

Machine Learning

4771

Instructor: Tony Jebara

Topic 4

- Tutorial: Matlab
- Perceptron, Online & Stochastic Gradient Descent
- Convergence Guarantee
- Perceptron vs. Linear Regression
- Multi-Layer Neural Networks
- Back-Propagation
- Demo: LeNet
- Deep Learning

Tutorial: Matlab

- Matlab is the most popular language for machine learning
- See www.cs.columbia.edu->computing->Software->Matlab
- Online info to get started is available at:
<http://www.cs.columbia.edu/~jebara/tutorials.html>
- Matlab tutorials
- List of Matlab function calls
- Example code: for homework #1 will use `polyreg.m`
- General: `help`, `lookfor`, `1:N`, `rand`, `zeros`, `A'`, `reshape`, `size`
- Math: `max`, `min`, `cov`, `mean`, `norm`, `inv`, `pinv`, `det`, `sort`, `eye`
- Control: `if`, `for`, `while`, `end`, `%`, `function`, `return`, `clear`
- Display: `figure`, `clf`, `axis`, `close`, `plot`, `subplot`, `hold on`, `fprintf`
- Input/Output: `load`, `save`, `ginput`, `print`,
- BBS and TA's are also helpful

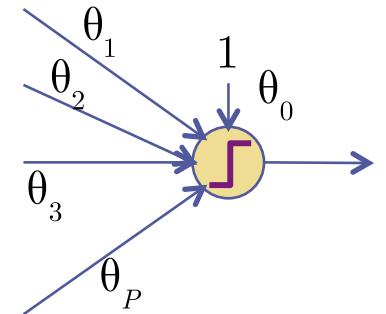
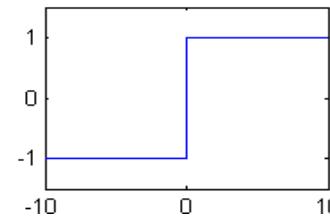
Perceptron (another Neuron)

- Classification scenario once again but consider +1, -1 labels

$$\mathcal{X} = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\} \quad x \in \mathbb{R}^D \quad y \in \{-1, 1\}$$

- A better choice for a classification squashing function is

$$g(z) = \begin{cases} -1 & \text{when } z < 0 \\ +1 & \text{when } z \geq 0 \end{cases}$$



- And a better choice is classification loss

$$L(y, f(\mathbf{x}; \theta)) = \text{step}(-yf(\mathbf{x}; \theta))$$

- Actually with above $g(z)$ any loss is like classification loss

$$R(\theta) = \frac{1}{4N} \sum_{i=1}^N (y_i - g(\theta^T x_i))^2 \equiv \frac{1}{N} \sum_{i=1}^N \text{step}(-y_i \theta^T x_i)$$

- What does this $R(\theta)$ function look like?

$$R(\theta) = \frac{1}{N} \sum_{i=1}^N (y_i - g(\theta^T x_i))^2$$

Perceptron & Classification Loss

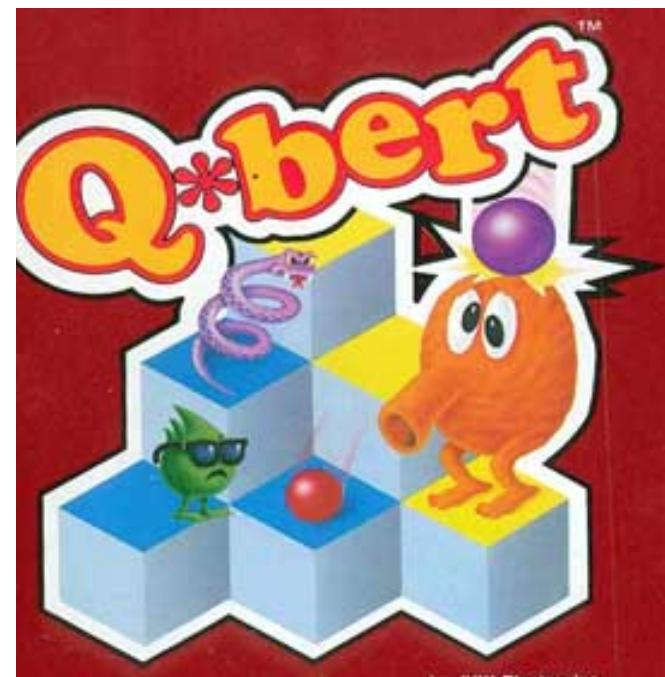
- Classification loss for the Risk leads to hard minimization
- What does this $R(\theta)$ function look like?

$$R(\theta) = \frac{1}{N} \sum_{i=1}^N \text{step}\left(-y_i \theta^T x_i\right)$$

if $y_i = -1, \theta^T x_i = +2 \Rightarrow \text{step}(-y_i \theta^T x_i) = +1$
 if $y_i = +1, \theta^T x_i = -2 \Rightarrow \text{step}(-y_i \theta^T x_i) = -1$
 if $y_i = -1, \theta^T x_i = -2 \Rightarrow \text{step}(-y_i \theta^T x_i) = 0$
 if $y_i = +1, \theta^T x_i = +2 \Rightarrow \text{step}(-y_i \theta^T x_i) = 0$

- Qbert-like, can't do gradient descent since the gradient is zero except at edges when a label flips

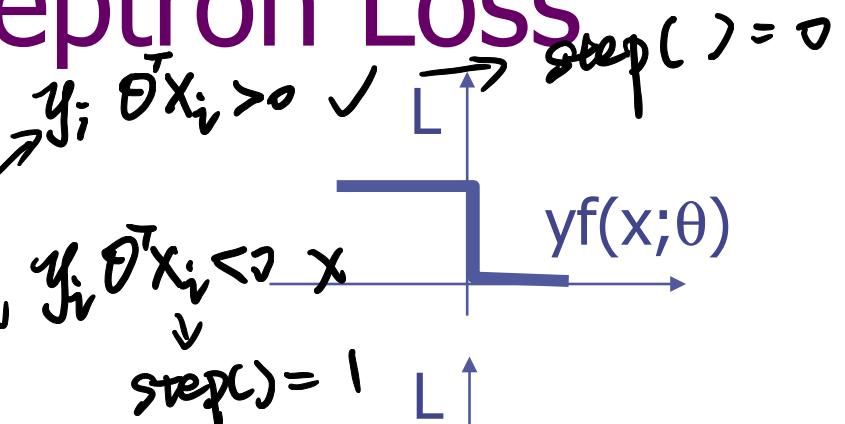
$$R_{\text{OOD}} = \frac{1}{n} \sum_{i \in \text{misclassified}} -y_i \theta^T x_i \quad \frac{\partial R}{\partial \theta} = \frac{1}{n} \sum_{i \in \text{misclassified}} -y_i x_i$$



Perceptron & Perceptron Loss

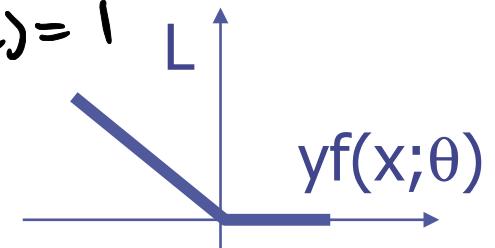
- Instead of Classification Loss

$$R(\theta) = \frac{1}{N} \sum_{i=1}^N \text{step}(-y_i \theta^T x_i)$$



- Consider Perceptron Loss:

$$R^{per}(\theta) = -\frac{1}{N} \sum_{i \in \text{misclassified}} y_i (\theta^T x_i)$$



- Instead of staircase-shaped R
get smooth piece-wise linear R

- Get reasonable gradients for gradient descent

$$\nabla_{\theta} R^{per}(\theta) = -\frac{1}{N} \sum_{i \in \text{misclassified}} y_i \mathbf{x}_i$$

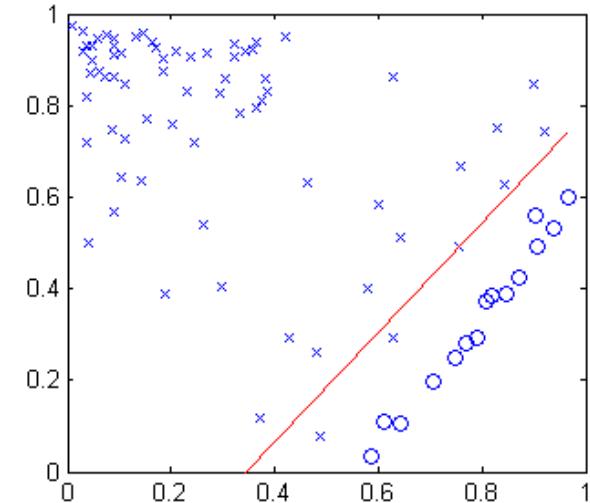
$$\theta^{t+1} = \theta^t - \eta \nabla_{\theta} R^{per} \Big|_{\theta^t} = \theta^t + \eta \frac{1}{N} \sum_{i \in \text{misclassified}} y_i \mathbf{x}_i$$

Gradients

Perceptron vs. Linear Regression

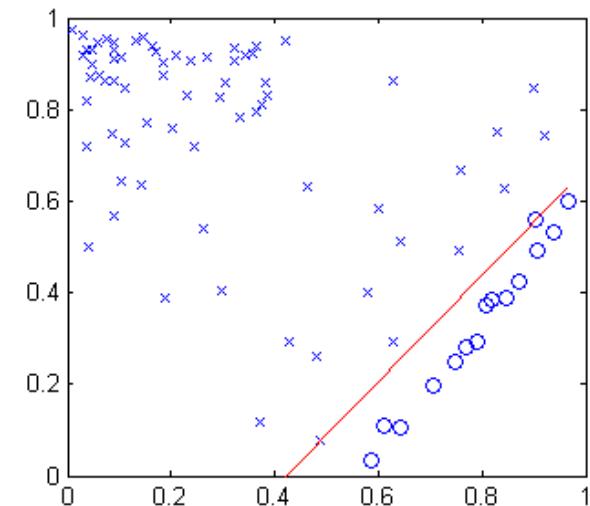
- Linear regression gets close but doesn't do perfectly

classification error = 2
squared error = 0.139



- Perceptron gets zero error

classification error = 0
perceptron err = 0



Stochastic Gradient Descent

- Gradient Descent vs. Stochastic Gradient Descent

All misclassified points

- Instead of computing the average gradient for all points and then taking a step

$$\nabla_{\theta} R^{per}(\theta) = -\frac{1}{N} \sum_{i \in \text{misclassified}} y_i \mathbf{x}_i$$

each mis point

- Update the gradient for each mis-classified point by itself

$$\nabla_{\theta} R^{per}(\theta) = -y_i \mathbf{x}_i \quad \text{if } i \text{ mis-classified}$$

- Also, set η to 1 without loss of generality

$$\theta^{t+1} = \theta^t - \eta \nabla_{\theta} R^{per} \Big|_{\theta^t} = \theta^t + \underbrace{y_i \mathbf{x}_i}_{\mathcal{D}} \quad \text{if } i \text{ mis-classified}$$

Online Perceptron

- Apply stochastic gradient descent to a perceptron
- Get the “online perceptron” algorithm:

initialize $t = 0$ and $\theta^0 = \vec{0}$
 $\underbrace{\text{while not converged}}_{\theta^{t+1} - \theta^t \geq \epsilon}\{$
 $\quad \underbrace{\text{pick } i \in \{1, \dots, N\}}_{\text{if } (y_i x_i^T \theta^t \leq 0)} \quad \{ \quad \theta^{t+1} = \theta^t + y_i x_i$
 $\quad \quad t = t + 1 \quad \} \}$

- Either pick i randomly or use a “for i=1 to N” loop
- If the algorithm stops, we have a theta that separates data
- The total number of mistakes we made along the way is t

Online Perceptron Theorem

Theorem: the online perceptron algorithm converges to zero error in finite t if we assume

- 1) all data inside a sphere of radius r: $\|\mathbf{x}_i\| \leq r \quad \forall i$
- 2) data is separable with margin γ : $y_i (\theta^*)^T \mathbf{x}_i \geq \gamma \quad \forall i$

Proof:

- Part 1) Look at inner product of current θ^t with θ^* assume we just updated a mistake on point i:

$$\hat{\theta} = \theta^{t-1} + y_i \cdot \mathbf{x}_i \quad (\theta^*)^T \theta^t = (\theta^*)^T \theta^{t-1} + y_i (\theta^*)^T \mathbf{x}_i \geq (\theta^*)^T \theta^{t-1} + \gamma$$

after applying t such updates, we must get:

$$(\theta^*)^T \theta^t = (\theta^*)^T \theta^t \geq t\gamma$$

$$\|\theta^t\|^2 \leq \|\theta^{t-1}\|^2 + r^2 = \|\theta^{t-2}\|^2 + r^2 + r^2 \leq \sum_{i=0}^t \|\theta^{t-i}\|^2 + r^2 + r^2 + \dots \leq \|\theta^*\|^2 + tr^2$$

Tony Jebara, Columbia University

Online Perceptron Proof

- Part 1) $(\theta^*)^T \theta^t = (\theta^*)^T \theta^t \geq t\gamma$

- Part 2)

$$\begin{aligned} \|\theta^t\|^2 &= \|\theta^{t-1} + y_i \mathbf{x}_i\|^2 = \|\theta^{t-1}\|^2 + 2y_i (\theta^{t-1})^T \mathbf{x}_i + \|\mathbf{x}_i\|^2 \\ &\leq \|\theta^{t-1}\|^2 + \|\mathbf{x}_i\|^2 \\ &\leq \|\theta^{t-1}\|^2 + r^2 \\ &\leq tr^2 \end{aligned}$$

A · B = negative

A *B*

since only update mistakes
middle term is negative

- Part 3) Angle between optimal & current solution

$$\cos(\theta^*, \theta^t) = \frac{(\theta^*)^T \theta^t}{\|\theta^t\| \|\theta^*\|} \geq \frac{t\gamma}{\|\theta^t\| \|\theta^*\|} \geq \frac{t\gamma}{\sqrt{tr^2} \|\theta^*\|}$$

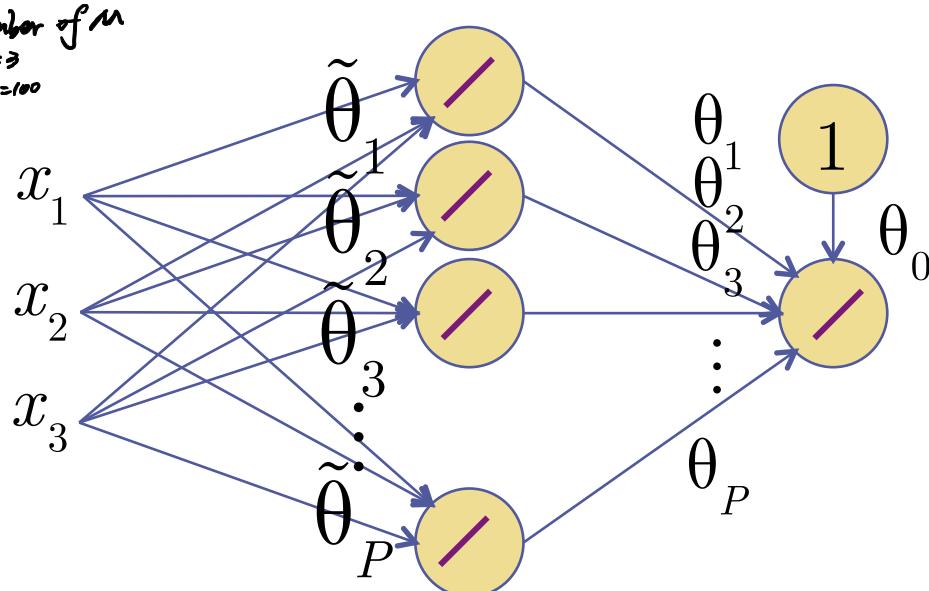
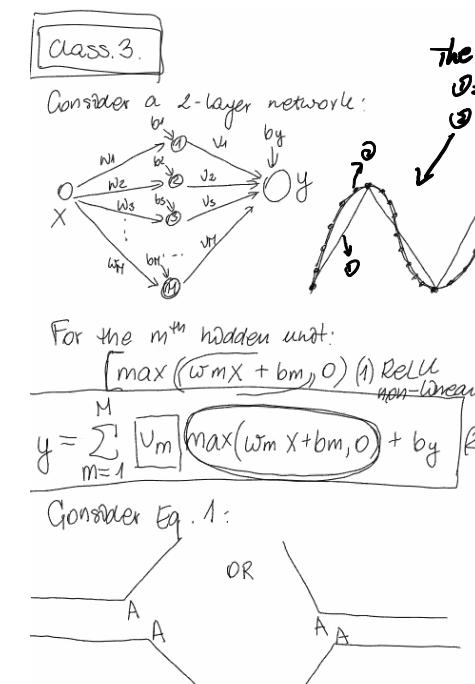
apply part 1
then part 2

- Since $\cos \leq 1 \Rightarrow \frac{t\gamma}{\sqrt{tr^2} \|\theta^*\|} \leq 1 \Rightarrow t \leq \frac{r^2}{\gamma^2} \|\theta^*\|^2$

...so t is finite!

Multi-Layer Neural Networks

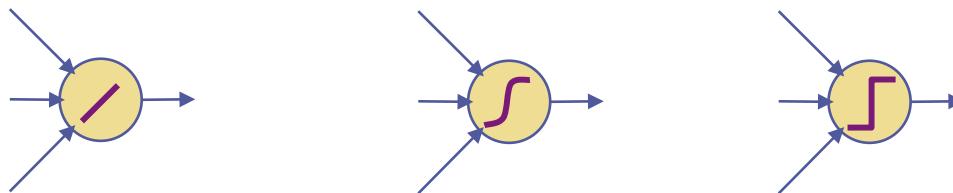
- What if we consider cascading multiple layers of network?
- Each output layer is input to the next layer
- Each layer has its own weights parameters
- Eg: each layer has linear nodes (not perceptron/logistic)



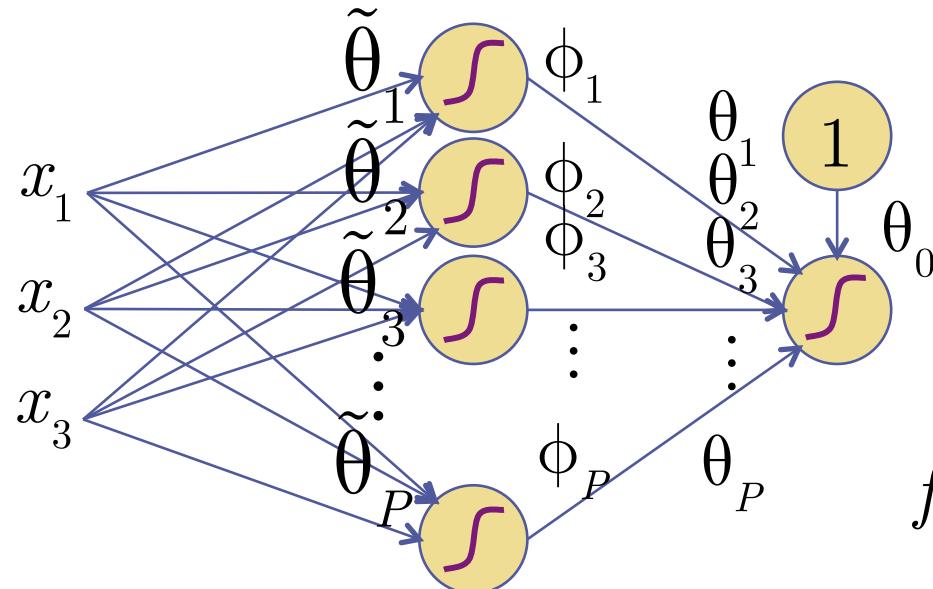
- Above Neural Net has 2 layers. What does this give?

Multi-Layer Neural Networks

- Need to introduce non-linearities between layers
- Avoids previous redundant linear layer problem



- Neural network can adjust the basis functions themselves...

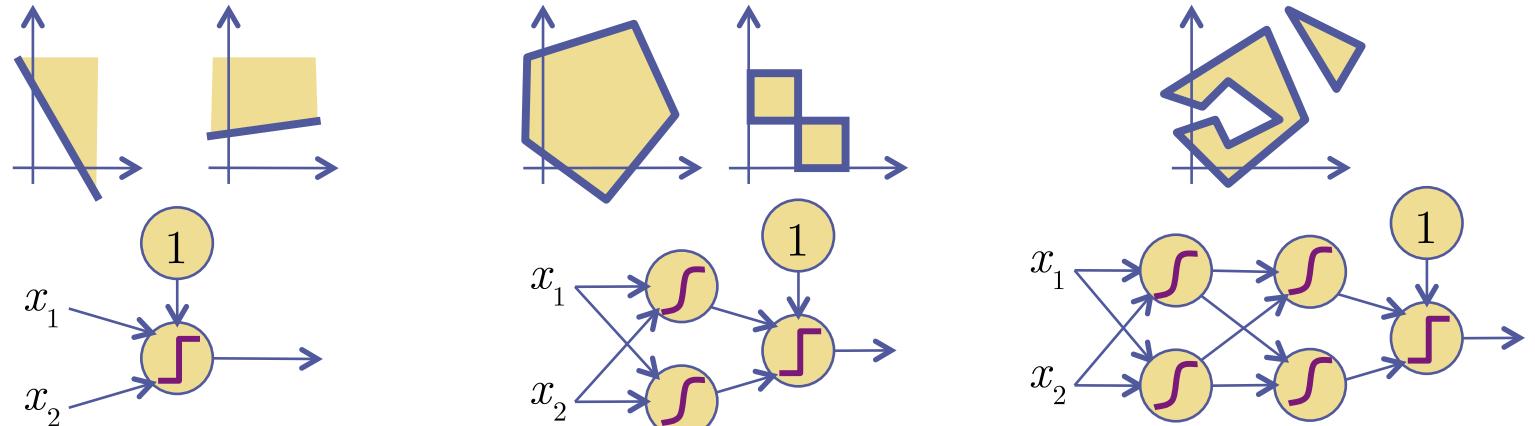


$$f(\mathbf{x}) = g\left(\sum_{i=1}^P \theta_i g\left(\tilde{\theta}_i^T \mathbf{x}\right)\right)$$

Multi-Layer Neural Networks

- Multi-Layer Network can handle more complex decisions
- 1-layer: is linear, can't handle XOR
- Each layer adds more flexibility (but more parameters!)
- Each node splits its input space with linear hyperplane
- 2-layer: if last layer is AND operation, get convex hull
- 2-layer: can do almost anything multi-layer can

by fanning out the inputs at 2nd layer



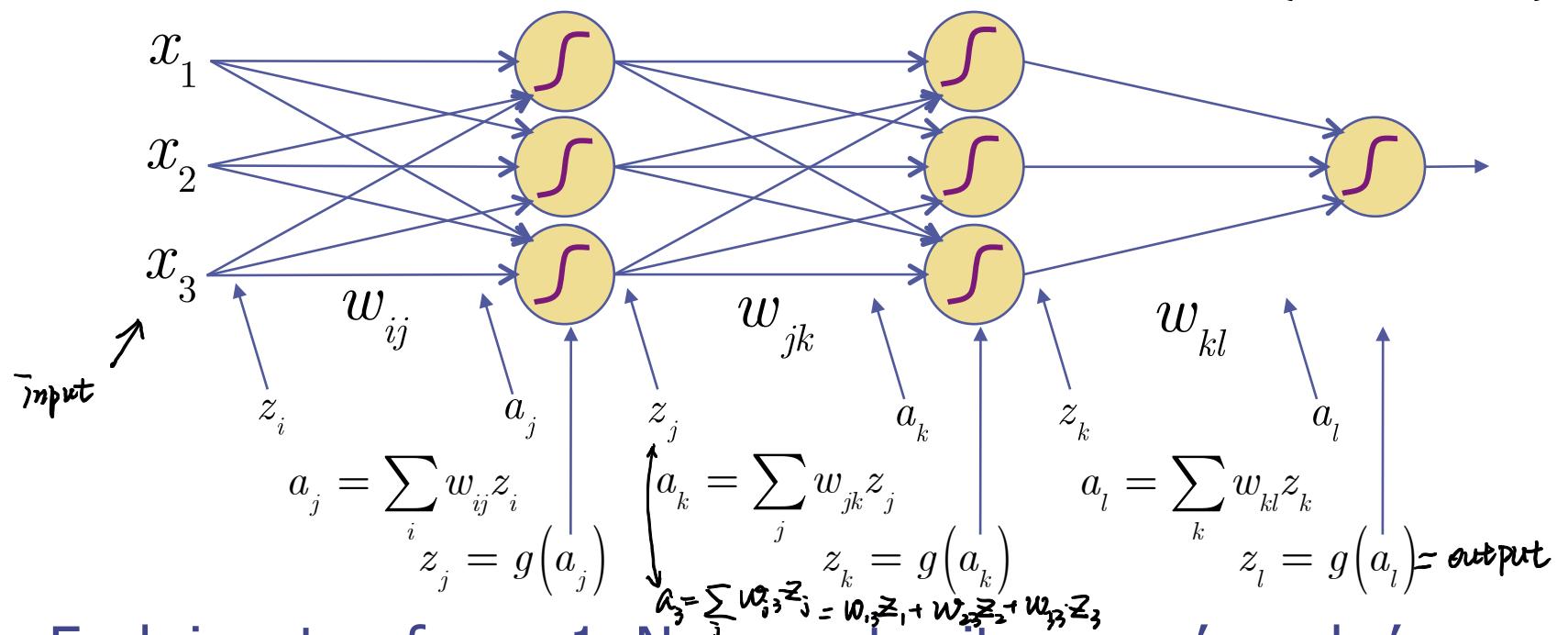
- Note: Without loss of generality, we can omit the 1 and θ_0

$$\begin{aligned}
 a_j &= \sum_i w_{ij} z_i \\
 z_j &= g(a_j) \\
 x_1, x_2, x_3 &\text{ inputs} \\
 &= z_1, z_2, z_3
 \end{aligned}
 \quad
 \begin{aligned}
 a_k &= \sum_j w_{jk} z_j \\
 z_k &= g(a_k) \\
 a_l &= \sum_l w_{lk} z_k \\
 z_l &= g(a_l) \Rightarrow \text{Output}
 \end{aligned}$$

Tony Jebara, Columbia University

Back-Propagation

- Gradient descent on squared loss is done layer by layer
- Layers: input, hidden, output. Parameters: $\theta = \{w_{ij}, w_{jk}, w_{kl}\}$

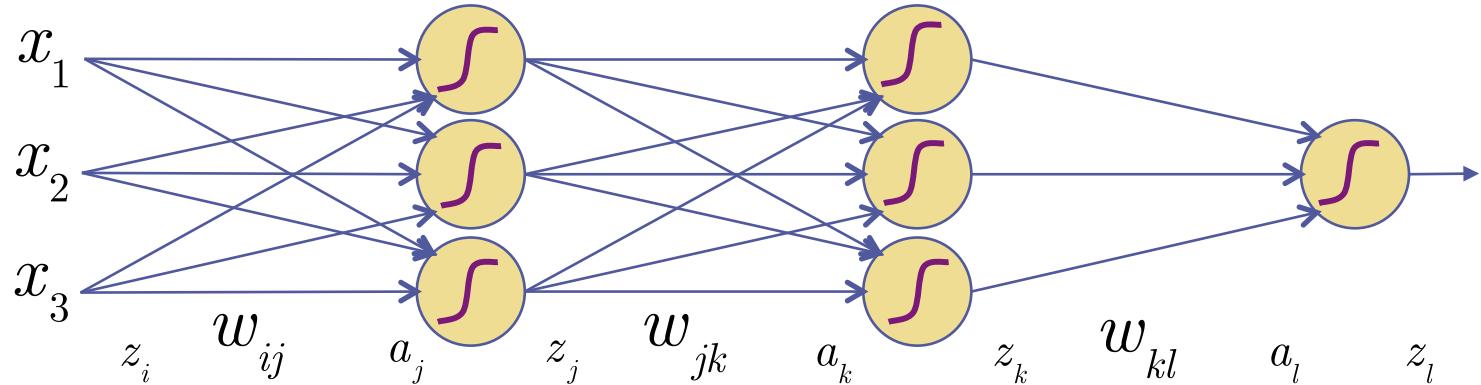


- Each input x_n for $n=1..N$ generates its own a 's and z 's
- Back-Propagation: Splits layer into its inputs & outputs
- Get gradient on output...back-track chain rule until input

Back-Propagation

- Cost function: $R(\theta) = \frac{1}{N} \sum_{n=1}^N L(y^n - f(x^n))$

$$= \frac{1}{N} \sum_n \frac{1}{2} \left(y^n - g\left(\sum_k w_{kl} g\left(\sum_j w_{jk} g\left(\sum_i w_{ij} x_i^n \right) \right) \right) \right)^2$$



- First compute output layer derivative:

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N} \sum_n \left[\frac{\partial L^n}{\partial a_l^n} \right] \left(\frac{\partial a_l^n}{\partial w_{kl}} \right)$$

Chain Rule

define $L^n := \frac{1}{2} (y^n - f(x^n))^2$

Back-Propagation

- Cost function:
$$\begin{aligned} R(\theta) &= \frac{1}{N} \sum_{n=1}^N L\left(y^n - f(x^n)\right) \\ &= \frac{1}{N} \sum_n \frac{1}{2} \left(y^n - g\left(\sum_k w_{kl} g\left(\sum_j w_{jk} g\left(\sum_i w_{ij} x_i^n\right)\right)\right) \right)^2 \end{aligned}$$
-
- x_1 \rightarrow z_i w_{ij} a_j \rightarrow z_j w_{jk} a_k \rightarrow z_k w_{kl} a_l \rightarrow z_l

- First compute output layer derivative:

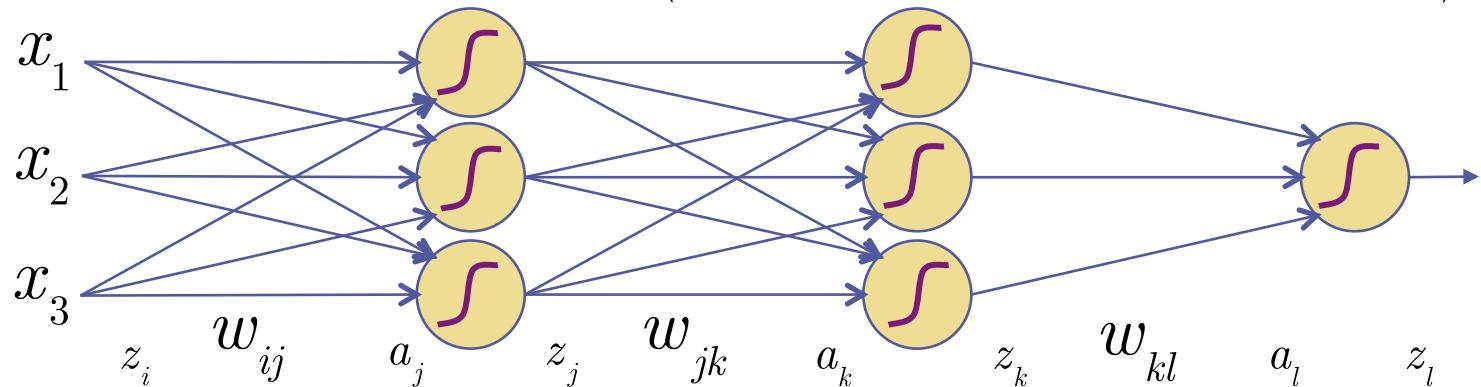
$$\begin{aligned} \frac{\partial R}{\partial w_{kl}} &= \frac{1}{N} \sum_n \left[\frac{\partial L^n}{\partial a_l^n} \left(\frac{\partial a_l^n}{\partial w_{kl}} \right) \right] \boxed{\text{Chain Rule}} \\ &\Downarrow \frac{1}{N} \sum_n \left[\frac{\partial \frac{1}{2} \left(y^n - g(a_l^n) \right)^2}{\partial a_l^n} \left(\frac{\partial a_l^n}{\partial w_{kl}} \right) \right] \end{aligned}$$

define $L^n := \frac{1}{2} \left(y^n - g(a_l^n) \right)^2$

Back-Propagation

- Cost function: $R(\theta) = \frac{1}{N} \sum_{n=1}^N L(y^n - f(x^n))$

$$= \frac{1}{N} \sum_n \frac{1}{2} \left(y^n - g\left(\sum_k w_{kl} g\left(\sum_j w_{jk} g\left(\sum_i w_{ij} x_i^n\right)\right)\right)\right)^2$$



- First compute output layer derivative:

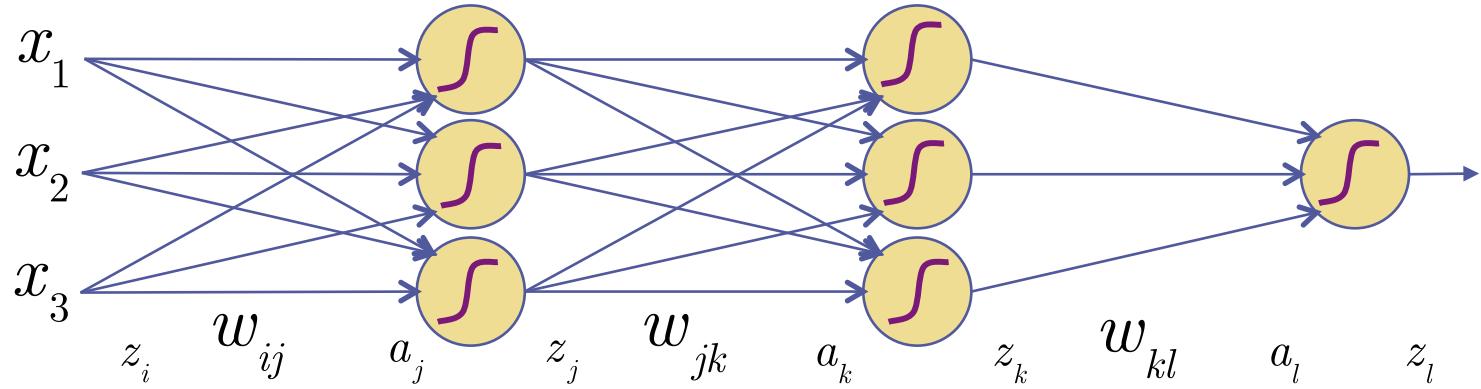
$$\begin{aligned} \frac{\partial R}{\partial w_{kl}} &= \frac{1}{N} \sum_n \left[\frac{\partial L^n}{\partial a_l^n} \left(\frac{\partial a_l^n}{\partial w_{kl}} \right) \right] \boxed{\text{Chain Rule}} \\ &= \frac{1}{N} \sum_n \left[\frac{\partial \frac{1}{2} (y^n - g(a_l^n))^2}{\partial a_l^n} \left(\frac{\partial a_l^n}{\partial w_{kl}} \right) \right] = \frac{1}{N} \sum_n \left[-(y^n - z_l^n) g'(a_l^n) \right] (z_k^n) \end{aligned}$$

define $L^n := \frac{1}{2} (y^n - f(x^n))^2$

Back-Propagation

- Cost function: $R(\theta) = \frac{1}{N} \sum_{n=1}^N L(y^n - f(x^n))$

$$= \frac{1}{N} \sum_n \frac{1}{2} \left(y^n - g\left(\sum_k w_{kl} g\left(\sum_j w_{jk} g\left(\sum_i w_{ij} x_i^n\right)\right)\right)\right)^2$$



- First compute output layer derivative:

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N} \sum_n \left[\frac{\partial L^n}{\partial a_l^n} \right] \left(\frac{\partial a_l^n}{\partial w_{kl}} \right)$$

Chain Rule

define $L^n := \frac{1}{2} (y^n - f(x^n))^2$
 $a_l :=$

Define as δ

$$\frac{\partial \delta_l (y^n - g(a_l^n))}{\partial a_l^n} = \frac{1}{N} \sum_n \left[\frac{\partial \frac{1}{2} (y^n - g(a_l^n))^2}{\partial a_l^n} \right] \left(\frac{\partial a_l^n}{\partial w_{kl}} \right) = \frac{1}{N} \sum_n \left[- (y^n - z_l^n) g'(a_l^n) \right] (z_k^n) = \frac{1}{N} \sum_n \delta_l^n z_k^n$$

$$\frac{\partial \delta_l (y^n - g(a_l^n))}{\partial a_l^n} = \frac{1}{N} \sum_n \left[- (y^n - z_l^n) g'(a_l^n) \right] (z_k^n) = - (y^n - z_l^n) g'(a_l^n)$$

Chain rule:

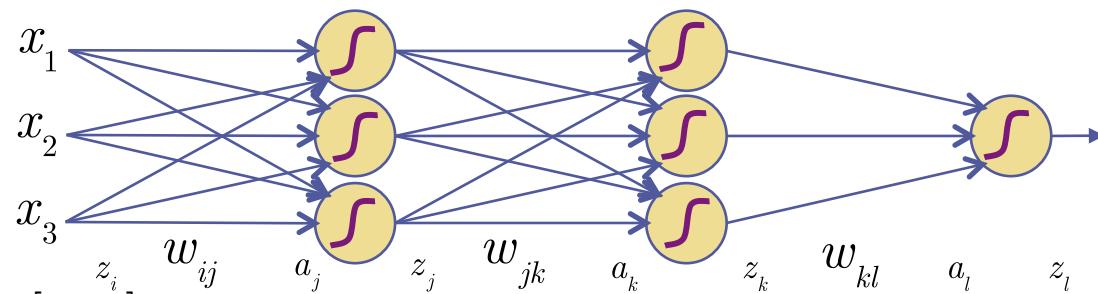
Tony Jebara, Colu

* If y is a function of x , for example $y = t(x)$ then:
 $\frac{\partial y}{\partial x} = \frac{\partial y}{\partial t} \frac{\partial t}{\partial x}$.

* Multivariate chain rule:
If $f(x, y)$ & $x(t)$ & $y(t)$ then
 $\frac{\partial f}{\partial t} = \frac{\partial f}{\partial x} \frac{\partial x}{\partial t} + \frac{\partial f}{\partial y} \frac{\partial y}{\partial t}$

Back-Propagation

• Cost function: $R(\theta) = \frac{1}{N} \sum_n \frac{1}{2} \left(y^n - g \left(\sum_k w_{kl} g \left(\sum_j w_{jk} g \left(\sum_i w_{ij} x_i^n \right) \right) \right) \right)^2$



$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N} \sum_n \left[\frac{\partial L^n}{\partial a_l^n} \right] \left[\frac{\partial a_l^n}{\partial w_{kl}} \right] = \frac{1}{N} \sum_n \left[- (y^n - z_l^n) g'(a_l^n) \right] (z_k^n) = \frac{1}{N} \sum_n \delta_l^n z_k^n$$

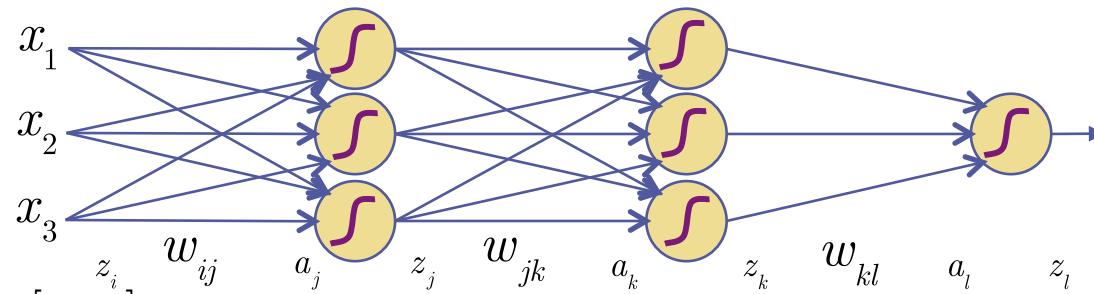
• Next, hidden layer derivative:

$$\begin{aligned} \frac{\partial R}{\partial w_{jk}} &= \frac{1}{N} \sum_n \left[\frac{\partial L^n}{\partial a_k^n} \right] \left[\frac{\partial a_k^n}{\partial w_{jk}} \right] = \frac{1}{N} \sum_n \left(\frac{\partial L^n}{\partial a_i^n} \cdot \frac{\partial a_i^n}{\partial a_k^n} \cdot \frac{\partial a_k^n}{\partial w_{jk}} \right) \\ &= \frac{1}{N} \sum_n \left(\delta_i \frac{\partial a_i^n}{\partial z_k} \cdot \frac{\partial z_k}{\partial a_k} \right) \left(\frac{\partial a_k}{\partial w_{jk}} \right) \end{aligned}$$

$$= \frac{1}{N} \sum_n \left(\underbrace{\delta_i \cdot w_{kl} \cdot g'(a_k)}_{\delta_j} \right) \cdot z_j$$

Back-Propagation

- Cost function: $R(\theta) = \frac{1}{N} \sum_n \frac{1}{2} \left(y^n - g\left(\sum_k w_{kl} g\left(\sum_j w_{jk} g\left(\sum_i w_{ij} x_i^n\right)\right)\right)\right)^2$



$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N} \sum_n \left[\frac{\partial L^n}{\partial a_l^n} \right] \left[\frac{\partial a_l^n}{\partial w_{kl}} \right] = \frac{1}{N} \sum_n \left[-\left(y^n - z_l^n \right) g'(a_l^n) \right] \left(z_k^n \right) = \frac{1}{N} \sum_n \delta_l^n z_k^n$$

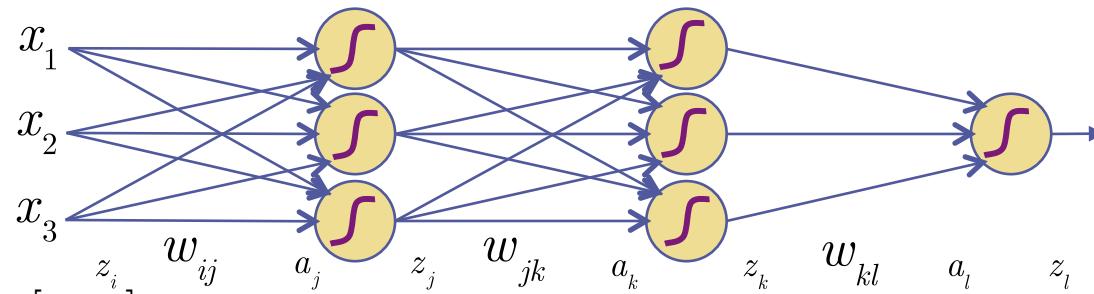
- Next, hidden layer derivative:

$$\frac{\partial R}{\partial w_{jk}} = \frac{1}{N} \sum_n \left[\frac{\partial L^n}{\partial a_k^n} \right] \left[\frac{\partial a_k^n}{\partial w_{jk}} \right] = \frac{1}{N} \sum_n \left[\sum_l \frac{\partial L^n}{\partial a_l^n} \frac{\partial a_l^n}{\partial a_k^n} \right] \left[\frac{\partial a_k^n}{\partial w_{jk}} \right]$$

Multivariate Chain Rule

Back-Propagation

- Cost function: $R(\theta) = \frac{1}{N} \sum_n \frac{1}{2} \left(y^n - g\left(\sum_k w_{kl} g\left(\sum_j w_{jk} g\left(\sum_i w_{ij} x_i^n\right)\right)\right)\right)^2$



$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N} \sum_n \left[\frac{\partial L^n}{\partial a_l^n} \right] \left[\frac{\partial a_l^n}{\partial w_{kl}} \right] = \frac{1}{N} \sum_n \left[-\left(y^n - z_l^n \right) g'(a_l^n) \right] (z_k^n) = \frac{1}{N} \sum_n \delta_l^n z_k^n$$

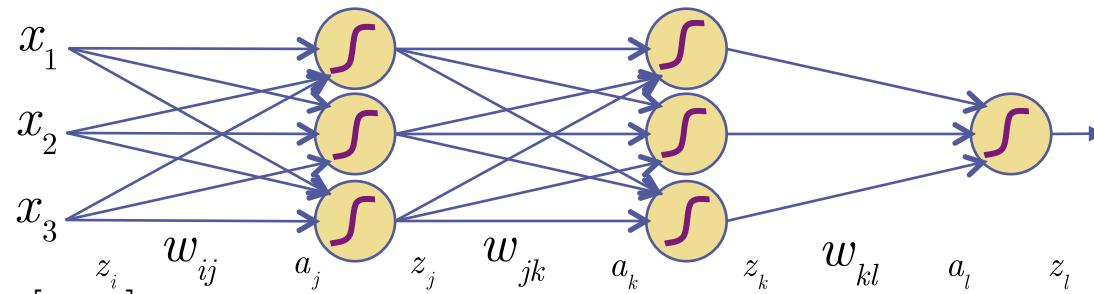
- Next, hidden layer derivative:

$$\begin{aligned} \frac{\partial R}{\partial w_{jk}} &= \frac{1}{N} \sum_n \left[\frac{\partial L^n}{\partial a_k^n} \right] \left[\frac{\partial a_k^n}{\partial w_{jk}} \right] = \frac{1}{N} \sum_n \left[\sum_l \frac{\partial L^n}{\partial a_l^n} \frac{\partial a_l^n}{\partial a_k^n} \right] \left[\frac{\partial a_k^n}{\partial w_{jk}} \right] \\ &= \frac{1}{N} \sum_n \left[\underbrace{\sum_l \delta_l^n \frac{\partial a_l^n}{\partial a_k^n}}_{\text{Multivariate Chain Rule}} \right] (z_j^n) \end{aligned}$$

Multivariate Chain Rule

Back-Propagation

- Cost function: $R(\theta) = \frac{1}{N} \sum_n \frac{1}{2} \left(y^n - g\left(\sum_k w_{kl} g\left(\sum_j w_{jk} g\left(\sum_i w_{ij} x_i^n\right)\right)\right)\right)^2$



$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N} \sum_n \left[\frac{\partial L^n}{\partial a_l^n} \left(\frac{\partial a_l^n}{\partial w_{kl}} \right) \right] = \frac{1}{N} \sum_n \left[-\left(y^n - z_l^n \right) g'(a_l^n) \right] (z_k^n) = \frac{1}{N} \sum_n \delta_l^n z_k^n$$

- Next, hidden layer derivative:

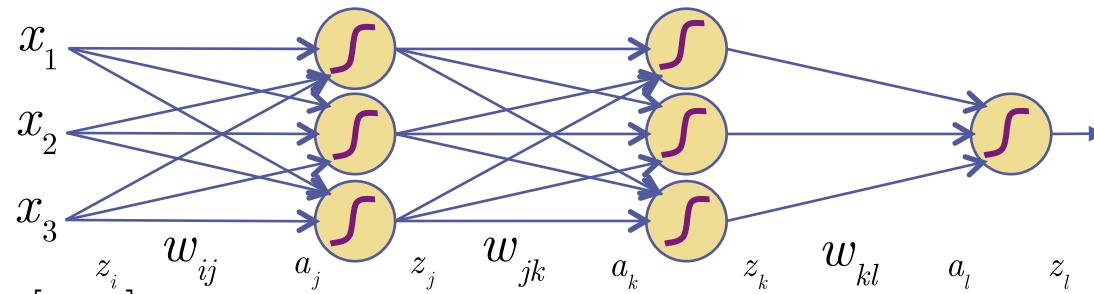
$$\begin{aligned} \frac{\partial R}{\partial w_{jk}} &= \frac{1}{N} \sum_n \left[\frac{\partial L^n}{\partial a_k^n} \left(\frac{\partial a_k^n}{\partial w_{jk}} \right) \right] = \frac{1}{N} \sum_n \left[\sum_l \frac{\partial L^n}{\partial a_l^n} \frac{\partial a_l^n}{\partial a_k^n} \left(\frac{\partial a_k^n}{\partial w_{jk}} \right) \right] \\ &= \frac{1}{N} \sum_n \left[\sum_l \delta_l^n \frac{\partial a_l^n}{\partial a_k^n} \right] (z_j^n) \end{aligned}$$

Multivariate Chain Rule

Recall $a_l = \sum_k w_{kl} g(a_k)$

Back-Propagation

- Cost function: $R(\theta) = \frac{1}{N} \sum_n \frac{1}{2} \left(y^n - g\left(\sum_k w_{kl} g\left(\sum_j w_{jk} g\left(\sum_i w_{ij} x_i^n\right)\right)\right)\right)^2$



$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N} \sum_n \left[\frac{\partial L^n}{\partial a_l^n} \right] \left[\frac{\partial a_l^n}{\partial w_{kl}} \right] = \frac{1}{N} \sum_n \left[- (y^n - z_l^n) g'(a_l^n) \right] (z_k^n) = \frac{1}{N} \sum_n \delta_l^n z_k^n$$

- Next, hidden layer derivative:

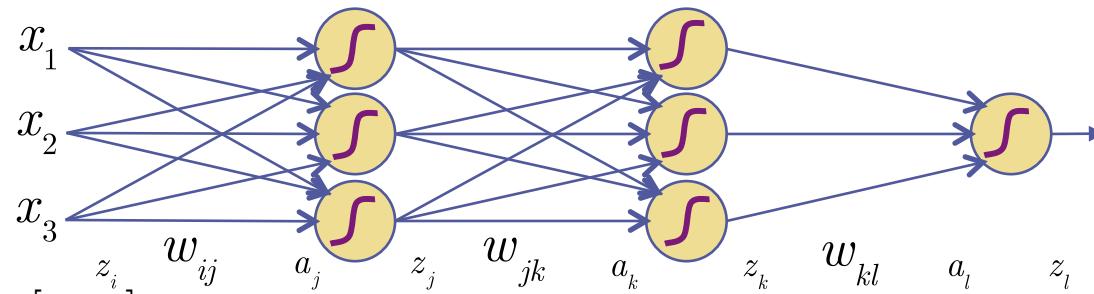
$$\begin{aligned} \frac{\partial R}{\partial w_{jk}} &= \frac{1}{N} \sum_n \left[\frac{\partial L^n}{\partial a_k^n} \right] \left[\frac{\partial a_k^n}{\partial w_{jk}} \right] = \frac{1}{N} \sum_n \left[\sum_l \frac{\partial L^n}{\partial a_l^n} \frac{\partial a_l^n}{\partial a_k^n} \right] \left[\frac{\partial a_k^n}{\partial w_{jk}} \right] \\ &= \frac{1}{N} \sum_n \left[\sum_l \delta_l^n \frac{\partial a_l^n}{\partial a_k^n} \right] (z_j^n) = \frac{1}{N} \sum_n \left[\sum_l \underbrace{\delta_l^n w_{kl}}_{\text{Recall}} g'(a_k^n) \right] (z_j^n) \end{aligned}$$

Multivariate Chain Rule

Recall $a_l = \sum_k w_{kl} g(a_k)$

Back-Propagation

- Cost function: $R(\theta) = \frac{1}{N} \sum_n \frac{1}{2} \left(y^n - g \left(\sum_k w_{kl} g \left(\sum_j w_{jk} g \left(\sum_i w_{ij} x_i^n \right) \right) \right) \right)^2$



$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N} \sum_n \left[\frac{\partial L^n}{\partial a_l^n} \right] \left[\frac{\partial a_l^n}{\partial w_{kl}} \right] = \frac{1}{N} \sum_n \underbrace{\left[- (y^n - z_l^n) g'(a_l^n) \right]}_{\delta_l} \left[z_k^n \right] = \frac{1}{N} \sum_n \delta_l^n z_k^n$$

- Next, hidden layer derivative:

$$\frac{\partial R}{\partial w_{jk}} = \frac{1}{N} \sum_n \left[\frac{\partial L^n}{\partial a_k^n} \right] \left[\frac{\partial a_k^n}{\partial w_{jk}} \right] = \frac{1}{N} \sum_n \left[\sum_l \frac{\partial L^n}{\partial a_l^n} \frac{\partial a_l^n}{\partial a_k^n} \right] \left[\frac{\partial a_k^n}{\partial w_{jk}} \right]$$

Multivariate Chain Rule

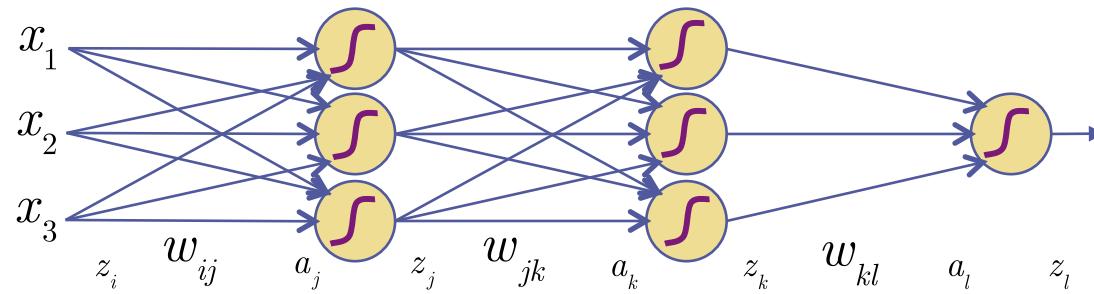
$$= \frac{1}{N} \sum_n \left[\sum_l \delta_l^n \frac{\partial a_l^n}{\partial a_k^n} \right] \left[z_j^n \right] = \frac{1}{N} \sum_n \underbrace{\left[\sum_l \delta_l^n w_{kl} g'(a_k^n) \right]}_{\delta_k} \left[z_j^n \right] = \frac{1}{N} \sum_n \delta_k^n z_j^n$$

Recall $a_l = \sum_k w_{kl} g(a_k)$

Define as δ δ_k

Back-Propagation

- Cost function: $R(\theta) = \frac{1}{N} \sum_n \frac{1}{2} \left(y^n - g \left(\sum_k w_{kl} g \left(\sum_j w_{jk} g \left(\sum_i w_{ij} x_i^n \right) \right) \right) \right)^2$



$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N} \sum_n \left[\frac{\partial L^n}{\partial a_l^n} \right] \left[\frac{\partial a_l^n}{\partial w_{kl}} \right] = \frac{1}{N} \sum_n \left[- (y^n - z_l^n) g'(a_l^n) \right] (z_k^n) = \frac{1}{N} \sum_n \delta_l^n z_k^n$$

$$\frac{\partial R}{\partial w_{jk}} = \frac{1}{N} \sum_n \left[\frac{\partial L^n}{\partial a_k^n} \right] \left[\frac{\partial a_k^n}{\partial w_{jk}} \right] = \frac{1}{N} \sum_n \left[\sum_l \delta_l^n w_{kl} g'(a_k^n) \right] (z_j^n) = \frac{1}{N} \sum_n \delta_k^n z_j^n$$

- Any previous (input) layer derivative: repeat the formula!

$$\frac{\partial R}{\partial w_{ij}} = \frac{1}{N} \sum_n \left[\frac{\partial L^n}{\partial a_j^n} \right] \left[\frac{\partial a_j^n}{\partial w_{ij}} \right] = \frac{1}{N} \sum_n \left[\sum_k \frac{\partial L^n}{\partial a_k^n} \frac{\partial a_k^n}{\partial a_j^n} \right] \left[\frac{\partial a_j^n}{\partial w_{ij}} \right] = \frac{1}{N} \sum_n \left[\sum_k \underbrace{\delta_k^n w_{jk} g'(a_j^n)}_{S_k} \right] (z_i^n) = \frac{1}{N} \sum_n \delta_j^n z_i^n$$

- What is this last z ?

\downarrow $\bar{\downarrow}$ $\frac{\partial a_k}{\partial z_j} \cdot \frac{\partial g(a_j)}{\partial a_j} = w_{jk} \cdot g'(a_j) = S_j$

Back-Propagation

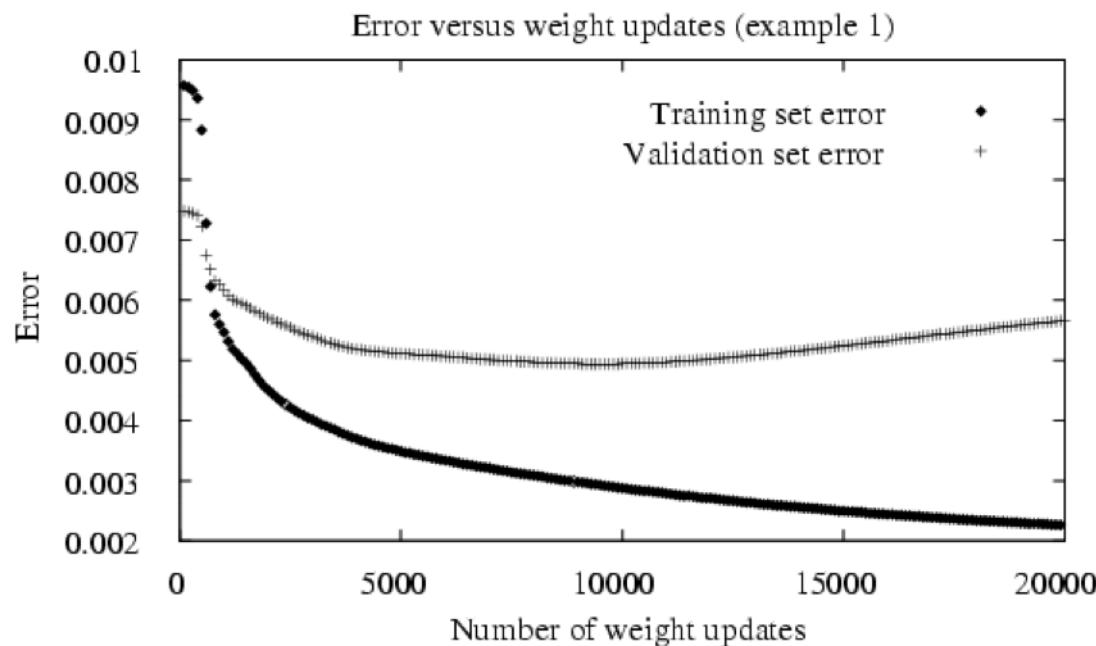
- Again, take small step in direction opposite to gradient

$$w_{ij}^{t+1} = w_{ij}^t - \eta \frac{\partial R}{\partial w_{ij}}$$

$$w_{jk}^{t+1} = w_{jk}^t - \eta \frac{\partial R}{\partial w_{jk}}$$

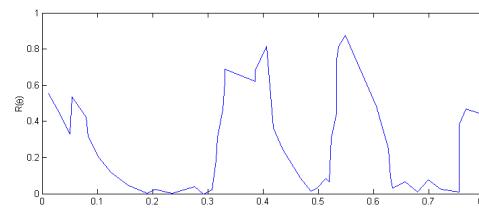
$$w_{kl}^{t+1} = w_{kl}^t - \eta \frac{\partial R}{\partial w_{kl}}$$

- Early stop before when error bottoms out on validation set



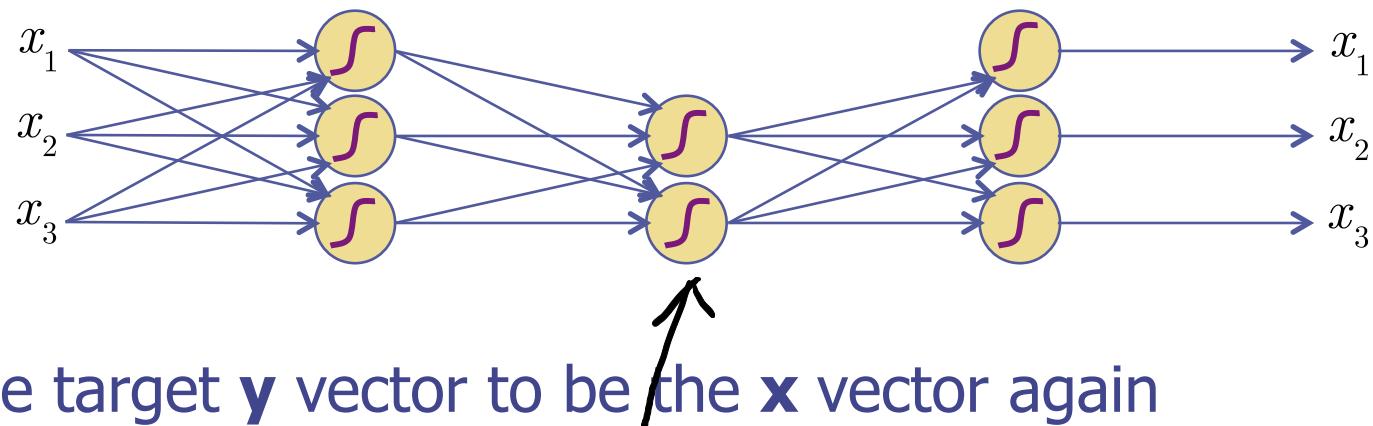
Neural Networks Demo

- Again, take small step in direction opposite to gradient
- Digits Demo: LeNet... <http://yann.lecun.com>
- Problems with back-prop
is that MLP over-fits...
- Other problems: hard to interpret, black-box
- What are the hidden inner layers doing?



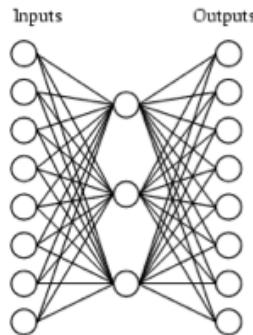
Auto-Encoders

- Make the neural net reconstruct the input vector



- Set the target y vector to be the x vector again
 - But, it gets narrow in the middle!
 - So, there is some information “compression”
 - This leads to better generalization
-
- This is unsupervised learning since we only use the x data

Auto-Encoders



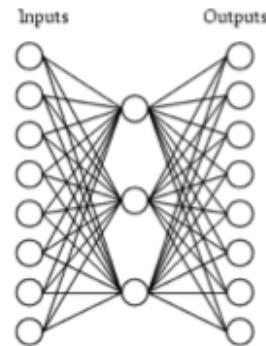
A target function:

Input	Output
10000000	\rightarrow 10000000
01000000	\rightarrow 01000000
00100000	\rightarrow 00100000
00010000	\rightarrow 00010000
00001000	\rightarrow 00001000
00000100	\rightarrow 00000100
00000010	\rightarrow 00000010
00000001	\rightarrow 00000001

Can this be learned??

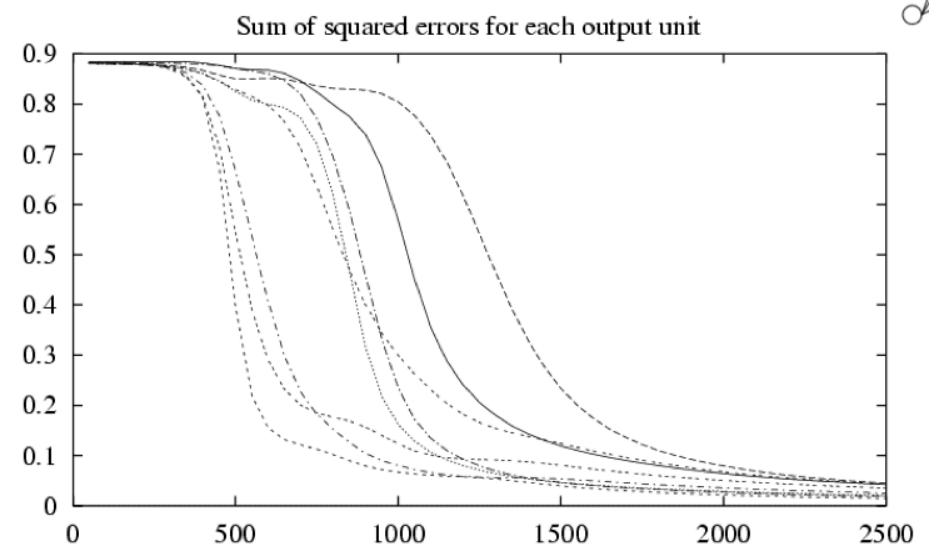
Auto-Encoders

A network:



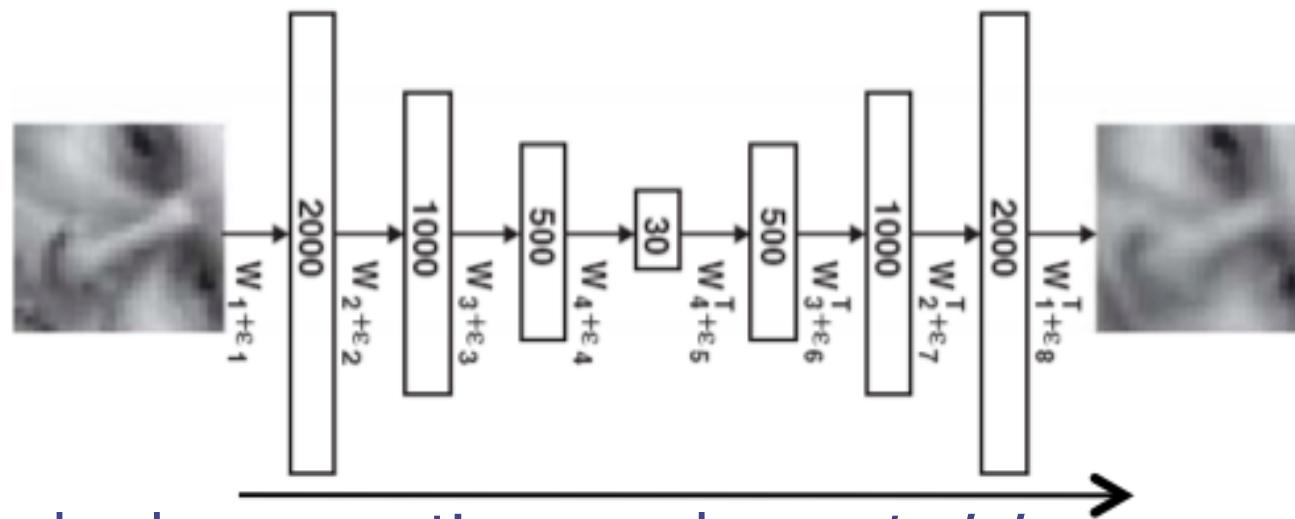
Learned hidden layer representation:

Input	Hidden Values	Output
10000000	→ .89 .04 .08	→ 10000000
01000000	→ .01 .11 .88	→ 01000000
00100000	→ .01 .97 .27	→ 00100000
00010000	→ .99 .97 .71	→ 00010000
00001000	→ .03 .05 .02	→ 00001000
00000100	→ .22 .99 .99	→ 00000100
00000010	→ .80 .01 .98	→ 00000010
00000001	→ .60 .94 .01	→ 00000001



Deep Learning

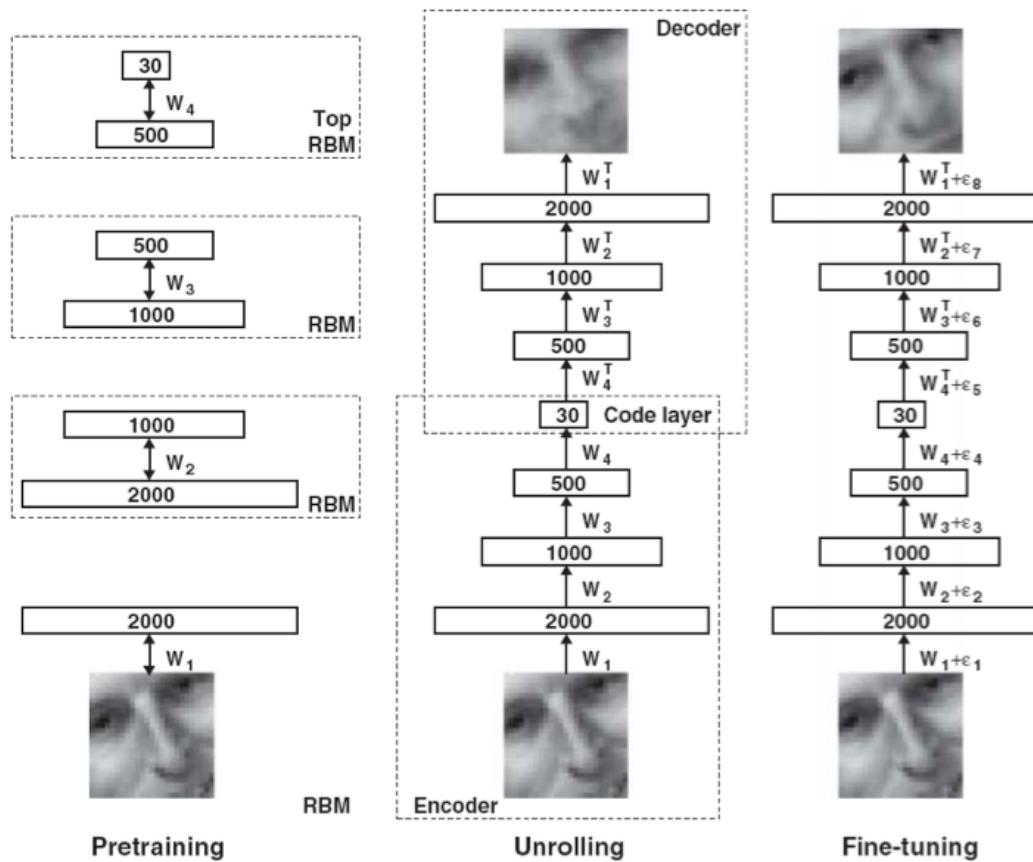
- We can stack several independently trained auto-encoders



- Using back-propagation, we do *pre-training*
- Train Net 1 to go from 2000 inputs to 1000 to 2000 inputs
- Train Net 2 to go from 1000 hidden values to 500 to 1000
- Train Net 3 to go from 500 hidden to 30 to 500
- Then, do *unrolling* - link up the learned networks as above

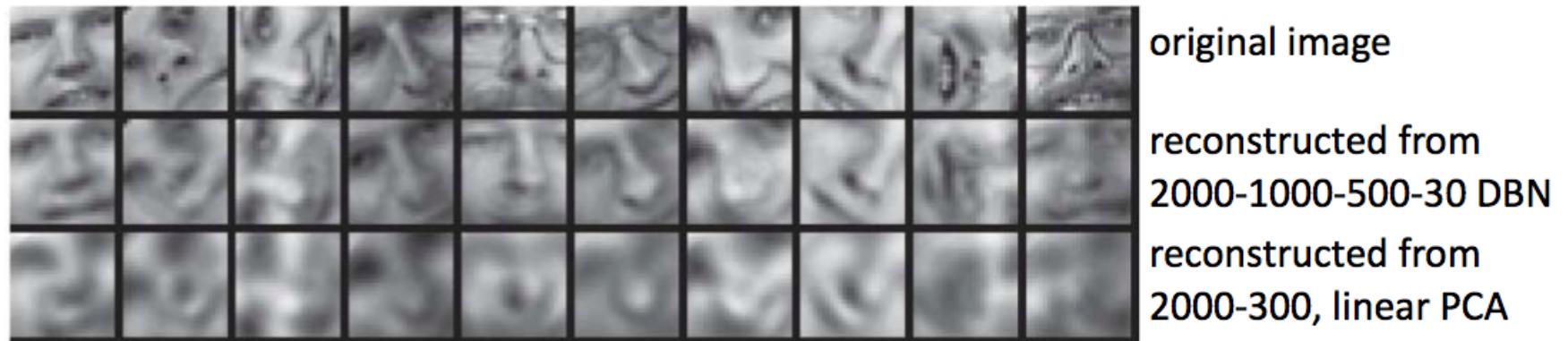
Deep Learning

- Then do *fine-tuning* of the overall neural network by running back-propagation on the whole thing to reconstruct \mathbf{x} from \mathbf{x}



Deep Learning

- Does good reconstruction!
- Beats PCA on images of unaligned faces.
- PCA is better when face images are aligned...
- We will cover PCA in a few lectures...



Minimum Training Error?

- Is minimizing Empirical Risk the right thing?
- Are Perceptrons and Neural Networks giving the best classifier?
- We are getting: minimum training error
not minimum testing error
- Perceptrons are giving a bunch of solutions:

