Cmput 466 / 551

# Artificial Neural Networks #2: Conjugate Gradient, …

Covering chapter (HTF) 11

+

"An Intro to Conjugate Gradient Method without Agonizing Pain"

R Greiner
Department of Computing Science
University of Alberta

Thanks: T Dietterich, R Parr, J Shewchuk

# Outline

- Introduction
  - Historical Motivation, non-LTU, Objective
  - Types of Structures
- Multi-layer Feed-Forward Networks
  - Sigmoid Unit
  - Backpropagation
- Tricks for Effectiveness
  - Efficiency: Conjugate Gradient, Line Search
  - Generalization: Alternative Error Functions
- Example: Face Recognition
- Hidden layer representations
- Towards Deeper Nets
- ~~Recurrent Networks~~

# Issues

Backprop will (at best)…

- … slowly …
  - Conjugate gradient
  - Line search, …
- … converge to LOCAL Opt …
  - Multiple restart
  - simulated annealing, …
- … wrt Training Data
  - Early stopping
  - regularization, …

# Gradient Descent

To optimize $J(\mathbf{w})$:
Initialize $\mathbf{w}^{(0)}$
For k = 1..$m$
$\mathbf{w}^{(k+1)} := \mathbf{w}^{(k)} + \alpha^{(k)}\, \mathbf{d}^{(k)}$

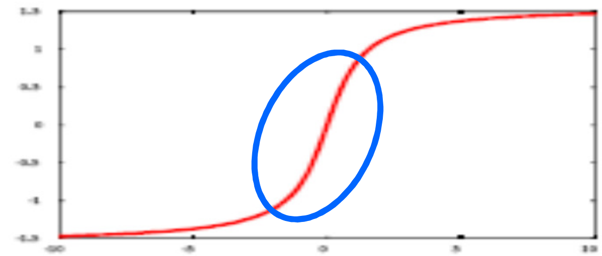
- General description:
  Want $\mathbf{w}^*$ that minimizes function $J(\mathbf{w})$
- So far. . .
  - $\mathbf{w}^{(0)}$ is random
  - $\alpha^{(k)} = 0.05$
  - $\mathbf{d}^{(k)} = \nabla J = \left\langle \frac{\partial J(\mathbf{w}^{(k)})}{\partial w_i^{(k)}} \right\rangle_i$ is derivative
  - $m$ = until bored...
- Alternatively...
  1. Use *small* random values for $\mathbf{w}^{(0)}$
  2. Use *conjugate gradient* for direction $\alpha^{(k)}$, $\mathbf{d}^{(k)}$
  3. Use *line search* for distance $\alpha^{(k)}$
  4. Use "cross tuning" for stopping criteria $m$
  5. Multiple restarts

Overfitting

Efficiency

Local Opt

# 1. Proper Initialization (**w**)



- Start in "linear regions"
  - Start all weights near 0,

    ⇒ sigmoid units in linear regions.

    ⇒ whole net ≈ one linear threshold unit

    ⇒ network ≈ linear in weights…

  so moves quickly...
  until in "correct region"

Specific for Sigmoid
and variants

- Break symmetry
  - Ensure each unit has different input weights
    (so hidden units move in different directions)
  - Set weight to random number in range

    $w_{i,j} \sim \text{Uniform}[-1,+1] \times \frac{1}{\sqrt{\text{Fan.In}}}$

# 2. Conjugate Gradient

- At step *r*, searching along direction (?gradient?) $\mathbf{d}^{(r)}$
  ... using $e(\alpha) = J(\,\mathbf{w}^{(r)} + \alpha\,\mathbf{d}^{(r)}\,)$
- At (local) minimum $\alpha^*$: $\boxed{\dfrac{\partial}{\partial\alpha} J\left(\mathbf{w}^{(r)} + \alpha\,\mathbf{d}^{(r)}\,\right) = 0}$
- Let $\mathbf{w}^{(r+1)} = \mathbf{w}^{(r)} + \alpha^*\,\mathbf{d}^{(r)}$
- $\dfrac{\partial}{\partial\alpha} J\left(\mathbf{w}^{(r)} + \alpha\,\mathbf{d}^{(r)}\,\right) = \nabla J(\mathbf{w}^{(r)} + \alpha\,\mathbf{d}^{(r)}\,)^T \cdot \dfrac{\partial}{\partial\alpha}\left(\mathbf{w}^{(r)} + \alpha\,\mathbf{d}^{(r)}\,\right)$

  $= \quad \nabla J(\mathbf{w}^{(r+1)})^T\,\mathbf{d}^{(r)} \qquad = 0$

# 2. Conjugate Gradient

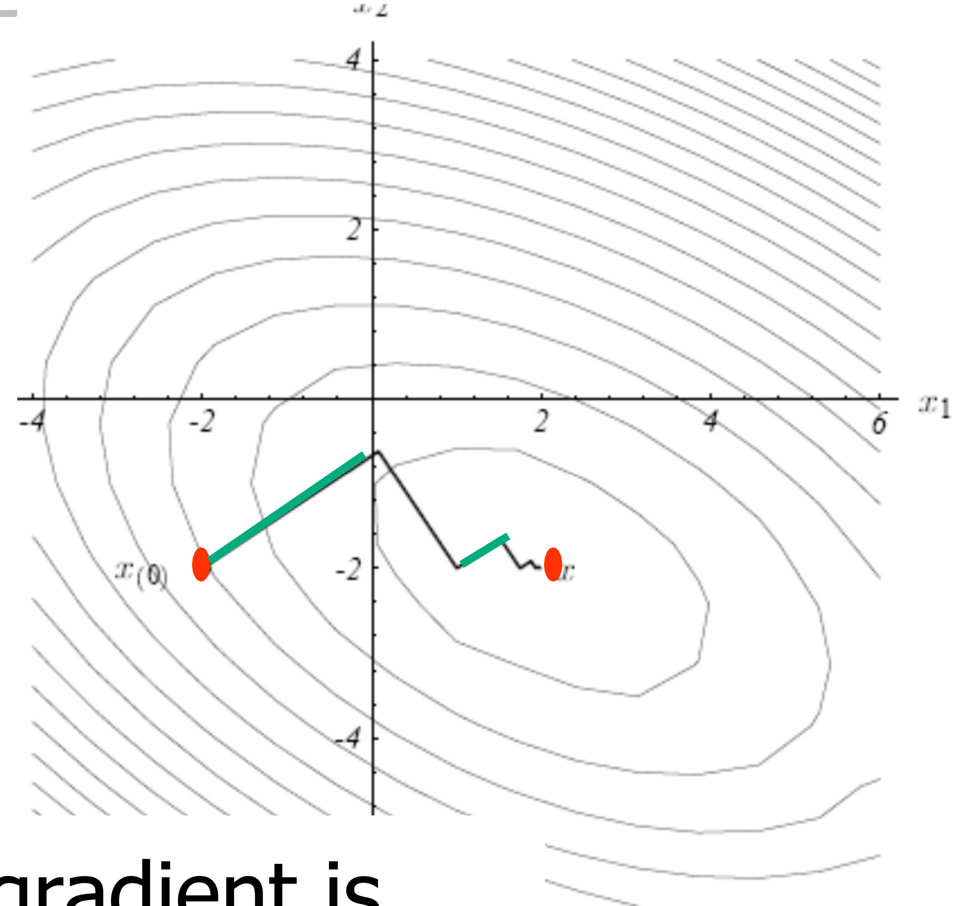
Jump

- At step *r*, searching along direction (?gradient?) $\mathbf{d}^{(r)}$ … using $q(\alpha) = J(\ \mathbf{w}^{(r)} + \alpha\ \mathbf{d}^{(r)}\ )$
- At (local) minimum $\alpha^*$: $\boxed{\frac{\partial}{\partial \alpha} J\left(\mathbf{w}^{(r)} + \alpha\, \mathbf{d}^{(r)}\ \right) = 0}$
- Let $\mathbf{w}^{(r+1)} = \mathbf{w}^{(r)} + \alpha^*\ \mathbf{d}^{(r)}$
  $\Rightarrow\quad \nabla J(\mathbf{w}^{(r+1)})^T\ \mathbf{d}^{(r)}\ = 0$
- Gradient $\nabla J(\mathbf{w}^{(r+1)})$ at $r+1^{st}$ step is ORTHOGONAL to previous search direction $\mathbf{d}^{(r)}$ !
- Is this the best direction??

"An Intro to Conjugate Gradient Method without Agonizing Pain"

# Problem with Steepest Descent

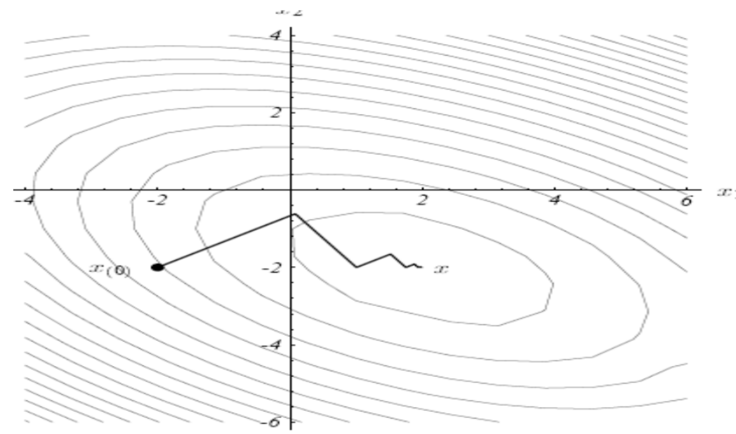
- Steepest Descent…
  from [-2,-2] to [2,-2]




- Path "zigzag"s as each gradient is orthogonal to the previous gradient…
  … but aligned with earlier gradients

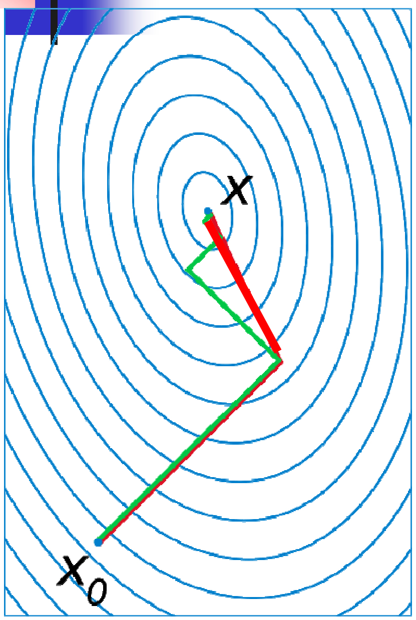
# Descend along Gradient?

- Q: Should we travel along Gradient?
- A: Not necessarily!

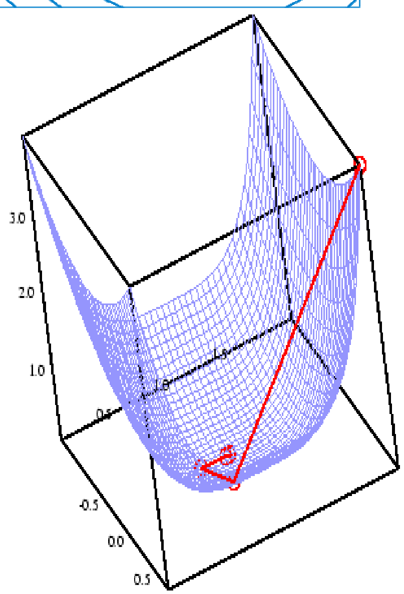


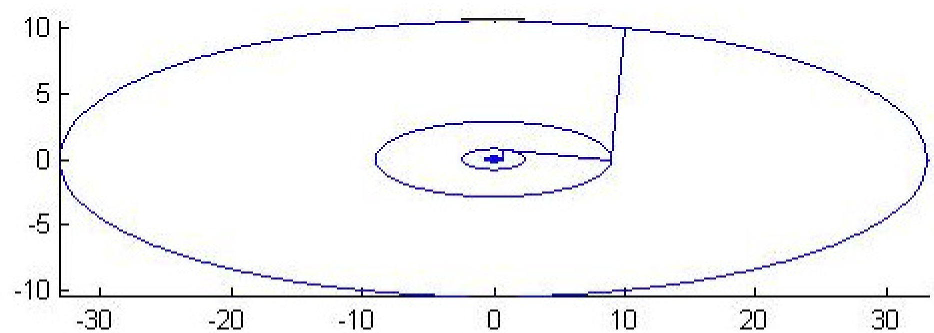
- If different curvatures along different axes: local negative gradient $-\nabla J$ will NOT point towards minimum !
- What to do?

# Is Gradient always most efficient?




- Each green line is gradient…
- Problematic when going down narrow canyon
- Red is better…



# Better…

$\mathbf{g}_r = \nabla J(\mathbf{w}_r)$

- Problem: Gradients { $\mathbf{g}_r$ } are NOT orthogonal to each other
  - so can "repeat" same directions
- Better to use other vector-directions { $\mathbf{d}_r$ } … where ∃ only n of them (dim of space)
  - "Conjugate":
    - Spanning
    - "Orthogonal" (wrt Hessian matrix)
- Then after n steps:
  must be at (local) optimum!!

# Conjugate Gradient Algorithm

- Notation… wrt iteration $j$
  - Weights: $\mathbf{w}_j$
  - Gradient: $\mathbf{g}_j = \nabla J(\mathbf{w}_j)$
  - Direction: $\mathbf{d}_j$
- Update parameters: $\mathbf{w}_{j+1} := \mathbf{w}_j + \alpha_j \mathbf{d}_j$
  - To find appropriate distance
  - To get DIRECTION $\mathbf{d}_j$
    - $\mathbf{d}_1 := -\mathbf{g}_1$
    - $\mathbf{d}_{j+1} := -\mathbf{g}_{j+1} + \beta_j \mathbf{d}_j$

$\alpha_j =$ tba

$\beta_j =$ tba

# Conjugate Gradient, IIa

$\mathbf{g}_r = \nabla J(\mathbf{w}_r) = \left[\frac{\partial J(\mathbf{w}^{(r)})}{\partial w_1}, \ldots, \frac{\partial J(\mathbf{w}^{(r)})}{\partial w_n}\right]$ is gradient, wrt $r^{th}$ iteration

- Let **d** be DIRECTION of change.

  Perhaps just use $\mathbf{d} = \mathbf{g}$ ? But ...
- On iteration $r$, by construction: $g(\mathbf{w}_{r+1})^T \mathbf{d}_r = 0$
- Want this to be true for next direction as well:

$$g(\mathbf{w}_{r+2})^T \mathbf{d}_r = 0$$

As

$$\mathbf{w}_{r+2} := \mathbf{w}_{r+1} + \alpha_{r+1} \mathbf{d}_{r+1}$$

need:

$$g(\mathbf{w}_{r+1} + \alpha_{r+1} \mathbf{d}_{r+1})^T \mathbf{d}_r = 0$$

# Conjugate Gradient, IIb

- First order Taylor expansion:

$$g(\mathbf{w}_{r+1} + \alpha_{r+1}\, \mathbf{d}_{r+1})^T$$
$$= g(\mathbf{w}_{r+1})^T + \alpha_{r+1} \mathbf{d}_{r+1}^T\, \nabla g(\mathbf{w}_{r+1} + \gamma\, \mathbf{d}_{r+1})$$

for some $\gamma \in (0, \alpha_{r+1})$

- Post-Multiply by $\mathbf{d}_r$ & use $g(\mathbf{w}_{r+1})^T\, \mathbf{d}_r = 0$ to get

$$\alpha_{r+1}\, \mathbf{d}_{r+1}^T\, \boxed{\nabla g(\mathbf{w}_{r+1} + \gamma\, \mathbf{d}_{r+1})}\, \mathbf{d}_r = 0$$

- Let $\mathcal{H}(\mathbf{w}) = \nabla g(\mathbf{w}) = \nabla(\nabla J(\mathbf{w}))$

… a $n \times n$ matrix of 2nd derivatives, evaluated at $\mathbf{w}$

# Hessian Matrix (Second Derivatives)

- Consider J(x, y) = $x^2$ +3xy – 5x
- $g(x,y) = \nabla J = \left[\frac{\partial J(x,y)}{\partial x}, \ \frac{\partial J(x,y)}{\partial y}\right] = [2x + 3y - 5, \ \ 3x]$
- $\mathcal{H} = \nabla\nabla J = \begin{bmatrix} \frac{\partial}{\partial x}\frac{\partial J(x,y)}{\partial x} & \frac{\partial}{\partial y}\frac{\partial J(x,y)}{\partial x} \\ \frac{\partial}{\partial x}\frac{\partial J(x,y)}{\partial y} & \frac{\partial}{\partial y}\frac{\partial J(x,y)}{\partial y} \end{bmatrix} = \begin{bmatrix} \frac{\partial}{\partial x}(2x+3y-5) & \frac{\partial}{\partial y}(2x+3y-5) \\ \frac{\partial}{\partial x}(3x) & \frac{\partial}{\partial y}(3x) \end{bmatrix}$

  $= \begin{bmatrix} 2 & 3 \\ 3 & 0 \end{bmatrix}$
- As J(x, y) is quadratic, $\mathcal{H}$ is constant

  If J(x, y) = $x^3y^2$ +… , then $\mathcal{H}$ is function of args
  - But we will assume is ≈constant (in neighborhood)...

# Conjugate Property

- Want: $\mathbf{d}_{r+1}{}^{T} \nabla g( \mathbf{w}_{r+1} + \gamma\, \mathbf{d}_{r+1} )\, \mathbf{d}_r = 0$

- Using $\mathcal{H}(\mathbf{w}_r) = \nabla g(\mathbf{w}_r) = \nabla( \nabla J(\mathbf{w}_r) )$

$$0 = \mathbf{d}_{r+1}{}^{T} \nabla g(\mathbf{w}_{r+1} + \gamma\, \mathbf{d}_{r+1} )\, \mathbf{d}_r$$
$$= \mathbf{d}_{r+1}{}^{T} \mathcal{H}(\mathbf{w}_{r+1} + \gamma\, \mathbf{d}_{r+1} )\, \mathbf{d}_r$$
$$\approx \mathbf{d}_{r+1}{}^{T} \mathcal{H}\, \mathbf{d}_r$$

- Challenge: How to find such $\mathbf{d}_r$ vectors?
- Assuming $J(\mathbf{w}) = J_0 + b^T\mathbf{w} + \frac{1}{2}\mathbf{w}^T \mathcal{H}\, \mathbf{w}$

  then $g(\mathbf{w}) = \nabla J(\mathbf{w}) = b + \mathcal{H}\, \mathbf{w}$
- $J$ is min at $\mathbf{w}^*$ s.t. $g(\mathbf{w}^*) = b + \mathcal{H}\, \mathbf{w}^* = 0$

# Conjugate Gradient, IV

- Spse ∃ n vectors "mutually conjugate wrt $\mathcal{H}$"

$$\mathbf{d}_j^T \mathcal{H} \mathbf{d}_i = 0 \quad \forall j \neq i$$

  Then $\{ \mathbf{d}_i \}$ linearly independent (if $\mathcal{H}$ pos def)
- Starting from $\mathbf{w}_1$; want minimum $\mathbf{w}^*$

  n dimensional space

  As $\{ \mathbf{d}_i \}$ spanning, $\mathbf{w}^* - \mathbf{w}_1 = \sum_{i=1}^{n} \alpha_i \, \mathbf{d}_i$
- As $\mathbf{w}_{j+1} = \mathbf{w}_j + \alpha_j \, \mathbf{d}_j$

$$\Rightarrow \mathbf{w}_j = \mathbf{w}_1 + \sum_{i=1}^{j-1} \alpha_i \, \mathbf{d}_i$$

- Series of steps,
  each parallel some conjugate direction,
  of magnitude $\alpha_j \in \Re$

# To find $\alpha_j$ ...

1. $g(\mathbf{w}_j) = \mathcal{H}\,\mathbf{w}_j + b$
2. $g(\mathbf{w}^*) = \mathbf{0}$
   $\Rightarrow \mathcal{H}\mathbf{w}^* = -\mathbf{b}$
3. $\mathbf{d}_j^T \mathcal{H}\,\mathbf{d}_i = 0$ if $i \neq j$

- To find value for $\alpha_j$ :
  - pre-multiply $\mathbf{w}^* - \mathbf{w}_1 = \sum_{i=1}^{n} \alpha_i\, \mathbf{d}_i$
  - by $\mathbf{d}_j^T \mathcal{H}$ :

$$\mathbf{d}_j^T(\mathcal{H}\,\mathbf{w}^* - \mathcal{H}\,\mathbf{w}_1) = \mathbf{d}_j^T\mathcal{H} \sum_{i=1}^{n} \alpha_i \mathbf{d}_j^T$$

$$\mathbf{d}_j^T(\boxed{-\mathbf{b}} - \mathcal{H}\,\mathbf{w}_1) = \sum_{i=1}^{n} \alpha_i \mathbf{d}_j^T\,\mathcal{H}\,\mathbf{d}_i = \alpha_j \boxed{\mathbf{d}_j^T\,\mathcal{H}\,\mathbf{d}_j}$$

#2 #3

$$\left(\begin{aligned} \mathbf{d}_j^T\mathcal{H}\,\mathbf{w}_j &= \mathbf{d}_j^T\mathcal{H}\,[\mathbf{w}_1 + \textstyle\sum_{i=1}^{j-1} \alpha_i\, \mathbf{d}_i\,] \\ &= \mathbf{d}_j^T\mathcal{H}\,\mathbf{w}_1 + \textstyle\sum_{i=1}^{j-1} \alpha_i \boxed{\mathbf{d}_j^T\mathcal{H}\,\mathbf{d}_i} = \mathbf{d}_j^T\mathcal{H}\,\mathbf{w}_1 \end{aligned}\right)$$

#3

$$\boxed{\alpha_j = -\frac{\mathbf{d}_j^T(\mathbf{b} + \mathcal{H}\mathbf{w}_1)}{\mathbf{d}_j^T\mathcal{H}\mathbf{d}_j} = -\frac{\mathbf{d}_j^T(\mathrm{b} + \mathcal{H}\mathbf{w}_j)}{\mathbf{d}_j^T\mathcal{H}\mathbf{d}_j} = -\frac{\mathbf{d}_j^T\boxed{\mathbf{g}_j}}{\mathbf{d}_j^T\mathcal{H}\mathbf{d}_j}}$$

#1

# Conjugate Gradient Algorithm

- Notation… wrt iteration j
  - Weights: $\mathbf{w}_j$
  - Gradient: $\mathbf{g}_j = \nabla J(\mathbf{w}_j)$
  - Direction: $\mathbf{d}_j$
- Update parameters: $\mathbf{w}_{j+1} := \mathbf{w}_j + \alpha_j \mathbf{d}_j$
  - To find appropriate distance

$$\alpha_j = -\frac{\mathbf{d}_j^T \boldsymbol{g}_j}{\mathbf{d}_j^T \mathcal{H}\, \mathbf{d}_j}$$

  - To get DIRECTION $\mathbf{d}_j$
    - $\mathbf{d}_1 := -\mathbf{g}_1$
    - $\mathbf{d}_{j+1} := -\mathbf{g}_{j+1} + \beta_j \mathbf{d}_j$

$\beta_j =$

tba

# Obtaining $\mathbf{d}_i$ from $\mathbf{g}_i$

- Given gradient $\mathbf{g}_{i+1}$
  let $\mathbf{d}_{i+1} := -\mathbf{g}_{i+1} + \beta_j\, \mathbf{d}_j$
- Find $\beta_j$ such that $\mathbf{d}_{j+1}^{\mathrm{T}} \mathcal{H}\, \mathbf{d}_j = 0$
  - $(-\mathbf{g}_{i+1} + \beta_j\, \mathbf{d}_j)^{\mathrm{T}}\, \mathcal{H}\, \mathbf{d}_j = 0$
  - $\mathbf{g}_{i+1}^{\mathrm{T}}\, \mathcal{H}\, \mathbf{d}_j = \beta_j\, \mathbf{d}_j^{T}\, \mathcal{H}\, \mathbf{d}_j$

$$\Rightarrow \quad \beta_j = \frac{\mathbf{g}_{j+1}^{\mathrm{T}}\, \mathcal{H}\, \mathbf{d}_j}{\mathbf{d}_j^{T}\, \mathcal{H}\, \mathbf{d}_j}$$

# Simpler version of $\beta_j = \frac{g_{j+1}{}^T \mathcal{H} \mathbf{d}_j}{\mathbf{d}_j{}^T \mathcal{H} \mathbf{d}_j}$

- Observe

$$\mathbf{g}_{j+1} - \mathbf{g}_j = [\mathcal{H}\, \mathbf{w}_{j+1} + b] - [\mathcal{H}\, \mathbf{w}_j + b]$$
$$= \mathcal{H}\, [\mathbf{w}_{j+1} - \mathbf{w}_j\,] \quad = \quad \mathcal{H}\, [\, \alpha_j\, \mathbf{d}_j] \quad = \; \alpha_j\, \mathcal{H}\, \mathbf{d}_j$$

- So… $\mathcal{H}\, \mathbf{d}_j \; = [\mathbf{g}_{j+1} - \mathbf{g}_j]/\, \alpha_j$

$$\beta_j = \frac{\mathbf{g}_{j+1}{}^T \mathcal{H} \mathbf{d}_j}{\mathbf{d}_j{}^T \mathcal{H} \mathbf{d}_j} = \frac{\mathbf{g}_{j+1}{}^T \left[\mathbf{g}_{j+1} - \mathbf{g}_j\;\right]/\alpha_j}{\mathbf{d}_j{}^T \left[\mathbf{g}_{j+1} - \mathbf{g}_j\;\right]/\alpha_j} = \frac{\mathbf{g}_{j+1}{}^T \left[\mathbf{g}_{j+1} - \mathbf{g}_j\;\right]}{\mathbf{d}_j{}^T \left[\mathbf{g}_{j+1} - \mathbf{g}_j\;\right]}$$

- "Hestenes-Stiefel" Version

# Alternative Version

1. $d_j^T g_{j+1} = 0$
2. $\mathbf{d}_j = -g_j + \beta_{j-1} \mathbf{d}_{j-1}$

- Consider DENOMINATOR: $d_j^T [\mathbf{g}_{j+1} - \mathbf{g}_j]$

$$
\begin{aligned}
\mathbf{d}_j^T [\mathbf{g}_{j+1} - \mathrm{g}_j] &= \mathbf{d}_j^T \mathbf{g}_{j+1} - \mathbf{d}_j^T \mathbf{g}_j \\
&= 0 - \left(-\mathbf{g}_j + \beta_{j-1} \mathbf{d}_{j-1}\right)^T \mathbf{g}_j \\
&= \mathrm{g}_j{}^T \mathrm{g}_j - \beta_{j-1}{}^T (\mathbf{d}_{j-1} \mathbf{g}_j) \\
&= \mathrm{g}_j^T \mathrm{g}_j
\end{aligned}
$$

- $$\beta_j = \frac{\mathbf{g}_{j+1}{}^T [\mathbf{g}_{j+1} - \mathbf{g}_j]}{\mathbf{d}_j^T [\mathbf{g}_{j+1} - \mathrm{g}_j]} = \frac{\mathbf{g}_{j+1}^T [\mathbf{g}_{j+1} - \mathbf{g}_j]}{\mathbf{g}_j^T \mathbf{g}_j}$$

- Polak-Ribiere version

# Computing Actual Direction $\mathbf{d}_j$

- $\mathbf{d}_{j+1} := -\mathbf{g}_{j+1} + \beta_j \mathbf{d}_j$ where $\beta_j = \dfrac{\mathbf{g}_{j+1}^T \mathcal{H} \mathbf{d}_j}{\mathbf{d}_j^T \mathcal{H} \mathbf{d}_j}$
- Assuming $\mathbf{J}$ is quadratic...
  - Hestenes-Stiefel: $\boxed{\beta_j = \dfrac{\mathbf{g}_{j+1}^T [\mathbf{g}_{j+1} - \mathbf{g}_j]}{\mathbf{d}_j^T [\mathbf{g}_{j+1} - \mathbf{g}_j]}}$
  - Polak-Ribiere: $\boxed{\beta_j = \dfrac{\mathbf{g}_{j+1}^T [\mathbf{g}_{j+1} - \mathbf{g}_j]}{\mathbf{g}_j^T \mathbf{g}_j}}$
  - Fletcher-Reeves: $\boxed{\beta_j = \dfrac{\mathbf{g}_{j+1}^T \mathbf{g}_{j+1}}{\mathbf{g}_j^T \mathbf{g}_j}}$
- If $J$ is NOT quadratic, Polak-Ribiere seems best
  [If gradients similar, $\beta \approx 0$, so $\approx$restarting!]

# Conjugate Gradient Algorithm

- Notation… wrt iteration $j$
  - Weights: $\mathbf{w}_j$
  - Gradient: $\mathbf{g}_j = \nabla J(\mathbf{w}_j)$
  - Direction: $\mathbf{d}_j$

Want $\mathbf{w}^* = \min_{\mathbf{w}} J(\mathbf{w})$

- Update parameters: $\mathbf{w}_{j+1} := \mathbf{w}_j + \alpha_j \mathbf{d}_j$
  - To find appropriate distance

    $$\alpha_j = -\frac{\mathbf{d}_j^T \mathbf{g}_j}{\mathbf{d}_j^T \mathcal{H} \mathbf{d}_j}$$

  - To get DIRECTION $\mathbf{d}_j$
    - $\mathbf{d}_1 := -\mathbf{g}_1$
    - $\mathbf{d}_{j+1} := -\mathbf{g}_{j+1} + \beta_j \mathbf{d}_j$

    $$\beta_j = \frac{\mathbf{g}_{j+1}^T [\mathbf{g}_{j+1} - \mathbf{g}_j]}{\mathbf{g}_j^T \mathbf{g}_j}$$

- If $\mathbf{J}$ quadratic, converge in $n$ steps!
  If not… sometimes reset: $\mathbf{d}_t := -\mathbf{g}_t$
- Compuational cost
  - Do not need to compute Hessian $\mathcal{H}$ for $\beta_j$ …
  - But… need $\mathcal{H}$ for $\alpha_j$

# Problem with $\alpha_j$

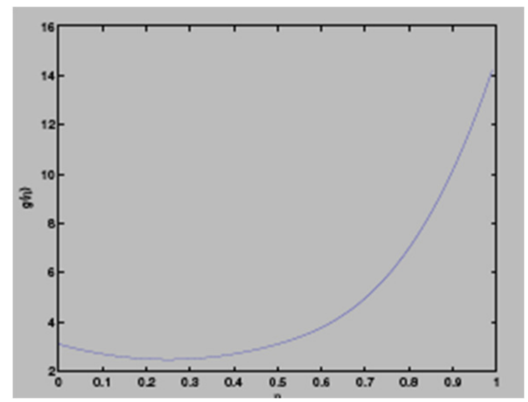
$$\alpha_j = -\frac{\mathbf{d}_j^T \boldsymbol{g}_j}{\mathbf{d}_j^T \boldsymbol{\mathcal{H}}\, \mathbf{d}_j}$$

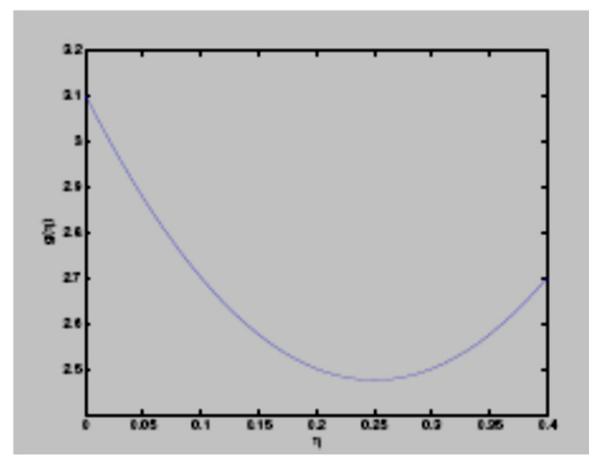
- $\alpha_j$ has closed form!
- But … requires $\mathcal{H}$
- Size: $\mathcal{H}$ is $n \times n$
  - So if $n = 1{,}000$, $\mathcal{H}$ has $\binom{1000}{2} \approx 500{,}000$ entries !
  - … if $n = 1{,}000{,}000$ …

  Challenging even if given $J(\cdot)$ … analytical
- What if need to estimate $\mathcal{H}$ empirically ??
  - … lots of samples …

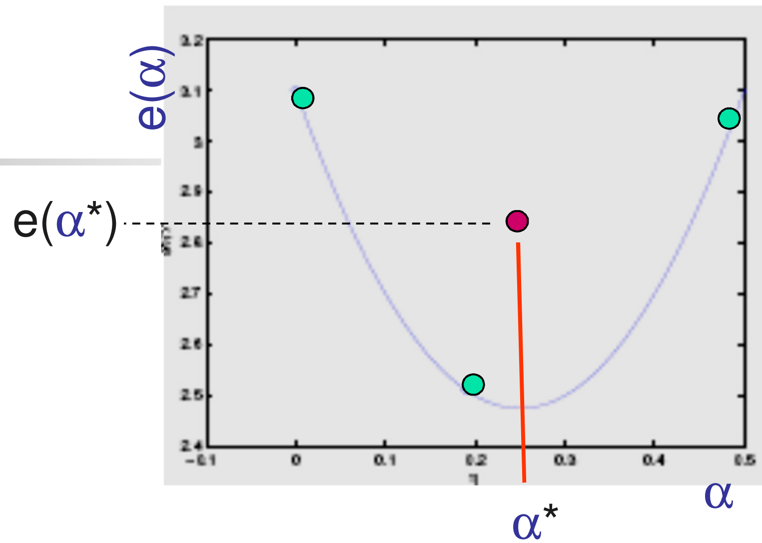
# 3. Line Search

- **Task**: Seek $\mathbf{w}$ that minimize $J(\mathbf{w})$
- Approach: Given direction $\mathbf{d} \in \Re^n$
  - New value $\mathbf{w}' := \mathbf{w} + \alpha\, \mathbf{d}$
  - But what value of $\alpha$?
- Good news: $\alpha \in \Re \Rightarrow$ 1 dim search!
- Let $e(\alpha) = J(\ \mathbf{w} + \alpha\, \mathbf{d}\ )$
  Want $\alpha^* = \text{argmin}_\eta \{\ e(\alpha)\ \}$
- **Line Search**:
  Near 0, $e(\alpha) \approx$ quadratic





# Line Search, con't



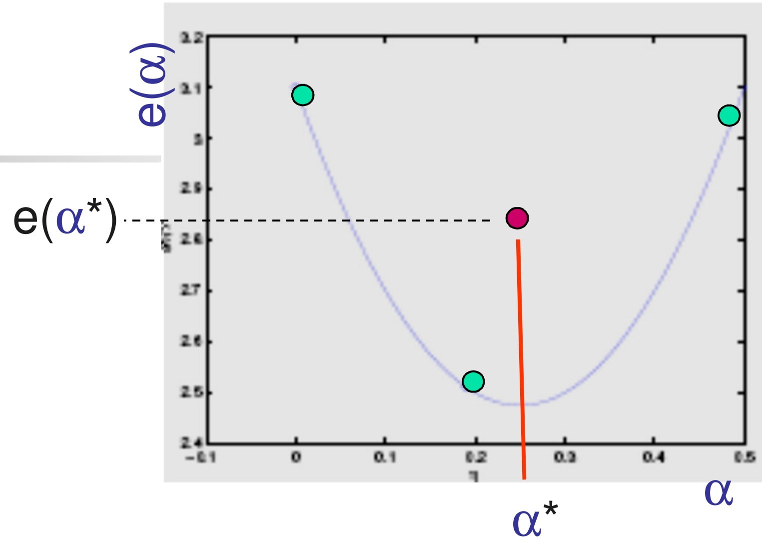

- Find 3 values s.t.
  - $\alpha_A < \alpha_B < \alpha_C$
  - $e(\alpha_A), e(\alpha_C) > e(\alpha_B)$
- Fit 2-D poly to

  $[\alpha_A, e(\alpha_A)], \ [\alpha_B, e(\alpha_B)], \ [\alpha_C, e(\alpha_C)]$

  $$\boxed{h_{\{A,B,C\}}(\alpha) = h(\alpha) = r\,\alpha^2 + s\,\alpha + t}$$
- Take min of this $h(\cdot)$ poly…

  $\Rightarrow \ \alpha^* = \operatorname{argmin}_\alpha h(\alpha)$
- Compute $e(\alpha^*)$
  - Compare $e(\alpha^*)$ to $h(\alpha^*)$ …

# Line Search, con't




- If $e(\alpha^*) \approx h(\alpha^*)$
  - Stop: found opt!
- Else:
  - Find 3 new points:
    - Note $\alpha_A \leq \alpha^* \leq \alpha_C$
    - Compare $\alpha^*$ to $\alpha_B$
      Compare $e(\alpha^*)$ to $e(\alpha_B)$
  - $\langle \alpha'_A, \alpha'_B, \alpha'_C \rangle :=$

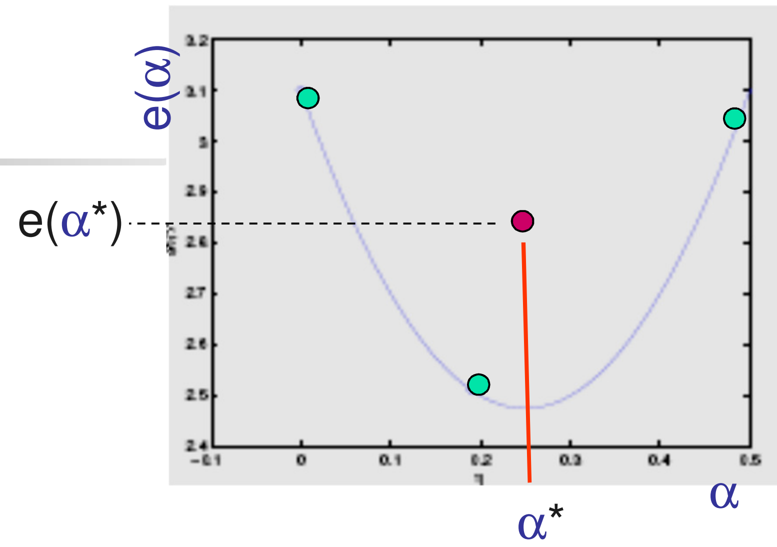
    $\langle \alpha^*, \alpha_B, \alpha_C \rangle$ if $\alpha^* < \alpha_B$ & $e(\alpha^*) > e(\alpha_B)$

    $\langle \alpha_A, \alpha^*, \alpha_C \rangle$ if $\alpha^* < \alpha_B$ & $e(\alpha^*) < e(\alpha_B)$

    $\langle \alpha_B, \alpha^*, \alpha_C \rangle$ if $\alpha^* > \alpha_B$ & $e(\alpha^*) < e(\alpha_B)$

    $\langle \alpha_A, \alpha_B, \alpha^* \rangle$ if $\alpha^* > \alpha_B$ & $e(\alpha^*) > e(\alpha_B)$
  - Recur

# Line Search, III




- This is for ONE ITERATION of general search

  Search can involve m iterations,

  Each iteration may involve 10's of eval's to get $\alpha^*$
- Issues:
  - How to find first 3 values?
  - Many other tricks... (Brent's Method)
  - Given assumptions, ANALYTIC form

# Issues

Backprop will (at best)…

- … slowly …
  - Conjugate gradient
  - Line search, …
- … converge to LOCAL Opt …
  - Multiple restarts
  - Simulated annealing, …
- … wrt Training Data
  - Early stopping
  - Regularization, …

# Local ≠ Global Optimum

- Techniques so far: Seek **LOCAL** minimal
- For Linear Separators: PERFECT
  ∃ 1 minimum
  ... if everything nearby looks "bad" ⇒ Done!
- Not true in general!

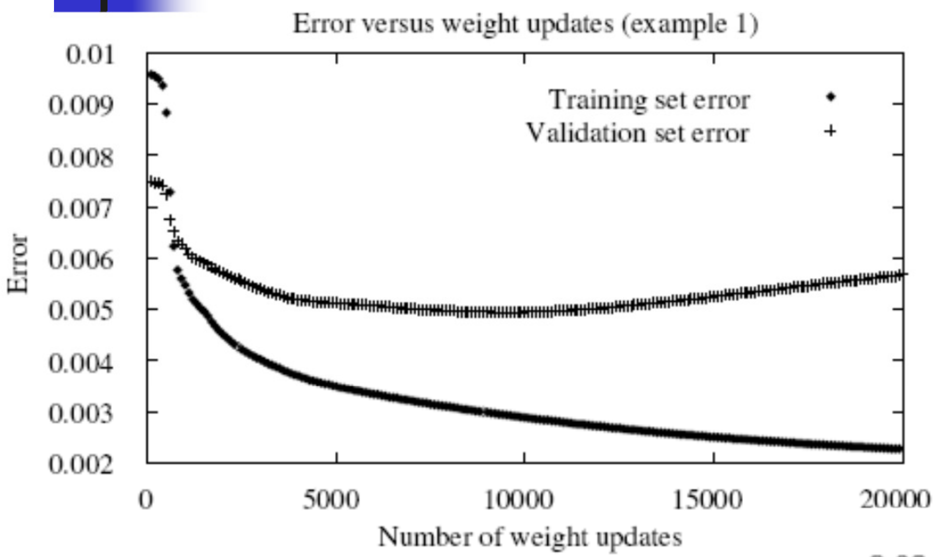
- Multiple Restarts
- Simulated Annealing
  Go wrong-way sometimes ...
  with diminishing probabilities

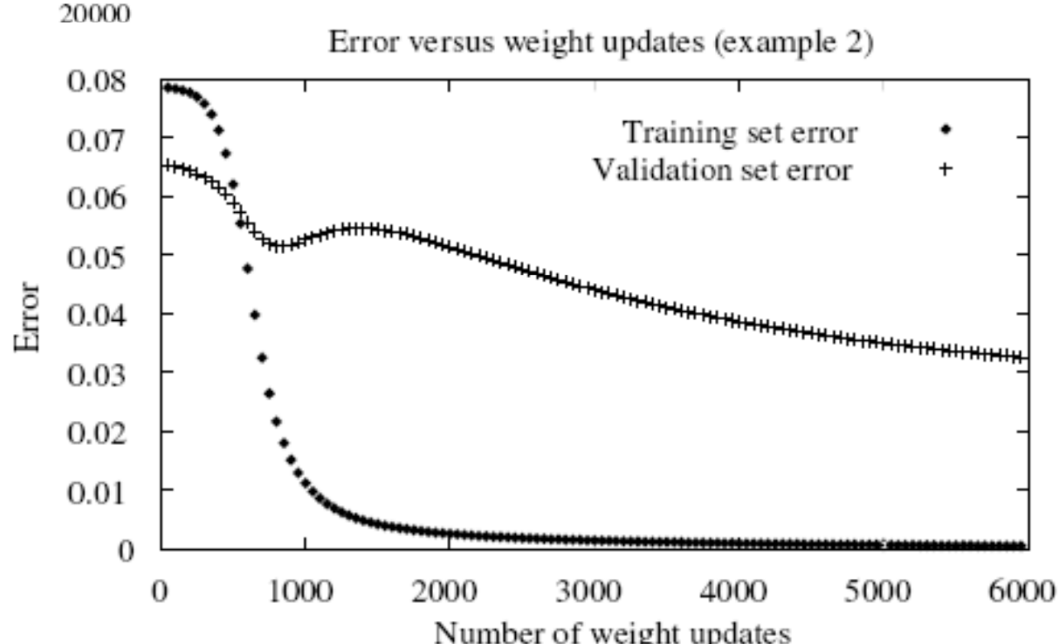
# Issues

Backprop will (at best)…

- … slowly …
  - Conjugate gradient
  - Line search, …
- … converge to LOCAL Opt …
  - Multiple restarts
  - Simulated annealing, …
- … wrt Training Data
  - Early stopping
  - Regularization, …

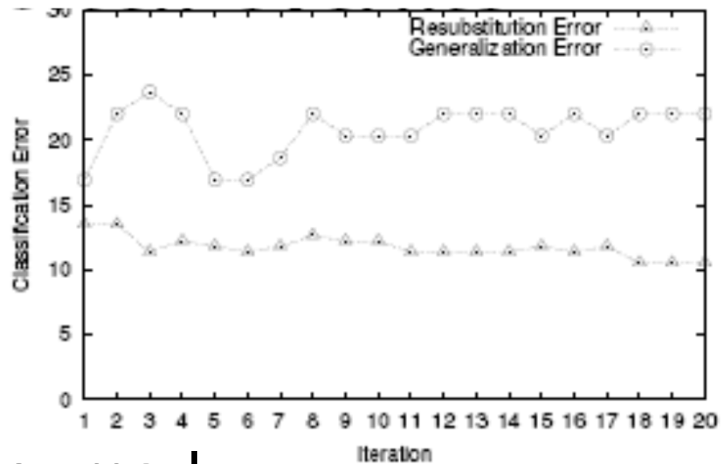
# Overfitting in ANNs



# When to Stop?

- After R iterations? (for fixed R)
  *?? What value of R?*
- When resubstitution error is suff. small?
  *No: often overfits*




- Use "validation data set"
  1. Do many iterations,
     then use weights from high-water mark
  2. Cross validation:
     Plot # iterations vs error $\rightarrow$ opt = $r_i$
     Let $\underline{r}$ =median($r_i$)
     Use all data, for $\underline{r}$ iterations

# Regularized Error Functions

- Penalize large weights: "Regularizing" … "weight decay"

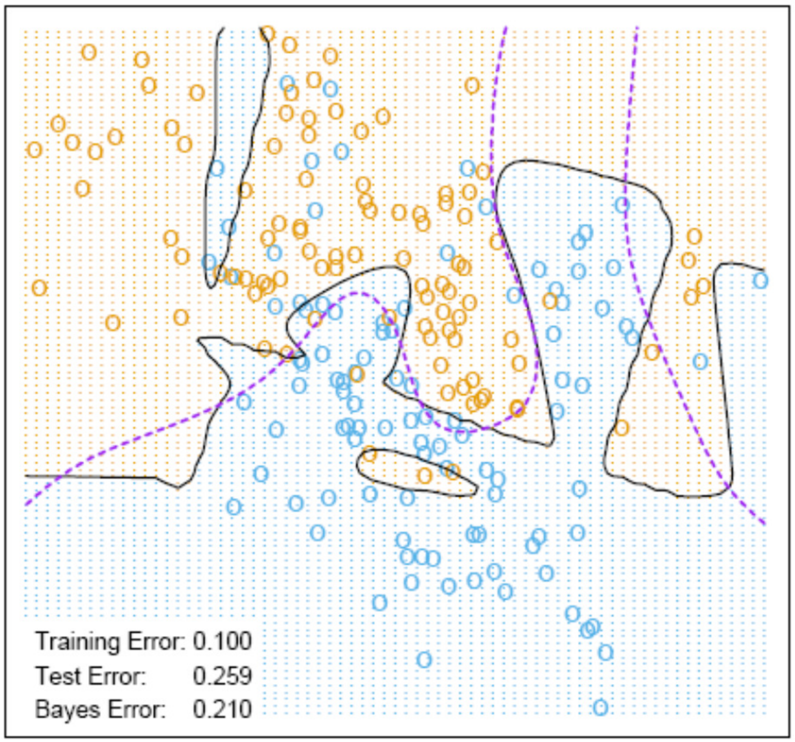
$$E(\mathbf{w}) = \frac{1}{2}\sum_{d\in D}\sum_{k\in outputs}(t_{kd}-o_{kd})^2 + \gamma\sum_{i,j} w_{ij}^2$$

or …

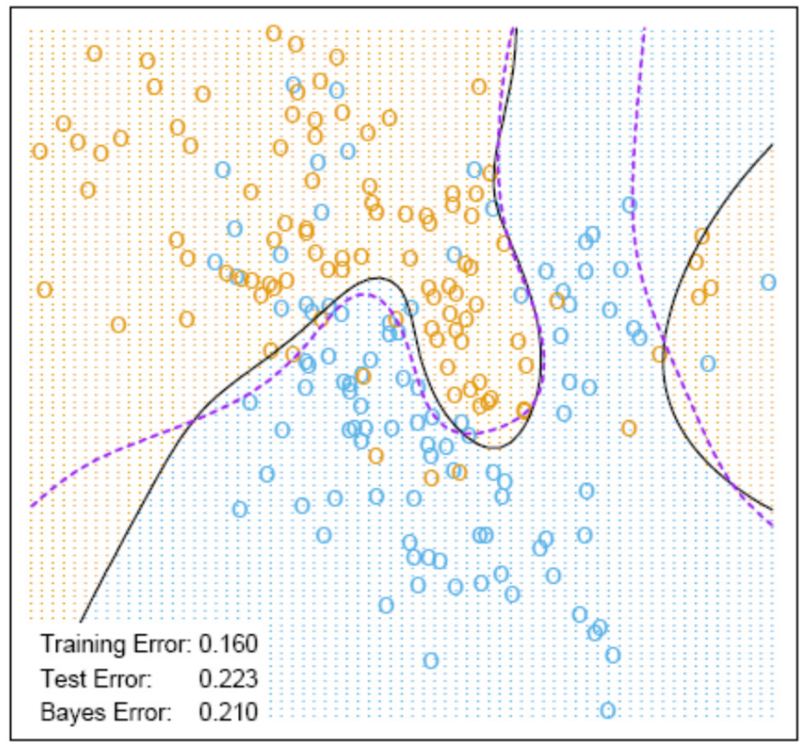
$$E(\mathbf{w}) = \frac{1}{2}\sum_{d\in D}\sum_{k\in outputs}(t_{kd}-o_{kd})^2 + \gamma\sum_{i,j}\frac{w_{ij}^2}{1+w_{ij}^2}$$

- ≈ ridge regression

# Example




No Weight Decay




Weight Decay=0.02

Neural Network - 10 Units

# Other Ideas

- Train on target slopes as well as values:
  (more constraints…)

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} \sum_{k \in outputs} \left[ (t_{kd} - o_{kd})^2 + \mu \sum_{j \in inputs} \left( \frac{\partial t_{kd}}{\partial x_d^j} - \frac{\partial o_{kd}}{\partial x_d^j} \right)^2 \right]$$

- Tie together weights:
  – eg, in phoneme recognition network
  (Fewer weights, …)
- Change structure

# Dynamically Modifying Network Structure

- So far, assume structure FIXED..
  ... only learning values of WEIGHTS
- Why not modify structure as well?

"Cascade Correlation"
1. Initially: NO hidden units
   ... just direct connections from input-output
2. Find best weights for this structure
3. If good fit: STOP.
   Otherwise... if significant residual error:
4. Produce new hidden unit
   from previous units,
   - connect to all output units
     w/weights CORRELATED
     with residual error

   Goto 2

"Optimal Brain Damage"
start w/ complex network,
prune "inessential" connections
Inessential if $w_{ij} \approx 0$
Remove node if all outboud $\approx 0$

… Deep Nets …

# Outline

- Introduction
  - Historical Motivation, non-LTU, Objective
  - Types of Structures
- Multi-layer Feed-Forward Networks
  - Sigmoid Unit
  - Backpropagation
- Tricks for Effectiveness
  - Efficiency: Conjugate Gradient, Line Search
  - Generalization: Alternative Error Functions
- Example: Face Recognition
- Hidden layer representations
- Towards Deeper Nets
- ~~Recurrent Networks~~

Skip …

# Neural Nets for Face Recognition

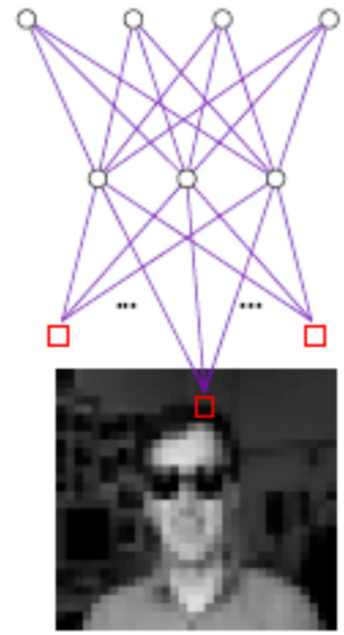
- **Performance Task:** Recognize DIRECTION of face
- **Framework**: Different people, poses, “glasses”, different background, . . .
- **Design Decisions:**
  - **Input Encoding:**
    - Just pixels? (subsampled? averaged?)
    - or perhaps lines/edges?
  - **Output Encoding:**
    - Single output ([0, 1/n] = #1, ... )
    - Set-of-n outputs (take highest value)
  - **Network structure**:
    - # of layers
    - How connected?
  - **Learning Parameters:** Stochastic?
    - Initial values of weights?
    - Learning rate $\eta$, Momentum $\alpha$, . . .
    - Size of Validation Set, . . .

# Neural Nets Used

left strt rght up

Typical input images

left strt rght up

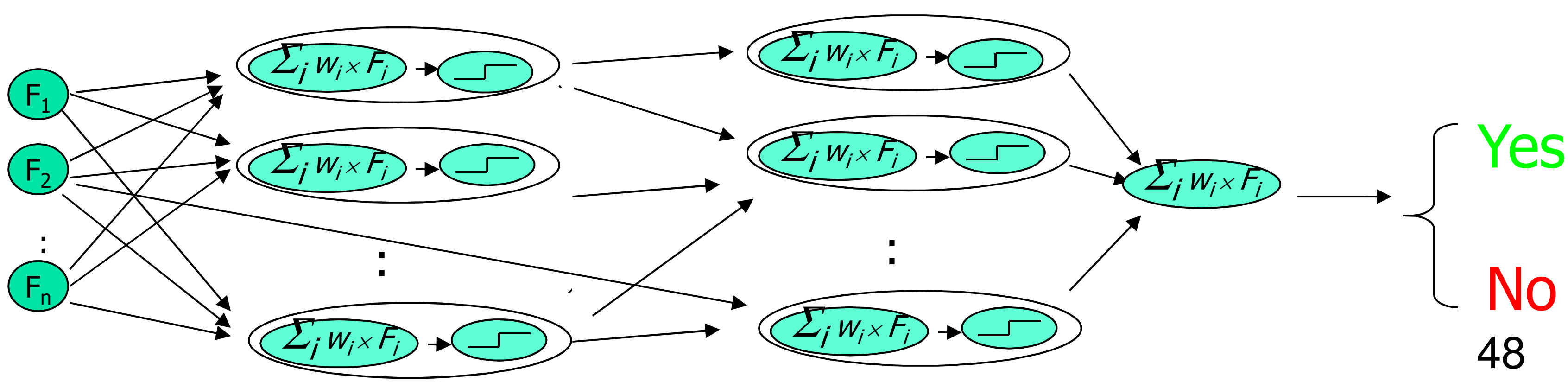


90% accurate learning head pose, and recognizing 1-of-20 faces

Jump

# Deep Learning

- Observation:
  - "Which Features" is more important than "Which Learner"
  - So… spend time finding (or generating) useful features !
- For k-layer Neural Net:
  - Think of first k-1 layers as LEARNING features
  - … as (complex) combinations of input variables

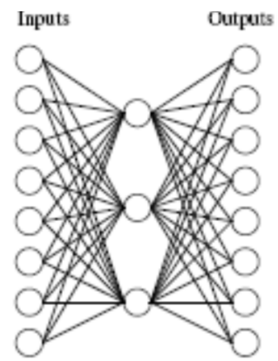


# Deep Learning ⇒ AutoEncoders

- Backprop (esp with Conjugate Gradient) works well for SHALLOW networks
  - 1 or 2 hidden layers
- But (in practice) shallow nets have limited expressibility
  - Features are not sufficiently rich
- So want DEEPER networks
- But "signal" for modifying weights does not "propagate" for deeper layers
- Need other tricks …
  - for finding features … autoencoders
  - initializing the weights

# Learning Hidden Layer Repr'n

- Auto-encoder:



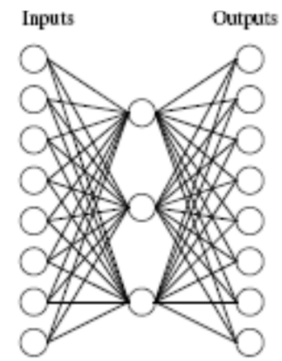

- Goal: Learn

| Input | | Output |
|---|---|---|
| 10000000 | → | 10000000 |
| 01000000 | → | 01000000 |
| 00100000 | → | 00100000 |
| 00010000 | → | 00010000 |
| 00001000 | → | 00001000 |
| 00000100 | → | 00000100 |
| 00000010 | → | 00000010 |
| 00000001 | → | 00000001 |

- Need to COMPRESS Data!

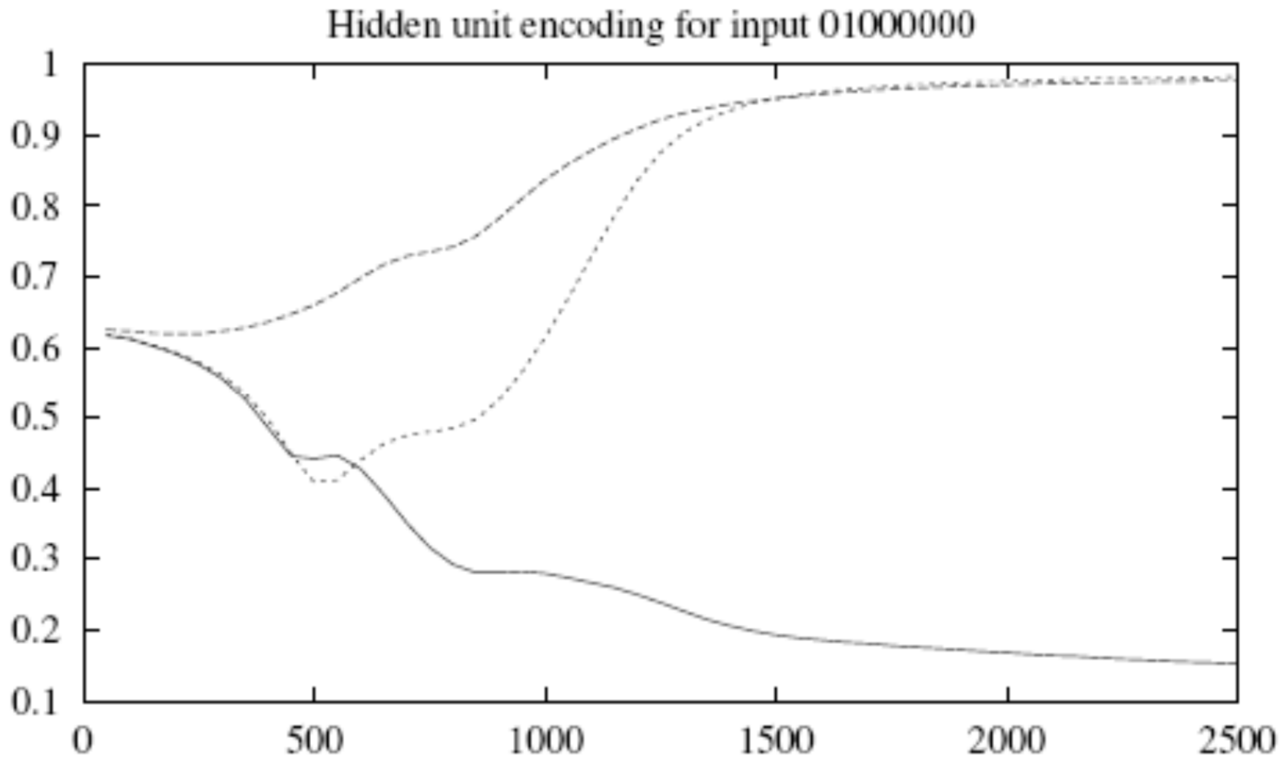
# Hidden Layer Representations

- Learned hidden layer representation:

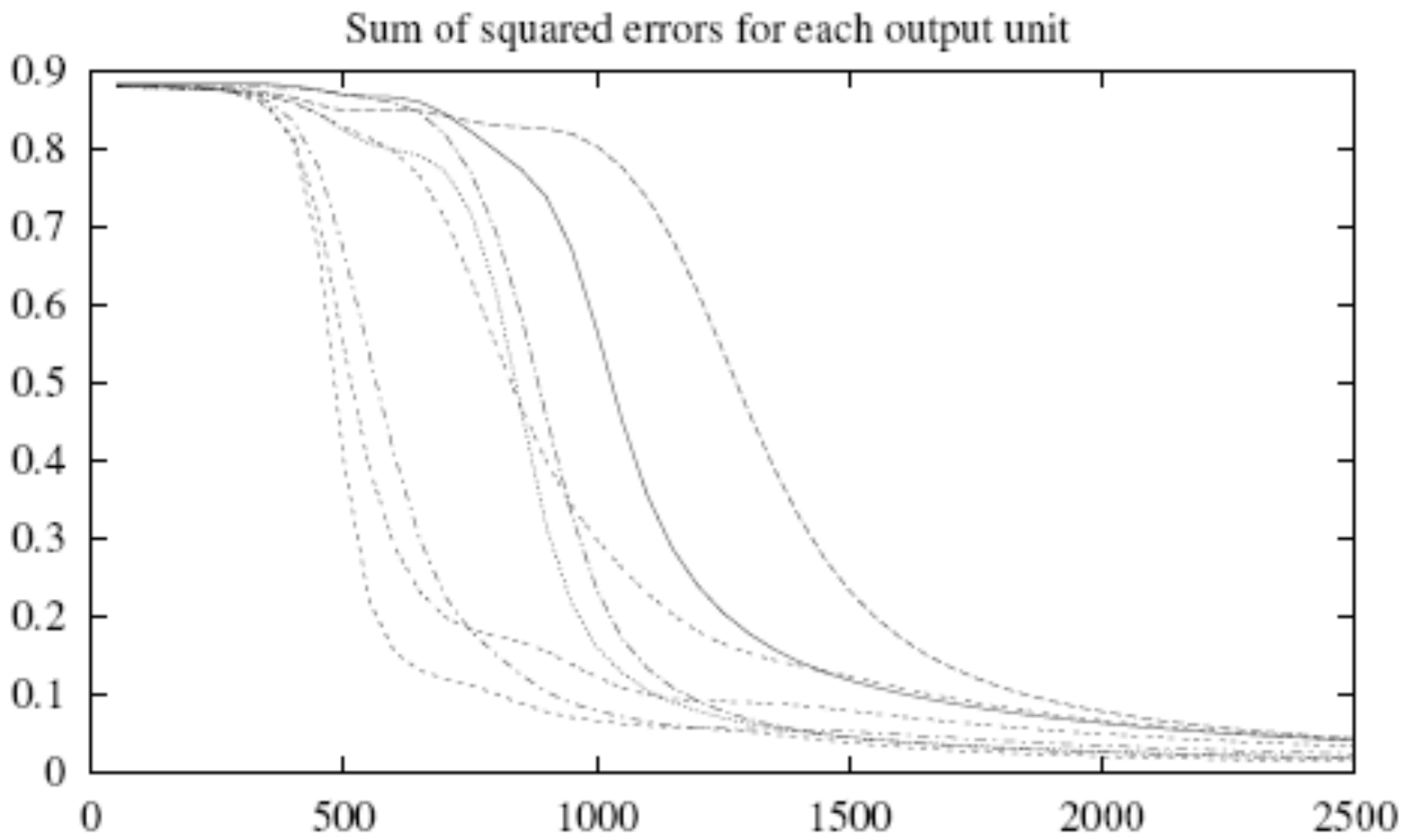



| Input | | Hidden Values | | | | Output |
|---|---|---|---|---|---|---|
| 10000000 | → | 1 | 0 | 0 | → | 10000000 |
| 01000000 | → | 0 | 0 | 1 | → | 01000000 |
| 00100000 | → | 0 | 1 | 0 | → | 00100000 |
| 00010000 | → | 1 | 1 | 1 | → | 00010000 |
| 00001000 | → | 0 | 0 | 0 | → | 00001000 |
| 00000100 | → | 0 | 1 | 1 | → | 00000100 |
| 00000010 | → | 1 | 0 | 1 | → | 00000010 |
| 00000001 | → | 1 | 1 | 0 | → | 00000001 |

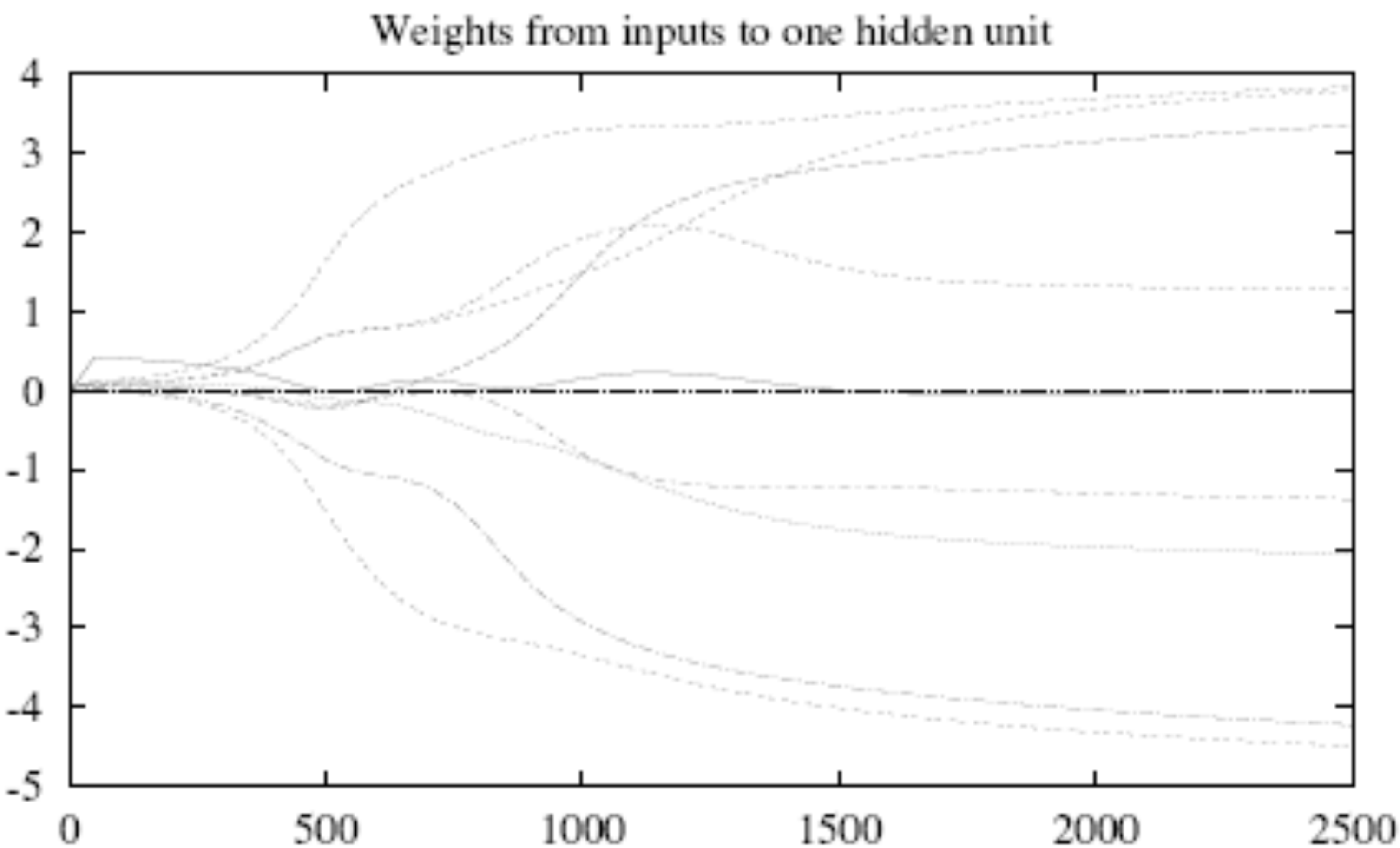
# Training Curve



# Training Curve #2



# Training Curve #3




Skip Recurrent Networks

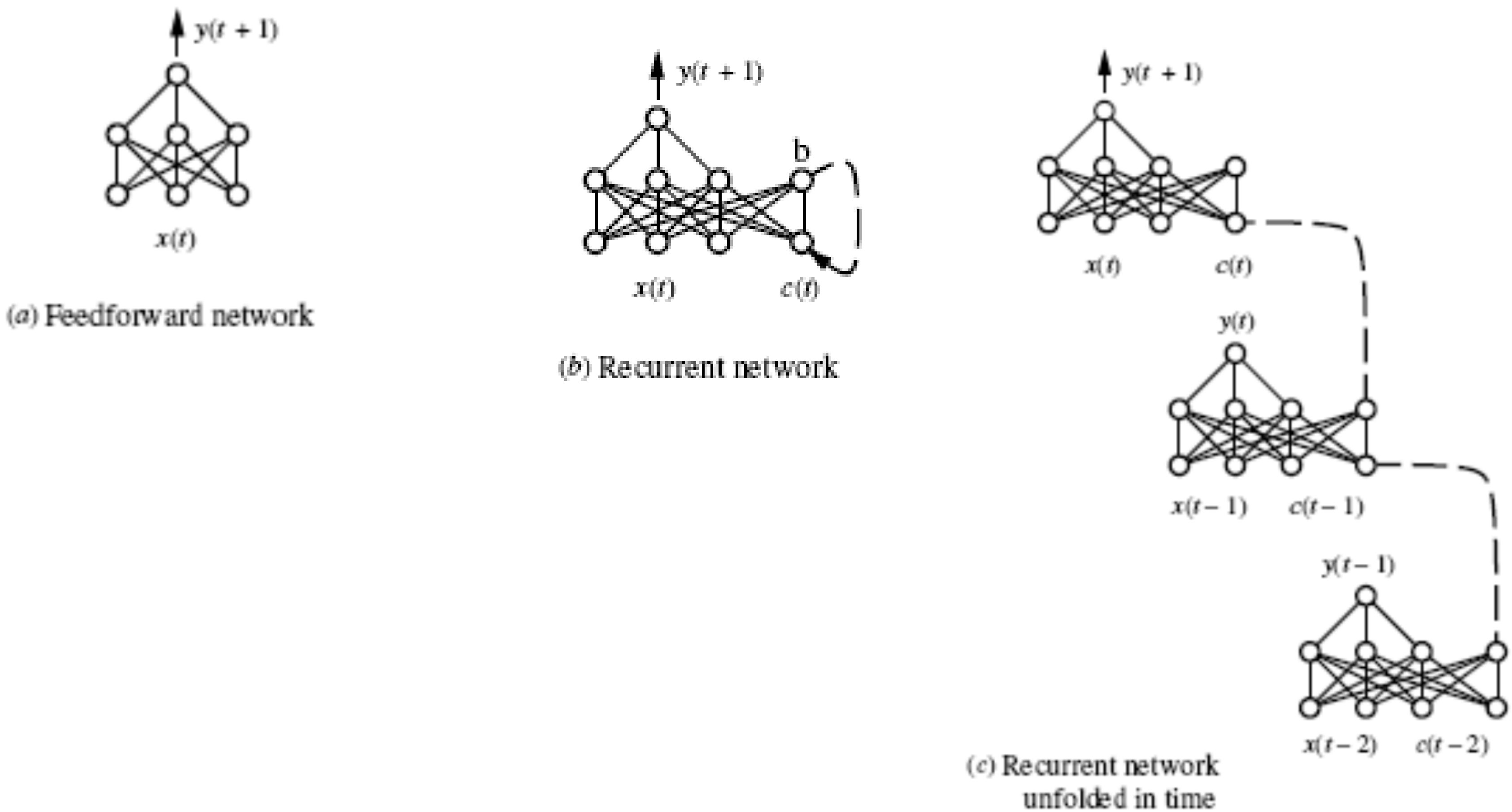
# Recurrent Networks

Not discussed, 2015

- Brain needs short-term memory, . . .
  $\Rightarrow$ feedforward network not sufficient.
- Brain has many feed-back connections.
  $\Rightarrow$ brain is recurrent network, with Cycles!
- Recurrent nets:
  - Can capture internal state.
    (activation keeps going around)
  - More complex agents
  - Much harder to analyze.
    … Unstable, Oscillate, Chaotic
- Main types:
  - Iterative model
  - Hopfield networks
  - Boltzmann machines

# Iterative Recurrent Network

Not discussed, 2015



# Hopfield Networks

Not discussed, 2015

- Symmetric connections ($W_{i,j} = W_{j,i}$)
  - Activation only $\{+1, -1\}$
  - $\sigma(.)$ is sign-function
- Train weights to obtain associative memory
  - eg, store patterns
- After learning, can "retrieve" patterns:
  - Set some node values,
  - other nodes settle to best pattern match

- **Theorem**:
  An N-unit Hopfield net can store up to 0.138N patterns reliably.

- Note: No explicit storage; all in the weights!

# Boltzmann Machines

Not discussed, 2015

- Symmetric connections ($w_{i,j} = w_{j,i}$)
- Activation only $\{+1, -1\}$, but stochastic
- $P(\ n_i = 1\ )$ depends on inputs
  - Network in constant motion,
    computing average output value of each node

  ... like simulated annealing
- Has nice (but slow) learning algorithm.
- Related to probabilistic reasoning

  ... belief networks!

# Other Topics

- Architecture
- Initialization
  - Incorporating Background Knowledge
  - KBANN, ...
- Better statistical models
  - When to use which system?
  - Other training techniques
  - Regularizing
- Other "internal" functions
  - Sigmoid
  - Radial Basis Function

# What to Remember

- Neural Nets can **represent** arbitrarily complex functions
- It can be challenging to **LEARN** the parameters, as multiple local optima
  - … gradient descent … using backpropagation
- Many tricks to make gradient descent work!
  - Line search
  - Conjugate gradient

… useful for ANY optimization (not just NN)