Stochastic Gradient Descent

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

Closely follows CB: 5.1 through 5.3.3

Backup reading: EA: 11-11.8
**Advanced Coverage of gradient descent in KM 8.1-8.4*

Goals

- Intro to non-linear Regression via neural networks
- What is (not) neural about neural nets?

- Understand how to apply Stochastic Gradient Descent based learning for neural networks
- Insights into why neural nets are so popular (and tricky)
  – How to finesse the bias-variance tradeoff
  – Benefits of online training
- Intro to deep learning

Beyond linear Regression

- linear in fixed transform ("phi") space
    - (nonlinear in original space)
- almost linear methods
    - piecewise linear (e.g. CART); locally linear regression, etc.

- general nonlinear forms (e.g. using basis functions, i.e. the "phi"s, that are also learnt)
    - e.g. feedforward neural networks (MLP, Conv Nets,…)
    - MLPs are universal approximators, just like polynomials.

Universal approximator family of functions: Given any *continuous* function, there is some member of that family that can exactly match it

How to Learn the Weights?

– **Linear Regression** with Cost E(w) = SSE or MSE
  - **direct** solution ("pseudo-inverse solution") gives optimal weights, w* (for given dataset, minimizing the empirical error)
  - **One Step:** Given initial guess, $w^{(0)}$, can find w* in one update step.
    – Newton's "root finding" step
  - **Iterative using Gradient Descent:** Given initial guess, $w^{(0)}$, iteratively update w by "going down the gradient" till you reach w*.

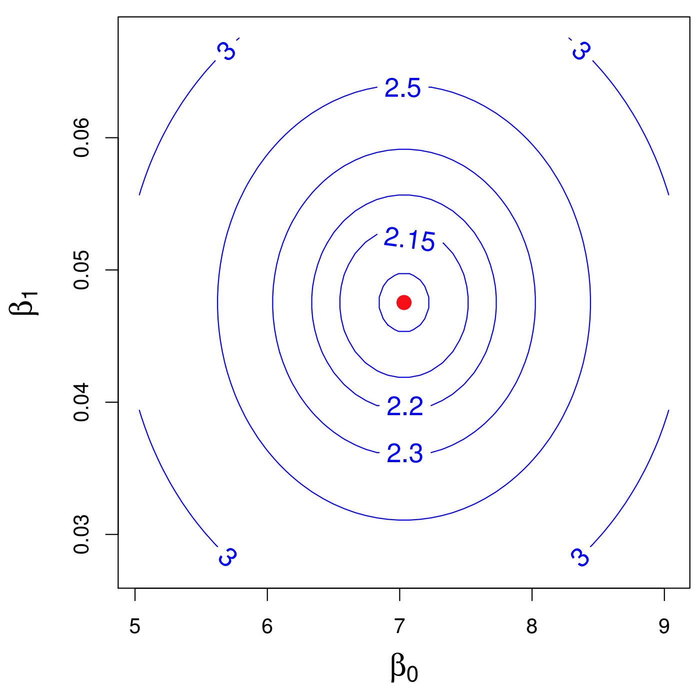
– **Nonlinear models:** E(w) is not quadratic, and in general non-convex.
  - weights are typically updated in an iterative manner (could be "on-line" or batch iterative)
  - SGD a popular choice, is on-line.

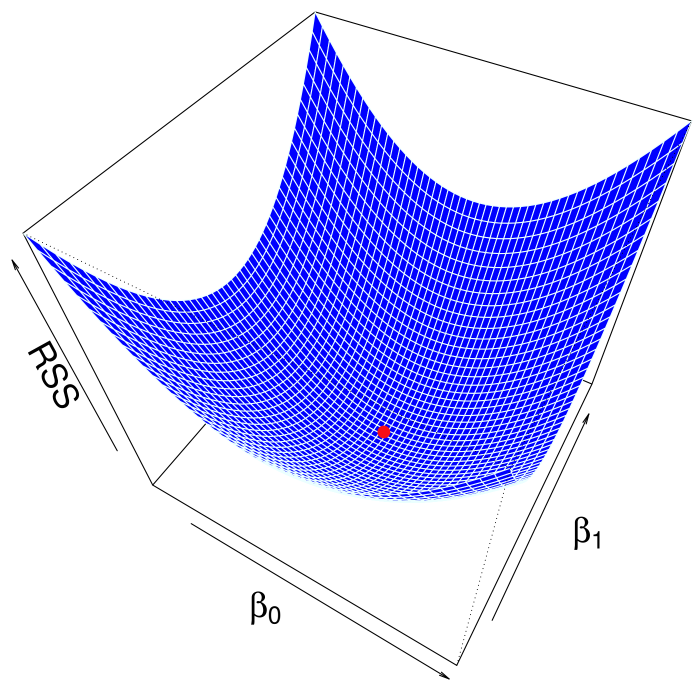
Gradient Descent for MLR

- **See (very introductory) Coursera Lectures by Andrew Ng, and Bishop Ch 5.1, 5.2**
- For a linear model, 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 weight dimensions? (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:

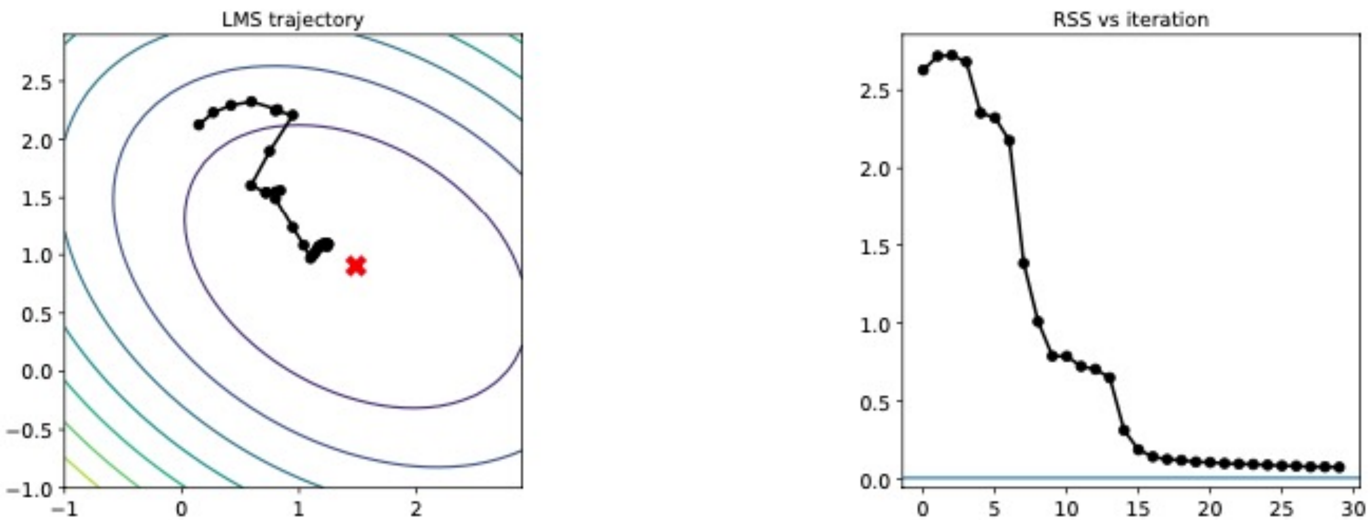
$$\begin{aligned} \mathbf{w}^{(\tau+1)} &= \mathbf{w}^{(\tau)} - \eta \nabla E_n \\ &= \mathbf{w}^{(\tau)} + \eta (t_n - \mathbf{w}^{(\tau)\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}_n)) \boldsymbol{\phi}(\mathbf{x}_n). \end{aligned}$$

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"

SGD example from KM pg. 290



(a) (b)

Figure 8.16: Illustration of the LMS algorithm. Left: we start from $\boldsymbol{\theta} = (-0.5, 2)$ and slowly converging to the least squares solution of $\hat{\boldsymbol{\theta}} = (1.45, 0.93)$ (red cross). Right: plot of objective function over time. Note that it does not decrease monotonically. Generated by code.probml.ai/book1/8.16.

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)

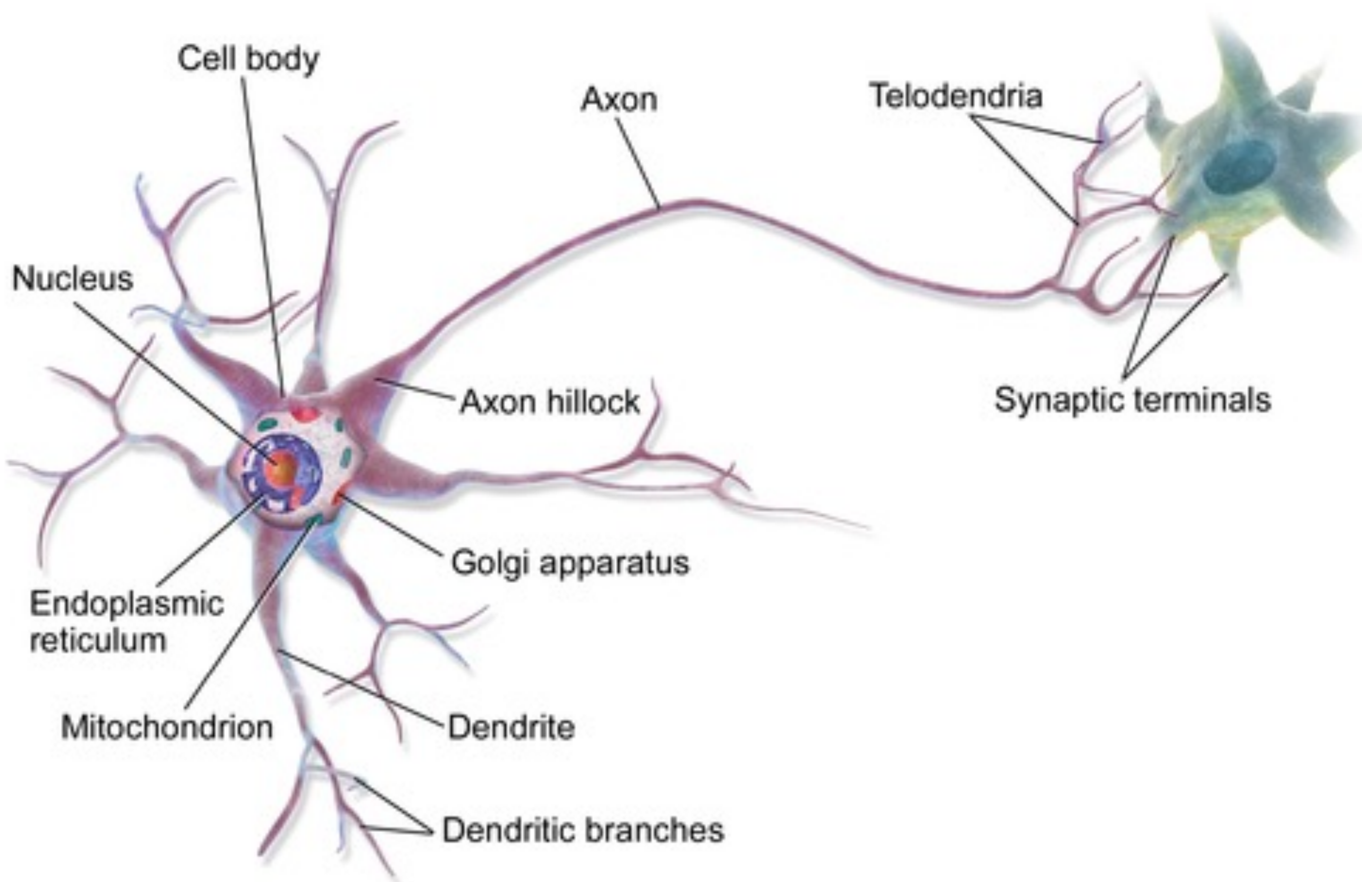
+ works for non-stationary environments as well as online settings

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 (vector of slopes), exist. (see “conjugate gradient” in the animation link on previous slide).

* Ruder’s Blog on advanced **gradient descent optimization algorithms.**

A Neuron

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



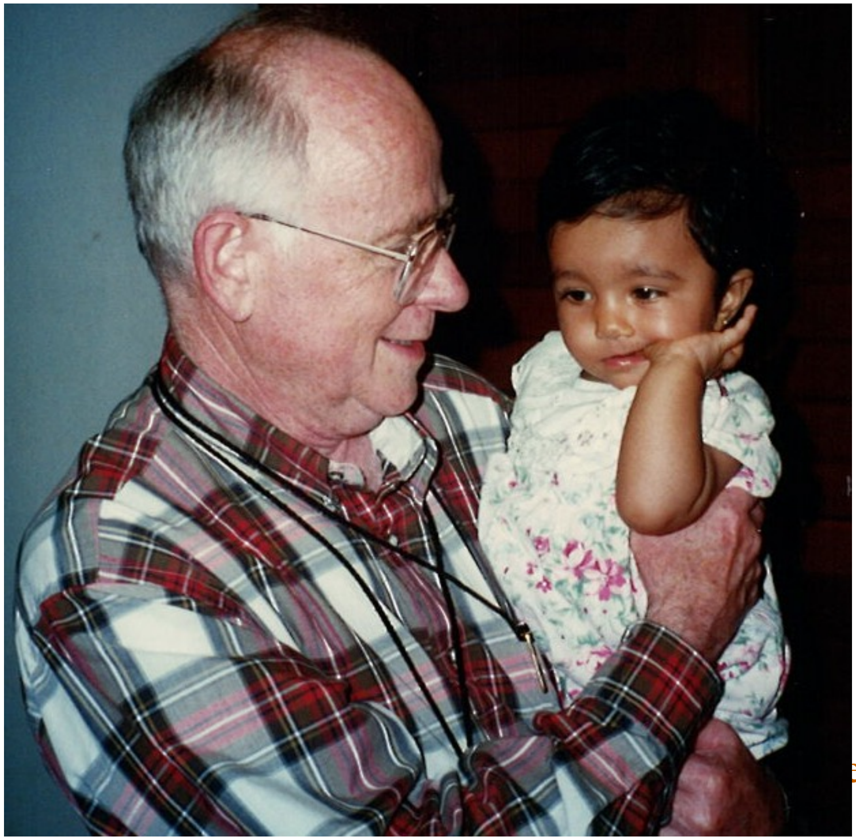
1-layer (Artificial) Neural Networks

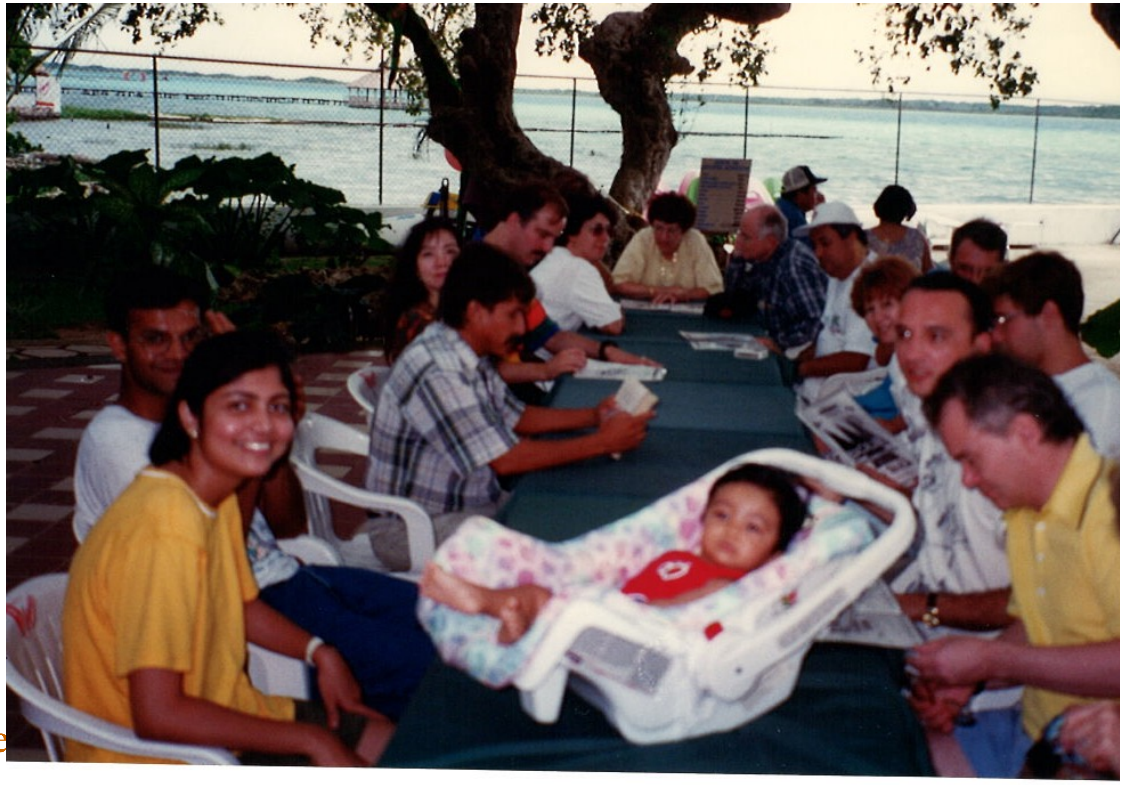
- Viewing (generalized) linear model as a “neural network” (draw below)

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

ADALINE

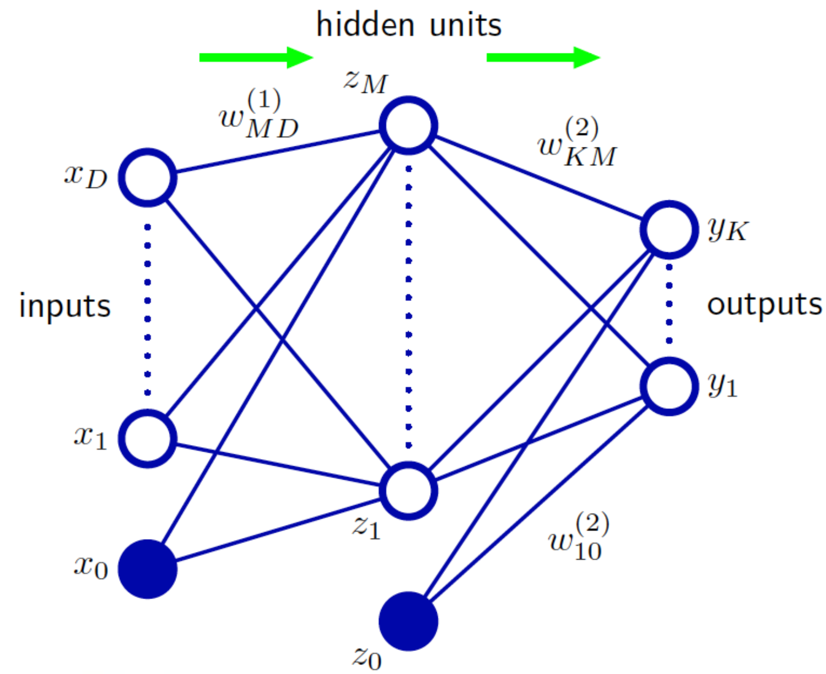
- https://www.youtube.com/watch?v=hc2Zj55j1zU





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 and hiddent layer is nonlinear with h() = tanh/sigmoid/gaussian/…

- 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 functior

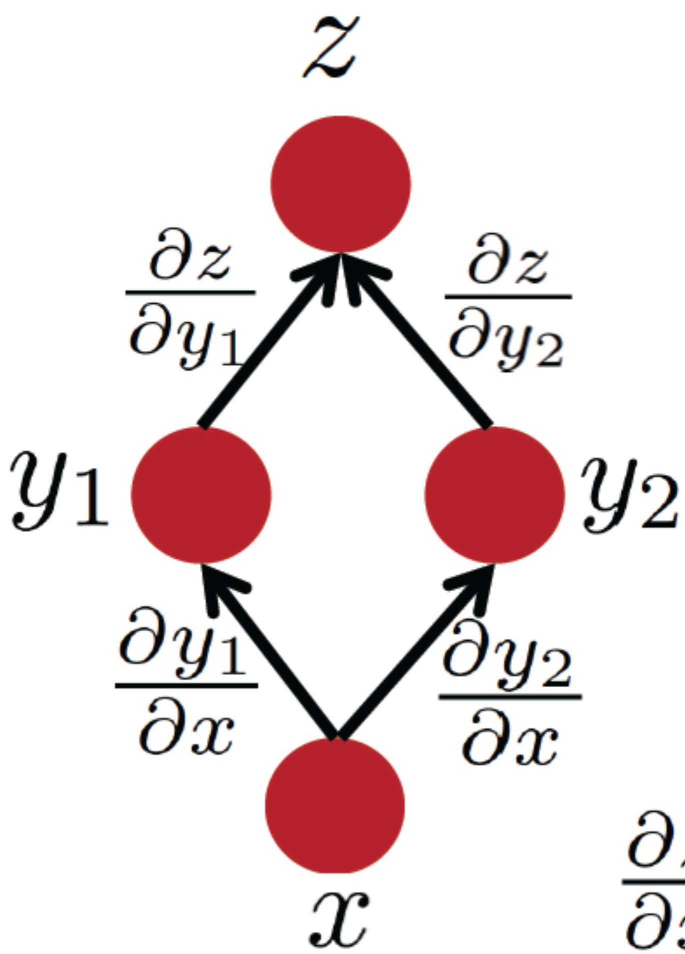
$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)} \qquad y_k = \sigma(a_k)$$

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

Depicts relations among 4 variables (not a neural net)



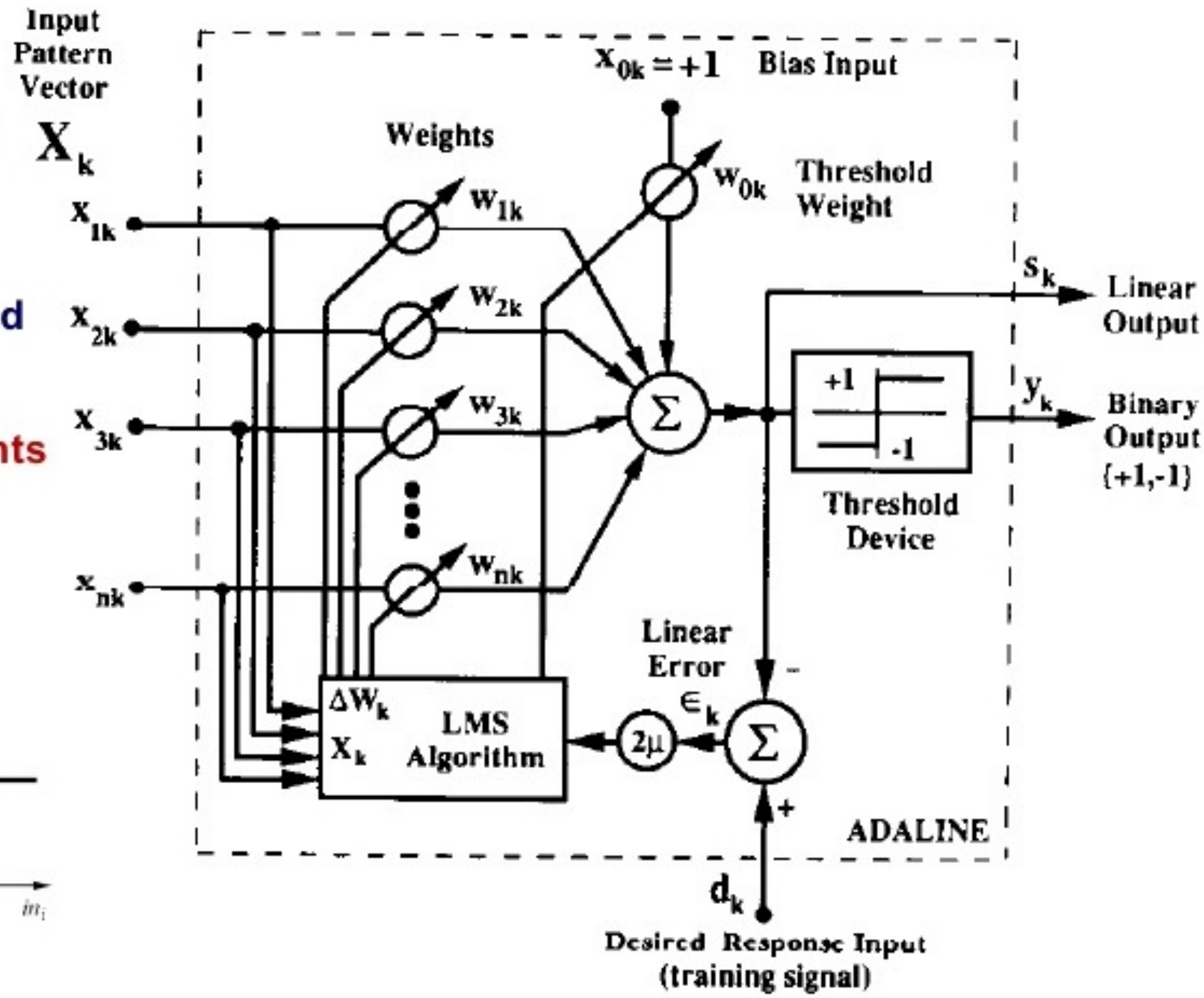
e.g. consider $z = (1-x)^3 + \sin(5x)$

Solve using $y1 = (1-x)$; $y2 = 5x$

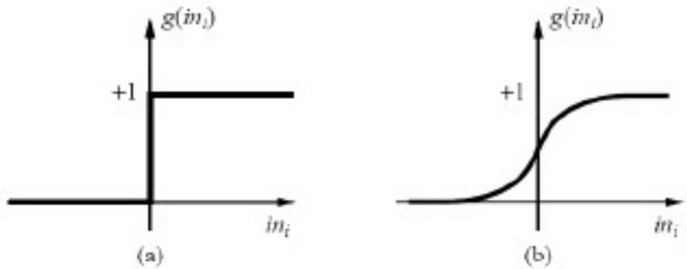
$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y_1}\frac{\partial y_1}{\partial x} + \frac{\partial z}{\partial y_2}\frac{\partial y_2}{\partial x}$$

Adaline

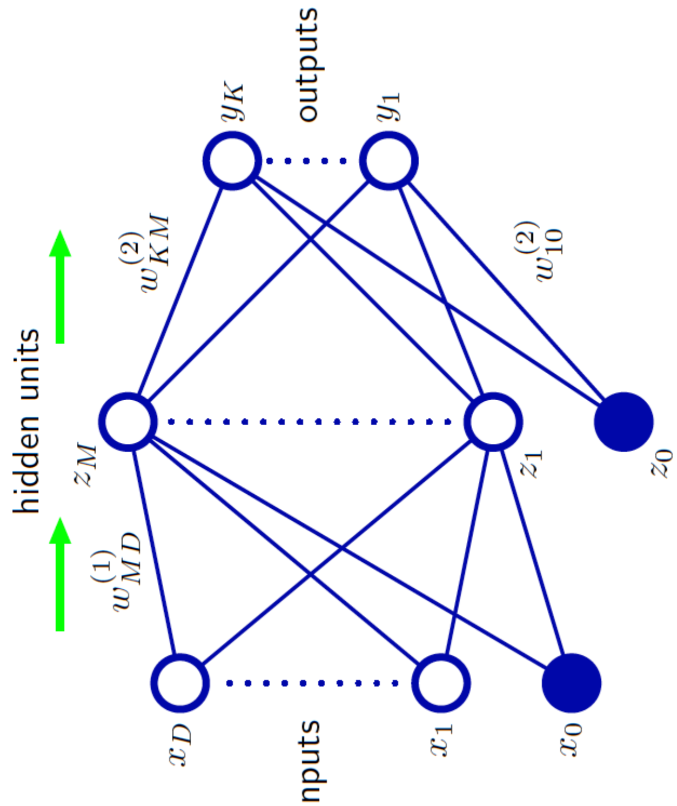
- Adaptive Linear Element
- Adaptive linear combiner cascaded with a hard-limiting quantizer
- Linear output transformed to binary by means of a threshold device
- Training = adjusting the weights



- Activation functions



Learning via Error Backpropagation



- Consider single output, y(x), sigmoid hidden units, linear output units

  y(x) = ?

Error = $(t-y)^2$

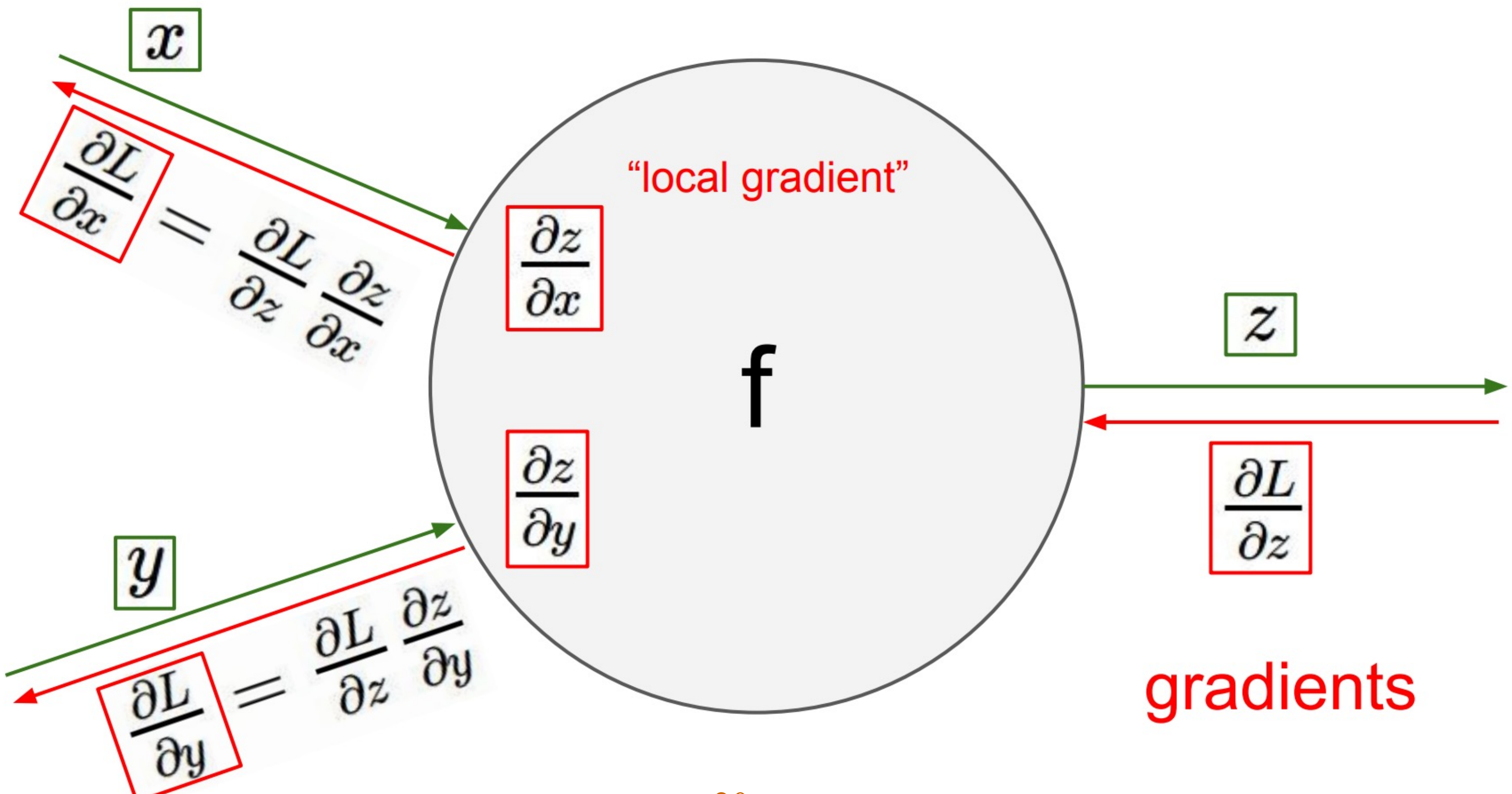
- So for given x, and using SGD, how will you update
  - A second layer weight?
  - A first layer weight?

- What changes if we have K outputs?

Backpropagation: Intuition

Simple video and follow-up

See Bishop 5.3.1 for application to MLP
Z = cell output, feeding into next layer that is closer to the final output
f= cell activation function. (relates z to inputs x, y)
Gradient of Error w.r.t. x (or y) involve previously computed gradient w.r.t. z



Local gradient times upstream gradient

Learning via Error Backpropagation

- 1. forward pass (from input layer through to output layer): compute outputs of successive layers, given the inputs to that layer
- 2. Compute error (loss) at final output layer by comparing with desired output values.
- 3. backward pass (from output layer to input layer): compute gradients of weights w.r.t. loss, layer by layer; and update all the weights.

- Cycle through all the data:
  - For batch settings: ONE EPOCH = one pass through the training dataset.

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

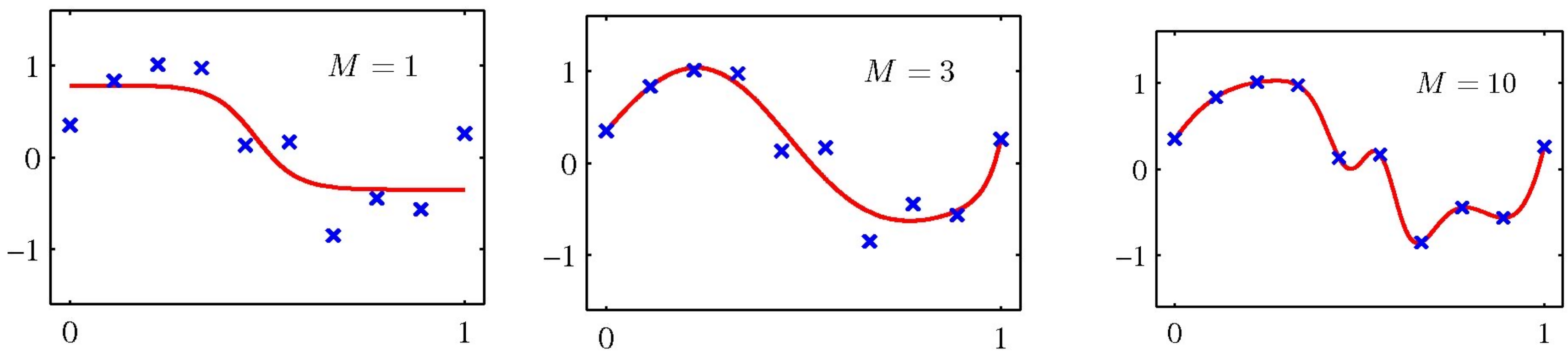
- Momentum: (a second order gradient descent method) https://distill.pub/2017/momentum/
- *More advanced methods, including demos (also see KM 8-8.4)

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

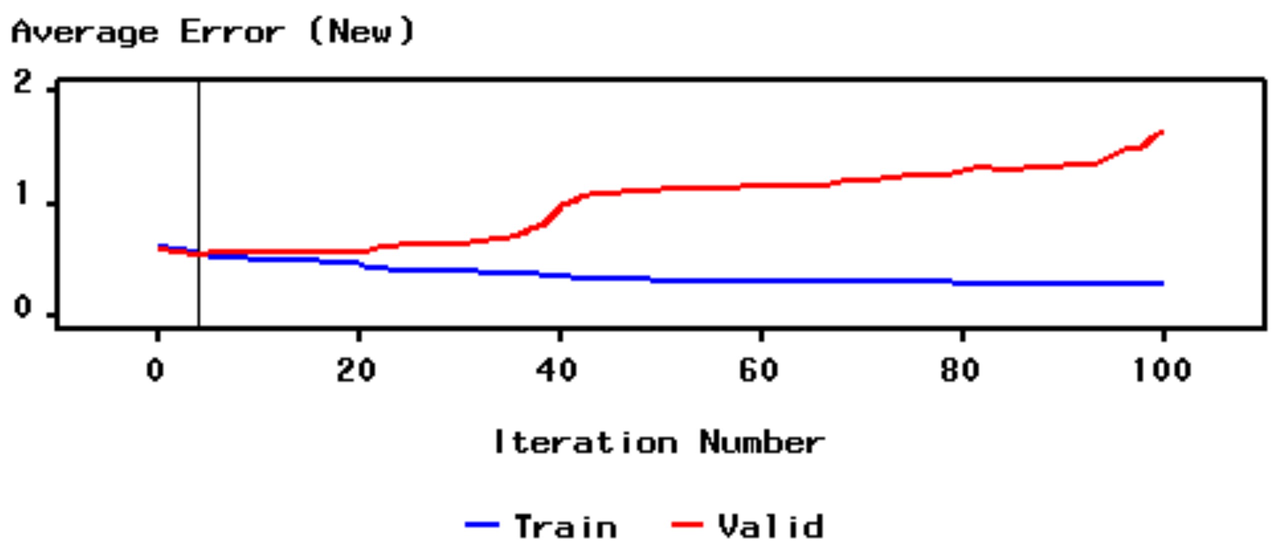
Fact: The "effective number of parameters" (or effective degrees of freedom) of an MLP increases with amount of training.

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

Adjusting Complexity

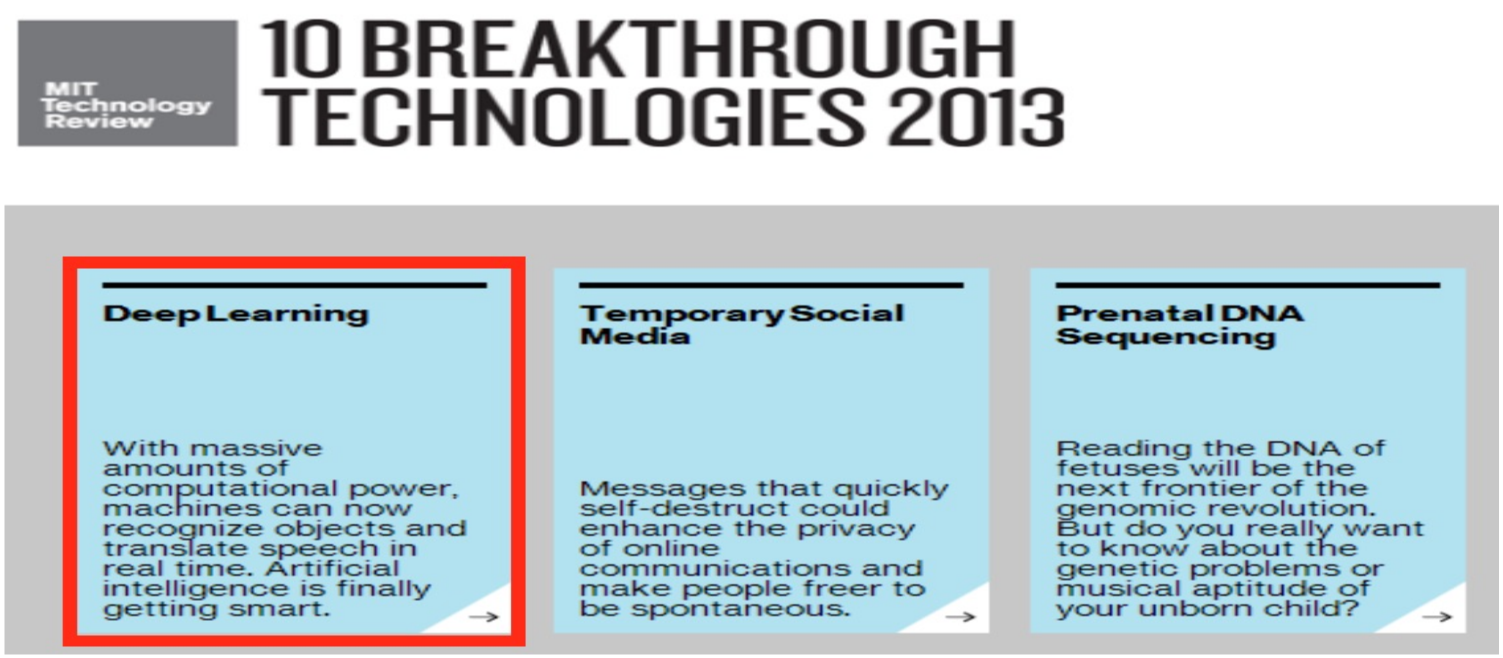
– "effective # of parameters" increases with # of epochs !!
  - Select adequately powerful model
  - while training, monitor performance using validation set
  - **stop training** when error on validation set reaches a minimum

  - 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 (2014)
https://www.cs.toronto.edu/~ranzato/publications/taigman_cvpr14.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.

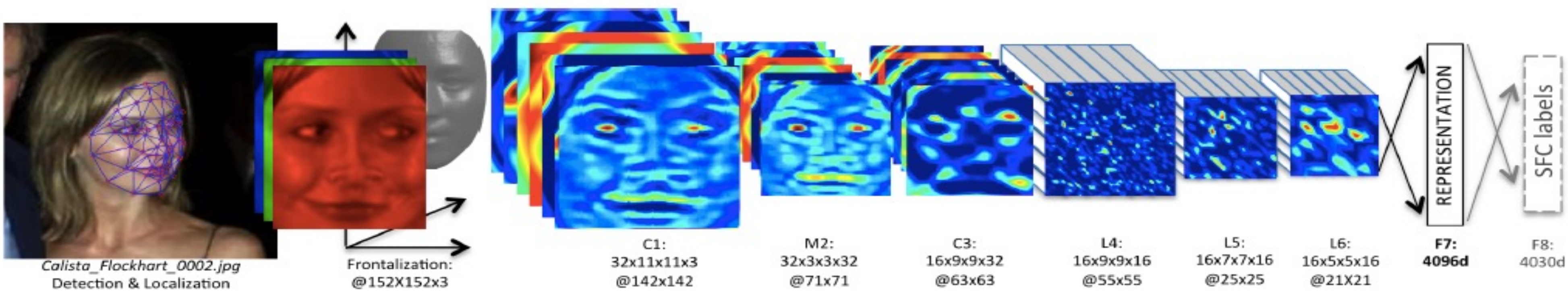
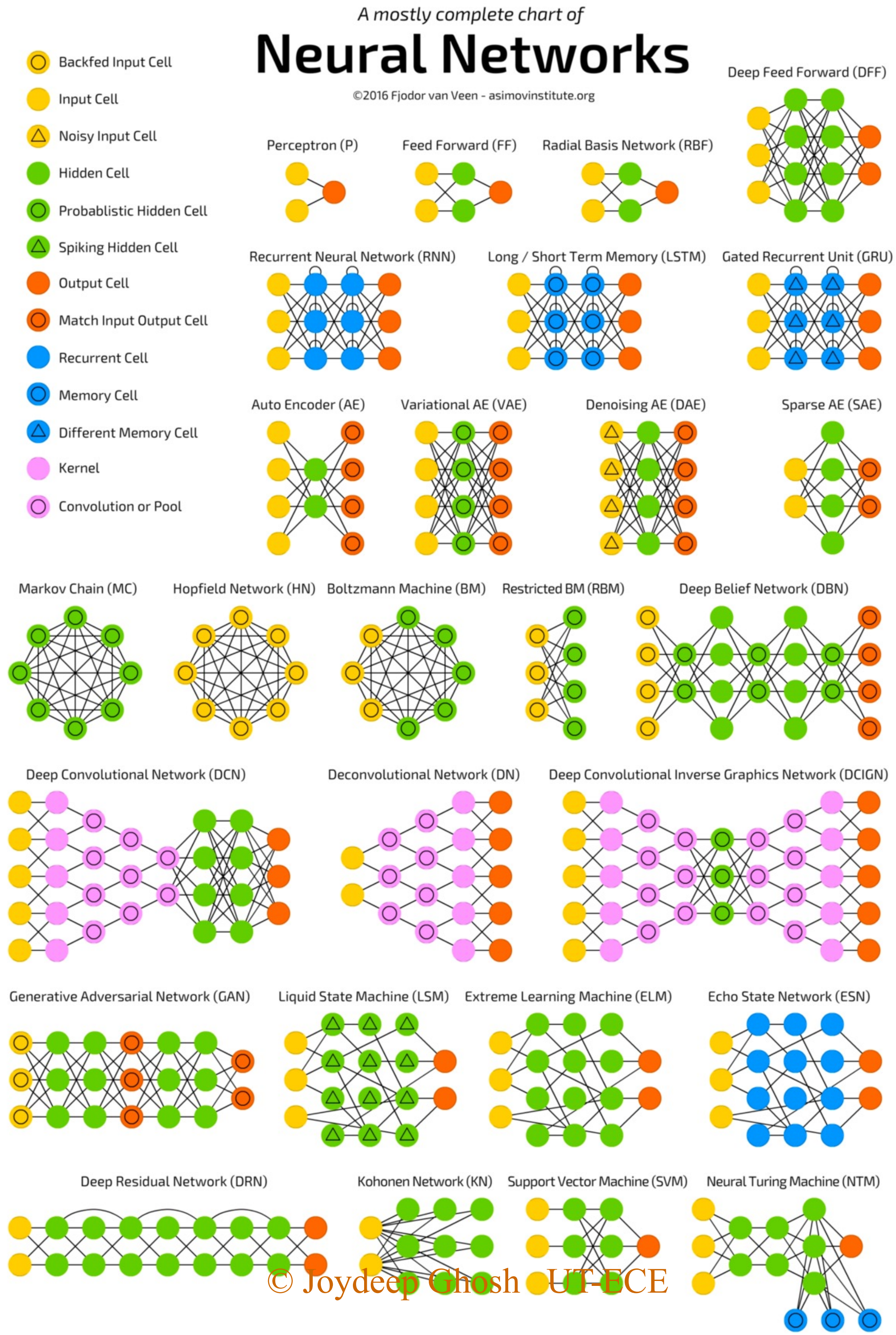


Figure 2. **Outline of the *DeepFace* architecture.** A front-end of a single convolution-pooling-convolution filtering on the rectified input, followed by three locally-connected layers and two fully-connected layers. Colors illustrate outputs for each layer. The net includes more than 120 million parameters,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



A mostly complete chart of
Neural Networks
©2016 Fjodor van Veen - asimovinstitute.org
Backfed Input Cell
Input Cell
Noisy Input Cell
Hidden Cell
Probablistic Hidden Cell
Spiking Hidden Cell
Output Cell
Match Input Output Cell
Recurrent Cell
Memory Cell
Different Memory Cell
Kernel
Convolution or Pool
Perceptron (P)
Feed Forward (FF)
Radial Basis Network (RBF)
Deep Feed Forward (DFF)
Recurrent Neural Network (RNN)
Long / Short Term Memory (LSTM)
Gated Recurrent Unit (GRU)
Auto Encoder (AE)
Variational AE (VAE)
Denoising AE (DAE)
Sparse AE (SAE)
Markov Chain (MC)
Hopfield Network (HN)
Boltzmann Machine (BM)
Restricted BM (RBM)
Deep Belief Network (DBN)
Deep Convolutional Network (DCN)
Deconvolutional Network (DN)
Deep Convolutional Inverse Graphics Network (DCIGN)
Generative Adversarial Network (GAN)
Liquid State Machine (LSM)
Extreme Learning Machine (ELM)
Echo State Network (ESN)
Deep Residual Network (DRN)
Kohonen Network (KN)
Support Vector Machine (SVM)
Neural Turing Machine (NTM)

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"

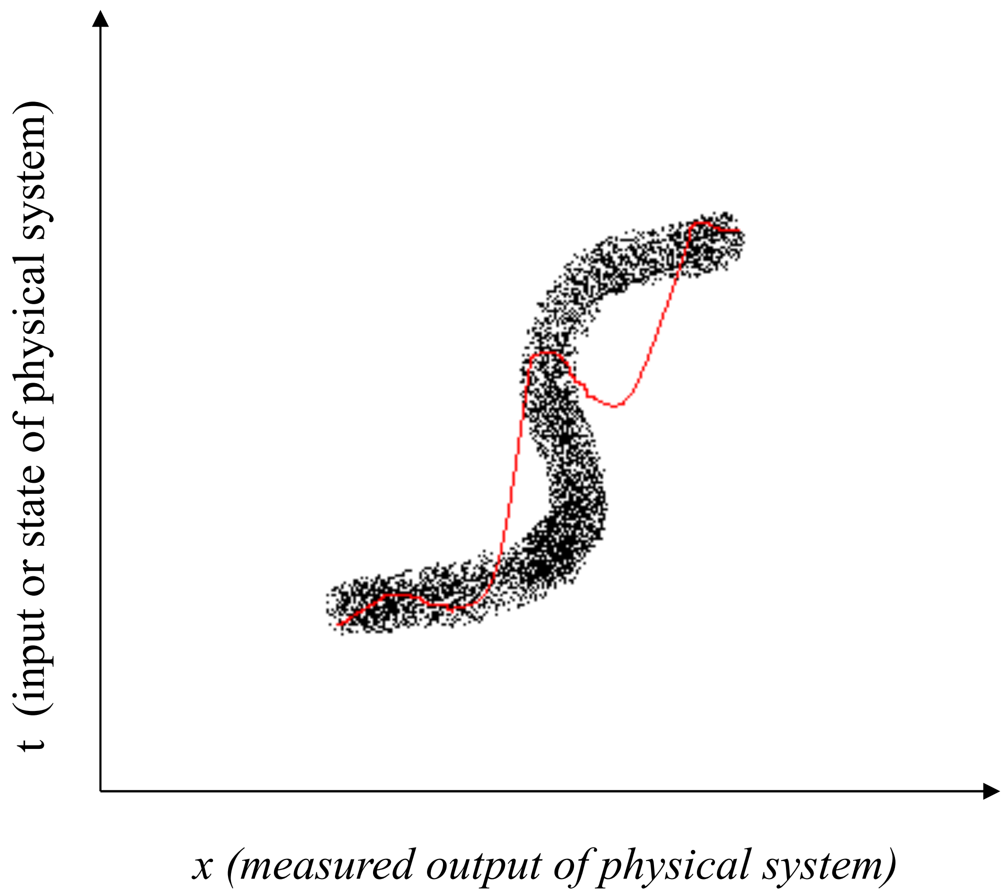
- Still lacking? try ensemble approaches specially GBDT/XGBoost, which also rank-orders the features.

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

- Solutions:
  - model joint pdf
  - build piecewise models
  - ….



Extras

(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…

Toy illustration for SGD

y = cost;

x is scalar parameter that can be adjusted

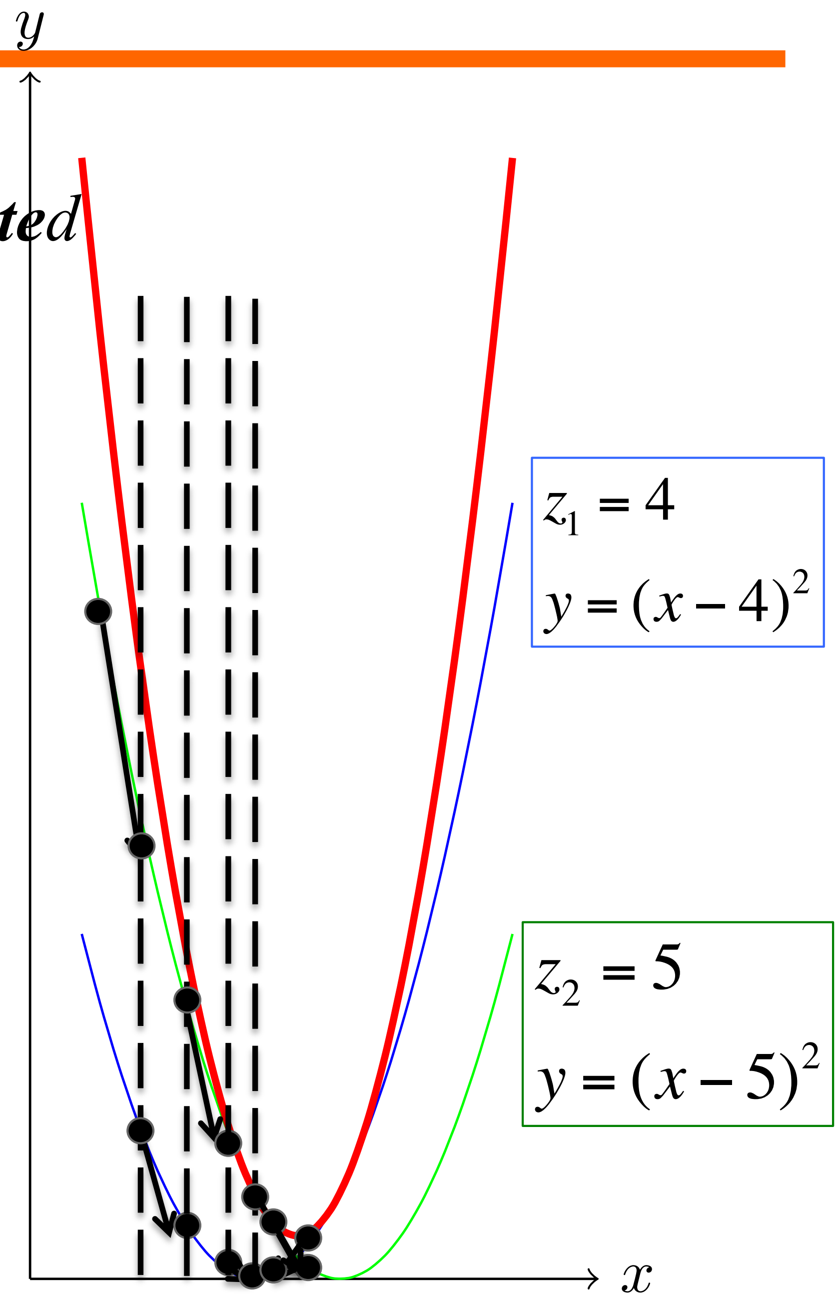
$$\boxed{y = (x-4)^2 + (x-5)^2}$$

$$\min_x \sum_i (x - z_i)^2 = \min_x \sum_i \ell(x, z_i)$$

$$\ell(x, z) = (x - z)^2$$

$$x^{t+1} = x^t - \eta \frac{\partial \ell(x^t, 4)}{\partial x}$$

$$x^{t+2} = x^{t+1} - \eta \frac{\partial \ell(x^{t+1}, 5)}{\partial x}$$



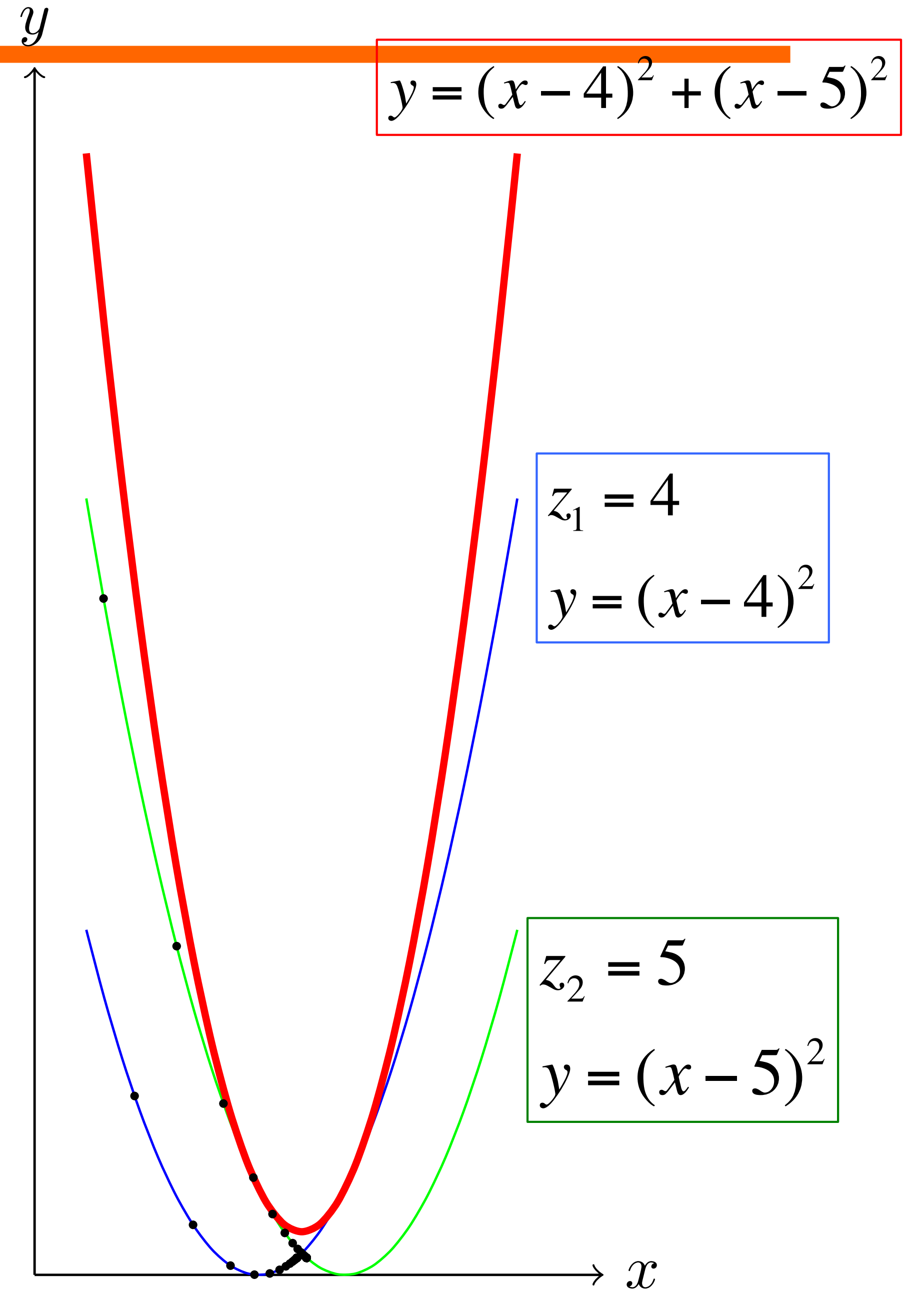
Toy illustration for SGD

$$\min_x \sum_i (x - z_i)^2 = \min_x \sum_i \ell(x, z_i)$$

$$\ell(x, z) = (x - z)^2$$

$$x^{t+1} = x^t - \eta \frac{\partial \ell(x^t, 4)}{\partial x}$$

$$x^{t+2} = x^{t+1} - \eta \frac{\partial \ell(x^{t+1}, 5)}{\partial x}$$



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

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