solve

$$\min_{\mathbf{X}} \frac{1}{2}\|\mathbf{A}\mathbf{X} - \mathbf{C}^\top\|_F^2$$

Iterative Regularization

Consider

$$\min_{\mathbf{x}} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|^2$$

- Assume that $\mathbf{A}$ has non-trivial null space
- The matrix $\mathbf{A}^\top\mathbf{A}$ is not invertible
- Can we still use iterative methods (CG, CGLS, ...)?

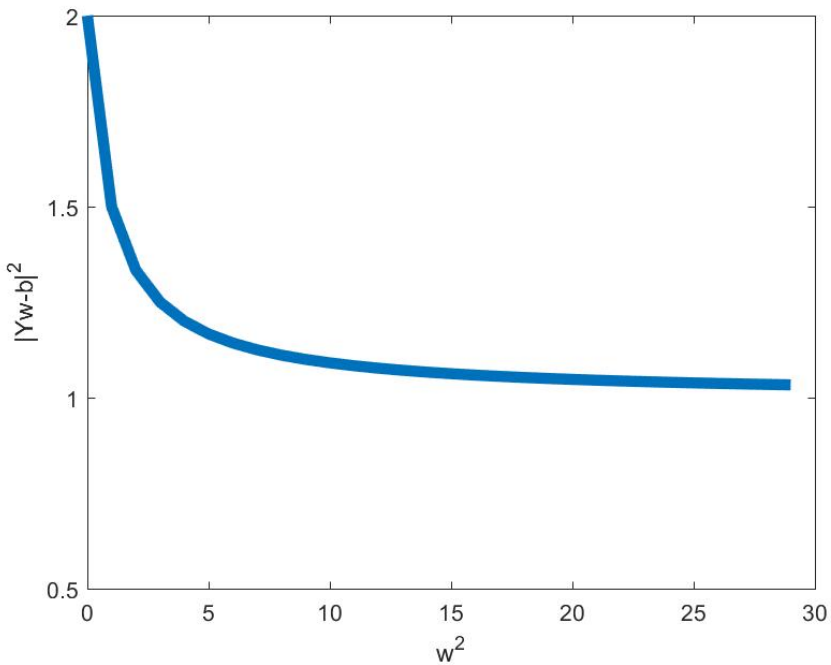
What are the properties of the iterates?

Excellent introduction to computational inverse problems [7, 19, 8]

Iterative Regularization: L-Curve

The CGLS algorithm has the following properties

- For each iteration $\|\mathbf{A}\mathbf{x}_k - \mathbf{c}\|^2 \leq \|\mathbf{A}\mathbf{x}_{k-1} - \mathbf{c}\|^2$
- If starting from $\mathbf{x} = 0$ then $\|\mathbf{x}_k\|^2 \geq \|\mathbf{x}_{k-1}\|^2$
- $\mathbf{x}_1, \mathbf{x}_2, \ldots$ converges to the minimum norm solution of the problem
- Plotting $\|\mathbf{x}_k\|^2$ vs $\|\mathbf{A}\mathbf{x}_k - \mathbf{c}\|^2$ typically has the shape of an L-curve



Cross Validation

Finding good least-squares solution requires good parameter selection.

- λ when using Tikhonov regularization (weight decay)
- number of iteration (for SD and CGLS)

Suppose that we have two different "solutions"

$$\begin{aligned} \mathbf{x}_1 \quad &\rightarrow \quad \|\mathbf{x}_1\|^2 = \eta_1 \quad \|\mathbf{A}\mathbf{x}_1 - \mathbf{c}\|^2 = \rho_1. \\ \mathbf{x}_2 \quad &\rightarrow \quad \|\mathbf{x}_2\|^2 = \eta_2 \quad \|\mathbf{A}\mathbf{x}_2 - \mathbf{c}\|^2 = \rho_2. \end{aligned}$$

How to decide which one is better?

Cross Validation

Goal: Gauge how well the model can predict new examples.

Let $\{\mathbf{A}_{\mathrm{CV}}, \mathbf{c}_{\mathrm{CV}}\}$ be data that is **not used** for the training

Idea: If $\|\mathbf{A}_{\mathrm{CV}}\mathbf{x}_1 - \mathbf{c}_{\mathrm{CV}}\|^2 \leq \|\mathbf{A}_{\mathrm{CV}}\mathbf{x}_2 - \mathbf{c}_{\mathrm{CV}}\|^2$, then $\mathbf{x}_1$ is a better solution that $\mathbf{x}_2$.

When the solution depends on some hyper-parameter(s) λ, we can phrase this as bi-level optimization problem

$$\lambda^* = \arg\min_{\lambda} \|\mathbf{A}_{\mathrm{CV}}\mathbf{x}(\lambda) - \mathbf{c}_{\mathrm{CV}}\|^2,$$

where $\mathbf{x}(\lambda) = \arg\min_{\mathbf{x}} \|\mathbf{A}\mathbf{x} - \mathbf{x}\|^2 + \lambda\|\mathbf{x}\|^2$.

Cross Validation

To assess the final quality of the solution cross validation is not sufficient (why?).

Need a final testing set.

Procedure

- Divide the data into 3 groups $\{\mathbf{A}_{\text{train}}, \mathbf{A}_{\text{CV}}, \mathbf{A}_{\text{test}}\}$.
- Use $\mathbf{A}_{\text{train}}$ to estimate $\mathbf{x}(\lambda)$
- Use $\mathbf{A}_{\text{CV}}$ to estimate λ
- Use $\mathbf{A}_{\text{test}}$ to assess the quality of the solution

Important - we are not allowed to use $\mathbf{A}_{\text{test}}$ to tune parameters!

Classification

Logistic Regression

Assume our data falls into two classes. Denote by $\mathbf{c}_{\text{obs}}(\mathbf{y})$ the probability that example $\mathbf{y} \in \mathbb{R}^{n_f}$ belongs to first category.

Since output of our classifier $f(\mathbf{y}, \boldsymbol{\theta})$ is supposed to be probability, use logistic sigmoid

$$\mathbf{c}(\mathbf{y}, \boldsymbol{\theta}) = \frac{1}{1 + \exp\left(-f(\mathbf{y}, \boldsymbol{\theta})\right)}.$$

Example (Linear Classification): If $f(\mathbf{y}, \boldsymbol{\theta})$ is a linear function (adding bias is easy), $\boldsymbol{\theta} = \mathbf{w} \in \mathbb{R}^{n_f}$ and

$$\mathbf{c}(\mathbf{y}, \mathbf{w}) = \frac{1}{1 + \exp(-\mathbf{w}^\top \mathbf{y})}.$$

for now: consider linear models for simplicity

Multinomial Logistic Regression

Suppose data falls into $n_c \geq 2$ categories and the components of $\mathbf{c}_{\text{obs}}(\mathbf{y}) \in [0,1]^{n_c}$ contain probabilities for each class.

Applying the logistic sigmoid to each component of $f(\mathbf{y}, \mathbf{W})$ not enough (probabilities must sum to one). Use

$$\mathbf{c}(\mathbf{y}, \mathbf{W}) = \left(\frac{1}{\mathbf{e}_{n_c}^\top \exp(\mathbf{W}\mathbf{y})} \right) \exp(\mathbf{W}\mathbf{y}).$$

Note: Division and exp are applied element-wise!

Logistic Regression - Loss Function

How similar are $\mathbf{c}(\cdot, \mathbf{W})$ and $\mathbf{c}_{\text{obs}}(\cdot)$?

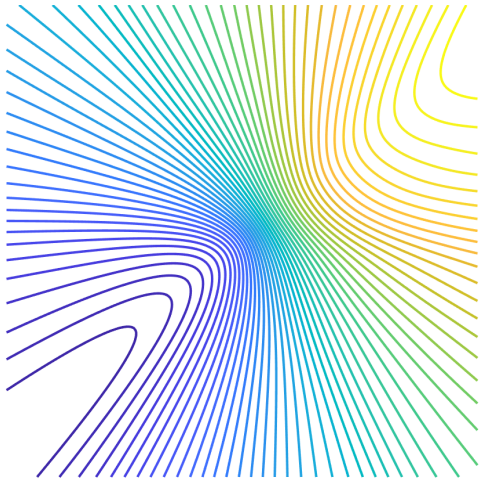
Naive idea: Let $\mathbf{Y} \in \mathbb{R}^{n_f \times n}$ be examples with class probabilities $\mathbf{C}_{\text{obs}} \in [0,1]^{n_c \times n}$, use

$$\phi(\mathbf{W}) = \frac{1}{2n} \sum_{j=1}^{n} \|\mathbf{c}(\mathbf{y}_j, \mathbf{W}) - \mathbf{c}_{j,\text{obs}}\|_F^2$$

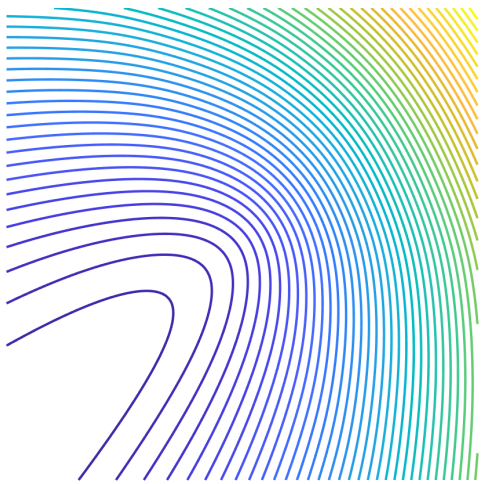
Problems

- ignores that $\mathbf{c}(\cdot, \mathbf{W})$ and $\mathbf{c}_{\text{obs}}(\cdot)$ are distributions.
- leads to non-convex objective function

Frobenius



Cross Entropy



Cross Entropy for Multinomial Logistic Regression

Similarly, for general case ($n_c \geq 2$ classes, n examples).
Recall:

$$\mathbf{C}(\mathbf{Y}, \mathbf{W}) = \exp(\mathbf{W}\mathbf{Y}) \operatorname{diag}\left(\frac{1}{\mathbf{e}_{n_c}^\top \exp(\mathbf{W}\mathbf{Y})}\right)$$

Get cross entropy by summing over all examples

$$E(\mathbf{C}_{\text{obs}}, \mathbf{C}(\mathbf{Y}, \mathbf{W})) = -\frac{1}{n}\operatorname{tr}(\mathbf{C}_{\text{obs}}^\top \log(\mathbf{C}(\mathbf{Y}, \mathbf{W}))).$$

We will also call this the *softmax* (cross-entropy) function.

Simplifying the Softmax Function

Let $\mathbf{S} = \mathbf{WY}$, then

$$E(\mathbf{C}_{\text{obs}}, \mathbf{S}) = -\frac{1}{n}\text{tr}\left(\mathbf{C}_{\text{obs}}^{\top} \log\left(\exp(\mathbf{S})\text{diag}\left(\frac{1}{\mathbf{e}_{n_c}^{\top}\exp(\mathbf{S})}\right)\right)\right).$$

Verify that this is equal to

$$\begin{aligned} E(\mathbf{C}_{\text{obs}}, \mathbf{S}) = &-\frac{1}{n}\mathbf{e}_{n_c}^{\top}\left(\mathbf{C}_{\text{obs}} \odot \mathbf{S}\right)\mathbf{e}_n \\ &+ \frac{1}{n}\mathbf{e}_{n_c}^{\top}\mathbf{C}_{\text{obs}} \log\left(\mathbf{e}_{n_c}^{\top}\exp(\mathbf{S})\right)^{\top} \end{aligned}$$

($\odot$ is Hadamard product, exp and log component-wise)

If $\mathbf{C}_{\text{obs}}$ has a unit row sum (why?) then $\mathbf{e}_{n_c}^{\top}\mathbf{C}_{\text{obs}}^{\top} = \mathbf{e}_n^{\top}$ and

$$E(\mathbf{C}_{\text{obs}}, \mathbf{S}) = -\frac{1}{n}\mathbf{e}_{n_c}^{\top}\left(\mathbf{C}_{\text{obs}} \odot \mathbf{S}\right)\mathbf{e}_n + \frac{1}{n}\log(\mathbf{e}_{n_c}^{\top}\exp(\mathbf{S}))\mathbf{e}_n$$

Numerical Considerations

Scale to prevent overflow. Note that for an arbitrary $s \in \mathbb{R}$ we have

$$E(\mathbf{C}_{\text{obs}}, \mathbf{WY} - s) = E(\mathbf{C}_{\text{obs}}, \mathbf{WY})$$

This prevents overflow, but may lead to underflow (and divisions by zero).

Note that s does not need to be the same in each row (example). Hence, we can choose $\mathbf{s} = \max(\mathbf{WY}, [], 1) \in \mathbb{R}^{1 \times n}$ to avoid underflow and overflow.

For stability use $E(\mathbf{C}_{\text{obs}}, \mathbf{S})$ where $\mathbf{S} = \mathbf{WY} - \mathbf{e}_{n_c}\mathbf{s}$.

Linear Classification

If $\mathbf{W}$ can separate the classes then the goal is to minimize the cross entropy (with some potential regularization)

$$\mathbf{W}^* = \arg\min_{\mathbf{W}} \quad -\frac{1}{n}\mathbf{e}_{n_c}^\top \left(\mathbf{C}_{\text{obs}} \odot \mathbf{S}\right)\mathbf{e}_n + \frac{1}{n}\log(\mathbf{e}_{n_c}^\top \exp(\mathbf{S}))\mathbf{e}_n$$
$$\text{subject to } \mathbf{S} = \mathbf{W}\mathbf{Y} - \mathbf{e}_{n_c}\mathbf{s}$$

This is a smooth convex optimization problem
$\Rightarrow$ many existing optimization techniques will work

For large-scale problems, use derivative-based optimization algorithm. (Examples: Steepest Descent, Newton-like methods, Stochastic Gradient Descent, ADMM, . . .)

Excellent references: [15, 4, 1, 6]

Part 1 Summary: Linear Models

- Intro to Deep Learning $\subsetneq$ Machine Learning
  - risks and promises of phenomenological models
  - importance of generalization
- (Review) Linear Regression
  - iterative and direct regularization
  - (generalized) cross validation
- Multinomial Logistic Regression
  - cross entropies
  - leads to convex optimization problem
- Linear models
  - well-understood, easy to use, not very powerful
  - next: add nonlinearity (through neural networks)

References

[1] A. Beck. *Introduction to Nonlinear Optimization*. Theory, Algorithms, and Applications with MATLAB. SIAM, Philadelphia, Oct. 2014.

[2] Y. Bengio et al. Learning deep architectures for AI. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.

[3] A. Bordes, S. Chopra, and J. Weston. Question Answering with Subgraph Embeddings. *arXiv preprint arXiv:1406.3676*, 2014.

[4] S. P. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, Mar. 2004.

[5] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural Language Processing (Almost) from Scratch. *Journal of Machine Learning Research*, 12:2493–2537, 2011.

[6] S. W. Fung, S. Tyrväinen, L. Ruthotto, and E. Haber. ADMM-SOFTMAX : An ADMM Approach for Multinomial Logistic Regression. *arXiv.org*, Jan. 2019.

[7] P. C. Hansen. *Rank-deficient and discrete ill-posed problems*. SIAM Monographs on Mathematical Modeling and Computation. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1998.

[8] P. C. Hansen. *Discrete inverse problems*, volume 7 of *Fundamentals of Algorithms*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2010.

[9] C. F. Higham and D. J. Higham. Deep Learning: An Introduction for Applied Mathematicians. *arXiv.org*, Jan. 2018.

References (cont.)

[10] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[11] S. Jean, K. Cho, R. Memisevic, and Y. Bengio. On Using Very Large Target Vocabulary for Neural Machine Translation. *arXiv preprint arXiv:1412.2007*, 2014.

[12] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 61:1097–1105, 2012.

[13] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[14] Y. LeCun, B. E. Boser, and J. S. Denker. Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*, pages 396–404, 1990.

[15] J. Nocedal and S. Wright. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer Science & Business Media, New York, Dec. 2006.

[16] R. Raina, A. Madhavan, and A. Y. Ng. Large-scale deep unsupervised learning using graphics processors. In *the 26th Annual International Conference*, pages 873–880. ACM, June 2009.

References (cont.)

[17] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386–408, 1958.

[18] D. Rumelhart, G. Hinton, and J. Williams, R. Learning representations by back-propagating errors. *Nature*, 323(6088):533–538, 1986.

[19] C. R. Vogel. *Computational Methods for Inverse Problems*. SIAM, Philadelphia, 2002.