Stochastic Gradient Descent

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

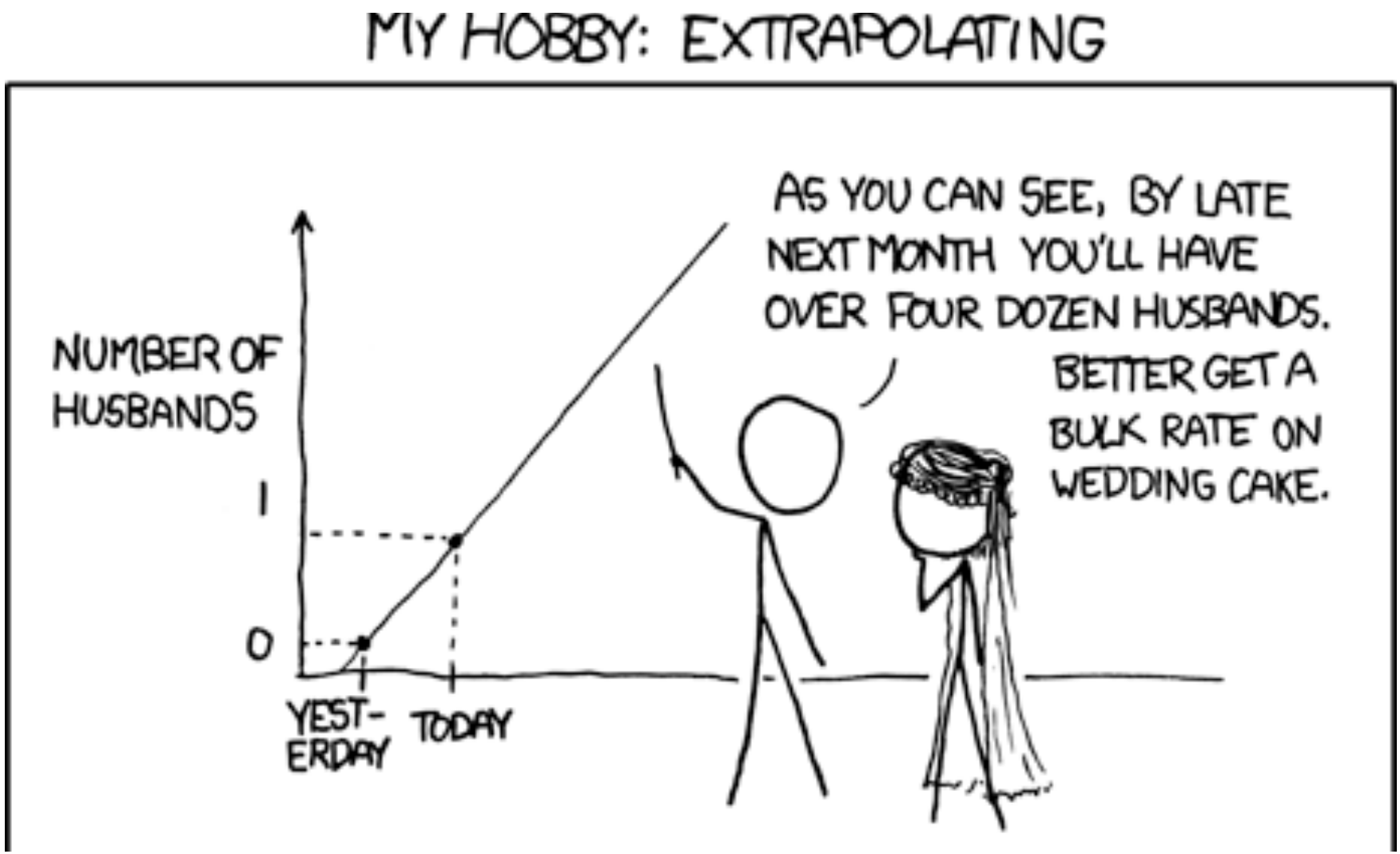
(Recap) Linear Regression

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

  Model: $E(y \mid \mathbf{x}) = \beta_0 + \beta_1 x_1 + \ldots + \beta_k x_k$ + 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



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
  - Universal approximator: Given a continuous function, there is some member of that family that can exactly it

How to Learn the Weights?

- **Linear Regression** with Cost E(w) = SSE or MSE
  - **direct** solution ("pseudo-inverse solution") gives optimal weights, w*
  - **One Step:** Given initial guess, $w^{(0)}$, can find w* in one update step.
    - Realize that E(w)) is quadratic in **w:** $E(\mathbf{w}) = \text{SSE}(\mathbf{w}) = \text{SSE}(\mathbf{w}^*) + (\mathbf{w}-\mathbf{w}^*)^T\mathbf{Q}(\mathbf{w}-\mathbf{w}^*)$
    - **w*** is the optimum solution, and **Q** a positive definite matrix.
    - So only one update of Newton's "root finding" step is needed.
  - **Iterative using Gradient Descent:** Given initial guess, $w^{(0)}$, iteratively update w by "going down the gradient" till you reach w*.

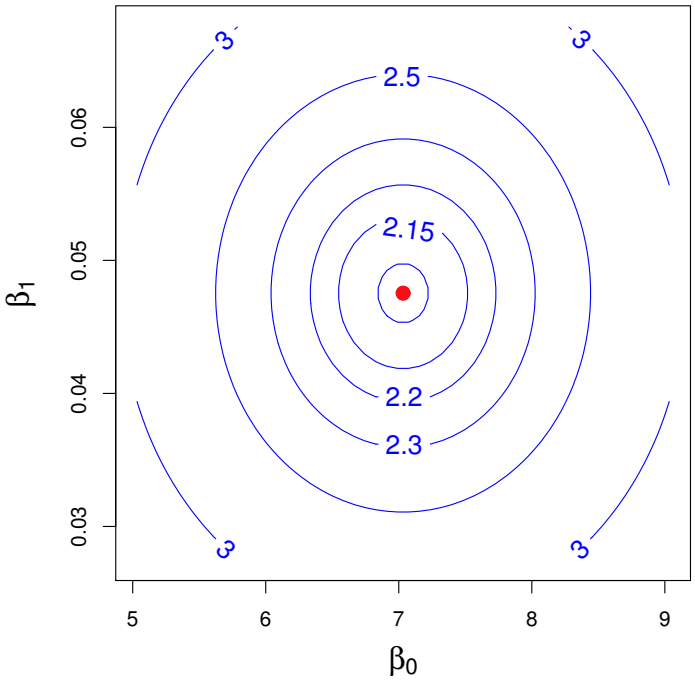
- **Nonlinear models:** E(w) is not quadratic, and in general non-convex.
  - weights are typically updated in an iterative manner (could be "on-line" or batch iterative)
  - SGD a popular choice, is on-line.

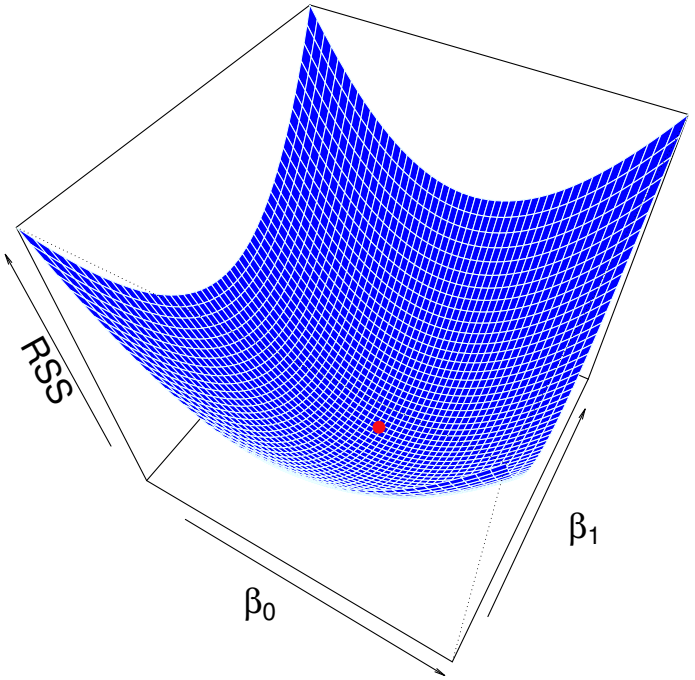
Gradient Descent for MLR

- **See (very introductory) Coursera Lectures by Andrew Ng, and Bishop Ch 5.1, 5.2**
- The cost function $E(\mathbf{w})$ is quadratic in $\mathbf{w}$
  - 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

- Learning rate η is crucial
  - Too low: slow convergence
  - Somewhat high: convergence with oscillations
  - Too high: unstable/divergence

What happens when gradients differ greatly across the weights? (Sketch below).

Gradient Descent for **Non-Linear Models**

E(w) is not quadratic, and in general non-convex, with *multiple local minima*.

Visualize GD as well as some second-order methods

http://www.benfrederickson.com/numerical-optimization/

(look at Gradient Descent example first, then Nelder-Mead)

- motivates adaptive learning rates, and second order methods that look at curvatures (2^{nd} derivative) in addition to gradients.

- Note: True gradient descent is a batch algorithm, involving all of the training data. (why?)

Stochastic gradient descent (SGD)

- "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 (inner loop) iteration number, n denoting the datapoint being considered, we get:

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

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)

Stokhos = "to aim"

Sequential Learning with SGD

- Learning rate: too small → too slow
  - Too large → oscillatory, or may even diverge
- Should η be fixed or adaptive (second order methods)?

- Is convergence needed or not?
  - Streaming data?
    - Non-stationary? May not want to converge!
    - If convergence is desired, then η should decrease with time.
      - (e.g. choose $\eta = a/t$)
  - Batch training data?
    - Two loops:
      - outer: # of epochs
      - inner (one per epoch) = one pass over the training dataset.
    - Typically stop when validation error "flattens" vs. # of epochs.

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)

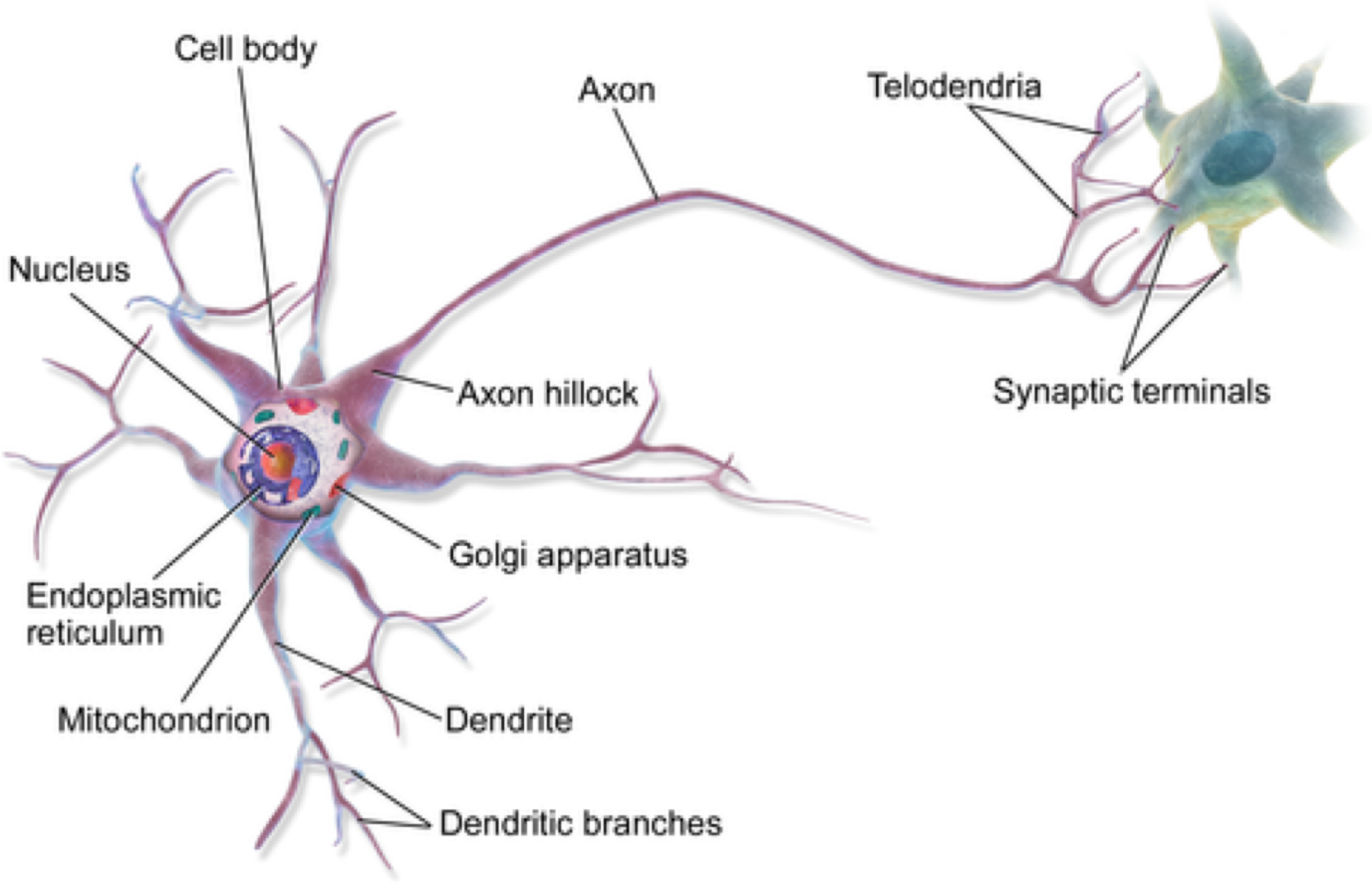
+ useful to scale inputs to help find learning rate (usually fixed)

+ 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. (see "conjugate gradient" in the animation link on previous slide).

A Neuron

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



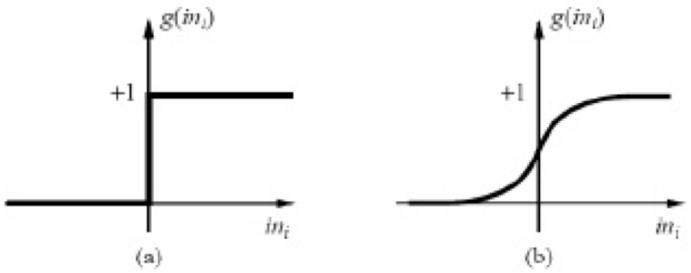
1-layer (Artificial) Neural Networks

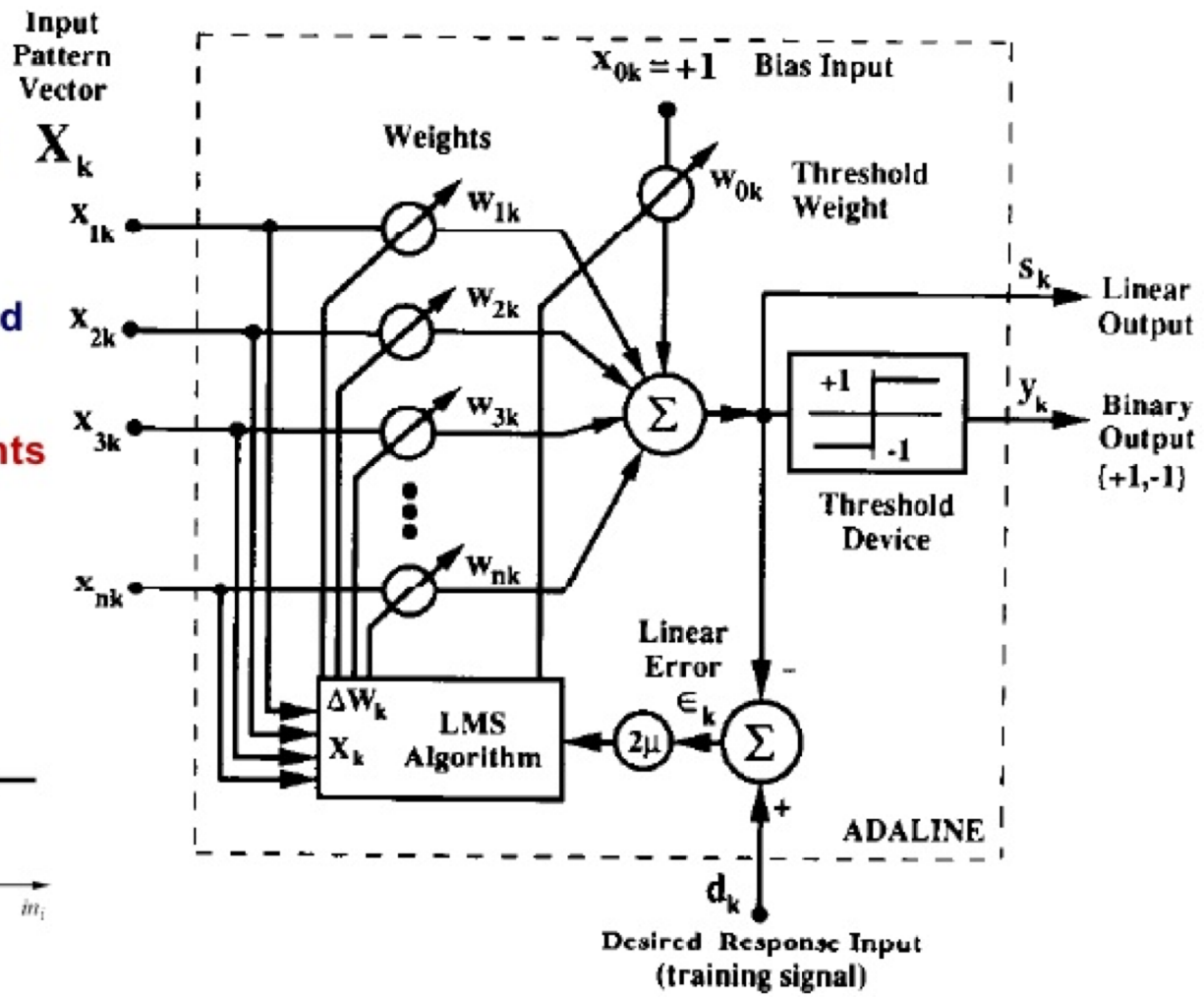
- Viewing (generalized) linear model as a "neural network" (draw below)

- Learning through SGD
- (déjà vu) LMS Algorithm, Widrow-Hoff rule from ADALINE (1960)

Adaline

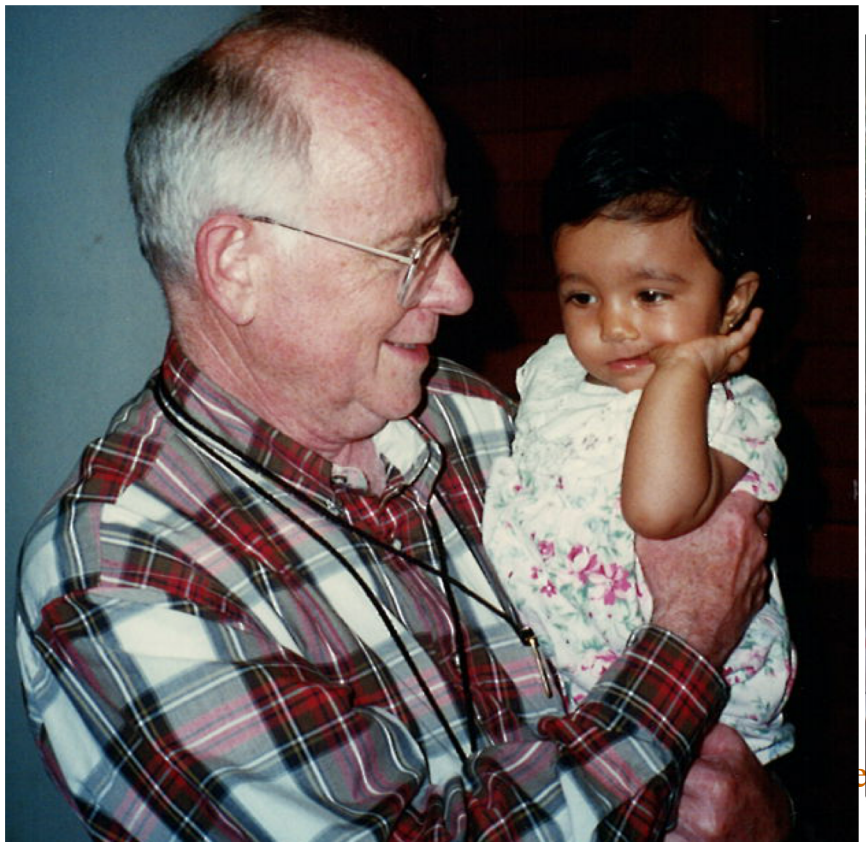
- Adaptive Linear Element
- Adaptive linear combiner cascaded with a hard-limiting quantizer
- Linear output transformed to binary by means of a threshold device
- Training = adjusting the weights
- Activation functions

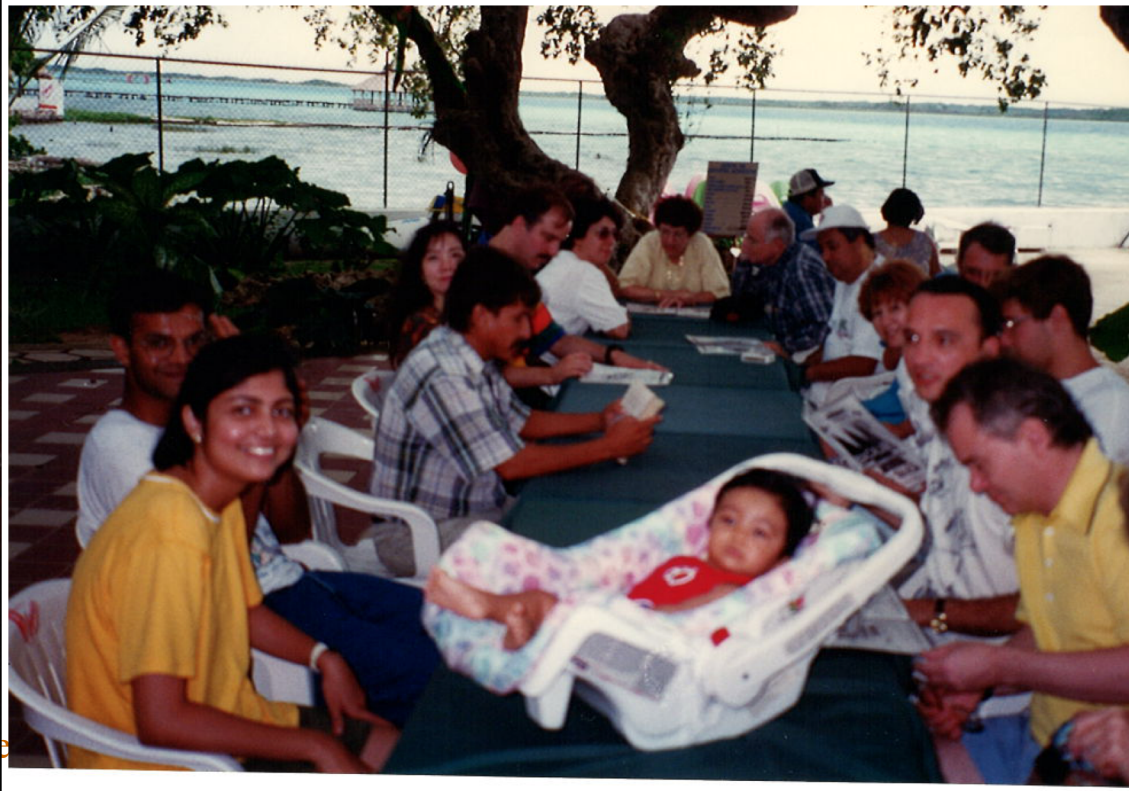




ADALINE

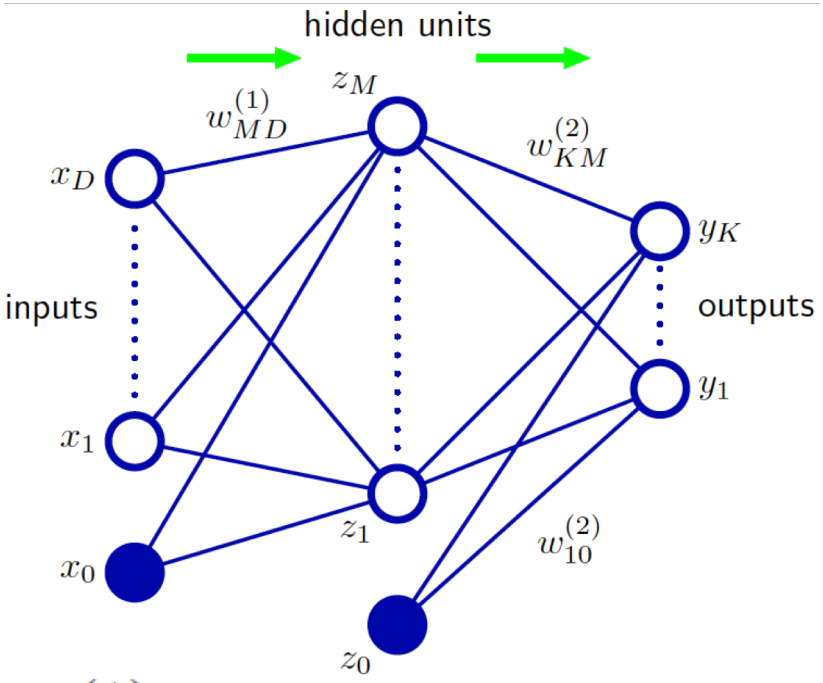
- https://www.youtube.com/watch?v=hc2Zj55j1zU





Multi-Layered Perceptrons (MLP) (most popular neural network before deep learning)

- Bishop Fig 5.1 (right). Typical 2-layered MLP for Regression (with one hidden layer of **M units** and 2 layers of adaptive weights):
  Choice of activation functions:
  - Hidden layer: tanh/sigmoid
  - Output layer: linear / sigmoid or softmax
- **Universal approximator** if output layer cells are linear



- Hidden layer with activation function h():

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \qquad z_j = h(a_j)$$

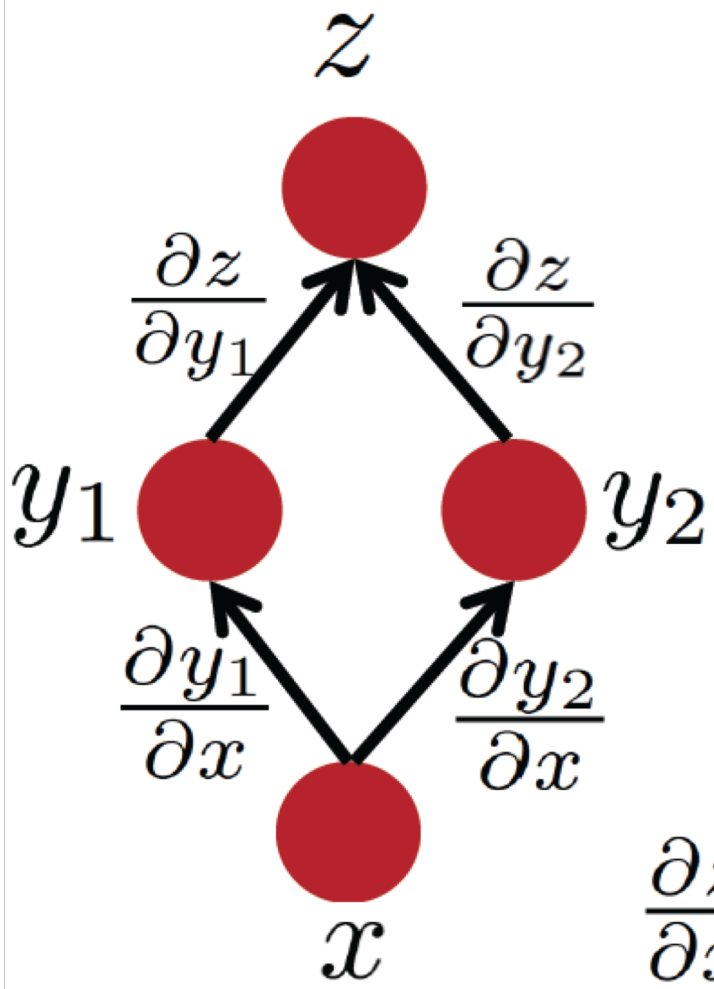
Output layer with activation function σ():

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)} \qquad y_k = \sigma(a_k)$$

Overall:

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

(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, using chain rule to "propagate" the error back from output towards the input nodes.

- Derived from chain rule for partial derivatives, applied to SGD
- Three stages:

1. Evaluate an "error signal" at the output units with net input a_k

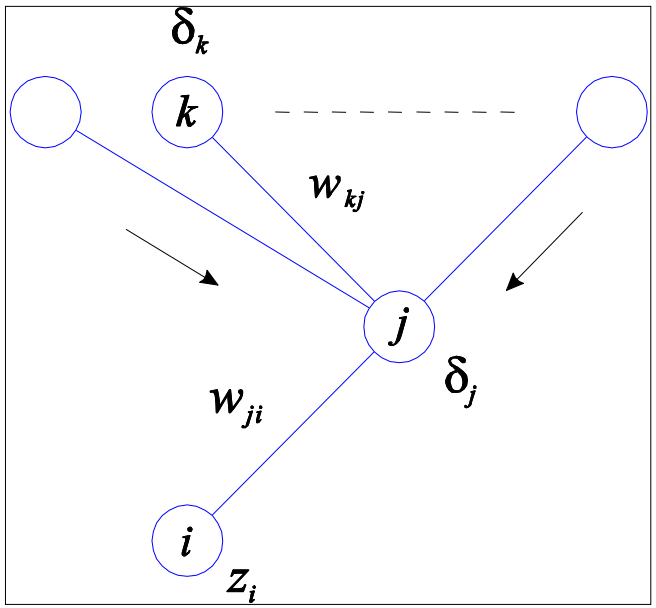
$$\delta_k = \frac{\partial E_n}{\partial a_k}$$

2. Propagate the signal backwards through the network

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

3. Evaluate derivatives

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



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

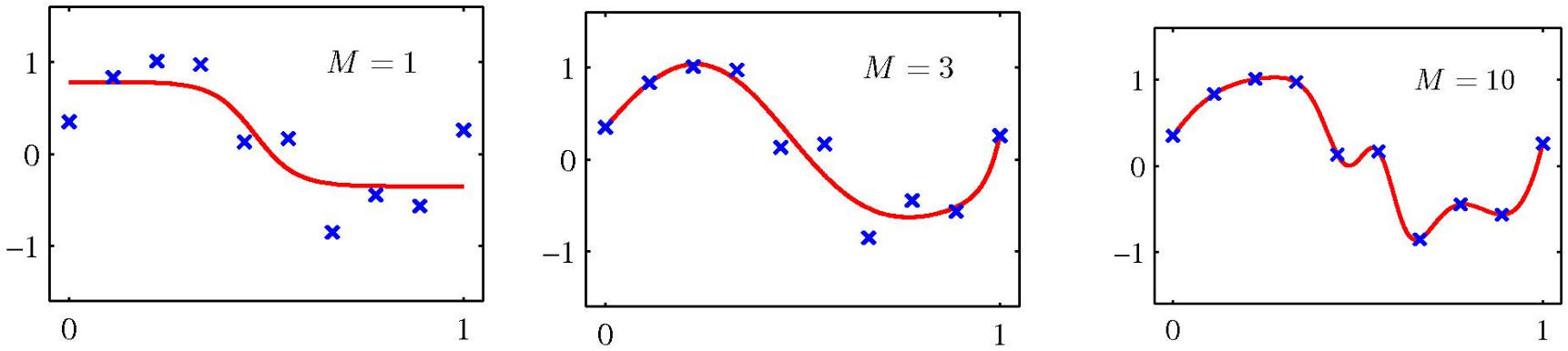
- http://playground.tensorflow.org/

- (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)



MLPs with 1, 3 and 10 hidden units, trained till MSE_train flattened out

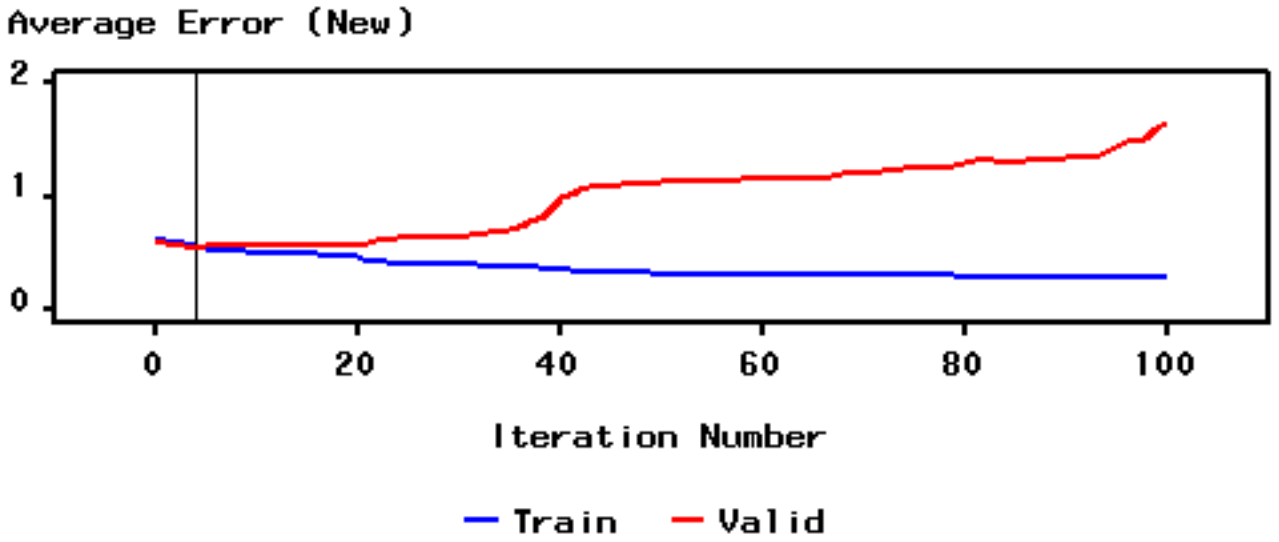
Fact: The "effective number of parameters" (or effective degrees of freedom) of an MLP increases with amount of training.

So, with less training M=10 solution looks like a network with M < 10 but with more training …

Adjusting Complexity

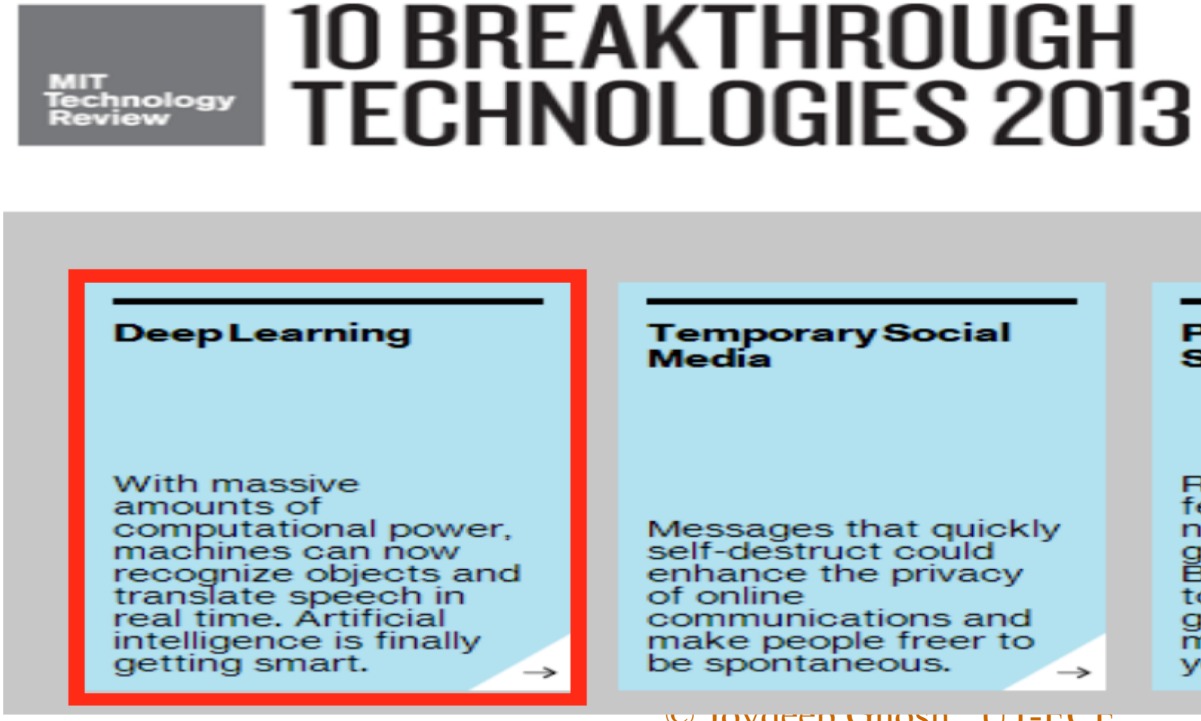
– "effective # of parameters" increases with # of epochs !!
  - Select adequately powerful model
  - while training, monitor performance using validation set
  - **stop training** when error on validation set reaches a minimum

  - 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 https://www.youtube.com/watch?v=ht6fLrar91U starting 32:00



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.

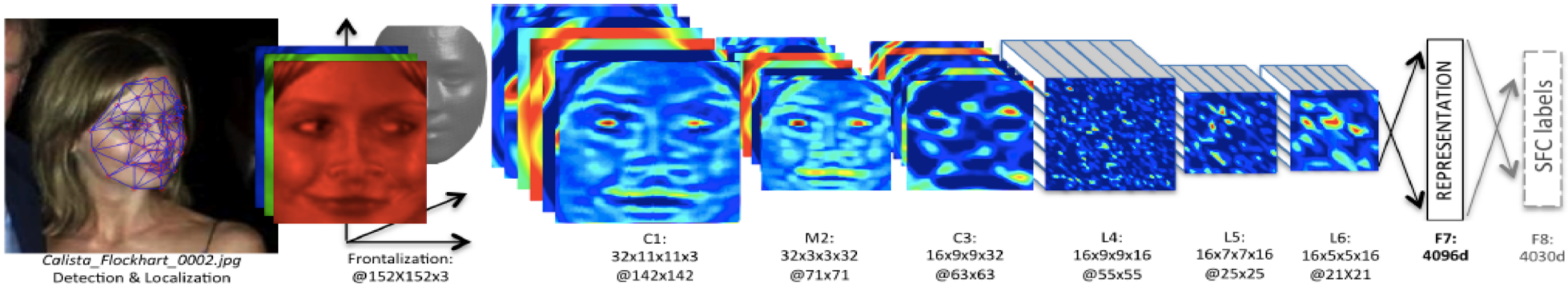
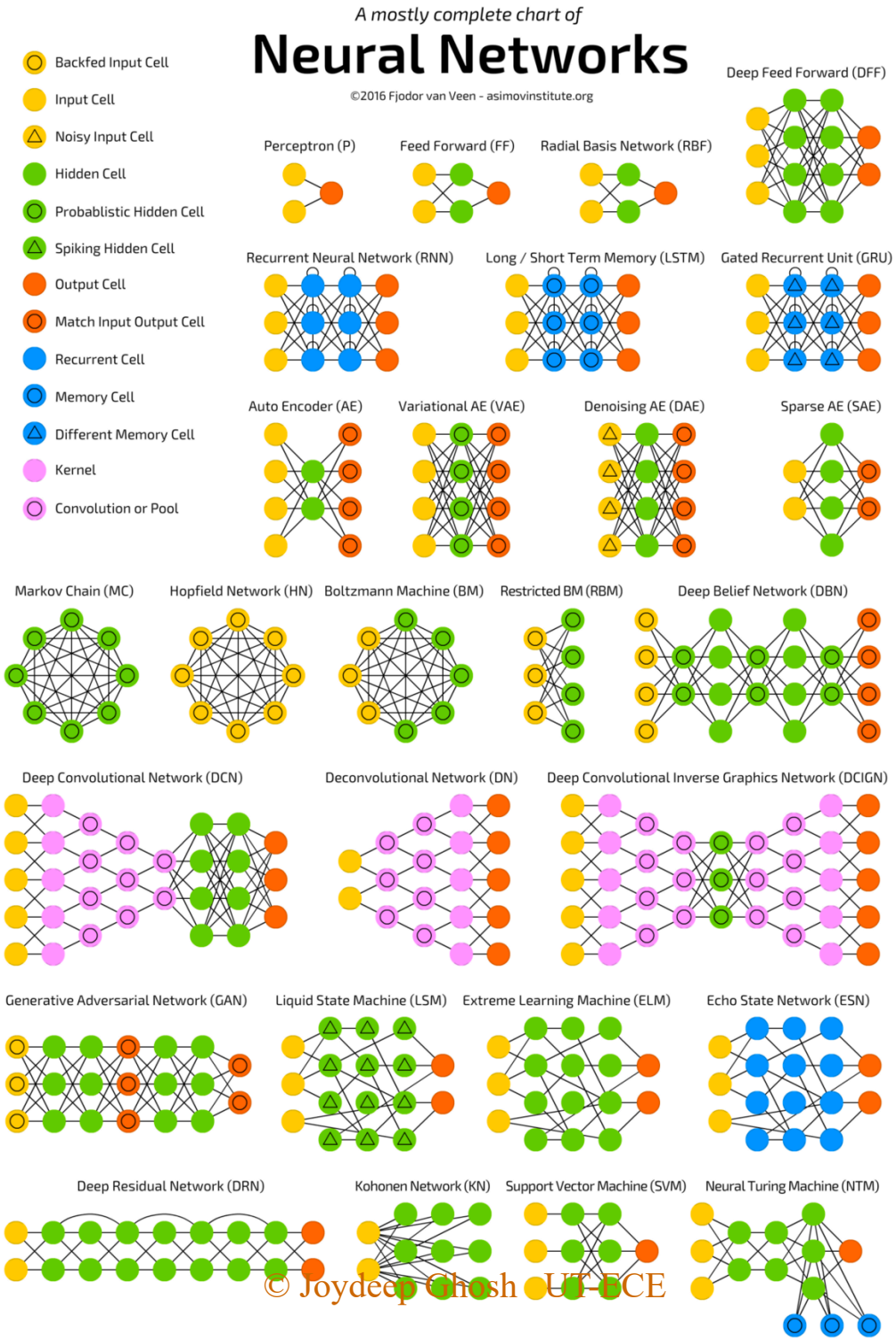


Figure 2. **Outline of the *DeepFace* architecture**. A front-end of a single convolution-pooling-convolution filtering on the rectified input, followed by three locally-connected layers and two fully-connected layers. Colors illustrate outputs for each layer. The net includes more than 120 million parameters,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



So which method should I choose?

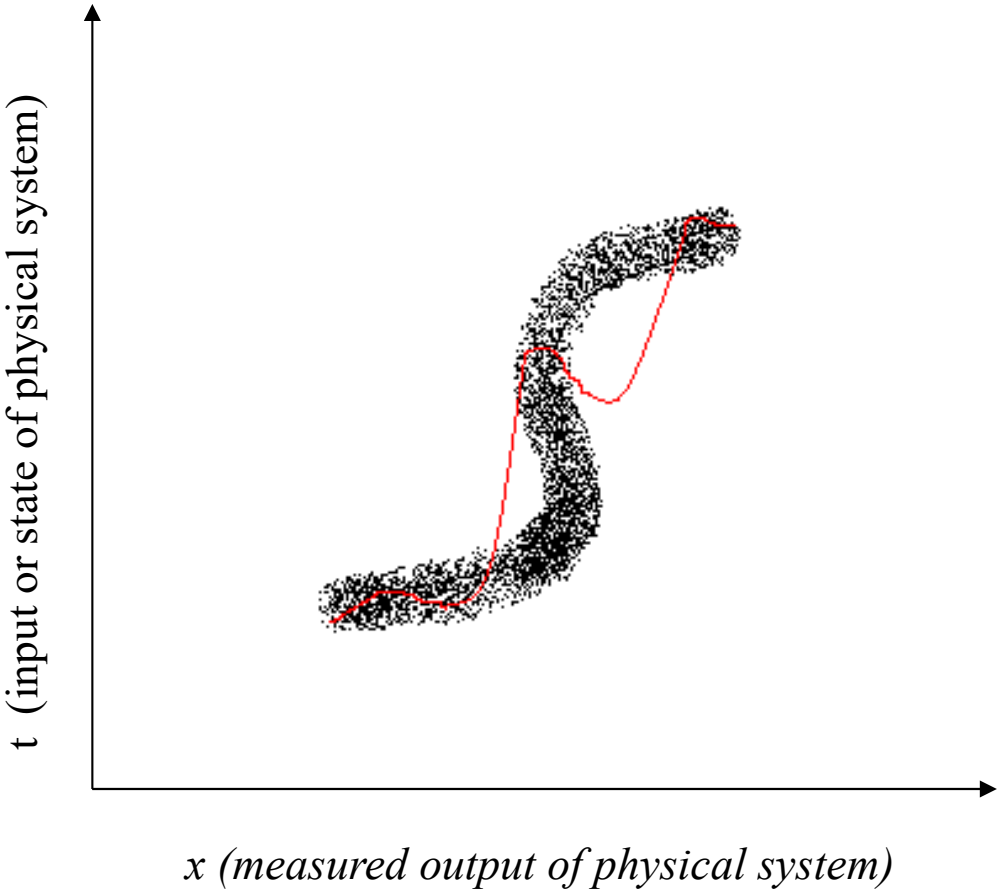
- Depends on type, complexity of problem; data size
  - (i) try linear regression first
    - Explore data; Study residues
    - do feature selection/transformation if needed
    - If robustness is needed, try "rlm" (R package), or use SVR.
  - (ii) Now try a set of powerful but not so interpretable 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**) + (zero-mean, unimodal) noise

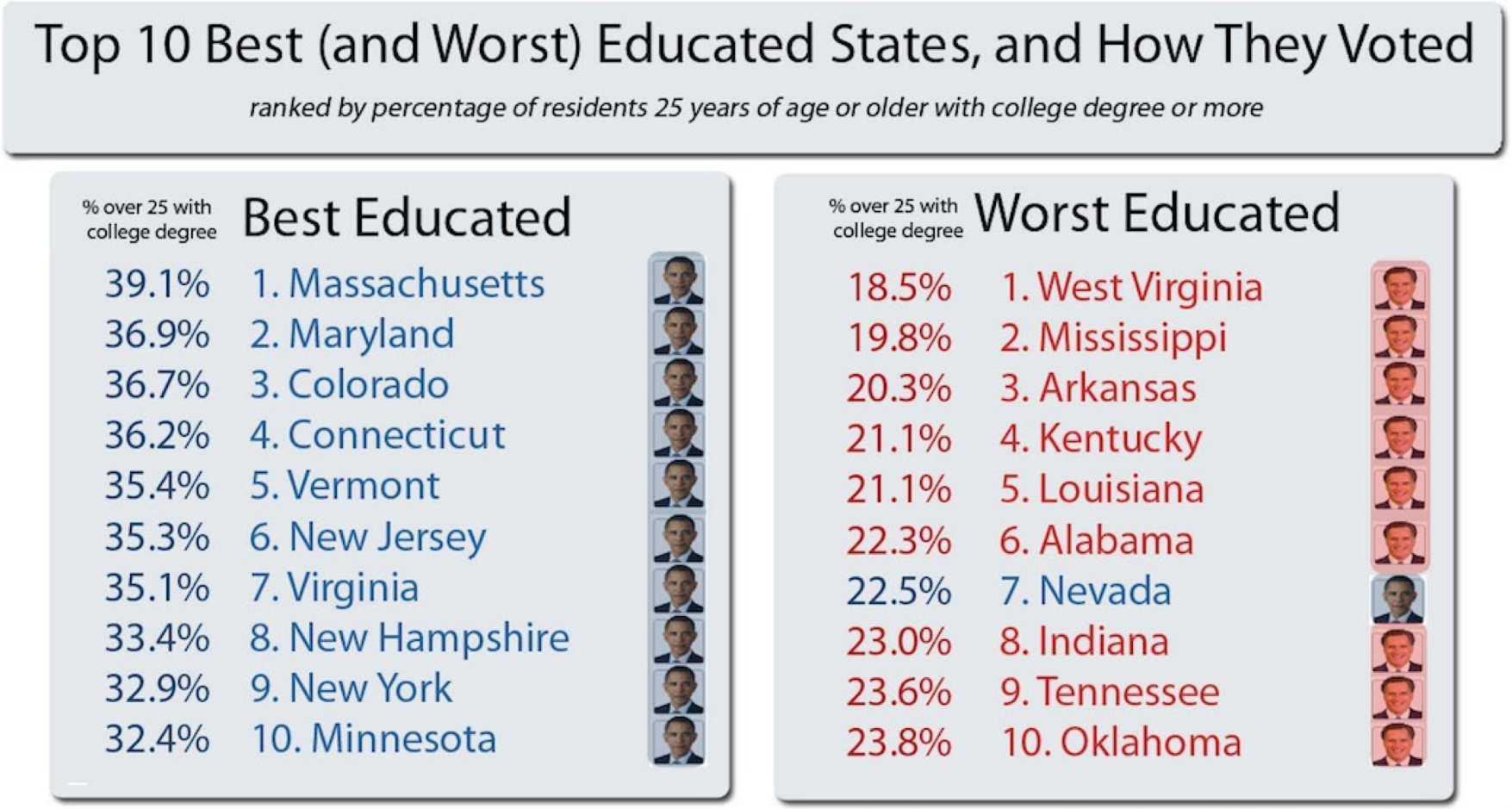
  So Least squares solution will bomb
- Solution: model joint pdf
  - Or piecewise models



Extras

Correlation vs. Causation

- Though it may still be actionable



Sequential Learning with SGD

- Learning rate: too small → too slow
  - Too large → 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 $\beta_1, \beta_2, \ldots, \beta_p = 0$. 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

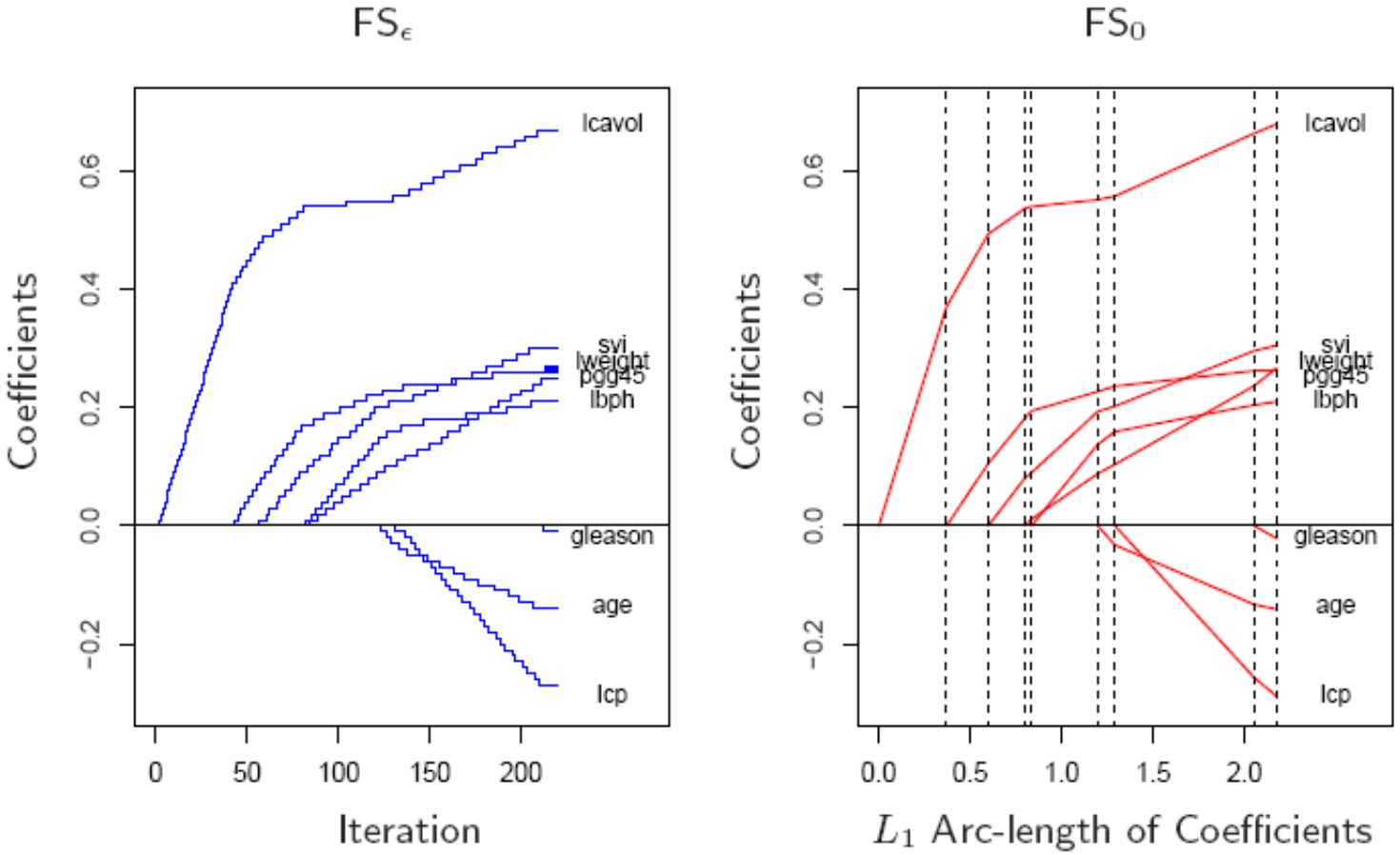


FIGURE 3.19. *Coefficient profiles for the prostate data. The left panel shows incremental forward stagewise regression with step size* $\epsilon = 0.01$. *The right panel shows the infinitesimal version* FS_0 *obtained letting* $\epsilon \to 0$. *This profile was fit by the modification 3.2b to the 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 $\mathbf{x}_j^{(0)} = \mathbf{x}_j,\ j = 1, \ldots, p$.
2. For $m = 1, 2, \ldots, p$
   (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 $\mathbf{z}_m$: $\mathbf{x}_j^{(m)} = \mathbf{x}_j^{(m-1)} - [\langle \mathbf{z}_m, \mathbf{x}_j^{(m-1)} \rangle / \langle \mathbf{z}_m, \mathbf{z}_m \rangle]\mathbf{z}_m,\ j = 1, 2, \ldots, p$.
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.