
Stochastic Gradient Descent

On-line learning and intro to (Feedforward)
Neural Networks

(Recap) Linear Regression

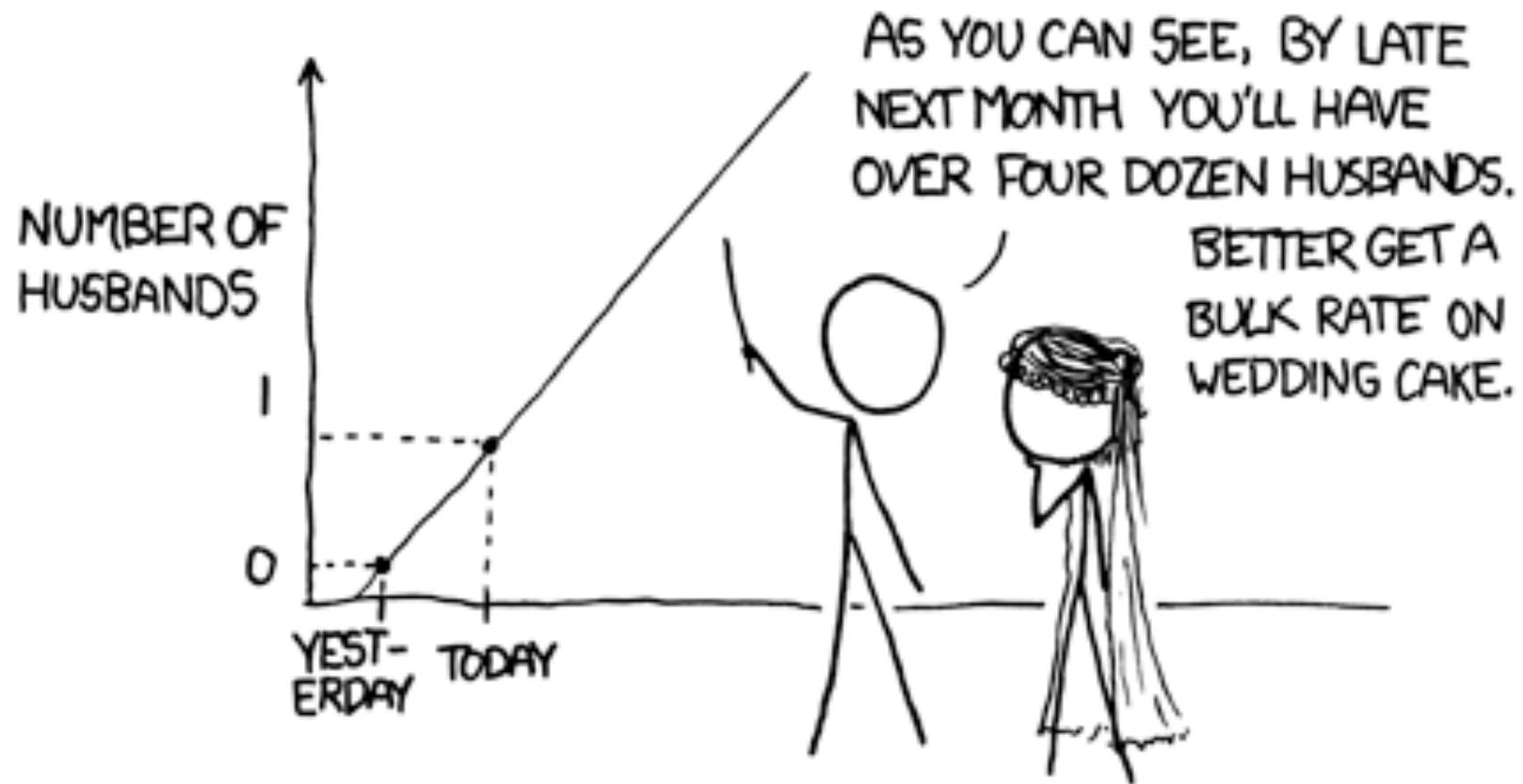
- Studied extensively and have well-developed theory (variable selection methods, extensions for dealing with correlated data, evaluation of results,...)

Model: $E(y | \mathbf{x}) = \beta_0 + \beta_1 x_1 + \dots + \beta_k x_k + \text{i.i.d. Gaussian error term}$

- Evaluating the coefficients
 - consider standard errors
 - Consider coefficients if both x and y's were normalized
 - Doing significance test to see if each term should be dropped.
 - Many predictors? Better to do forward search...

Knowing Scope of Models

MY HOBBY: EXTRAPOLATING

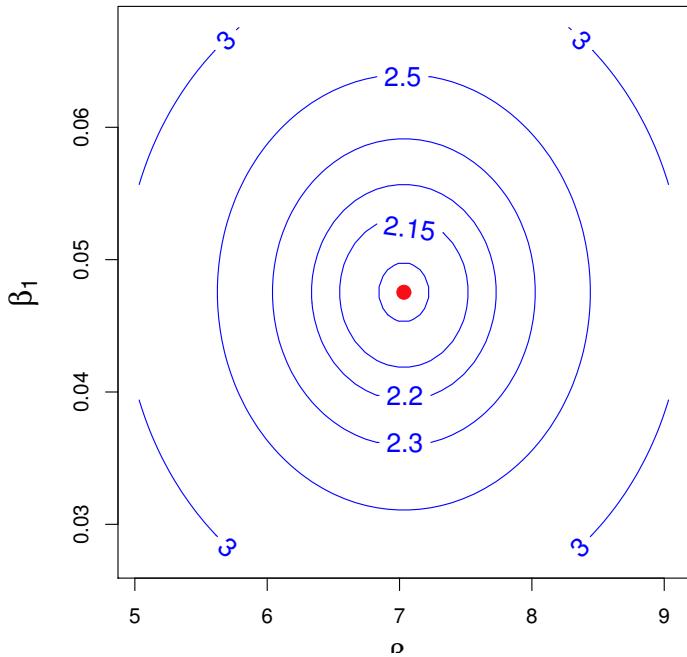


Beyond linear Regression

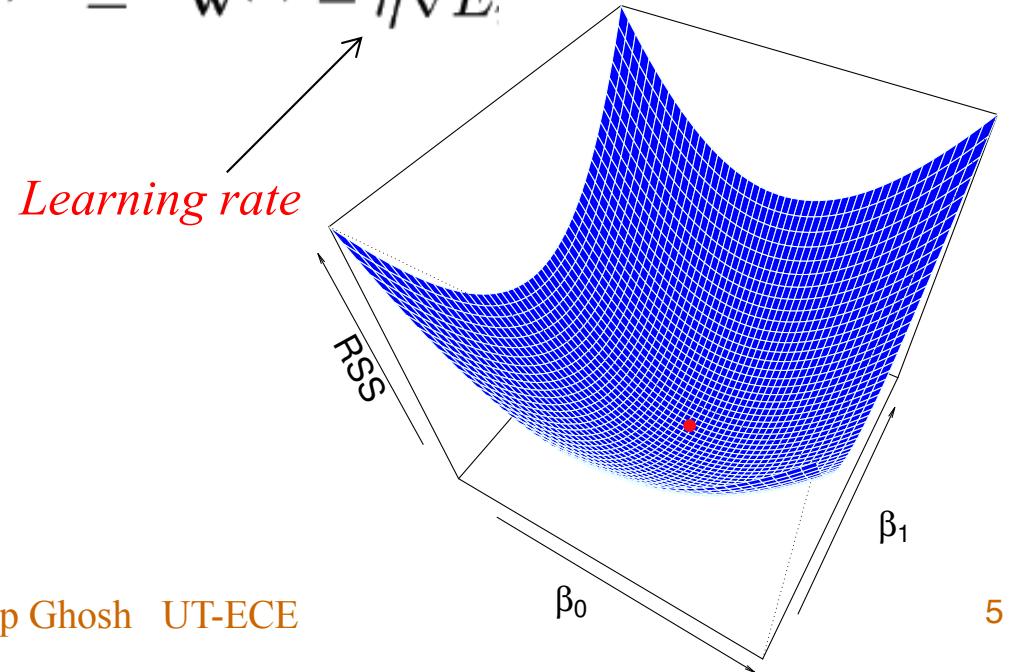
- almost linear methods
 - generalized linear models
 - Multi-level models
 - additive models
 - piecewise linear (e.g. CART); locally linear regression,
- nonlinear models
 - linear in fixed transform (“phi”) space (Fourier, Gabor, polynomial...)
 - general nonlinear forms (basis functions are adaptive as well)
 - e.g. feedforward neural networks (MLP, RBF,...)
 - MLP, RBFs, Polynomials are all **universal approximators**
- For nonlinear models, weight are typically updated in an iterative manner (could be “on-line” or batch iterative)
- SGD a popular choice, is on-line.

Gradient Descent for Linear Models

- See (very introductory) Coursera Lectures by Andrew Ng, and Bishop Ch 5.1, 5.2
- The cost function $E(\mathbf{w})$ is quadratic in \mathbf{w}
 - $E(\mathbf{w}) = \text{SSE}(\mathbf{w}) = \text{SSE}(\mathbf{w}^*) + (\mathbf{w}-\mathbf{w}^*)^T \mathbf{Q} (\mathbf{w}-\mathbf{w}^*)$
 - \mathbf{w}^* is the optimum solution, and \mathbf{Q} a positive definite matrix.
 - SSE = sum of squared error (also called Residual Sum Squares or RSS)
 - Hence error surface is convex in weight space, so weights can be obtained by doing gradient descent (**incrementally moving down the cost surface in weight space**)



$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E$$

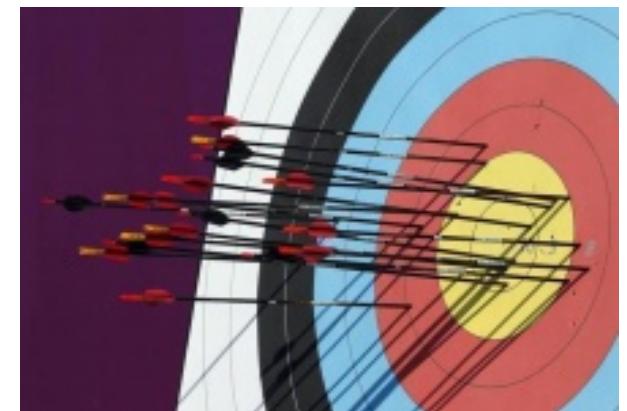


Learning Rate

- Learning rate $\eta(t)$ is crucial
 - Too low, too slow; too high, unstable
- True gradient descent is batch algorithm, slow (but sure)
- “on-line” version (**stochastic gradient descent or SGD**): replace true gradient by “instantaneous” gradient, that reduces error only on the new instance (pun intended!)
- E.g. for linear models, with τ as time/iteration number, n denoting the datapoint being considered, we get:

$$\begin{aligned}\mathbf{w}^{(\tau+1)} &= \mathbf{w}^{(\tau)} - \eta \nabla E_n \\ &= \mathbf{w}^{(\tau)} + \eta(t_n - \mathbf{w}^{(\tau)T} \phi(\mathbf{x}_n)) \phi(\mathbf{x}_n).\end{aligned}$$

- This is known as the *least-mean-squares (LMS) algorithm or the Widrow-Hoff rule.*
 - Note: Data items considered one at a time
(a.k.a. online learning)



Sequential Learning with SGD

- Learning rate: too small \rightarrow too slow
 - Too large \rightarrow oscillatory, or may even diverge
- Should η be fixed or adaptive (second order methods)?
- Is convergence needed or not?
 - Non-stationary? May not want to converge!
 - If convergence is desired, then η should decrease with time.
 - (e.g. choose $\eta = a/t$)
 - Visualize SGD as well as some second-order methods.

Why SGD?

Better for large data sets

+ Often faster than batch gradient descent

 Can do “mini-batches” as a practical trade-off

+ Less prone to local minima, so often applied to complex models (and correspondingly complex error surfaces)

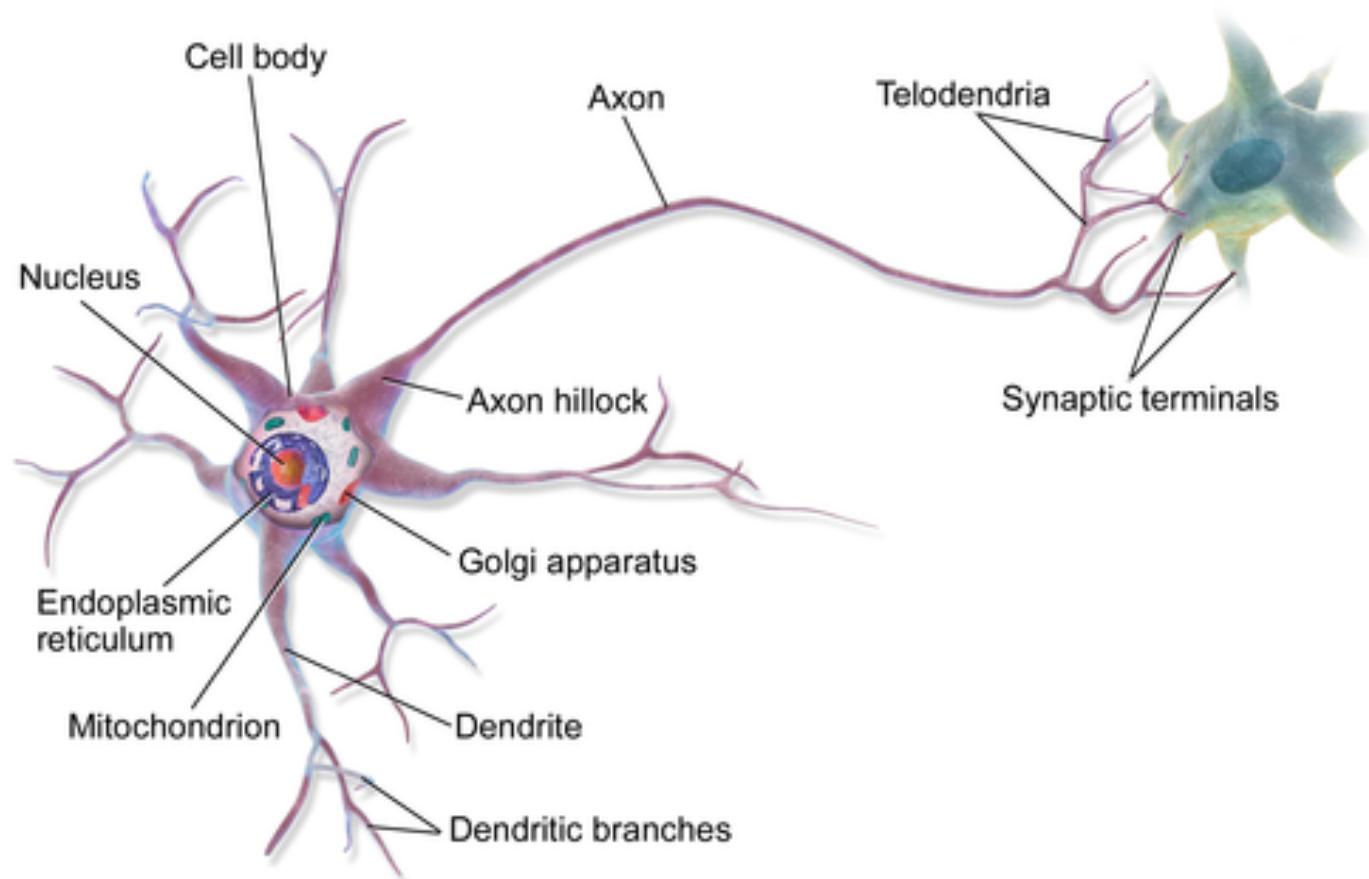
+ useful to scale inputs to help find learning rate (usually fixed)

+ works for non-stationary environments as well as online settings

Also used for more complex error surfaces, though many second-order methods, that also consider the Hessian (matrix of curvatures) in addition to the Jacobian, or vector of slopes, exist.

A Neuron

From <https://en.wikipedia.org/wiki/Neuron>

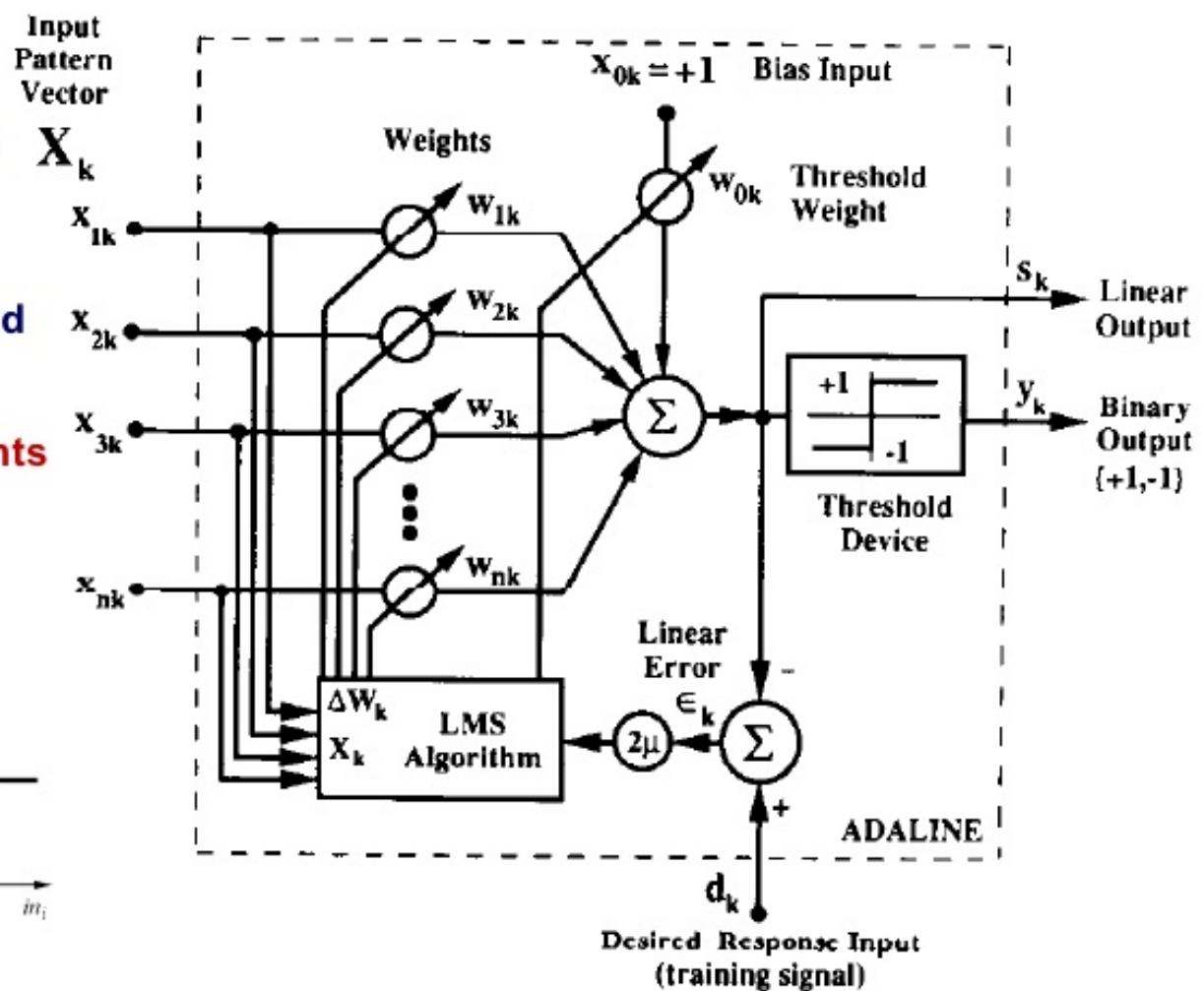
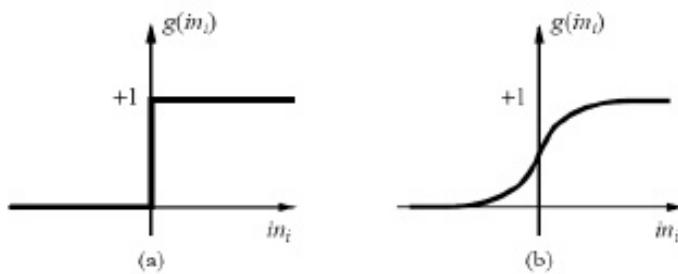


1-layer (Artificial) Neural Networks

- Viewing (generalized) linear model as a “neural network”
- Learning through SGD
- (déjà vu) LMS Algorithm, Widrow-Hoff rule from ADALINE (1960)

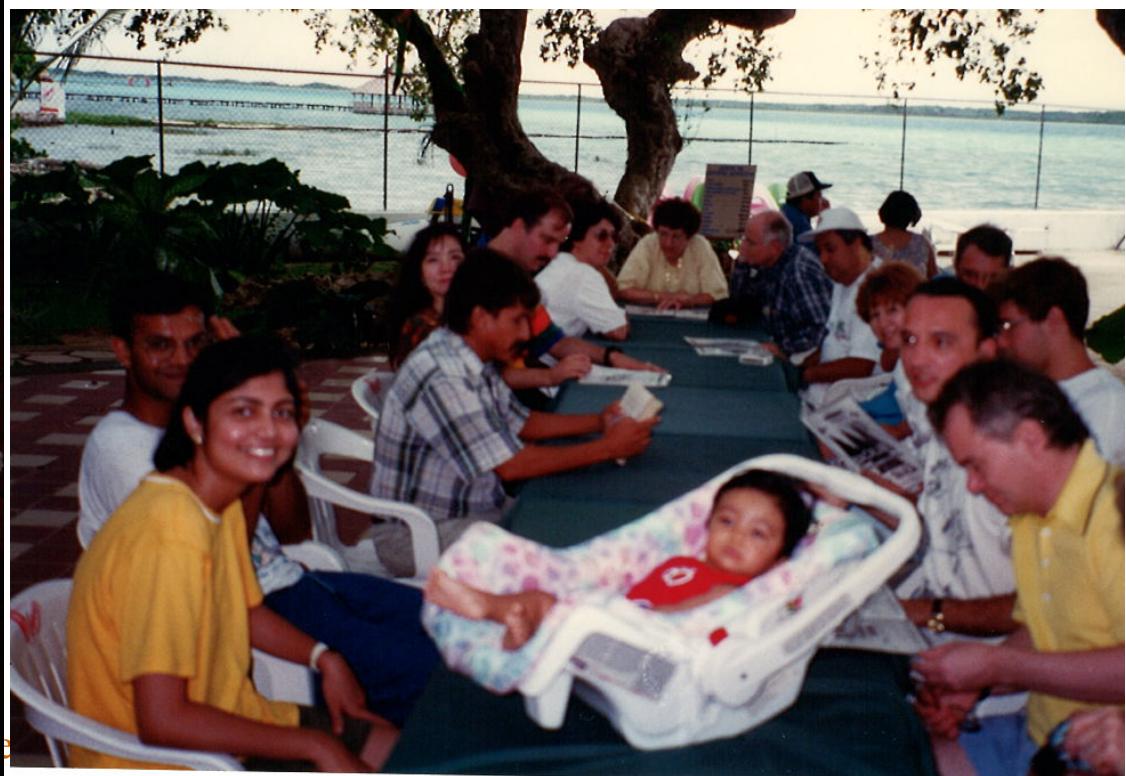
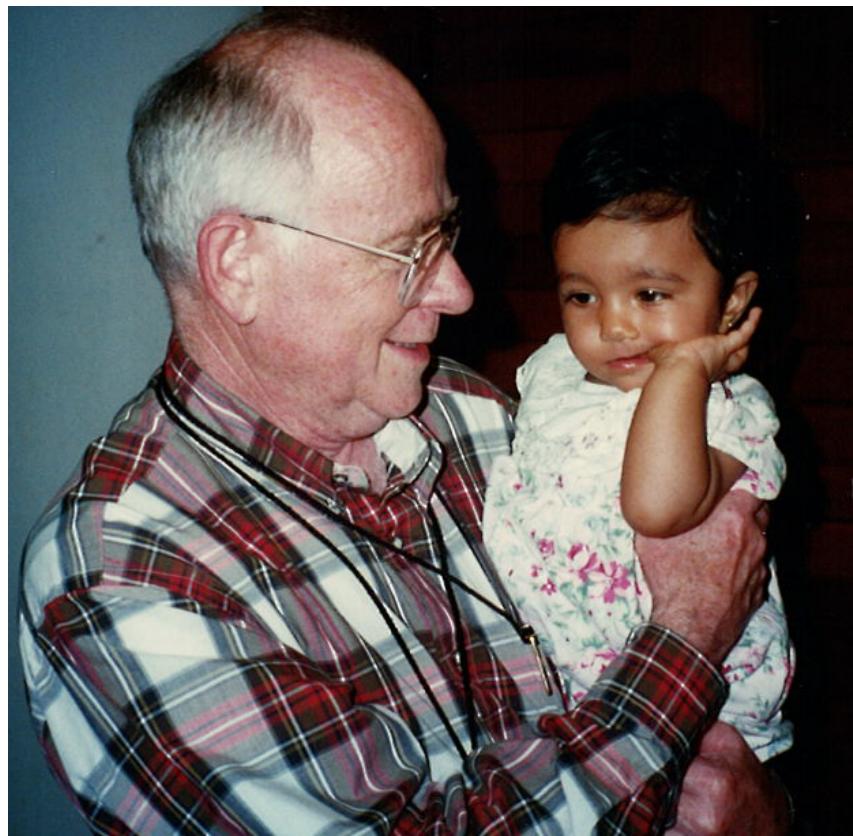
Adaline

- Adaptive Linear Element
- Adaptive linear combiner cascaded with a hard-limiting quantizer
- Linear output transformed to binary by means of a threshold device
- Training = adjusting the weights
- Activation functions



ADALINE

- <https://www.youtube.com/watch?v=hc2Zj55j1zU>

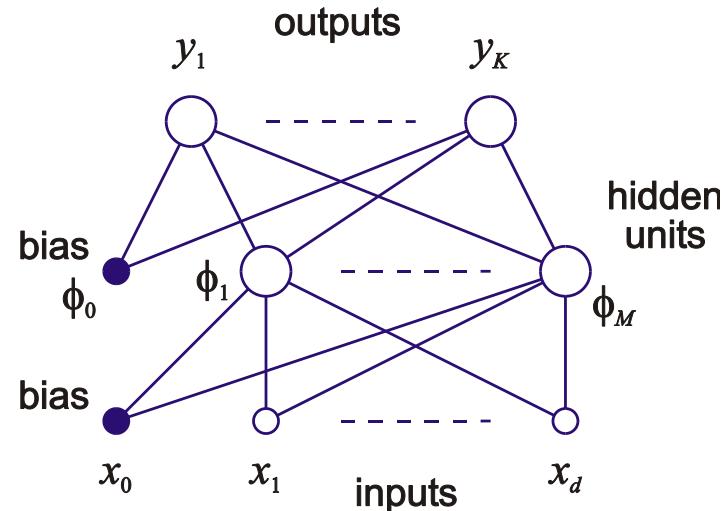


Multi-Layered Perceptrons (MLP)

(most popular neural network before deep learning)

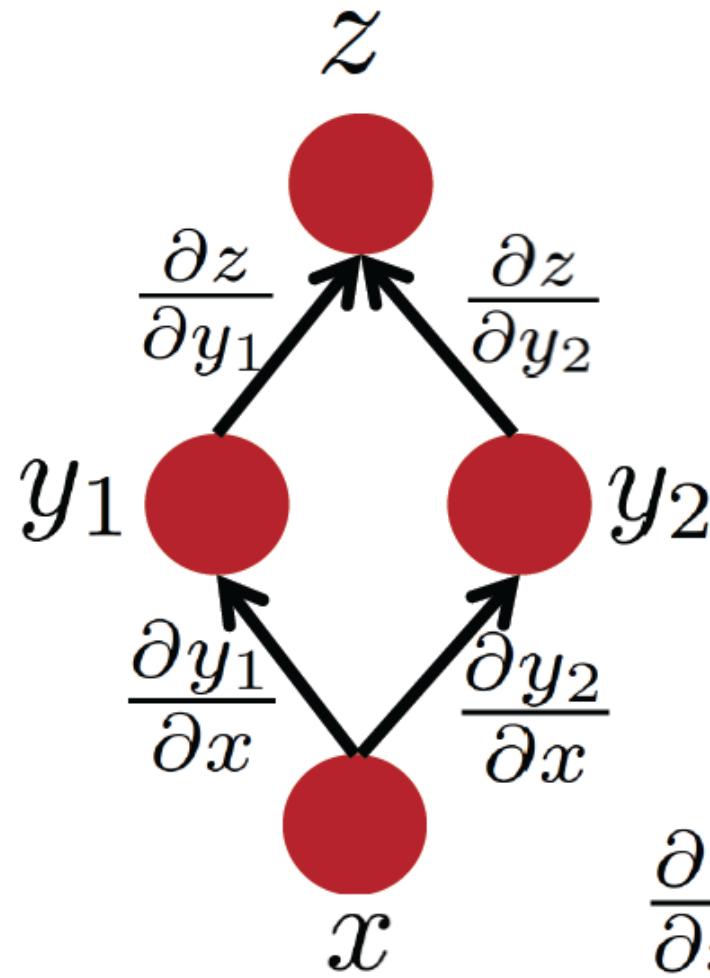
- 1 hidden layer → 2 layered network.
- Choice of activation functions:
- Hidden layer: tanh/sigmoid
 - Why not LTU
- Output layer: linear / sigmoid or softmax
- Typical MLP for Regression (2 layers of adaptive weights):

$$y_k(\mathbf{x}, \mathbf{w}) = \sum_{j=1}^M w_{kj}^{(2)} \tanh \left(\sum_i w_{ji}^{(1)} x_i + w_{j0}^{(1)} \right) + w_{k0}$$



Animation at: <http://playground.tensorflow.org/>

(Multi-path) Chain Rule from Calculus*



$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y_1} \frac{\partial y_1}{\partial x} + \frac{\partial z}{\partial y_2} \frac{\partial y_2}{\partial x}$$

Weight Update through Error Back-propagation*

Takeaway: weights are updated through (online) SGD

- Derived from chain rule for partial derivatives
- Three stages:
 1. Evaluate an “error signal” at the output units with net input a_k

$$\delta_k = \frac{\partial E_n}{\partial a_k}$$

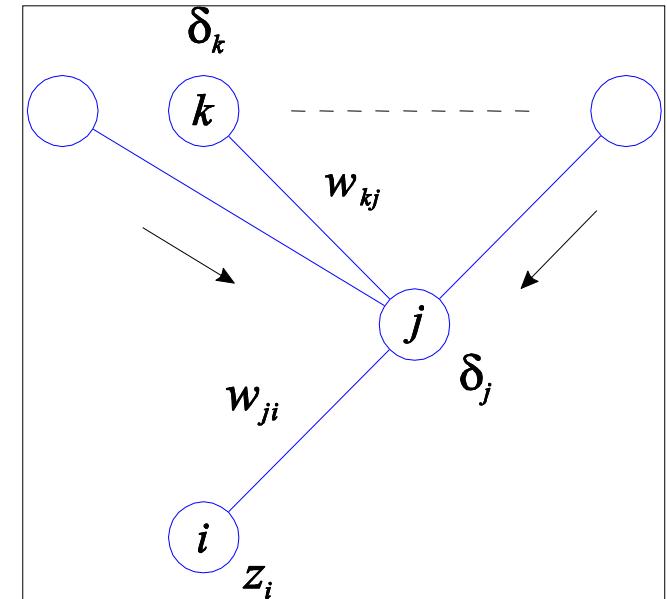
2. Propagate the signal backwards through the network

$$\delta_j = g'(a_j) \sum_k w_{kj} \delta_k$$

3. Evaluate derivatives

$$\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i$$

4. Use this for SGD



Design Parameters to be Aware about

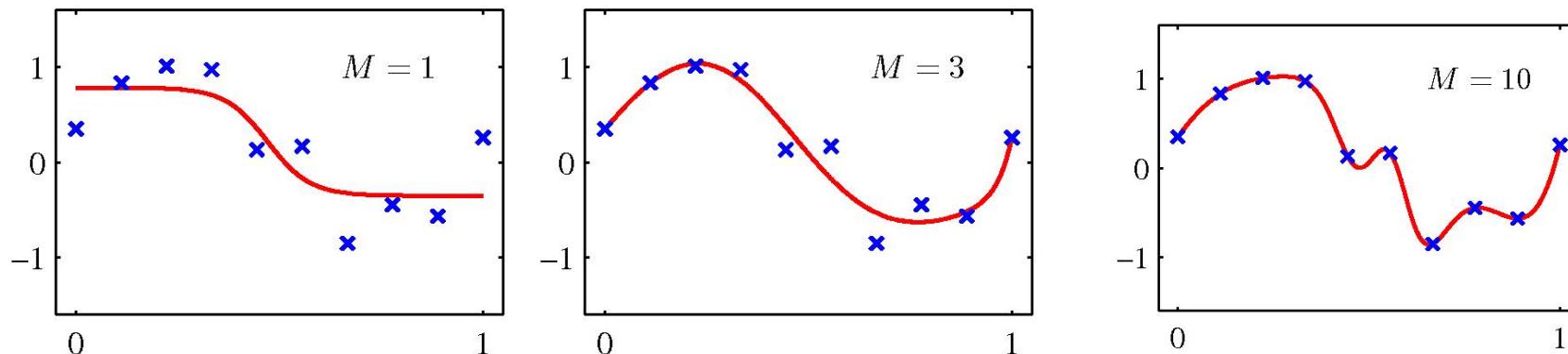
- # of hidden units/nodes
 - model complexity
- # of epochs/iterations
 - how long do you train: best is use a cross-validation set to decide when to stop
- Activation function
 - typically tanh or logistic, a.k.a. sigmoid
- Learning rate (SGD is used to update weights)
 - Speed of training
- Momentum: (a second order gradient descent method)
<https://distill.pub/2017/momentum/>

Visualizing the Workings of an MLP

- <http://playground.tensorflow.org/>
- (both regression and classification examples with different degrees of difficulty).
- TRY IT OUT!

Model Complexity for MLP

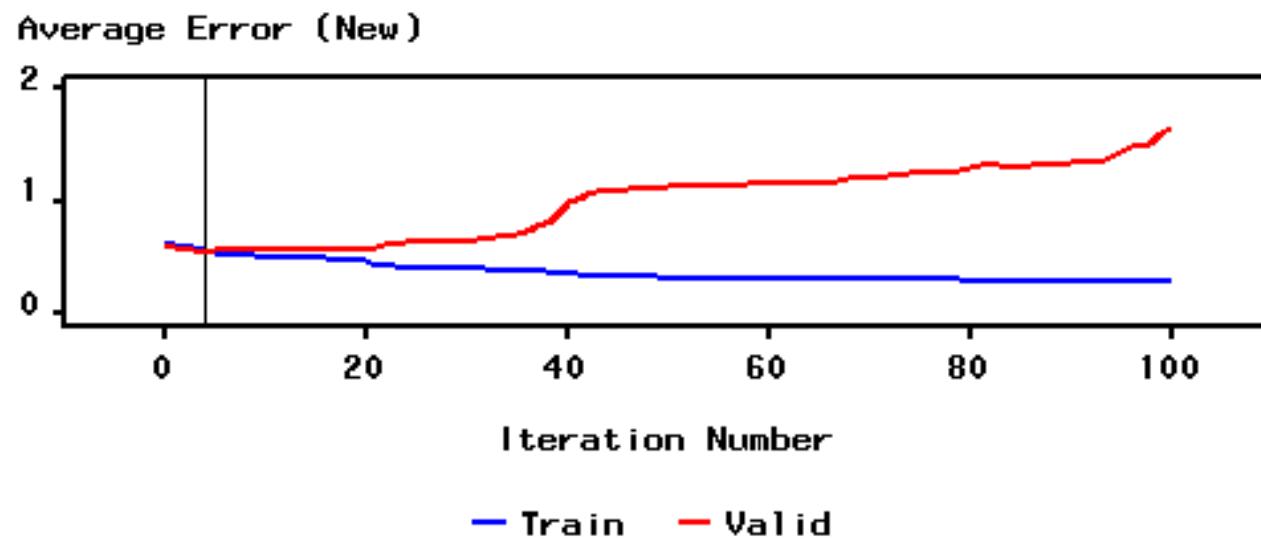
- complexity related to # of hidden units AND amount of training (number of passes or epochs through the data)



With less training M=10 solution looks like M < 10...

Adjusting Complexity

- “effective # of parameters” increases with # of epochs !!
 - Select adequately powerful model
 - while training, monitor performance using validation set
 - **stop training** when error on validation set reaches a minimum
 - SAS fig.



Deep Learning

- Amazing improvements in speech recognition, NLP, recognizing objects in images,...
- See <http://deeplearning.net/>
 - For hype/investment, see Frank Chen video towards bottom of <http://www.piventures.in/> starting 32:00



The image displays three cards from the "10 Breakthrough Technologies 2013" list by MIT Technology Review. The cards are arranged horizontally and have a light blue background with black text. Each card has a red arrow pointing to the right in the bottom right corner.

- Deep Learning**
With massive amounts of computational power, machines can now recognize objects and translate speech in real time. Artificial intelligence is finally getting smart. →
- Temporary Social Media**
Messages that quickly self-destruct could enhance the privacy of online communications and make people freer to be spontaneous. →
- Prenatal DNA Sequencing**
Reading the DNA of fetuses will be the next frontier of the genomic revolution. But do you really want to know about the genetic problems or musical aptitude of your unborn child? →

© Joydeep Ghosh UT-ECE

Going Deep

See tutorial at: <http://www.iro.umontreal.ca/~bengioy/talks/mlss-austin.pdf>

From Facebook's Deepface paper

<https://facebook.com/download/233199633549733/deepface.pdf>

Our method reaches an accuracy of 97.35% on the Labeled Faces in the Wild (LFW) dataset,
reducing the error of the current state
of the art by more than 27%, closely approaching human-level performance.

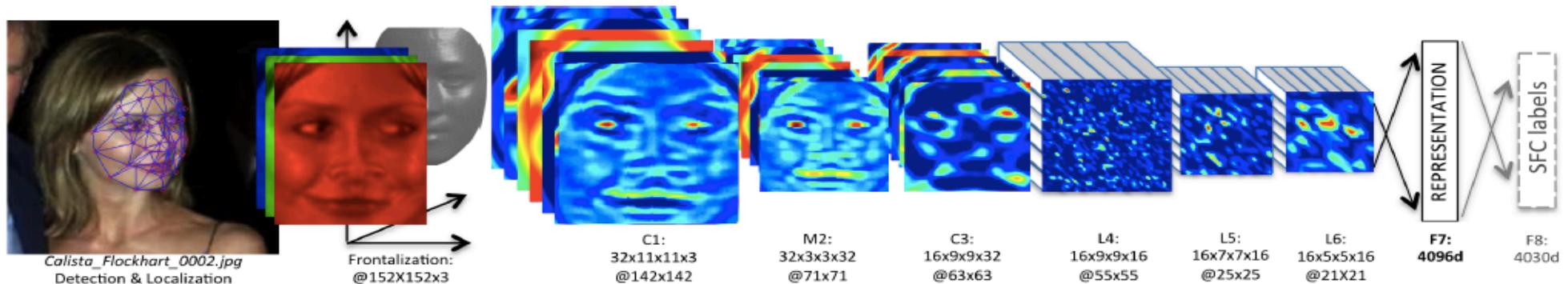


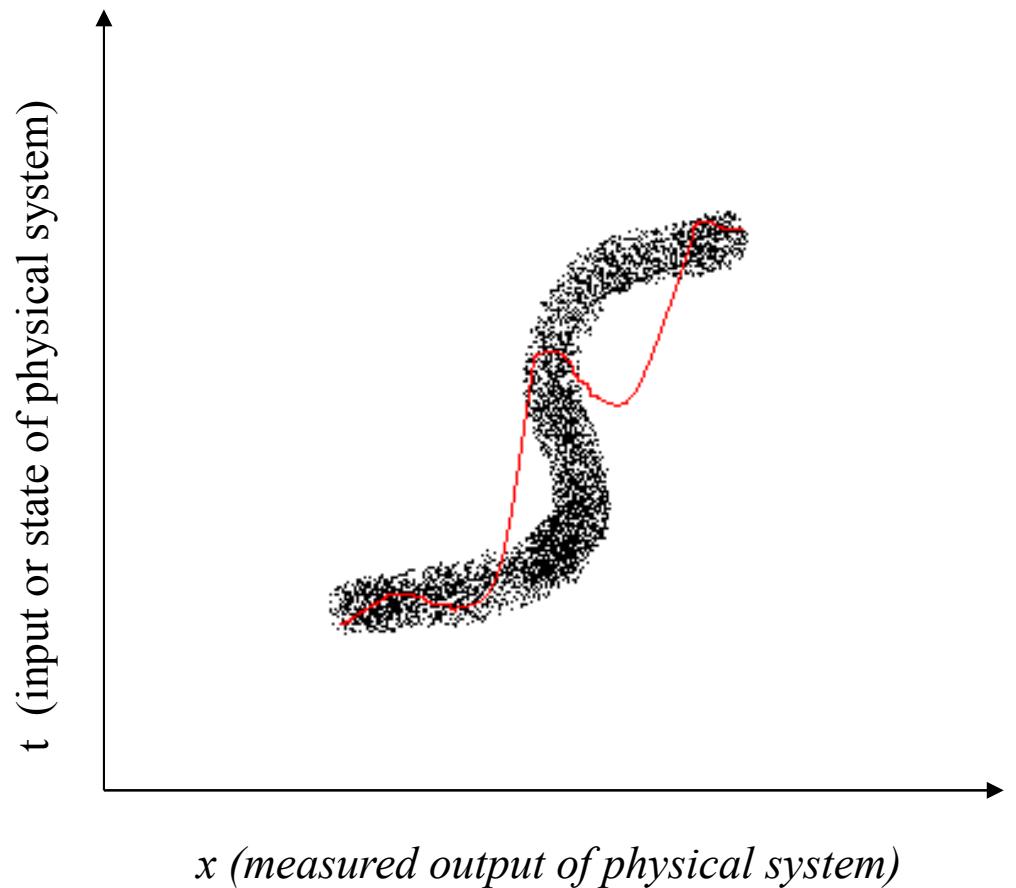
Figure 2. Outline of the *DeepFace* architecture. A front-end of a single convolution-pooling-convolution filtering on the rectified input, followed by three locally-connected layers and two fully-connected layers. Colors illustrate outputs for each layer. The net includes more than 120 million parameters, where more than 95% come from the local and fully connected layers.

So which method should I choose?

- Depends on type, complexity of problem; data size
 - (i) try linear regression first
 - Explore data; Study residues
 - do feature selection/transformation if needed
 - If robustness is needed, try “rlm” (R package), or use SVR.
 - (ii) Now try a set of powerful but not so interpretable models (MLP, deep learning, etc)
 - (estimate complexity of fit using a few trial runs)
 - How much is the gap between (i) and (ii)?
 - Consider Decision tree based regression if interpretation is important or a piecewise constant answer is more “actionable”
- Still lacking? try ensemble approaches specially gbdt, which also rank orders the features.

Caution: Modeling Inverse Problems

- Forward problem reasonably characterized by a function, but not the reverse problem
- t is not $h(x) + (\text{zero-mean, unimodal}) \text{ noise}$
So Least squares solution will bomb
- Solution: model joint pdf
 - Or piecewise models



Extras

Correlation vs. Causation

- Though it may still be actionable

Top 10 Best (and Worst) Educated States, and How They Voted

ranked by percentage of residents 25 years of age or older with college degree or more

% over 25 with
college degree

Best Educated

39.1%	1. Massachusetts
36.9%	2. Maryland
36.7%	3. Colorado
36.2%	4. Connecticut
35.4%	5. Vermont
35.3%	6. New Jersey
35.1%	7. Virginia
33.4%	8. New Hampshire
32.9%	9. New York
32.4%	10. Minnesota



% over 25 with
college degree

Worst Educated

18.5%	1. West Virginia
19.8%	2. Mississippi
20.3%	3. Arkansas
21.1%	4. Kentucky
21.1%	5. Louisiana
22.3%	6. Alabama
22.5%	7. Nevada
23.0%	8. Indiana
23.6%	9. Tennessee
23.8%	10. Oklahoma



Research Statistics provided by FoxBusiness.com, based on education data from the U.S. Census Bureau's American Community Survey. 24/7 Wall St. identified the U.S. states with the largest and smallest percentages of residents 25 or older with a college degree or more. <http://www.foxbusiness.com/personal-finance/2012/10/15/americas-best-and-worst-educated-states/>

Sequential Learning with SGD

- Learning rate: too small \rightarrow too slow
 - Too large \rightarrow oscillatory, or may even diverge
- Should η be fixed or adaptive (second order methods)?
- Is convergence needed or not?
 - Non-stationary? May not want to converge!
 - If convergence is desired, then η should decrease with time.
 - * Robbins-Monroe Sequence is adequate (e.g. $\eta(t) = 1/t$)
 - Sum of absolute values (of sequence of η values) is unbounded
 - Sum of squared values is finite
 - Values in decreasing sequence

Incremental Forward Stagewise Regression (HTF 3.8)

Algorithm 3.4 *Incremental Forward Stagewise Regression— FS_ϵ .*

1. Start with the residual \mathbf{r} equal to \mathbf{y} and $\beta_1, \beta_2, \dots, \beta_p = 0$. All the predictors are standardized to have mean zero and unit norm.
 2. Find the predictor \mathbf{x}_j most correlated with \mathbf{r}
 3. Update $\beta_j \leftarrow \beta_j + \delta_j$, where $\delta_j = \epsilon \cdot \text{sign}[\langle \mathbf{x}_j, \mathbf{r} \rangle]$ and $\epsilon > 0$ is a small step size, and set $\mathbf{r} \leftarrow \mathbf{r} - \delta_j \mathbf{x}_j$.
 4. Repeat steps 2 and 3 many times, until the residuals are uncorrelated with all the predictors.
-

HTF, pg 87

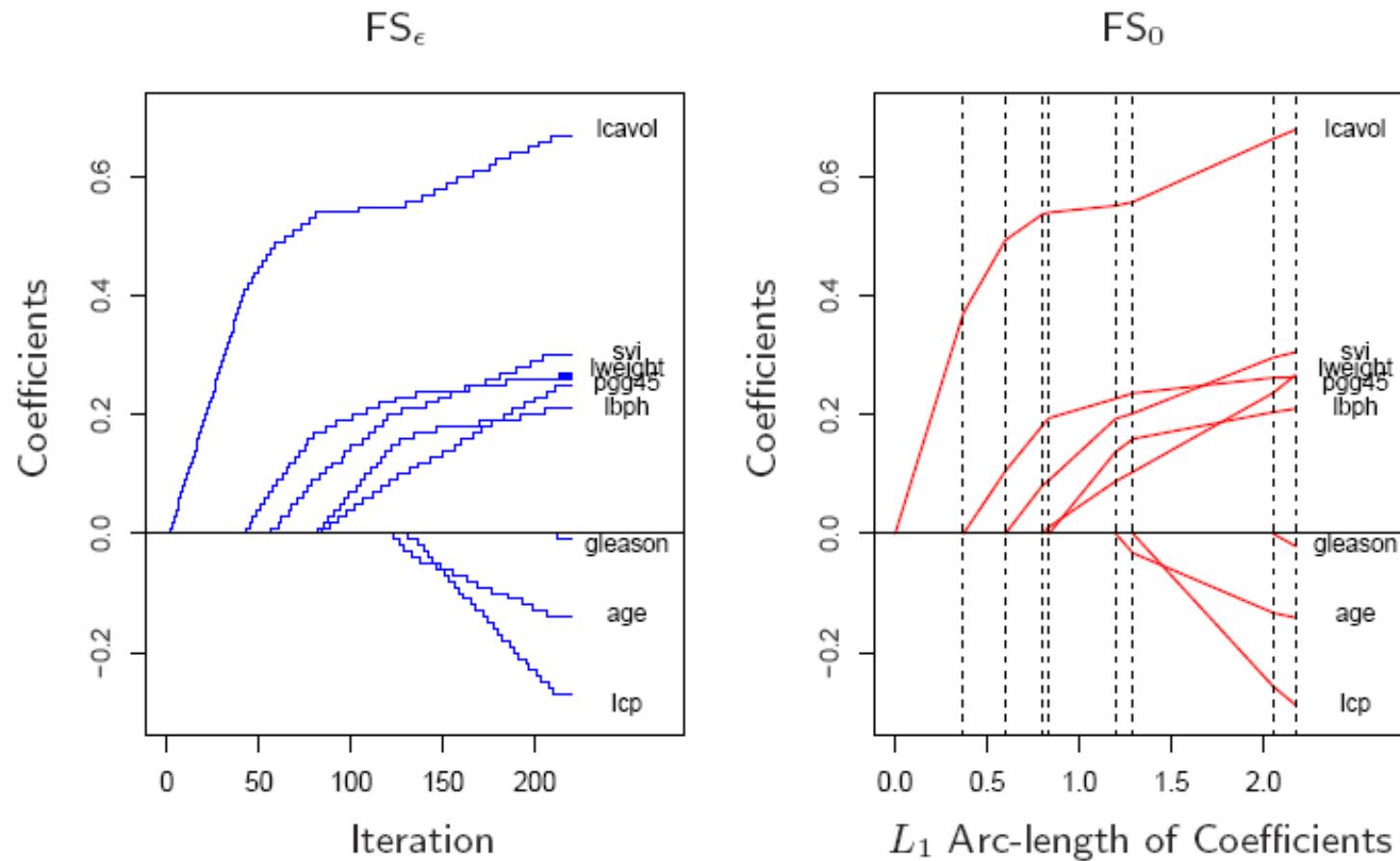


FIGURE 3.19. Coefficient profiles for the prostate data. The left panel shows incremental forward stagewise regression with step size $\epsilon = 0.01$. The right panel shows the infinitesimal version FS_0 obtained letting $\epsilon \rightarrow 0$. This profile was fit by the modification 3.2b to the LAR Algorithm 3.2. In this example the FS_0 profiles are monotone, and hence identical to those of lasso and LAR.

Regression on Derived Input Projections (HTF 3.5)

- PCR: Principal Component Regression
- PLS: Partial Least Squares

pls package contains both. Typically their effect is similar to ridge regression, but shrinking of coefficients not so smooth

Algorithm 3.3 Partial Least Squares.

1. Standardize each \mathbf{x}_j to have mean zero and variance one. Set $\hat{\mathbf{y}}^{(0)} = \bar{y}\mathbf{1}$, and $\mathbf{x}_j^{(0)} = \mathbf{x}_j$, $j = 1, \dots, p$.
2. For $m = 1, 2, \dots, p$
 - (a) $\mathbf{z}_m = \sum_{j=1}^p \hat{\varphi}_{mj} \mathbf{x}_j^{(m-1)}$, where $\hat{\varphi}_{mj} = \langle \mathbf{x}_j^{(m-1)}, \mathbf{y} \rangle$.
 - (b) $\hat{\theta}_m = \langle \mathbf{z}_m, \mathbf{y} \rangle / \langle \mathbf{z}_m, \mathbf{z}_m \rangle$.
 - (c) $\hat{\mathbf{y}}^{(m)} = \hat{\mathbf{y}}^{(m-1)} + \hat{\theta}_m \mathbf{z}_m$.
 - (d) Orthogonalize each $\mathbf{x}_j^{(m-1)}$ with respect to \mathbf{z}_m : $\mathbf{x}_j^{(m)} = \mathbf{x}_j^{(m-1)} - [\langle \mathbf{z}_m, \mathbf{x}_j^{(m-1)} \rangle / \langle \mathbf{z}_m, \mathbf{z}_m \rangle] \mathbf{z}_m$, $j = 1, 2, \dots, p$.
3. Output the sequence of fitted vectors $\{\hat{\mathbf{y}}^{(m)}\}_1^p$. Since the $\{\mathbf{z}_\ell\}_1^m$ are linear in the original \mathbf{x}_j , so is $\hat{\mathbf{y}}^{(m)} = \mathbf{X} \hat{\beta}^{\text{pls}}(m)$. These linear coefficients can be recovered from the sequence of PLS transformations.