Principles of Machine Learning
Approximating Real-Valued Functions from Data
Neural Networks and Deep Learning

Vasant Honavar
Artificial Intelligence Research Laboratory
College of Information Sciences and Technology
Bioinformatics and Genomics Graduate Program
The Huck Institutes of the Life Sciences
Pennsylvania State University

vhonavar@ist.psu.edu
http://vhonavar.ist.psu.edu
http://faculty.ist.psu.edu/vhonavar

Learning Real-Valued Functions

- Learning to approximate real-valued functions
- Bayesian recipe for learning real-valued functions
- Brief digression – continuity, differentiability, Taylor series approximation of functions
- Learning linear functions using gradient descent in weight space
- Universal function approximation theorem
- Learning nonlinear functions using gradient descent in weight space
- Practical considerations and examples

Review: Bayesian Recipe for Learning

$$P(h \mid D) = \frac{P(D \mid h)P(h)}{P(D)}$$

- *P* (*h*) = prior probability of hypothesis *h*
- *P* (*D*) = prior probability of training data *D*
- *P* (*h* | *D*) = probability of *h* given D
- *P* (*D* | *h*) = probability of *D* given *h*

Bayesian recipe for learning

Choose the most likely hypothesis given the data

$$h_{MAP} = \arg\max_{h \in H} P(h \mid D) \qquad \text{(Maximum a posteriori hypothesis)}$$

$$= \arg\max_{h \in H} \frac{P(D \mid h) P(h)}{P(D)}$$

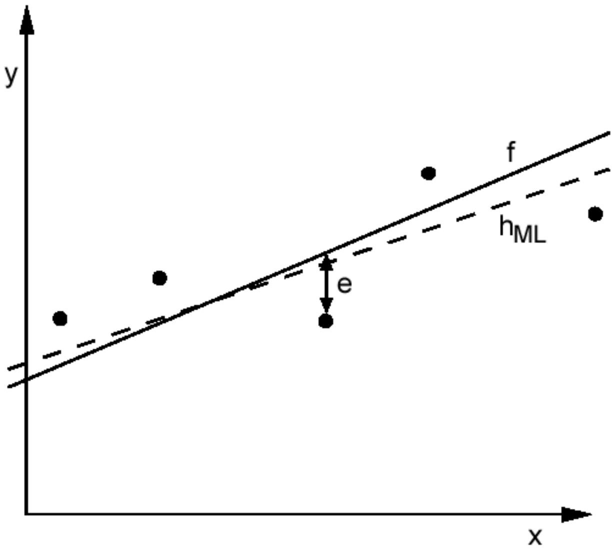
$$= \arg\max_{h \in H} P(D \mid h) P(h)$$

If $\forall h_i, h_j \in H \;\; P(h_i) = P(h_j),$

$$h_{ML} = \arg\max_{h \in H} P(D \mid h) \qquad \text{(Maximum likelihood hypothesis)}$$

Learning a Real Valued Function

- Consider a real-valued target function *f*
- Training examples $\langle x_i, d_i \rangle$, where d_i is noisy training value $d_i = f(x_i) + e_i$
- e_i is random variable (noise) drawn independently for each x_i according to Gaussian distribution with zero mean
- $\Rightarrow d_i$ *has mean* $f(x_i)$ and same variance



Then the maximum likelihood hypothesis h_{ML} is one that minimizes the sum of squared error:

$$h_{ML} = \arg\min_{h \in H} \sum_{i=1}^{m} (d_i - h(x_i))^2$$

Learning a Real Valued Function

$$\begin{aligned}
h_{ML} &= \arg\max_{h \in H} P(h \mid D) \\
&= \arg\max_{h \in H} P(D \mid h) \\
&= \arg\max_{h \in H} \prod_{i=1}^{m} P(d_i, X_i \mid h) \\
&= \arg\max_{h \in H} \prod_{i=1}^{m} P(d_i \mid h, X_i) P(X_i) \\
&= \arg\max_{h \in H} \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\left(\frac{d_i - h(x_i)}{\sigma}\right)^2} \prod_{i=1}^{m} P(X_i)
\end{aligned}$$

Maximize natural log of this instead...

Learning a Real Valued Function

$$h_{ML} = \arg\max_{h \in H} \sum_{i=1}^{m} \ln \frac{1}{\sqrt{2\pi\sigma^2}} - \tfrac{1}{2}\left(\frac{d_i - h(x_i)}{\sigma}\right)^2$$

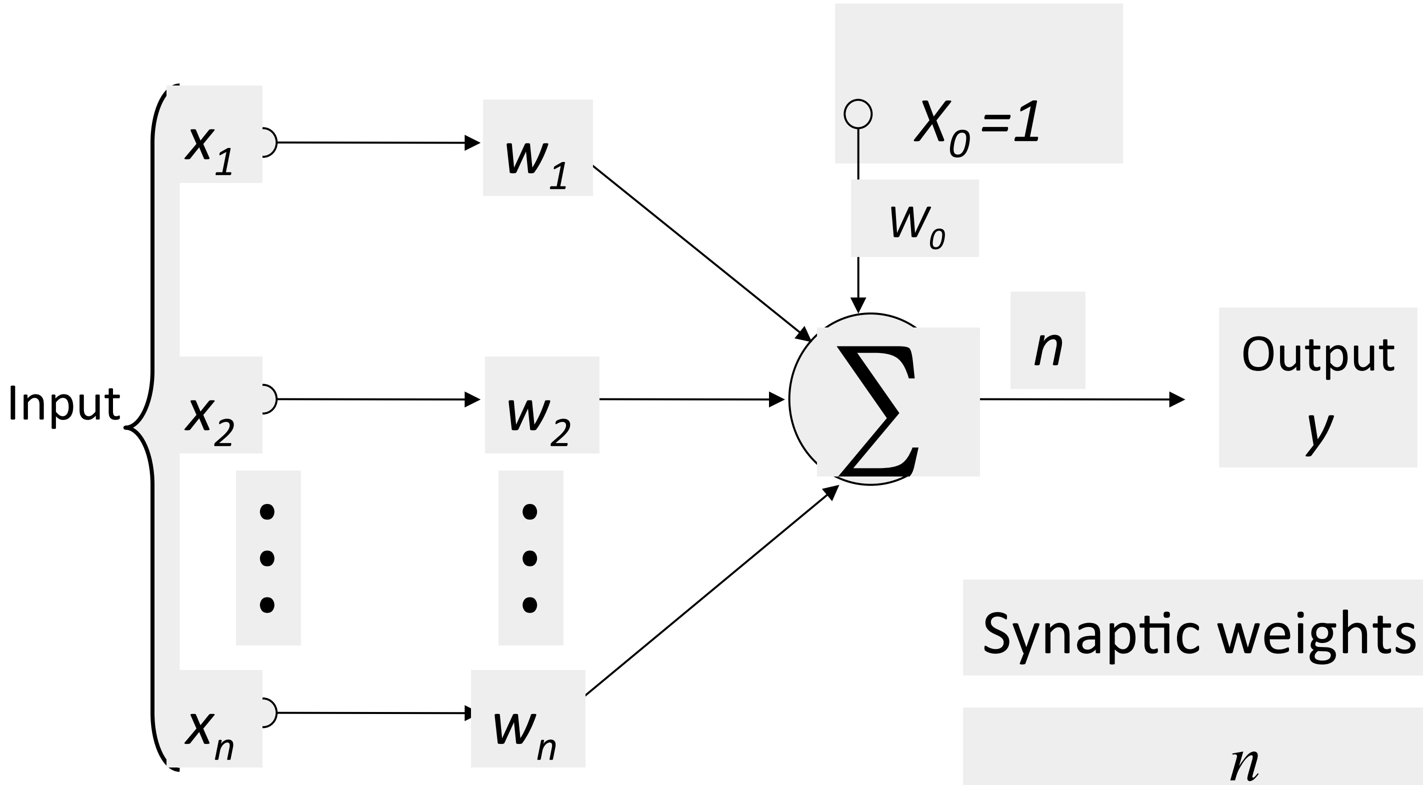
$$= \arg\max_{h \in H} \sum_{i=1}^{m} -\tfrac{1}{2}\left(\frac{d_i - h(x_i)}{\sigma}\right)^2$$

$$= \arg\max_{h \in H} \sum_{i=1}^{m} -(d_i - h(x_i))^2$$

$$= \arg\min_{h \in H} \sum_{i=1}^{m} (d_i - h(x_i))^2$$

Maximum Likelihood hypothesis is one that minimizes the mean squared error!

Approximating a linear function using a linear neuron



Synaptic weights

$$y = \sum_{i=0}^{n} w_i x_i$$

Learning Task

$\mathbf{W} = [W_0 \ldots\ldots\ldots W_n]^T$ is the weight vector

$\mathbf{X}_p = \left[X_{0p} \ldots . X_{np}\right]^T$ is the pth training sample

$y_p = \sum_i W_i X_{ip} = \mathbf{W} \bullet \mathbf{X}_p$ is the output of the neuron for input $\mathbf{X}_p$

$\mathbf{X}_p = f\left(\mathbf{X}_p\right)$ is the desired output for input $\mathbf{X}_p$

$e_p = \left(d_p - y_p\right)$ is the *error* of the neuron on input $\mathbf{X}_p$

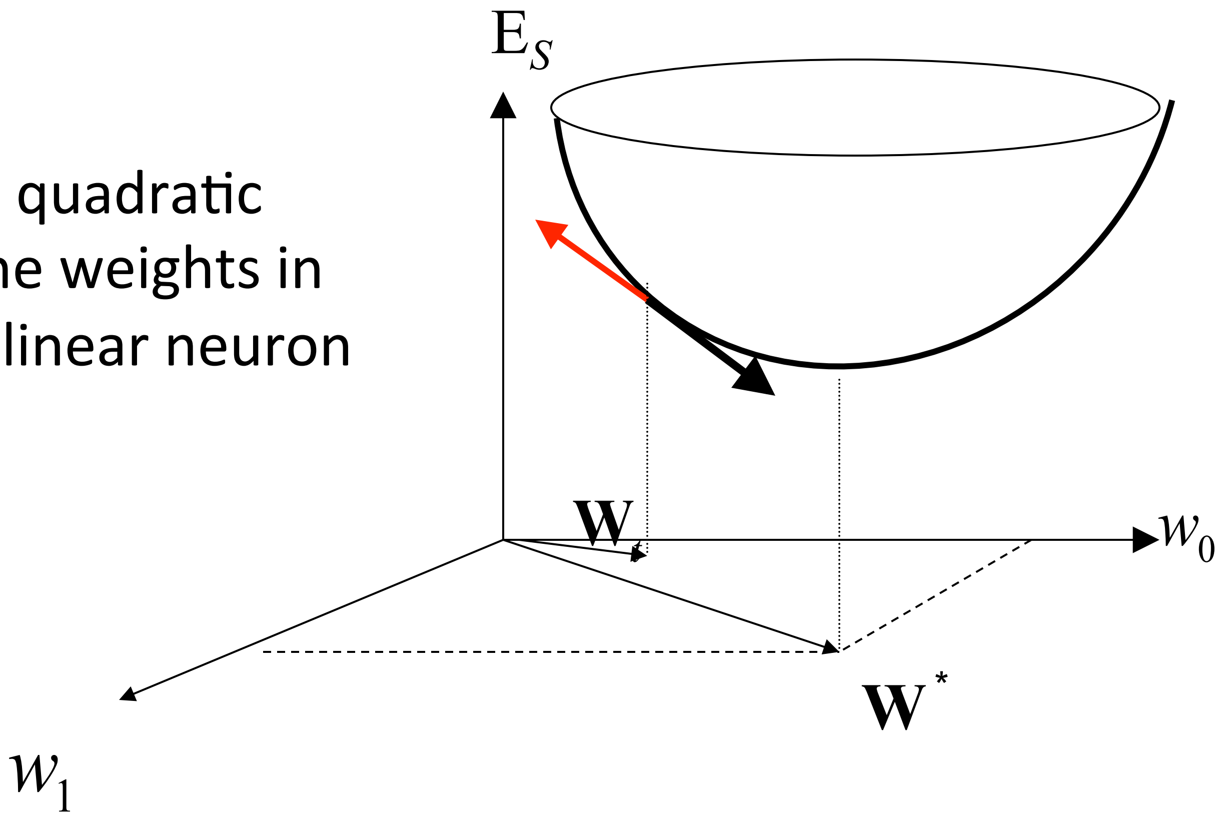
$S = \left\{\left(\mathbf{X}_p, d_p\right)\right\}$ is`a (multi) set of training examples

$E_S(\mathbf{W}) = E_S(W_0, W_1, \ldots\ldots\ldots W_n) = \frac{1}{2}\sum_p e_p^2$ is the estimated error of $\mathbf{W}$ on training set S

Goal: Find $\mathbf{W}^* = \underset{\mathbf{W}}{\arg\min}\, E_S(\mathbf{W})$

Learning linear functions

The error is a quadratic function of the weights in the case of a linear neuron



Learning linear functions

$$w_i \leftarrow w_i - \eta \frac{\partial E}{\partial w_i}$$

$$\frac{\partial E}{\partial w_i} = \frac{1}{2}\frac{\partial}{\partial w_i}\left\{\sum_p e_p^2\right\} = \frac{1}{2}\left(\sum_p \frac{\partial}{\partial w_i}\left(e_p^2\right)\right)$$

$$= \frac{1}{2}\left(\sum_p \left(2e_p\right)\frac{\partial e_p}{\partial w_i}\right) = \sum_p e_p\left(\frac{\partial e_p}{\partial y_p}\right)\left(\frac{\partial y_p}{\partial w_i}\right) = \sum_p e_p(-1)\left(\frac{\partial}{\partial w_i}\left(\sum_{j=0}^{n} w_j x_{jp}\right)\right)$$

$$= -\sum_p \left(d_p - y_p\right)\left(\frac{\partial}{\partial w_i}\left(w_i x_{ip} + \sum_{j \neq i} w_j x_{jp}\right)\right)$$

$$= -\sum_p \left(d_p - y_p\right)\left(\frac{\partial}{\partial w_i}\left(w_i x_{ip}\right) + \frac{\partial}{\partial w_i}\left(\sum_{j \neq i} w_j x_{jp}\right)\right)$$

$$= -\sum_p \left(d_p - y_p\right)x_{ip}$$

$$w_i \leftarrow w_i + \eta \sum_p \left(d_p - y_p\right) x_{ip}$$

Least Mean Square Error (LMSE) Learning Rule

$$w_i \leftarrow w_i + \eta \sum_p \left(d_p - y_p\right) x_{ip}$$

Batch Update

Per sample Update

$$w_i \leftarrow w_i + \eta\left(d_p - y_p\right) x_{ip}$$

Choice of learning rate

In theory, infinitesimally small (why?)

Per sample Update $$0 < \eta < \frac{2}{\left\| \mathbf{X}_p \right\|^2}$$

Batch Update $$0 < \eta < \frac{1}{\lambda_{\max}}$$

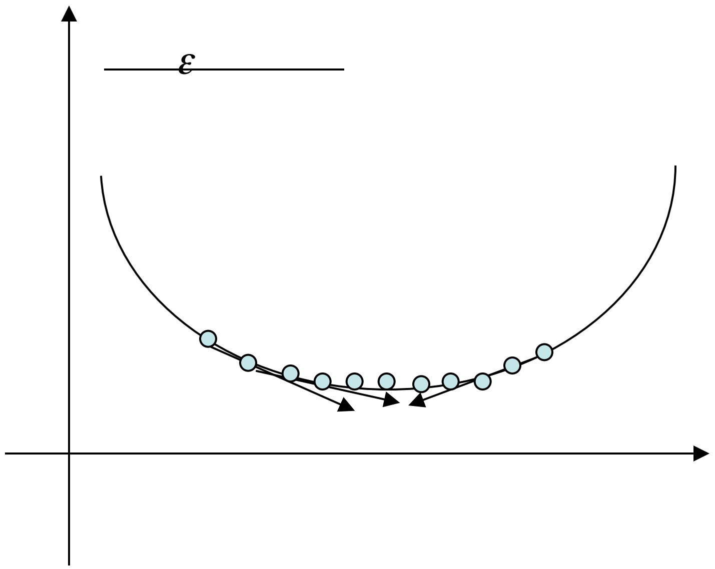
λ_{max} is the largest Eigen value of the Hessian of E_S (matrix of second order partial derivatives of E_S with respect to the weights)

Eigen values of a matrix A are given by solutions of $|A - \lambda I| = 0$

Problem with Gradient Descent

- Difficult to find the appropriate step size
  - Small η → slow convergence
  - Large η → oscillation
- Convergence conditions
  - Robbins-Monroe conditions

$$\sum_{t=0}^{\infty} \eta_t \rightarrow \infty, \quad \sum_{t=0}^{\infty} \eta_t^2 < \infty$$



General Algorithm

- Algorithm (Model algorithm for n-dimensional unconstrained minimization). Let x_k be the current estimate of x^*.
  - [Test for convergence] If the conditions for convergence are satisfied, the algorithm terminates with x_k as the solution.
  - [Compute a search direction] Compute a non-zero *n*-vector p_k, the direction of the search.
- Different algorithms differ primarily in their choice of the search direction

Newton Method

- Utilizing the second order derivative
- Expand the objective function to the second order around x_0

$$f(x) \approx f(x_0) + a(x - x_0) + \frac{b}{2}(x - x_0)^2$$

$$a = f'(x)\big|_{x=x_0}, \quad b = f''(x)\big|_{x=x_0}$$

- The minimum point is $x = x_0 - a/b$
- Newton method for optimization

$$x^{new} \leftarrow x^{old} - \frac{f'(x)\big|_{x=x^{old}}}{f''(x)\big|_{x=x^{old}}}$$

- Guaranteed to converge when the objective function is convex

Newton' s method

- Broader view can be obtained by local quadratic approximation, which is equivalent to Newton's method
- In multidimensional optimization, we seek zero of gradient, so *Newton iteration* has form

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \boldsymbol{H}_f^{-1}(\boldsymbol{x}_k)\nabla f(\boldsymbol{x}_k)$$

where $\boldsymbol{H}_f(\boldsymbol{x})$ is *Hessian* matrix of second partial derivatives of f,

$$\{\boldsymbol{H}_f(\boldsymbol{x})\}_{ij} = \frac{\partial^2 f(\boldsymbol{x})}{\partial x_i \partial x_j}$$

Newton' s method

- Do not explicitly invert Hessian matrix, but instead solve linear system

$$\boldsymbol{H}_f(\boldsymbol{x}_k)\boldsymbol{s}_k = -\nabla f(\boldsymbol{x}_k)$$

for Newton step $\boldsymbol{s}_k$, then take as next iterate

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{s}_k$$

- Convergence rate of Newton's method for minimization is normally quadratic
- As usual, Newton's method is unreliable unless started close enough to solution to converge

Example

- Use Newton's method to minimize

$$f(\boldsymbol{x}) = 0.5x_1^2 + 2.5x_2^2$$

- Gradient and Hessian are given by

$$\nabla f(\boldsymbol{x}) = \begin{bmatrix} x_1 \\ 5x_2 \end{bmatrix} \quad \text{and} \quad \boldsymbol{H}_f(\boldsymbol{x}) = \begin{bmatrix} 1 & 0 \\ 0 & 5 \end{bmatrix}$$

- Taking $\boldsymbol{x}_0 = \begin{bmatrix} 5 \\ 1 \end{bmatrix}$, we have $\nabla f(\boldsymbol{x}_0) = \begin{bmatrix} 5 \\ 5 \end{bmatrix}$
- Linear system for Newton step is $\begin{bmatrix} 1 & 0 \\ 0 & 5 \end{bmatrix} \boldsymbol{s}_0 = \begin{bmatrix} -5 \\ -5 \end{bmatrix}$, so $\boldsymbol{x}_1 = \boldsymbol{x}_0 + \boldsymbol{s}_0 = \begin{bmatrix} 5 \\ 1 \end{bmatrix} + \begin{bmatrix} -5 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$, which is exact solution for this problem, as expected for quadratic function

Newton's method

- If objective function f has continuous second partial derivatives, then Hessian matrix H_f is symmetric, and near minimum it is positive definite
- Thus, linear system for step to next iterate can be solved in only about half of work required for LU factorization
- Far from minimum, $H_f(x_k)$ may not be positive definite, so Newton step s_k may not be *descent direction* for function, i.e., we may not have

$$\nabla f(x_k)^T s_k < 0$$

- In this case, alternative descent direction can be computed, such as negative gradient or direction of negative curvature, and then perform line search

Quasi-Newton methods

- Newton's method costs $\mathcal{O}(n^3)$ arithmetic and $\mathcal{O}(n^2)$ scalar function evaluations per iteration for dense problem
- Many variants of Newton's method improve reliability and reduce overhead
- *Quasi-Newton* methods have form

$$x_{k+1} = x_k - \alpha_k B_k^{-1} \nabla f(x_k)$$

  where α_k is line search parameter and B_k is approximation to Hessian matrix
- Many quasi-Newton methods are more robust than Newton's method, are superlinearly convergent, and have lower overhead per iteration, which often more than offsets their slower convergence rate

Quasi-Newton Methods

- Involve approximating the Hessian matrix
- For example, we could replace the Hessian matrix with the identity matrix I
- In this case the search direction would be:

$$p_k = -\mathbf{I} \nabla_x f(x_k)$$

- Question: What is the resulting algorithm?

Quasi-Newton Method

- Obviously substituting the identity matrix uses no real information from the Hessian matrix.
- An alternative would be to systematically derive a matrix H_k which uses curvature information akin to the Hessian matrix.
- The search direction would then be:

$$p_k = -H_k^{-1} \nabla_x f(x_k)$$

Quasi-Newton Methods

- One class of Quasi-Newton methods "build" an approximation of the Hessian matrix **H** or **B**, the inverse of the Hessian matrix
- **H** is initialized to **I**
- Consider a Taylor series expansion around x_k

$$\nabla_x f(x_{k+1}) = \nabla_x f(x_k) + \nabla^2_{xx} f(x_k)(x_{k+1} - x_k)$$

$$g_{k+1} = g_k + \mathbf{H}_{k+1} p_k$$

$$g_{k+1} - g_k = \mathbf{H}_{k+1} p_k$$

$$q_k = \mathbf{H}_{k+1} p_k$$

$$\text{Let } \mathbf{H}_{k+1}^{-1} = \mathbf{B}_{k+1}$$

$$\text{Then } \mathbf{B}_{k+1} q_k = p_k$$

Quasi-Newton Methods

- One way to generate $\mathbf{B}_{k+1}$ without computing the second derivatives is to update the current $\mathbf{B}_k$ using information available at the current iteration, say $\mathbf{B}_k^u$
- There is no unique solution for $\mathbf{B}_k^u$
- General form of the update:

$$\mathbf{B}_k^u = auu' + bvv'$$

$$\text{subject to } \mathbf{B}_{k+1} q_k = \left(\mathbf{B}_k + \mathbf{B}_k^u\right) q_k = p_k$$

- Different methods differ in terms of how $\mathbf{B}_k$ is updated (i.e., choice of constants a, b, and vectors u, v')
- Quasi-Newton methods that choose $b = 0$ yield rank 1 updates
- Quasi-Newton methods that choose $b \neq 0$ yield rank 2 updates

David Fletcher Powell Method

- Rank one updates are simple, but have limitations; Rank 2 updates are preferred
- One of the first and perhaps one of the most clever rank 2 updates is due to Davidson, Fletcher, and Powell

$$\mathbf{B}_{k+1} = \mathbf{B}_k + \mathbf{B}_k^u = \mathbf{B}_k + auu' + bvv'$$

$$p_k = \left(\mathbf{B}_k + auu' + bvv'\right) q_k$$

- DFP method chooses a, b as follows:

$$p_k = \mathbf{B}_k q_k + auu' q_k + bvv' q_k$$

$$u = p_k, \quad v = \mathbf{B}_k q_k, \quad au' q_k = 1, \quad bv' q_k = -1$$

- DFP update:

$$\mathbf{B}_{k+1} = \mathbf{B}_k + \frac{p_k p_k'}{p_k' q_k} - \frac{\mathbf{B}_k q_k q_k' \mathbf{B}_k}{q_k' \mathbf{B}_k q_k}$$

Broyden Fletcher Goldfarb Fano Method

- Recall that: $q_k = \mathbf{H}_{k+1} p_k; \quad \mathbf{H}_{k+1}^{-1} q_k = p_k$

$$\mathbf{B}_{k+1} = \mathbf{H}_{k+1}^{-1}; \quad \mathbf{B}_{k+1} q_k = p_k$$

- So any formula for update of **B** can be transformed into one for update of **H** by interchanging the role of p and q.
- BFGS update for H is obtained from DFP update for B:

$$\mathbf{H}_{k+1} = \mathbf{H}_k + \frac{q_k q_k'}{q_k' q_k} - \frac{\mathbf{H}_k p_k p_k' \mathbf{H}_k}{p_k' \mathbf{H}_k p_k}$$

- BFGS update for B is obtained from taking the inverse of H:

$$\mathbf{B}_{k+1} = \mathbf{B}_k + \left(\frac{1 + q_k' \mathbf{B}_k q_k}{q_k' p_k} \right) \frac{p_k p_k'}{p_k' q_k} - \frac{p_k q_k' \mathbf{B}_k + \mathbf{B}_k q_k p_k'}{q_k' p_k}$$

Conjugate gradient method

- Another method that does not require explicit second derivatives, and does not even store approximation to Hessian matrix, is *conjugate gradient* (CG) method
- CG generates sequence of conjugate search directions, implicitly accumulating information about Hessian matrix
- For quadratic objective function, CG is theoretically exact after at most n iterations, where n is dimension of problem
- CG is effective for general unconstrained minimization as well

CG method

```
$x_0$ = initial guess
$g_0 = \nabla f(x_0)$
$s_0 = -g_0$
for $k = 0, 1, 2, \ldots$
    Choose $\alpha_k$ to minimize $f(x_k + \alpha_k s_k)$
    $x_{k+1} = x_k + \alpha_k s_k$
    $g_{k+1} = \nabla f(x_{k+1})$
    $\beta_{k+1} = (g_{k+1}^T g_{k+1})/(g_k^T g_k)$
    $s_{k+1} = -g_{k+1} + \beta_{k+1} s_k$
end
```

- Alternative formula for β_{k+1} is

$$\beta_{k+1} = ((g_{k+1} - g_k)^T g_{k+1})/(g_k^T g_k)$$

CG method example

- Use CG method to minimize $f(x) = 0.5x_1^2 + 2.5x_2^2$
- Gradient is given by $\nabla f(x) = \begin{bmatrix} x_1 \\ 5x_2 \end{bmatrix}$
- Taking $x_0 = \begin{bmatrix} 5 & 1 \end{bmatrix}^T$, initial search direction is negative gradient,

$$s_0 = -g_0 = -\nabla f(x_0) = \begin{bmatrix} -5 \\ -5 \end{bmatrix}$$

- Exact minimum along line is given by $\alpha_0 = 1/3$, so next approximation is $x_1 = \begin{bmatrix} 3.333 & -0.667 \end{bmatrix}^T$, and we compute new gradient,

$$g_1 = \nabla f(x_1) = \begin{bmatrix} 3.333 \\ -3.333 \end{bmatrix}$$

Example (cont.)

- So far there is no difference from steepest descent method
- At this point, however, rather than search along new negative gradient, we compute instead

$$\beta_1 = (g_1^T g_1)/(g_0^T g_0) = 0.444$$

which gives as next search direction

$$s_1 = -g_1 + \beta_1 s_0 = \begin{bmatrix} -3.333 \\ 3.333 \end{bmatrix} + 0.444 \begin{bmatrix} -5 \\ -5 \end{bmatrix} = \begin{bmatrix} -5.556 \\ 1.111 \end{bmatrix}$$

- Minimum along this direction is given by $\alpha_1 = 0.6$, which gives exact solution at origin, as expected for quadratic function

Truncated Newton methods

- Another way to reduce work in Newton-like methods is to solve linear system for Newton step by iterative method
- Small number of iterations may suffice to produce step as useful as true Newton step, especially far from overall solution, where true Newton step may be unreliable anyway
- Good choice for linear iterative solver is CG method, which gives step intermediate between steepest descent and Newton-like step
- Since only matrix-vector products are required, explicit formation of Hessian matrix can be avoided by using finite difference of gradient along given vector

Remarks

- Both DFP and BFGS methods have theoretical properties that guarantee superlinear (fast) convergence rate and global convergence under certain conditions.
- However, both methods could fail for general nonlinear problems.
- DFP is highly sensitive to inaccuracies in line searches.
- Both methods can get stuck on a saddle-point. In Newton's method, a saddle-point can be detected during modifications of the (true) Hessian. Therefore, search around the final point when using quasi-Newton methods.
- Update of Hessian becomes "corrupted" by round-off and other inaccuracies.
- All kind of "tricks" such as scaling and preconditioning exist to boost the performance of the methods.

Nonlinear Conjugate Gradient

- Even though conjugate gradient is derived for a quadratic objective function, it can be applied directly to other nonlinear functions
- Several variants:
  - Fletcher-Reeves conjugate gradient (FR-CG)
  - Polak-Ribiere conjugate gradient (PR-CG)
    - More robust than FR-CG
- Compared to Newton method
  - No need for computing the Hessian matrix
  - No need for storing the Hessian matrix

Limited-Memory Quasi-Newton

- Quasi-Newton
  - Avoid computing the inverse of Hessian matrix
  - But, it still requires computing the **B** matrix which is as large as the **H**
- Limited-Memory Quasi-Newton (L-BFGS)
  - Avoids explicitly computing **B** matrix
  - Computes the updates based on a small history of p and y vectors.
  - Linear time, linear space

Free Software

- http://www.ece.northwestern.edu/~nocedal/software.html
  - L-BFGS
  - L-BFGSB

Linear Conjugate Gradient Method

- Consider optimizing the quadratic function

$$\vec{x}^* = \arg\min_{\vec{x}} \frac{\vec{x}^T \mathbf{A} \vec{x}}{2} + \vec{b}^T \vec{x}$$

- Conjugate vectors
  - The set of vectors $\{\vec{p}_1, \vec{p}_2, \ldots, \vec{p}_l\}$ is said to be conjugate with respect to a matrix **A** if $$\vec{p}_i^T \mathbf{A} \vec{p}_j = 0, \quad \text{for any } i \neq j$$
  - Important property $$\vec{x} = \alpha_1 \vec{p}_1 + \alpha_2 \vec{p}_2 + \ldots + \alpha_l \vec{p}_l$$
    - The quadratic function can be optimized by simply optimizing the function along individual directions in the conjugate set.
  - Optimal solution: is the minimizer along the kth conjugate direction

Momentum update

$$w_i(t+1) = w_i(t) + \Delta w_i(t)$$

$$\Delta w_i(t) = -\eta \left.\frac{\partial E}{\partial w_i}\right|_{w_i = w_i(t)} + \alpha \Delta w_i(t-1) \text{ where } 0 < \alpha < 1$$

$$= -\eta \sum_{\tau=0}^{t} \alpha^{t-\tau} \left.\frac{\partial E}{\partial w_i}\right|_{w_i = w_i(\tau)}$$

The momentum update allows effective learning rate to increase when feasible and decrease when necessary. Converges for $0 \leq \alpha < 1$

Learning approximations of nonlinear functions from data – the generalized delta rule

- Motivations
- Universal function approximation theorem (UFAT)
- Derivation of the generalized delta rule
- Back-propagation algorithm
- Practical considerations
- Applications

Motivations

- Psychology – Empirical inadequacy of behaviorist theories of learning – simple reward-punishment based learning models are incapable of learning functions (e.g., exclusive OR) which are readily learned by animals (e.g., monkeys)
- Artificial Intelligence – the need for learning highly <u>nonlinear</u> functions where the form of the nonlinear relationship is unknown a-priori
- Statistics – Limitations of linear regression in fitting data when the relationship is highly nonlinear and the form of the relationship is unknown
- Control – Need for nonlinear control methods

Kolmogorov' s theorem (Kolmogorov, 1940)

- Any continuous function $g(x_1,..x_N) = [0,1]^N \rightarrow \Re$ can be expressed in the form

$$g(x_1,..x_N) = \sum_{j=1}^{2N+1} g_j\left(\sum_i u_{ij}(x_i)\right) \qquad \forall (x_1,...x_N) \in [0,1]^N; N \geq 2)$$

by choosing proper nonlinearities g_j and the weights and u_{ij}

Universal function approximation theorem (UFAT) (Cybenko, 1989)

- Let $\varphi : \Re \rightarrow \Re$ be a non-constant, bounded (hence non-linear), monotone, continuous function. Let I_N be the N-dimensional unit hypercube in $\Re^N$.
- Let $C(I_N) = \{f : I_N \rightarrow \Re\}$ be the set of <u>all</u> continuous functions with domain I_N *and* range $\Re$. Then for any function $f \in C(I_N)$ and any $\varepsilon > 0$, $\exists$ an integer L and a sets of real values $\theta, \alpha_j, \theta_j, w_{ji}$ $(1 \leq j \leq L;\ 1 \leq i \leq N)$ such that

$$F(x_1, x_2 \ldots x_N) = \sum_{j=1}^{L} \boldsymbol{\alpha}_j \boldsymbol{\phi}\left(\sum_{i=1}^{N} w_{ji} x_i - \boldsymbol{\theta}_j \right) - \boldsymbol{\theta}$$

is a uniform approximation of f – that is,

$$\forall (x_1, \ldots x_N) \in I_N, \quad \left| F(x_1, \ldots x_N) - f(x_1, \ldots x_N) \right| < \varepsilon$$

Universal function approximation theorem (UFAT)

$$F(x_1, x_2 \ldots x_n) = \sum_{j=1}^{L} \alpha_j \varphi\left(\sum_{i=1}^{N} w_{ji} x_i - \theta_j \right) - \theta$$

- Unlike Kolmogorov's theorem, UFAT requires only one kind of nonlinearity to approximate any arbitrary nonlinear function to any desired accuracy
- The sigmoid function satisfies the UFAT requirements

$$\varphi(z) = \frac{1}{1+e^{-az}}; a > 0$$

$$\lim_{z \to -\infty} \varphi(z) = 0; \quad \lim_{z \to +\infty} \varphi(z) = 1$$

Similar universal approximation properties can be guaranteed for other functions – e.g., radial basis functions

Universal function approximation theorem

- UFAT guarantees the existence of arbitrarily accurate approximations of continuous functions defined over bounded subsets of $\Re^N$
- UFAT tells us the representational power a certain class of multi-layer networks relative to the set of continuous functions defined on bounded subsets of $\Re^N$
- UFAT is not constructive – it does not tell us how to choose the parameters to construct a desired function
- To learn an unknown function from data, we need an algorithm to search the hypothesis space of multilayer networks
- Generalized delta rule allows nonlinear function to be learned from the training data

Alternatives

- Brute force – select a <u>complete</u> set of nonlinear basis functions (e.g., all polynomials of degree from 0 to N) to map the N-dimensional input into a very high dimensional feature space where a linear mapping to desired outputs exists – needs too many parameters to be determined from a limited number of training samples
- Additive models – $$g\left(\sum_{i=1}^{N} g_i(x_i) + w_0\right)$$
- Select some nonlinear functions and try to adjust the parameters of the chosen nonlinear functions to fit the data – it is hard to know a priori <u>which</u> nonlinear functions to choose

Alternatives

- Projection pursuit – closely related model but differing in algorithmic details

$$\sum_{j=1}^{L} w_j f_j\left(\sum_{i=1}^{N} w_{ji}x_i + w_{j0}\right) + w_0$$

different parameters are learned in groups – first

$w_{10} \ldots w_{1N}$ then $w_{20} \ldots w_{2N}$ through $w_{J0} \ldots w_{JN}$

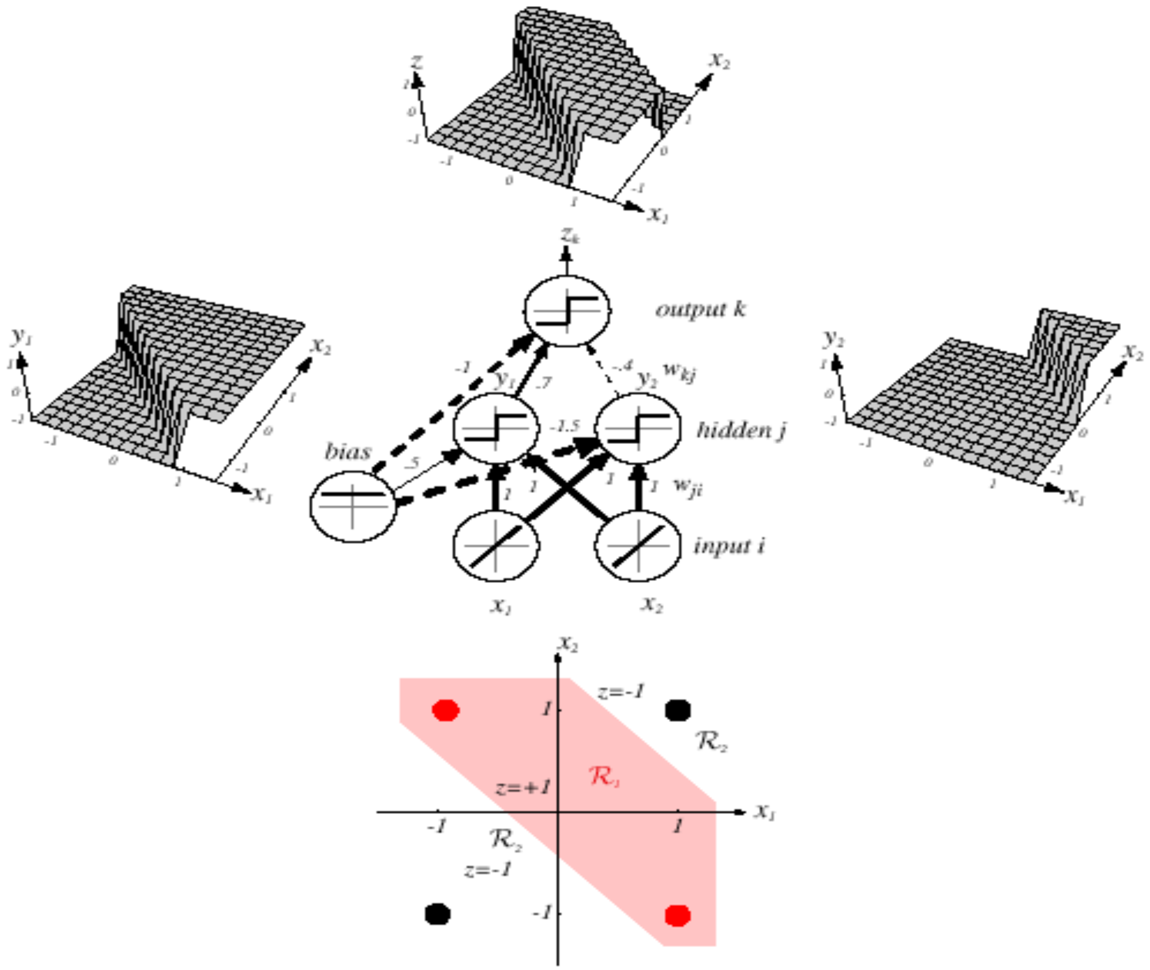
followed by $w_j \; (j = 0 \ldots J)$

iterating until some desired error criterion is met

Feed-forward neural networks

- A feed-forward n-layer network consists of n layers of nodes
- 1 layer of Input nodes
- *n*-2 layers of Hidden nodes
- 1 layer of Output nodes
- interconnected by modifiable weights from input nodes to the hidden nodes and the hidden nodes to the output nodes
- More general topologies (e.g., with connections that skip layers, e.g., direct connections between input and output nodes) are possible

A three layer network that approximates the exclusive or function



Three-layer feed-forward neural network

- A single *bias* unit is connected to each unit other than the input units
- Net *input*

$$n_j = \sum_{i=1}^{d} x_i w_{ji} + w_{j0} = \sum_{i=0}^{d} x_i w_{ji} \equiv \mathbf{W}_j . \bullet \mathbf{X},$$

- where the subscript i indexes units in the input layer, j in the hidden; w_{ji} denotes the input-to-hidden layer weights at the hidden unit j.
- The output of a hidden unit is a nonlinear function of its net input. That is, $y_j = f(n_j)$ e.g.,

$$y_j = \frac{1}{1+e^{-n_j}}$$

Three-layer feed-forward neural network

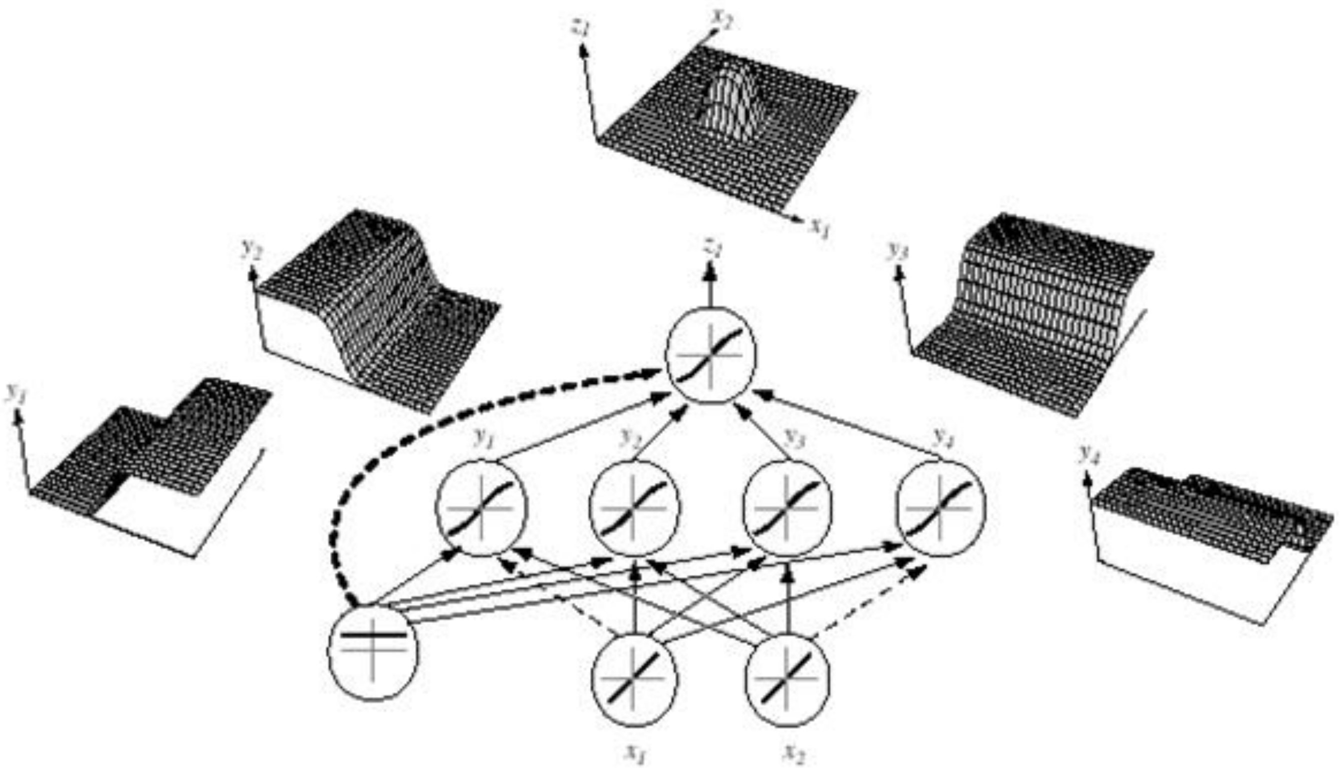
- Each output unit similarly computes its net activation based on the hidden unit signals as:

$$n_k = \sum_{j=1}^{n_H} y_j w_{kj} + w_{k0} = \sum_{j=0}^{n_H} y_j w_{kj} = \mathbf{W}_k \bullet \mathbf{Y},$$

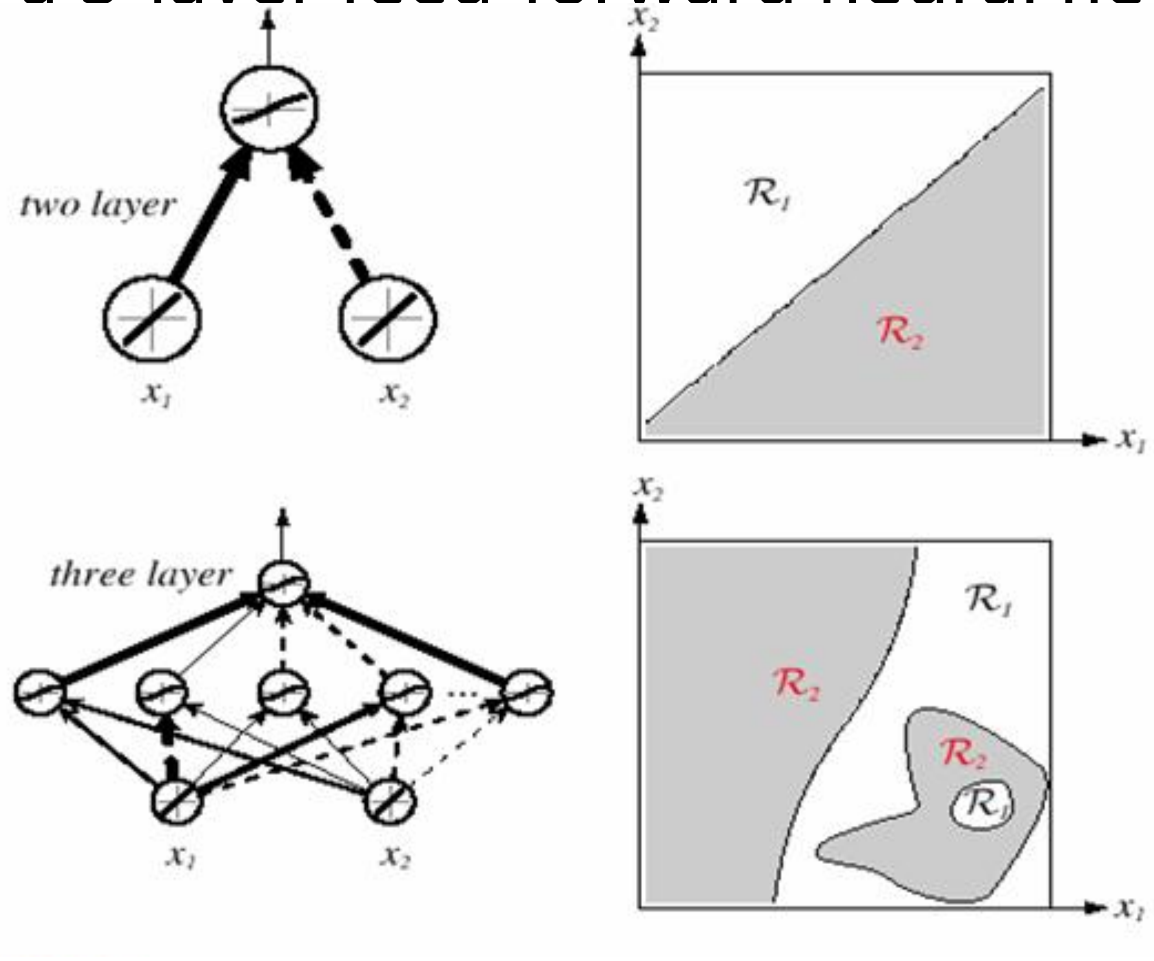
- where the subscript k indexes units in the ouput layer and n_H denotes the number of hidden units
- The output can be a linear or nonlinear function of the net input e.g.,

$$z_k = n_k$$

Computing nonlinear functions using a feed-forward neural network



Realizing non linearly separable class boundaries using a 3-layer feed-forward neural network



Learning nonlinear real-valued functions

- Given a training set determine
- Network structure – number of hidden nodes or more generally, network topology
    - Start small and grow the network
    - Start with a sufficiently large network and prune away the unnecessary connections
- For a given structure, determine the parameters (weights) that minimize the error on the training samples (e.g., the mean squared error)
- For now, we focus on the latter

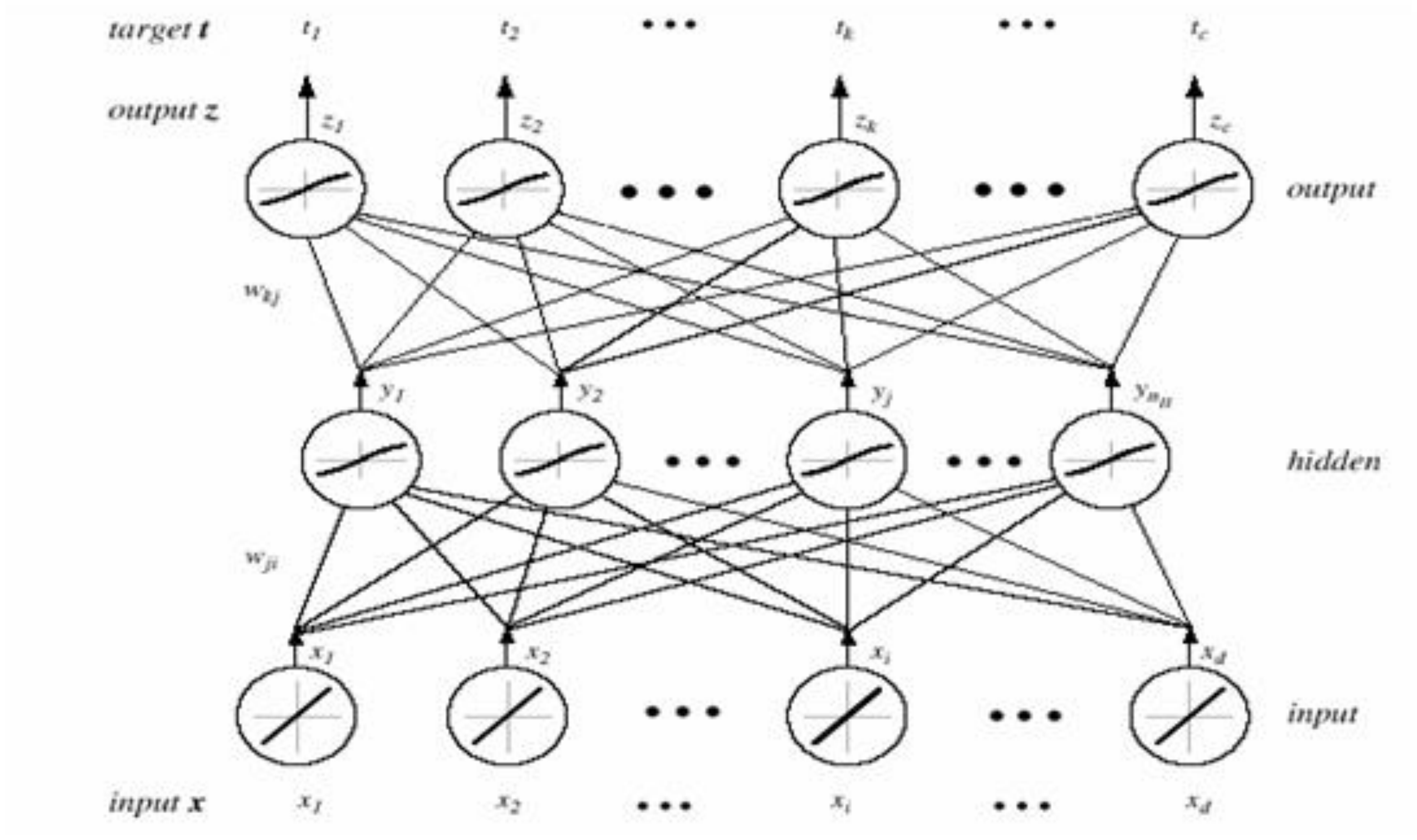
Generalized delta rule – error back-propagation

- Challenge – we know the desired outputs for nodes in the output layer, but not the hidden layer
- Need to solve the credit assignment problem – dividing the credit or blame for the performance of the output nodes among hidden nodes
- Generalized delta rule offers an elegant solution to the credit assignment problem in feed-forward neural networks in which each neuron computes a differentiable function of its inputs
- Solution can be generalized to other kinds of networks, including networks with cycles

Feed-forward networks

- Forward operation (computing output for a given input based on the current weights)
- Learning – modification of the network parameters (weights) to minimize an appropriate error measure
- Because each neuron computes a differentiable function of its inputs
  - If error is a differentiable function of the network outputs, the error is a differentiable function of the weights in the network – so we can perform gradient descent!

A fully connected 3-layer network



Generalized delta rule

- Let t_{kp} be the k-th target (or desired) output for input pattern $\mathbf{X}_p$ and z_{kp} be the output produced by k-th output node and let $\mathbf{W}$ represent all the weights in the network
- Training error: $$E_S(\mathbf{W}) = \frac{1}{2}\sum_p \sum_{k=1}^{M} (t_{kp} - z_{kp})^2 = \sum_p E_p(\mathbf{W})$$
- The weights are initialized with pseudo-random values and are changed in a direction that will reduce the error:

$$\Delta w_{ji} = -\eta \frac{\partial E_S}{\partial w_{ji}} \qquad \Delta w_{kj} = -\eta \frac{\partial E_S}{\partial w_{kj}}$$

Generalized delta rule

$\eta > 0$ is a suitable the learning rate $\mathbf{W} \leftarrow \mathbf{W} + \Delta\mathbf{W}$

Hidden–to-output weights

$$\frac{\partial E_p}{\partial w_{kj}} = \frac{\partial E_p}{\partial n_{kp}} \cdot \frac{\partial n_{kp}}{\partial w_{kj}}$$

$$\frac{\partial n_{kp}}{\partial w_{kj}} = y_{jp}$$

$$\frac{\partial E_p}{\partial n_{kp}} = \frac{\partial E_p}{\partial z_{kp}} \cdot \frac{\partial z_{kp}}{\partial n_{kp}} = -(t_{kp} - z_{kp})(1)$$

$$w_{kj} \leftarrow w_{kj} - \eta \frac{\partial E_p}{\partial w_{kj}} = w_{kj} + (t_{kp} - z_{kp}) y_{jp} = w_{kj} + \delta_{kp} y_{jp}$$

Generalized delta rule

Weights from input to hidden units

$$\frac{\partial E_p}{\partial w_{ji}} = \sum_{k=1}^{M} \frac{\partial E_p}{\partial z_{kp}} \frac{\partial z_{kp}}{\partial w_{ji}} = \sum_{k=1}^{M} \frac{\partial E_p}{\partial z_{kp}} \frac{\partial z_{kp}}{\partial y_{jp}} \cdot \frac{\partial y_{jp}}{\partial n_{jp}} \cdot \frac{\partial n_{jp}}{\partial w_{ji}}$$

$$= \sum_{k=1}^{M} \frac{\partial}{\partial z_{kp}} \left[\frac{1}{2} \sum_{l=1}^{M} (t_{lp} - z_{lp})^2 \right] (w_{kj})(y_{jp})(1 - y_{jp})(x_{ip})$$

$$= -\sum_{k=1}^{M} (t_{kp} - z_{kp})(w_{kj})(y_{jp})(1 - y_{jp})(x_{ip})$$

$$= -\underbrace{\left(\sum_{k=1}^{M} \delta_{kp}(w_{kj})(y_{jp})(1 - y_{jp}) \right)}_{\delta_{jp}} (x_{ip})$$

$$= -\delta_{jp} x_{ip}$$

$$w_{ji} \leftarrow w_{ji} + \eta \delta_{jp} x_{ip}$$

Back propagation algorithm

- Start with small random initial weights
- Until desired stopping criterion is satisfied do
- Select a training sample from S
- Compute the outputs of all nodes based on current weights and the input sample
- Compute the weight updates for output nodes
- Compute the weight updates for hidden nodes
- Update the weights

Using neural networks for classification

Network outputs are real valued.

How can we use the networks for classification?

$$F(\mathbf{X}_p) = \arg\max_k z_{kp}$$

Classify a pattern by assigning it to the class that corresponds to the index of the output node with the largest output for the pattern

Training multi-layer networks – Some Useful Tricks

- Initializing weights to small random values that place the neurons in the linear portion of their operating range for most of the patterns in the training set improves speed of convergence e.g.,

$$w_{ji} = \pm \frac{1}{2N} \sum_{i=1,\ldots,N} \frac{1}{|x_i|}$$

For input to hidden layer weights with the sign of the weight chosen at random

$$w_{kj} = \pm \frac{1}{2N} \sum_{i=1,\ldots,N} \left(\frac{1}{\varphi\left(\sum w_{ji} x_i\right)} \right)$$

For hidden to output layer weights with the sign of the weight chosen at random

Some Useful Tricks

- Use of momentum term allows the effective learning rate for each weight to adapt as needed and helps speed up convergence – in a network with 2 layers of weights,

$$\left.\begin{aligned}
w_{ji}(t+1) &= w_{ji}(t) + \Delta w_{ji}(t) \\
\Delta w_{ji}(t) &= -\eta \left.\frac{\partial E_S}{\partial w_{ji}}\right|_{w_{ji}=w_{ji}(t)} + \alpha \Delta w_{ji}(t-1) \\
w_{kj}(t+1) &= w_{kj}(t) + \Delta w_{kj}(t) \\
\Delta w_{kj}(t) &= -\eta \left.\frac{\partial E_S}{\partial w_{ji}}\right|_{w_{kj}=w_{kj}(t)} + \alpha \Delta w_{kj}(t-1)
\end{aligned}\right\}$$

where $0 < \alpha, \eta < 1$ with typical values of

$\eta = 0.5$ to 0.6, $\alpha = 0.8$ to 0.9

Some Useful Tricks

- Use sigmoid function which satisfies φ(–z)=–φ(z) helps speed up convergence

$$\varphi(z) = a\left(\frac{1-e^{-bz}}{1+e^{-bz}}\right)$$

$$a = 1.716, \mathrm{b} = \frac{2}{3} \Rightarrow \left.\frac{\partial \varphi}{\partial z}\right|_{z=0} \approx 1$$

and $\varphi(z)$ is linear in the range $-1 < z < 1$

Some Useful Tricks

- Randomize the order of presentation of training examples from one pass to the next helps avoid local minima
- Introduce small amounts of noise in the weight updates (or into examples) during training helps improve generalization – minimizes over fitting, makes the learned approximation more robust to noise, and helps avoid local minima
- If using the suggested sigmoid nodes in the output layer, set target output for output nodes to be 1 for target class and -1 for all others

Some useful tricks

- Regularization helps avoid over fitting and improves generalization

$$R(\mathbf{W}) = \lambda E(\mathbf{W}) + (1-\lambda)C(\mathbf{W}); \; 0 \le \lambda \le 1$$

$$C(\mathbf{W}) = \frac{1}{2}\left(\sum_{ji} w_{ji}^2 + \sum_{kj} w_{kj}^2\right)$$

$$-\frac{\partial C}{\partial w_{ji}} = -w_{ji} \text{ and } -\frac{\partial C}{\partial w_{kj}} = -w_{kj}$$

Start with λ close to 1 and gradually lower it during training. When λ <1, it tends to drive weights toward zero setting up a tension between error reduction and complexity minimization

Some Useful Tricks

Input and output encodings

- Do not eliminate *natural* proximity in the input or output space
  - Do not normalize input patterns to be of unit length if the length is likely to be relevant for distinguishing between classes
- Do not introduce *unwarranted* proximity as an artifact
  - Do not use $\log_2 M$ outputs to encode M classes, use M outputs instead to avoid spurious proximity in the output space
- Use error correcting codes when feasible

Some Useful Tricks

Examples of a good code

- Binary thermometer codes for encoding real values
    - Suppose we can use 10 bits to represent a value between -1.0 and +1.0
    - We can quantize the interval [-1, 1] into 10 equal parts
    - 0.38 in thermometer code is 1111000000
    - 0.60 in thermometer code is 1111110000
    - Note values that are close along the real number line have thermometer codes that are close in Hamming distance

Example of a bad code

- Ordinary binary representations of integers

Some Useful Tricks

- Normalizing inputs – know when and when not to normalize
- Scale each component of the input separately to lie between -1 and 1 with mean of 0 and standard deviation of 1

$$\mu_i = \frac{1}{P}\sum_{q=1}^{P} x_{iq}$$

$$\sigma_i^2 = \frac{1}{P}\sum_{q=1}^{P} x_{iq}^2 - \mu_i^2$$

$$x_{ip} \leftarrow \frac{\left(x_{ip} - \mu_i\right)}{\sigma_i}$$

Some Useful Tricks

Initializing weights (revisited)

Suppose weights are uniformly distributed between $-w$ and $+w$

Standardized input to a hidden neuron is distributed between $-w\sqrt{N}$ and $+w\sqrt{N}$

We want this to fall between -1 and +1 $\Rightarrow \left(w = \frac{1}{\sqrt{N}} \right)$

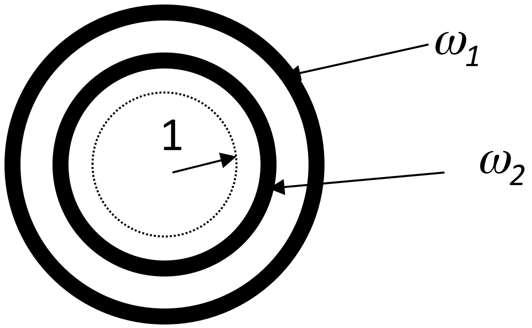
$$\Rightarrow -\frac{1}{\sqrt{N}} < w_{ji} < \frac{1}{\sqrt{N}}$$

$$-\frac{1}{\sqrt{n_H}} < w_{kj} < \frac{1}{\sqrt{n_H}}$$

Some Useful Tricks

- Normalizing inputs – know when and when not to normalize
- Normalizing each input pattern so that it is of unit length is commonplace, but often inappropriate

$$\mathbf{X}_p \leftarrow \frac{\mathbf{X}_p}{\|\mathbf{X}_p\|}$$

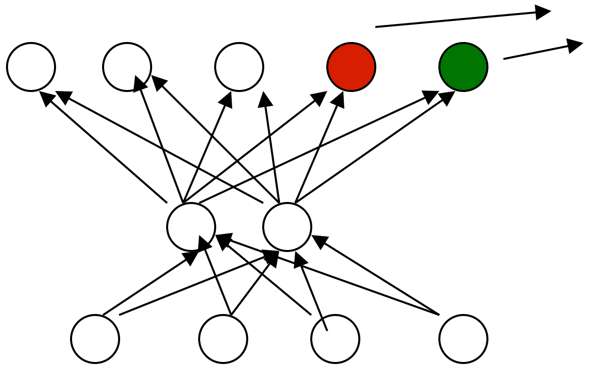


Some Useful Tricks

- Use of problem specific information (if known) speeds up convergence and improves generalization
- In networks designed for translation-invariant visual image classification, building in translation invariance as a constraint on the weights helps
- If we know the function to be approximated is smooth, we can build that in as part of the criterion to be minimized – minimize in addition to the error, the gradient of the error with respect to the <u>inputs</u>

Some Useful Tricks

- Manufacture training data – training networks with translated and rotated patterns if translation and rotation invariant recognition is desired
- Incorporate hints during training
- Hints are used as additional outputs during training to help shape the hidden layer representation



Hint nodes (e.g., vowels versus consonants in training a phoneme recognizer)

Some Useful Tricks

- Reducing the effective number of free parameters (degrees of freedom) helps improve generalization
- Regularization
- Preprocess the data to reduce the dimensionality of the input –
  - Train a neural network with output same as input, but with fewer hidden neurons than the number of inputs
  - Use the hidden layer outputs as inputs to a second network to do function approximation

Some Useful Tricks

- Choice of appropriate error function is critical – do not blindly minimize sum squared error – there are many cases where other criteria are appropriate
- Example

$$E_S(\mathbf{W}) = \sum_{p=1}^{P} \sum_{k=1}^{M} t_{kp} \ln\left(\frac{t_{kp}}{z_{kp}} \right)$$

is appropriate for minimizing the distance between the target probability distribution over the *M* output variables and the probability distribution represented by the network

Some Useful Tricks

- Interpreting the outputs as class conditional probabilities
- Use exponential output nodes

$$n_{kp} = \sum_{j=0}^{n_H} w_{kj} y_{jp}$$

$$\text{linear output } z_{kp} = \left(\frac{n_{kp}}{\sum_{l=1}^{M} n_{lp}} \right)$$

$$\text{exponential output } z_{kp} = \left(\frac{e^{n_{kp}}}{\sum_{l=1}^{M} n_{kp}} \right)$$

Bayes classification and Neural Networks

$$P(\omega_k \mid \mathbf{X}) = \frac{P(\mathbf{X} \mid \omega_k) P(\omega_k)}{\sum_{l=1}^{M} P(\mathbf{X} \mid \omega_l) P(\omega_l)}$$

$$\left.\begin{array}{l} t_k(\mathbf{X}_p) = t_{kp} = 1 \text{ if } \mathbf{X}_p \in \omega_k \\ t_k(\mathbf{X}_p) = t_{kp} = 0 \text{ if } \mathbf{X}_p \notin \omega_k \end{array}\right\} k\text{th target output}$$

$$g_k(\mathbf{X}_p; \mathbf{W}) = k\text{th output for input } \mathbf{X}_p$$

$$E_S(\mathbf{W}) = \sum_{p=1}^{P} \left(g_k(\mathbf{X}_p; \mathbf{W}) - t_{kp} \right)^2$$

$$= \sum_{\mathbf{X}_p \in \omega_k} \left(g_k(\mathbf{X}_p; \mathbf{W}) - 1 \right)^2 + \sum_{\mathbf{X}_p \notin \omega_k} \left(g_k(\mathbf{X}_p; \mathbf{W}) - 0 \right)^2$$

Bayes classification and Neural Networks

$$\lim_{|S|\to\infty}\frac{1}{|S|}E_S(\mathbf{W}) = P(\omega_k)\int (g_k(\mathbf{X};\mathbf{W})-1)^2 P(\mathbf{X}\mid\omega_k)d\mathbf{X} + P(\omega_{i\neq k})\int g_k{}^2(\mathbf{X};\mathbf{W})P(\mathbf{X}\mid\omega_{i\neq k})d\mathbf{X}$$

$$= \int g_k{}^2(\mathbf{X};\mathbf{W})P(\mathbf{X})d\mathbf{X} - 2\int g_k(\mathbf{X};\mathbf{W})P(\mathbf{X},\omega_k)d\mathbf{X} + \int P(\mathbf{X},\omega_k)d\mathbf{X}$$

$$= \int (g_k(\mathbf{X};\mathbf{W}) - P(\omega_k\mid\mathbf{X}))^2 P(\mathbf{X})d\mathbf{X} + \underbrace{\int P(\omega_k\mid\mathbf{X})P(\omega_{i\neq k}\mid\mathbf{X})P(\mathbf{X})d\mathbf{X}}_{\text{independent of } \mathbf{W}}$$

Because generalized delta rule minimizes this quantity with respect to **W**, we have

$$g_k(\mathbf{X};\mathbf{W}) \approx P(\omega_k\mid\mathbf{X})$$

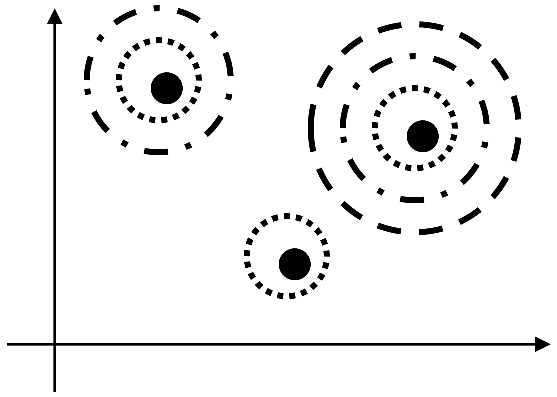
Assuming that the network is expressive enough to represent

$$P(\omega_k\mid\mathbf{X})$$

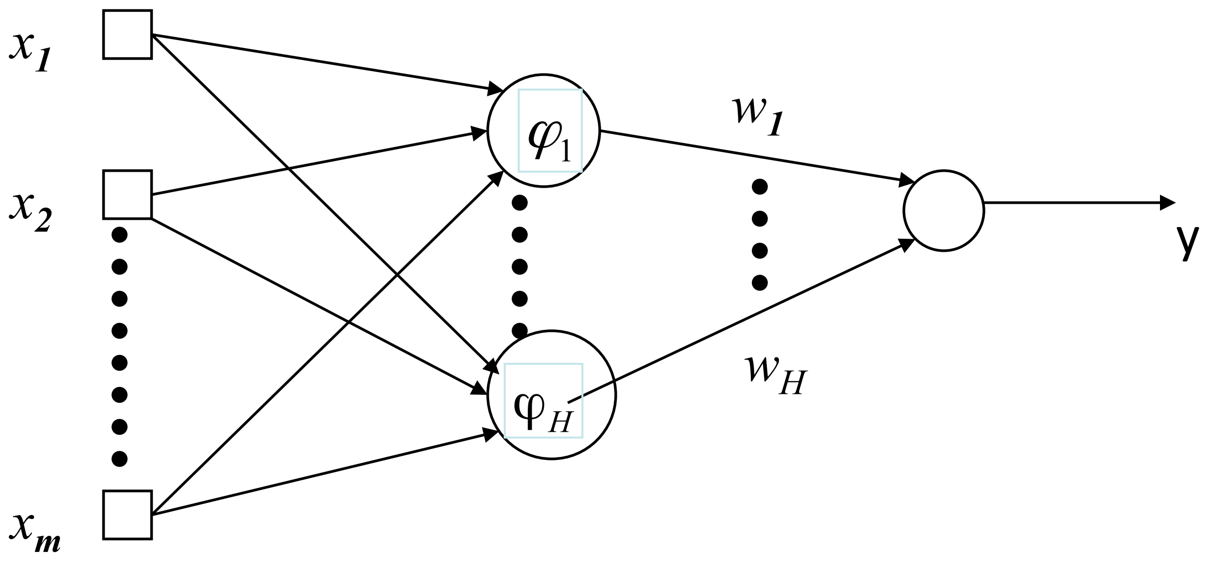
Radial-Basis Function Networks

- A function is approximated as a linear combination of radial basis functions (RBF). RBFs capture local behaviors of functions.

- RBFs represent <u>local</u> receptive fields



Radial Basis Function Networks



- Hidden layer applies a non-linear transformation from the input space to the hidden space.
- Output layer applies a linear transformation from the hidden space to the output space.

ϕ-separability of patterns

$$\varphi(\mathbf{x}) = <\varphi_1(\mathbf{x}),\ldots,\varphi_H(\mathbf{x})>$$

$$\varphi_i$$

Hidden layer representation

$$\{\varphi_i(\mathbf{x})\}_{i=1}^{H}$$

A (binary) partition, also called dichotomy, (C_1, C_2) of the training set C is ϕ-separable if there is a vector w of dimension H such that:

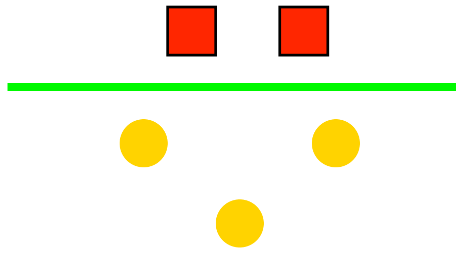
$$\mathbf{W} \bullet \varphi(\mathbf{X}) > 0 \quad \mathbf{X} \in C_1$$

$$\mathbf{W} \bullet \varphi(\mathbf{X}) < 0 \quad \mathbf{X} \in C_2$$

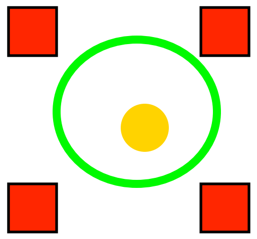
Examples of ϕ-separability

- Separating surface: $\mathbf{A} \bullet \varphi(\mathbf{X}) = 0$
- Examples of separable partitions (C1,C2):

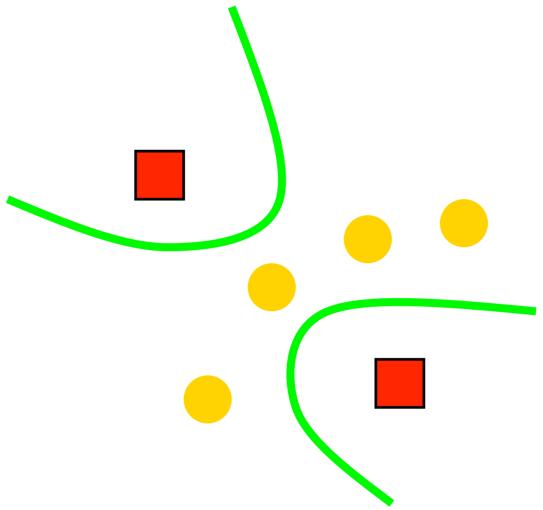
Linearly separable:



Spherically separable:



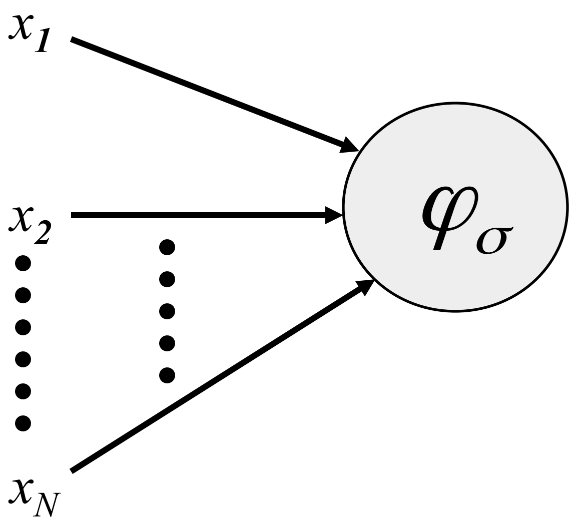
Quadratically separable:



Example of a radial basis function

- Hidden units: use a radial basis function

$\phi\sigma(|| \mathbf{X} - \mathbf{W}||^2)$ the output depends on the distance of the input x from the center t



$\phi_\sigma(|| \mathbf{X} - \mathbf{W}||^2)$

W is called center
σ is called spread
center and spread are parameters

Radial basis function

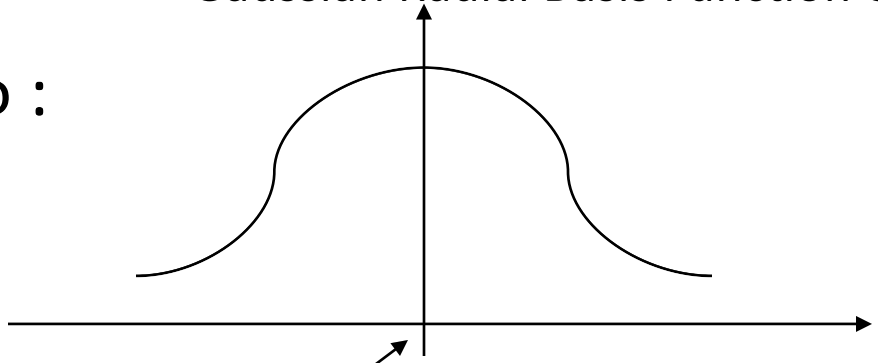
- A hidden neuron is more sensitive to data points near its center. This sensitivity may be tuned by adjusting the spread σ.

- Larger spread $\Rightarrow$ less sensitivity

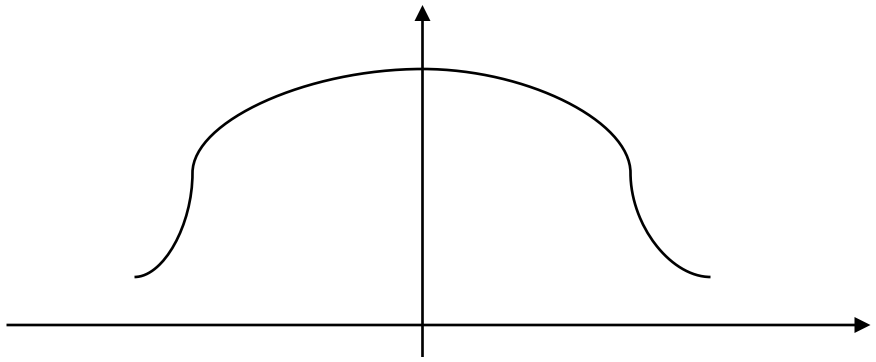
- Neurons in the visual cortex have locally tuned frequency responses.

Gaussian Radial Basis Function ϕ

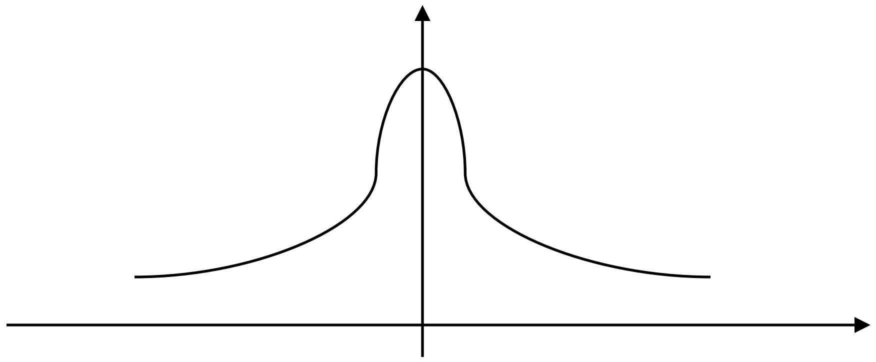
ϕ :



σ is a measure of how spread the curve is:



Large σ



Small σ

Types of ϕ

- Multiquadrics

$$\varphi(r) = (r^2 + c^2)^{1/2}$$

$$c > 0$$

$$r = \| \mathbf{X} - \mathbf{W} \|$$

- Inverse multiquadrics

$$\varphi(r) = \frac{1}{(r^2 + c^2)^{1/2}}$$

$$c > 0$$

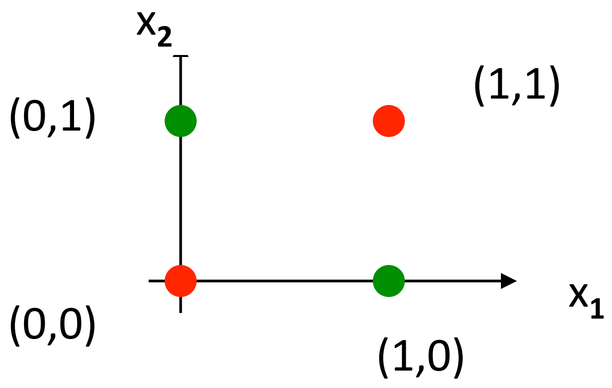
- Gaussian functions:

$$\varphi(r) = \exp\left(-\frac{r^2}{2\sigma^2} \right)$$

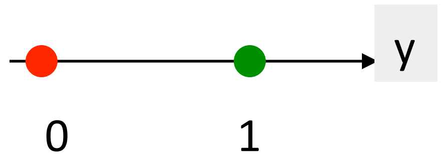
$$\sigma > 0$$

Implementing Exclusive OR using an RBF network

- Input space:



- Output space:



- Construct an RBF pattern classifier such that:

  (0,0) and (1,1) are mapped to 0, class C1

  (1,0) and (0,1) are mapped to 1, class C2

Exclusive OR revisited

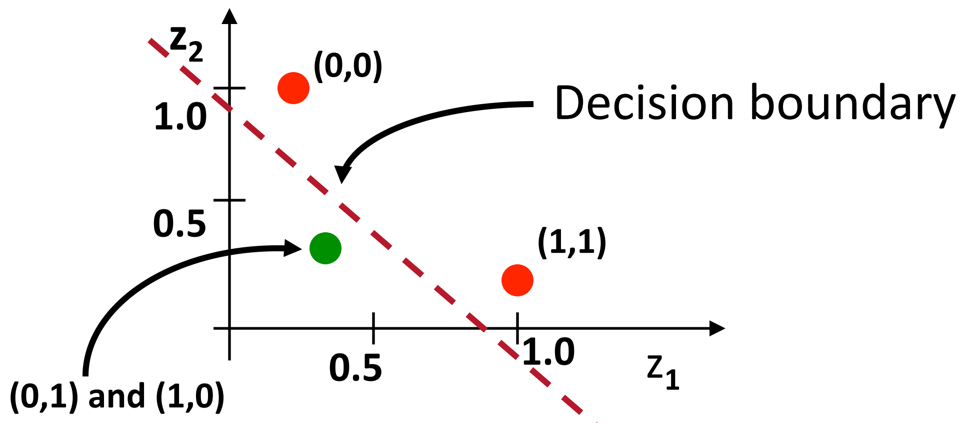
In the feature (hidden) space:

$$\varphi_1(x_1,x_2) = e^{-\|\mathbf{X}-\mathbf{W}_1\|^2} = z_1$$

$$\varphi_2(x_1,x_2) = e^{-\|\mathbf{X}-\mathbf{W}_2\|^2} = z_2$$

$$\mathbf{W}_1 = [1,1]^T$$

$$\mathbf{W}_2 = [0,0]^T$$



When mapped into the feature space < z_1 , z_2 >, C1 and C2 become *linearly separable.* So a linear classifier with φ_1(x) and φ_2(x) as inputs can be used to solve the XOR problem.

RBF Learning Algorithm

$$\Delta\sigma_j = -\eta_{\sigma_j} \frac{\partial E_S}{\partial \sigma_j}$$

$$\Delta\alpha_j = -\eta_j \frac{\partial E_S}{\partial \alpha_j}$$

$$\Delta w_{ji} = -\eta_{ji} \frac{\partial E_S}{\partial w_{ji}}$$

Depending on the specific function can be computed using the chain rule of calculus

RBF Learning Algorithm

$$z_{jp} = e^{-\frac{-\|\mathbf{x}_p - \mathbf{w}_j\|^2}{2\sigma_j^2}}$$

$$y_p = \sum_{j=0}^{L} \alpha_j z_{jp}$$

$$E_p = \frac{1}{2}\left(t_p - y_p\right)^2$$

$$\mathbf{X}_p = \left[x_{1p} \ldots\ldots\ldots x_{Np}\right]^T$$

$$\mathbf{W}_j = \left[w_{j1} \ldots\ldots w_{jN}\right]^T$$

RBF Learning Algorithm

$$\Delta\alpha_j = -\eta_j \frac{\partial E_p}{\partial \alpha_j} = \eta_j (t_p - y_p) z_{jp}$$

$$\alpha_j \leftarrow \alpha_j + \eta_j (t_p - y_p) z_{jp}$$

$$\frac{\partial E_p}{\partial w_{ji}} = \frac{\partial E_p}{\partial y_p} \frac{\partial y_p}{\partial z_{jp}} \frac{\partial z_{jp}}{\partial w_{ji}}$$

$$= -(t_p - y_p) \alpha_j \left(\frac{z_{jp}}{\sigma_j^2} \right) (x_{ip} - w_{ji})$$

$$w_{ji} = w_{ji} + \eta_{ji} (t_p - y_p) \alpha_j \left(\frac{z_{jp}}{\sigma_j^2} \right) (x_{ip} - w_{ji})$$

RBF Learning Algorithm

$$\frac{\partial E_p}{\partial \sigma_j} = \frac{\partial E_p}{\partial y_p}\frac{\partial y_p}{\partial z_{jp}}\frac{\partial z_{jp}}{\partial \sigma_j}$$

$$= -\left(t_p - y_p\right)\alpha_j\left(-z_{jp}\right)\left(\left(\frac{2}{\sigma_j}\right)\left(\ln z_{jp}\right)\right)$$

$$\sigma_j \leftarrow \sigma_j - \eta_j\left(t_p - y_p\right)\alpha_j\left(z_{jp}\right)\left(\left(\frac{2}{\sigma_j}\right)\left(\ln z_{jp}\right)\right)$$

RBF Learning Algorithm (continued)

Some useful facts

$$\|V\|^2 = V^T V \quad \text{(norm)}$$

$$\|V\|_C^2 = (CV)^T (CV) = V^T C^T C V \quad \text{(weighted norm)}$$

$$\|V\|_C^2 = \|V\|^2 \text{ if } C^T C = \text{identity matrix}$$

$$\frac{d}{d\mathbf{X}}(A\mathbf{X}) = A$$

$$\frac{d}{d\mathbf{X}}\left(\mathbf{X}^T A\mathbf{X}\right) = 2A\mathbf{X} \text{ (when A is a symmetric matrix)}$$

$$\frac{d}{dA}\left(\mathbf{X}^T A\mathbf{X}\right) = \mathbf{X}^T\mathbf{X}$$

RBF Learning Algorithm

$$z_{jp} = e^{-\frac{1}{2}(\mathbf{x}_p - \mathbf{w}_j)^T \Sigma_j (\mathbf{x}_p - \mathbf{w}_j)}$$

$$y_p = \sum_{j=0}^{L} \alpha_j z_{jp}$$

$$E_p = \frac{1}{2}\left(t_p - y_p\right)^2$$

$$\mathbf{X}_p = \left[x_{1p} \ldots\ldots\ldots x_{Np}\right]^T$$

$$\mathbf{W}_j = \left[w_{j1} \ldots\ldots w_{jN}\right]^T$$

Exercise: Derive the weight update equations from first principles

RBF Learning Algorithm

$$\Delta\alpha_j = -\eta_j \frac{\partial E_p}{\partial \alpha_j} = \eta_j (t_p - y_p) z_{jp}$$

$$\alpha_j \leftarrow \alpha_j + \eta_j (t_p - y_p) z_{jp}$$

$$\frac{\partial E_p}{\partial w_{ji}} = \frac{\partial E_p}{\partial y_p} \frac{\partial y_p}{\partial z_{jp}} \frac{\partial z_{jp}}{\partial w_{ji}}$$

$$= -(t_p - y_p)\alpha_j \left(\frac{z_{jp}}{\sigma_j^2}\right)(x_{ip} - w_{ji})$$

$$w_{ji} = w_{ji} + \eta_{ji} (t_p - y_p)\alpha_j \left(\frac{z_{jp}}{\sigma_j^2}\right)(x_{ip} - w_{ji})$$

$$\frac{\partial E_p}{\partial \sigma_j} = \frac{\partial E_p}{\partial y_p} \frac{\partial y_p}{\partial z_{jp}} \frac{\partial z_{jp}}{\partial \sigma_j}$$

$$= -(t_p - y_p)\alpha_j (-z_{jp})\left(\left(\frac{2}{\sigma_j}\right)(\ln z_{jp})\right)$$

RBF Learning Algorithm (continued)

More general form of radial basis function

C_j^{-1} is the inverse of an $N \times N$ covariance matrix

Note that the covariance matrix is symmetric

$$z_{jp} = e^{-\left(\mathbf{W}_j - \mathbf{X}_p\right)^T C_j^{-1} \left(\mathbf{W}_j - \mathbf{X}_p\right)}$$

Exercise: derive a learning rule for an RBF network with such neurons in the hidden layer and linear neurons in the output layer

RBF Learning Algorithm (continued)

Some useful facts

$\|V\|^2 = V^T V$ (norm)

$\|V\|_C^2 = (CV)^T (CV) = V^T C^T CV$ (weighted norm)

$\|V\|_C^2 = \|V\|^2$ if $C^T C$ = identity matrix

$$\frac{d}{d\mathbf{X}}(A\mathbf{X}) = A$$

$$\frac{d}{d\mathbf{X}}(\mathbf{X}^T A\mathbf{X}) = 2A\mathbf{X}$$ (when A is a symmetric matrix)

$$\frac{d}{dA}(\mathbf{X}^T A\mathbf{X}) = \mathbf{X}^T\mathbf{X}$$

RBF Learning Algorithm

- Initialize the parameters -- centers of the hidden neurons are typically initialized to coincide with a subset of the training set
- Use gradient descent to adjust the parameters using the training data until the desired performance criterion is satisfied

From Neural Networks to Deep Neural Networks

Typical goal of machine learning

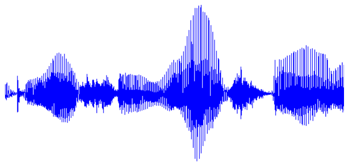
<u>input</u> <u>output</u>

images/video

$\xrightarrow{}\boxed{\text{ML}}\xrightarrow{}$

Label: “Motorcycle”
Suggest tags
Image search
…

audio



$\xrightarrow{}\boxed{\text{ML}}\xrightarrow{}$

Speech recognition
Music classification
Speaker identification
…

text

The New York Times
www.nytimes.com
HOME PAGE | TODAY'S PAPER | VIDEO | MOST POPULAR | TIMES TOPICS
Get Home Delivery-Bay Area | Log In | Register Now | TimesPeople

The New York Times
Tuesday, March 8, 2011 Last Update: 7:47 PM ET

Search
Follow Us | Subscribe to Home Delivery-Bay Area | Personalize Your Weather

Switch to Global Edition »
JOBS
REAL ESTATE
AUTOS
ALL CLASSIFIEDS
WORLD
U.S.
POLITICS
NEW YORK
BUSINESS
DEALBOOK
TECHNOLOGY
SPORTS
SCIENCE
HEALTH
OPINION
ARTS
Books
Movies
Music
Television
Theater
STYLE
Dining & Wine
Fashion & Style
Home & Garden
Weddings/Celebrations
TRAVEL
All Blogs
Cartoons
Classifieds
Corrections
Crossword/Games
Education
First Look
Learning Network

Opposition in Libya Struggles to Form a United Front
By ANTHONY SHADID and KAREEM FAHIM 45 minutes ago
Libyan rebels are facing a great and perhaps fatal obstacle: the need to forge a substitute to Col. Muammar el-Qaddafi in a state that he embodied.
- Qaddafi Forces Renew Assault Against Rebels on 2 Fronts 2:29 PM ET
- At War: Qaddafi's Arms Bazaar, Slowly Exposed

The Lede: Updates From the Region
By J. DAVID GOODMAN Live
Colonel Qaddafi appears at a hotel and shuns the press; Saudi Arabia braces for protests.

Tuesday TimesCast

Today's TimesCast
Libyan rebels try to set up a government; journalists wait for Colonel Qaddafi; Democrats and Republicans differ on the budget.

Amy Sancetta/Associated Press

Ohio State Suspends Tressel for 2 Games
By THE ASSOCIATED PRESS 8 minutes ago
Football coach Jim Tressel was suspended for two games and fined $250,000 for violating N.C.A.A. rules.

Gates Sees Crucial Test for U.S. in Afghanistan
By ELISABETH BUMILLER 5:02 PM ET
Defense Secretary Robert M. Gates said that the United States faces an "acid test" this spring and summer to determine if gains in the war are sustainable.

Prosecutors Want Psychiatric Testing for Loughner
By MARC LACEY 2:26 PM ET
Prosecutors want to know if Jared L. Loughner, the suspect in the Tucson shooting, is competent to stand trial.

In a Wealthy Suburb, Concern Over School Taxes
By LOUIS UCHITELLE 2:22 PM ET
Even in Bronxville, N.Y., where typical income is high, a look at the property tax trajectory has residents worried.

OPINION »
Fixes: Coming Together to Give Schools a Boost
A group of Cincinnati leaders are improving schools by using data for progress, not punishment.
- Brooks: New Humanism
- Herbert: Flailing After Muslims
- Cohen: Libyan Closure
- Editorial: Peter King's Obsession
- Kristof Blog: Do Women Leaders Matter?
- Brooks Blog: Body and Spirit

MARKETS » At close 03/08/2011

S.&P. 500	Dow	Nasdaq
1,321.82	12,214.38	2,765.77
+11.69	+124.35	+20.14
+0.89%	+1.03%	+0.73%

GET QUOTES My Portfolios »
Stock, ETFs, Funds Go

TRAVEL »
FRUGAL TRAVELER BLOG
Night Life and Beaches in Brazil, for Less Than the Cost of Rio
Seth Kugel discovers a beachy vacation with night life in the Amazon.

Cambodia's Eco-Tourism
Roads and bridges, and political stability, have led to new accommodations.

Log In With Facebook
Log in to see what your friends are sharing on nytimes.com. Privacy Policy | What's This?

WHAT'S POPULAR NOW
- Gates Calls for a Final Push to Eradicate Polio
- Museums Compete for NASA's Soon-to-be-Retired Space Shuttles

$\xrightarrow{}\boxed{\text{ML}}\xrightarrow{}$

Web search
Anti-spam
Machine translation
…

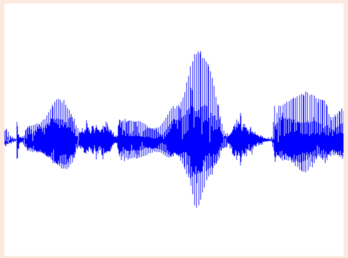
Typical goal of machin

Feature engineering

input | tput

images/video

audio



text

ML → Label: "Motorcycle"
Suggest tags
Image search
…

ML → Speech recognition
Music classification
Speaker identification
…

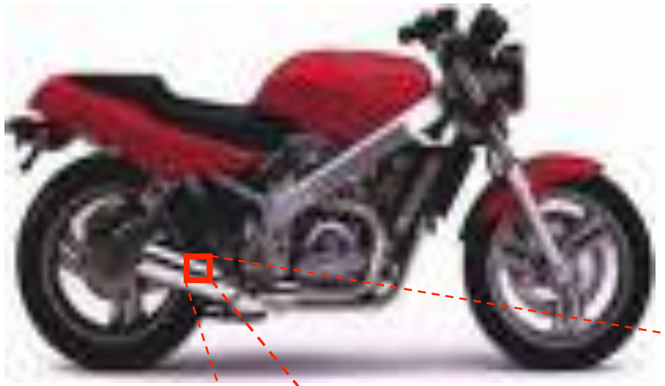
ML → Web search
Anti-spam
Machine translation
…

Object classification

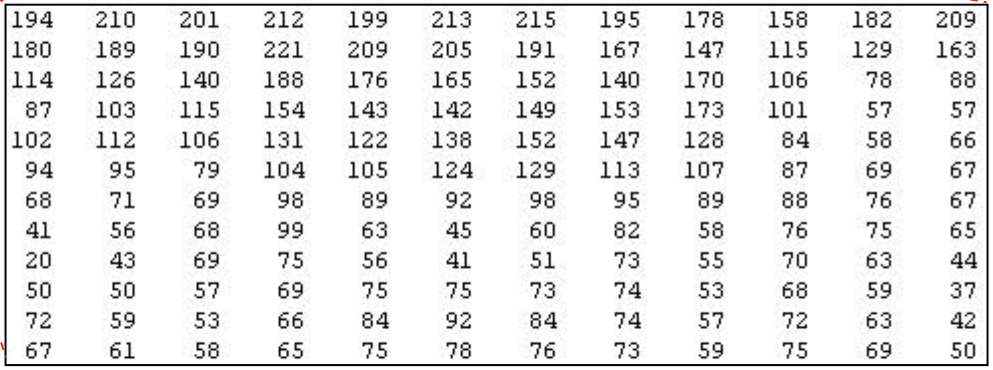
$\longrightarrow \boxed{\text{ML}} \longrightarrow$ “motorcycle”

Why is this hard?

You see this:

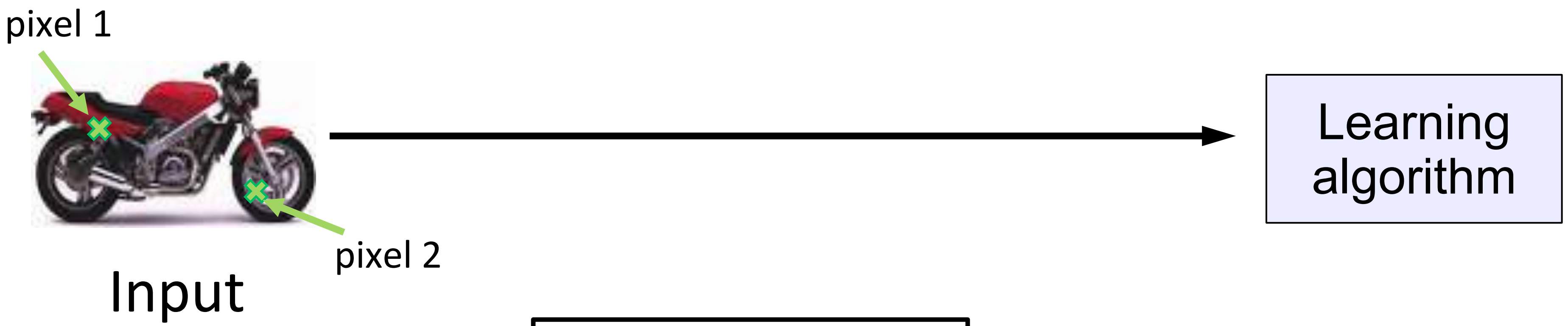


But the camera sees this:

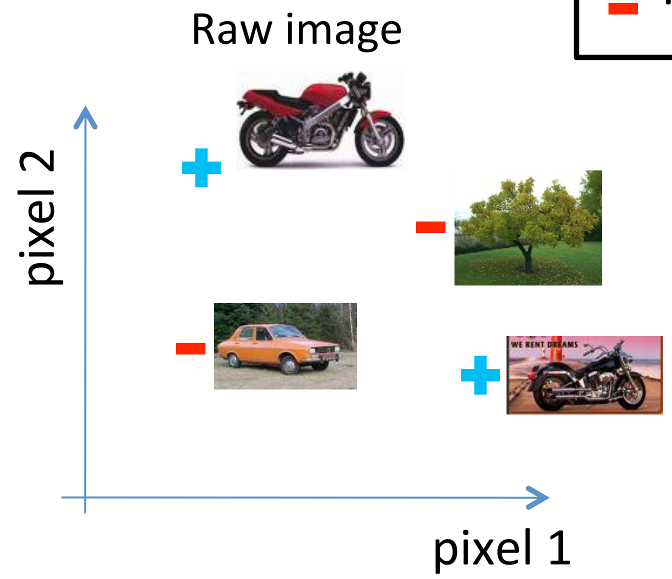


194	210	201	212	199	213	215	195	178	158	182	209
180	189	190	221	209	205	191	167	147	115	129	163
114	126	140	188	176	165	152	140	170	106	78	88
87	103	115	154	143	142	149	153	173	101	57	57
102	112	106	131	122	138	152	147	128	84	58	66
94	95	79	104	105	124	129	113	107	87	69	67
68	71	69	98	89	92	98	95	89	88	76	67
41	56	68	99	63	45	60	82	58	76	75	65
20	43	69	75	56	41	51	73	55	70	63	44
50	50	57	69	75	75	73	74	53	68	59	37
72	59	53	66	84	92	84	74	57	72	63	42
67	61	58	65	75	78	76	73	59	75	69	50

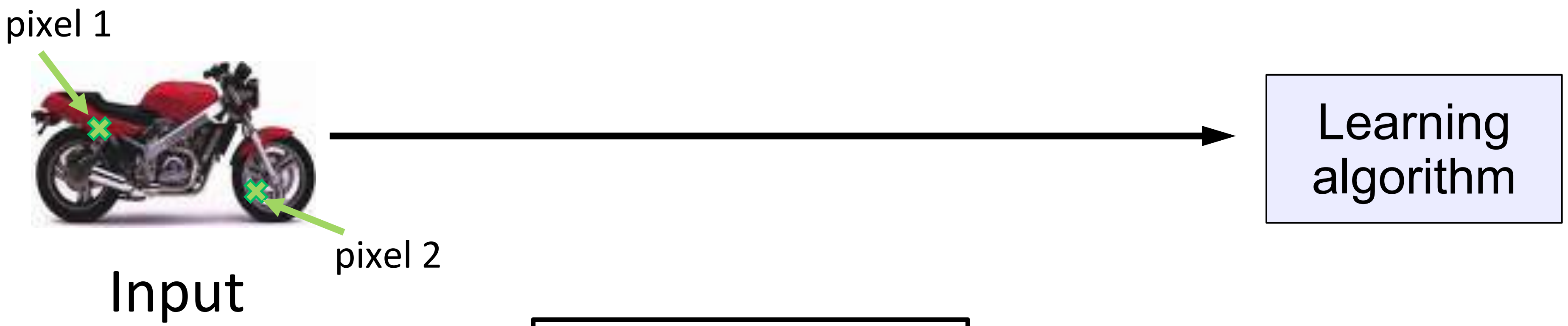
Pixel-based representation



+ Motorbikes
− “Non”-Motorbikes

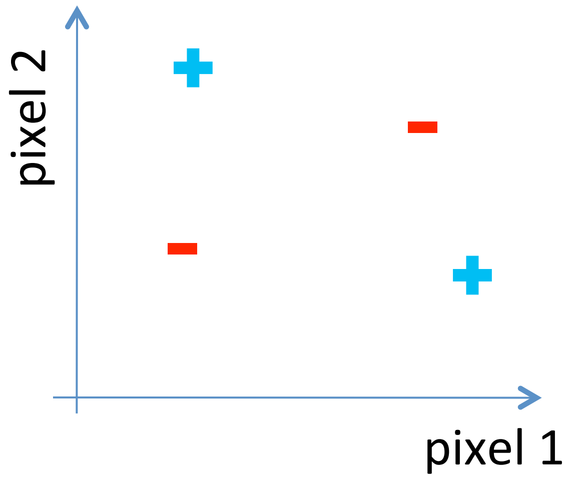


Pixel-based representation

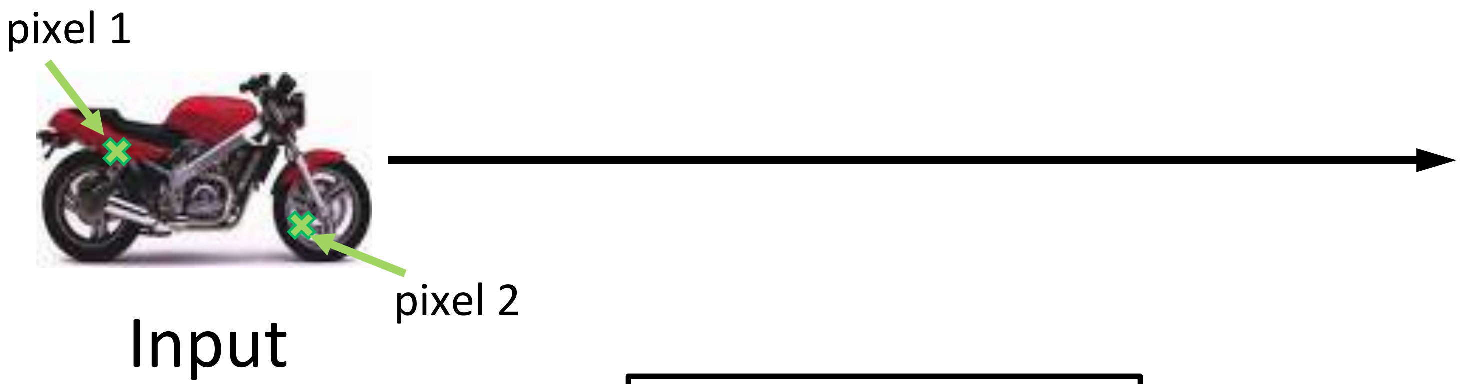


Motorbikes
"Non"-Motorbikes

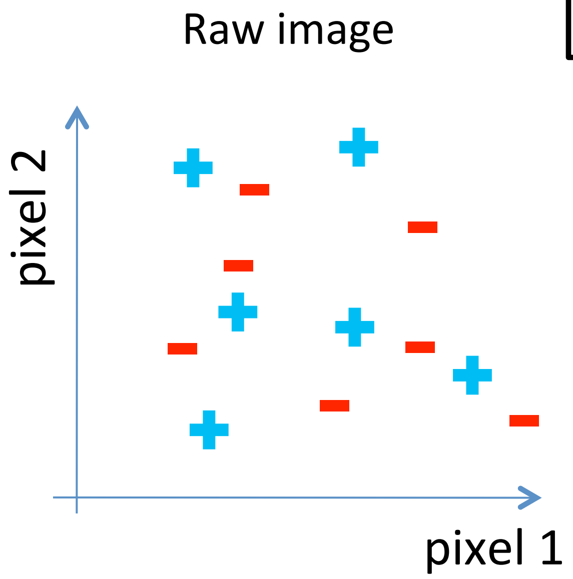
Raw image



Pixel-based representation



+ Motorbikes
– "Non"-Motorbikes



What we want

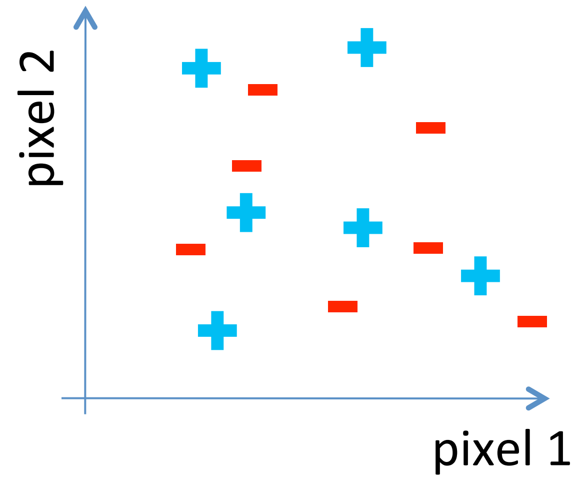
Input

$$\begin{tikzcd} \text{Input} \arrow[r] & \boxed{\text{Feature representation}} \arrow[r] & \boxed{\text{Learning algorithm}} \end{tikzcd}$$

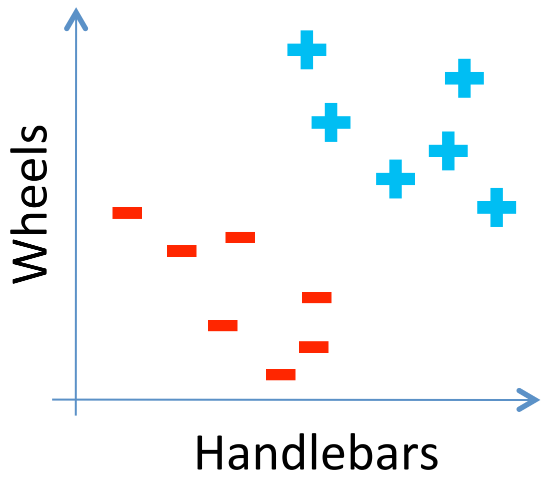
E.g., Does it have Handlebars? Wheels?

✚	Motorbikes
▬	"Non"-Motorbikes

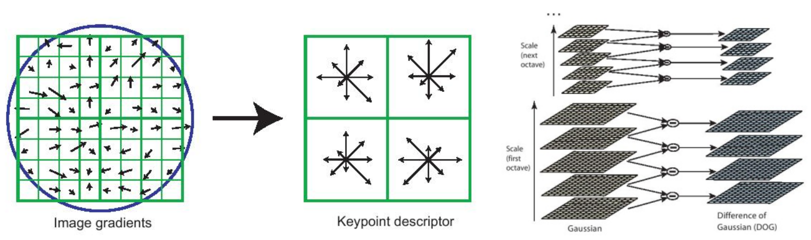
Raw image



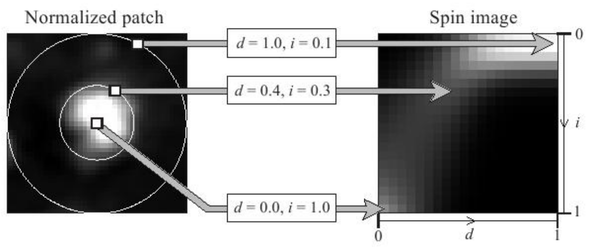
Features



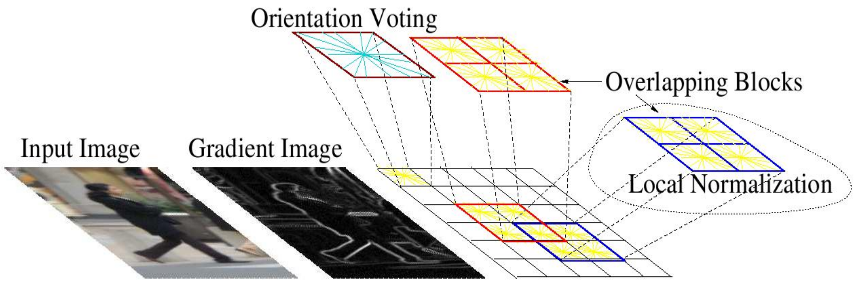
Some feature representations



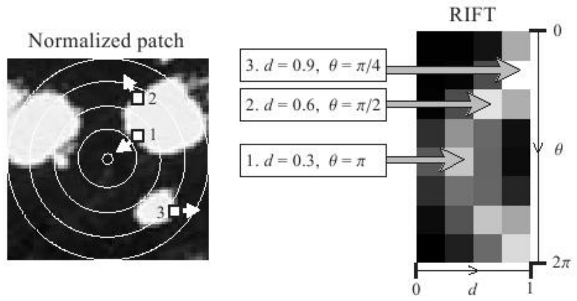
SIFT



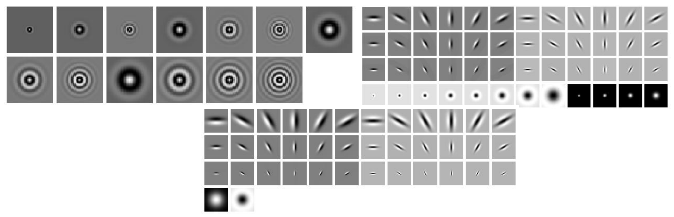
Spin image



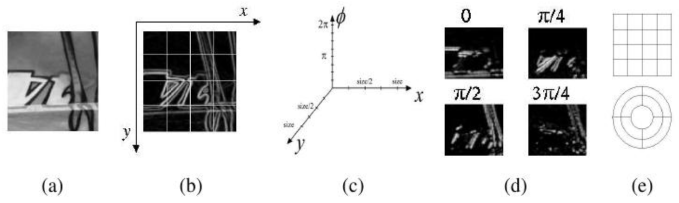
HoG


RIFT
Normalized patch
3. $d = 0.9, \theta = \pi/4$
2. $d = 0.6, \theta = \pi/2$
1. $d = 0.3, \theta = \pi$
0
θ
2π
0
d
1

RIFT

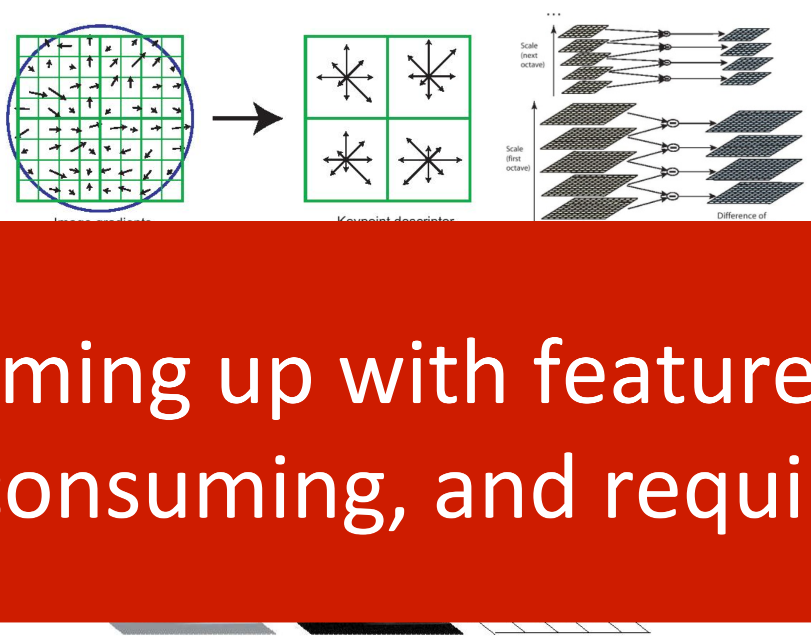


Textons



GLOH

Some feature representations

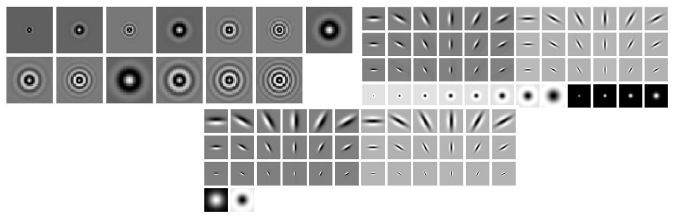


HoG

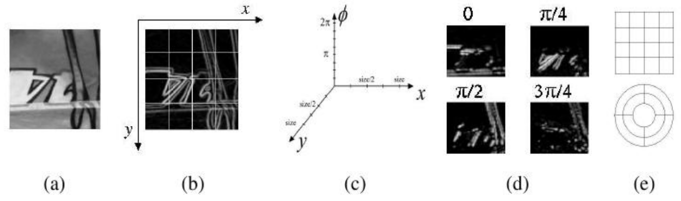
Normalized patch
Spin image
d = 1.0, i = 0.1
d = 0.4, i = 0.3
d = 0.0, i = 1.0
0
i
1
0
d
1

RIFT

Coming up with features is often difficult, time-consuming, and requires expert knowledge.

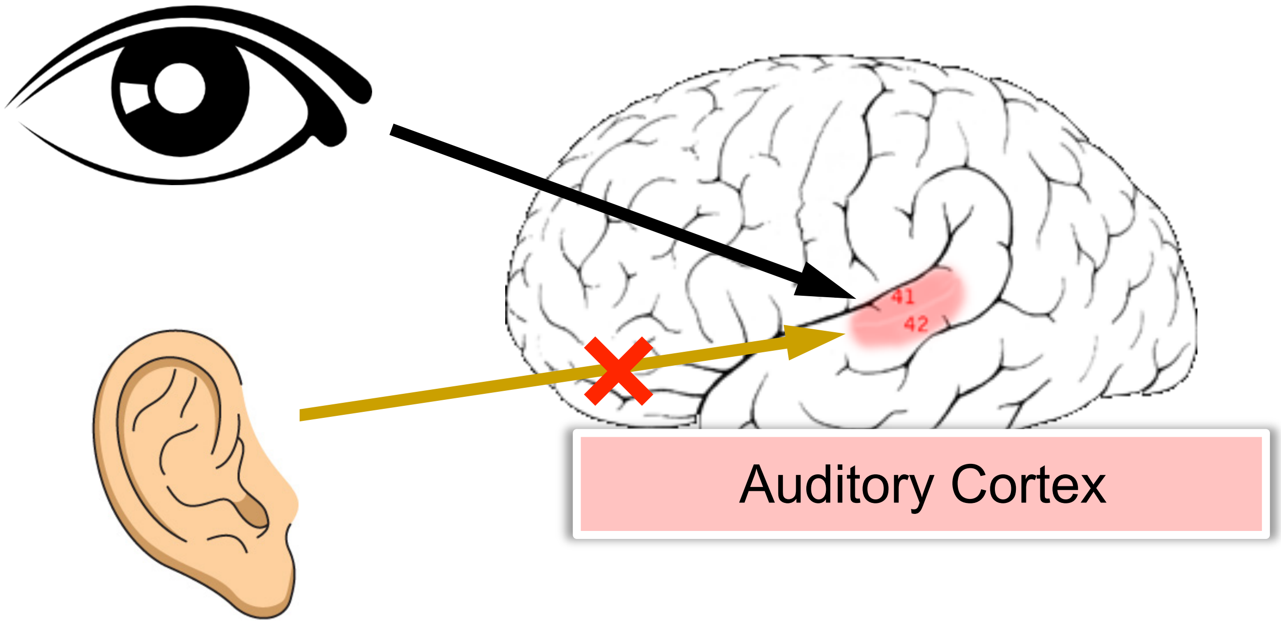


Textons



GLOH

The brain: inspiration for deep learning



Auditory cortex learns to see!

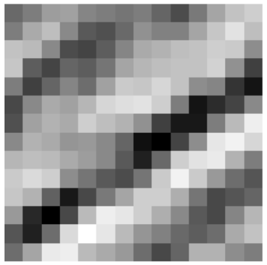
[Roe et al., 1992]

Basic idea of deep learning

- Also referred to as representation learning or unsupervised feature learning (with subtle distinctions)
- Is there some way to extract **meaningful features** from data **even without knowing the task** to be performed?
- Then, throw in some hierarchical structure to make it 'deep'

Feature learning problem

- Given a 14x14 image patch x, can represent it using 196 real numbers.



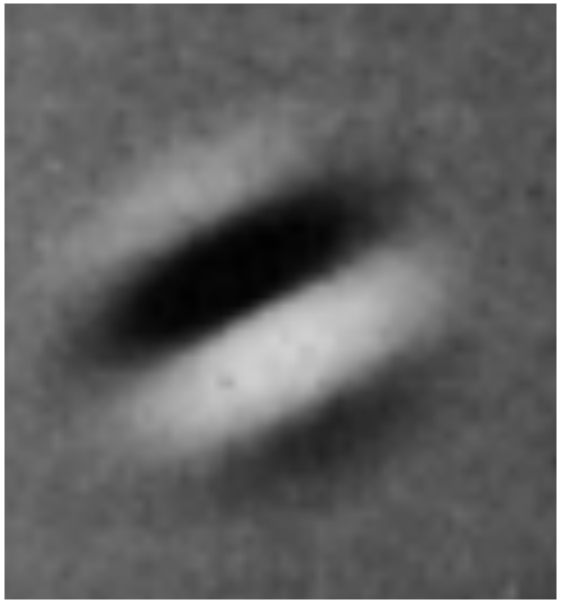
$$\begin{bmatrix} 255 \\ 98 \\ 93 \\ 87 \\ 89 \\ 91 \\ 48 \\ \dots \end{bmatrix}$$

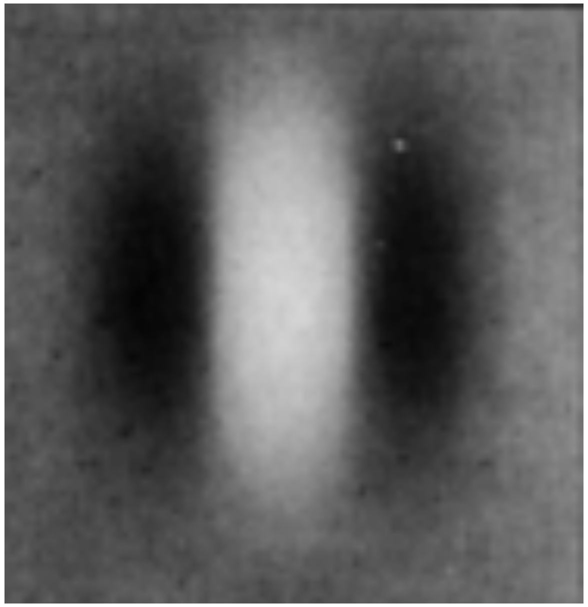
- Problem: Can we find a learn a better feature vector to represent this?

First stage of visual processing: V1

V1 is the first stage of visual processing in the brain.

Neurons in V1 typically modeled as edge detectors:


Neuron #1 of visual cortex (model)


Neuron #2 of visual cortex (model)

Learning sensor representations

Sparse coding (Olshausen & Field,1996)

Input: Images $x^{(1)}, x^{(2)}, \ldots, x^{(m)}$ (each in $R^{n \times n}$)

Learn: Dictionary of bases $\phi_1, \phi_2, \ldots, \phi_k$ (also $R^{n \times n}$), so that each input x can be approximately decomposed as:

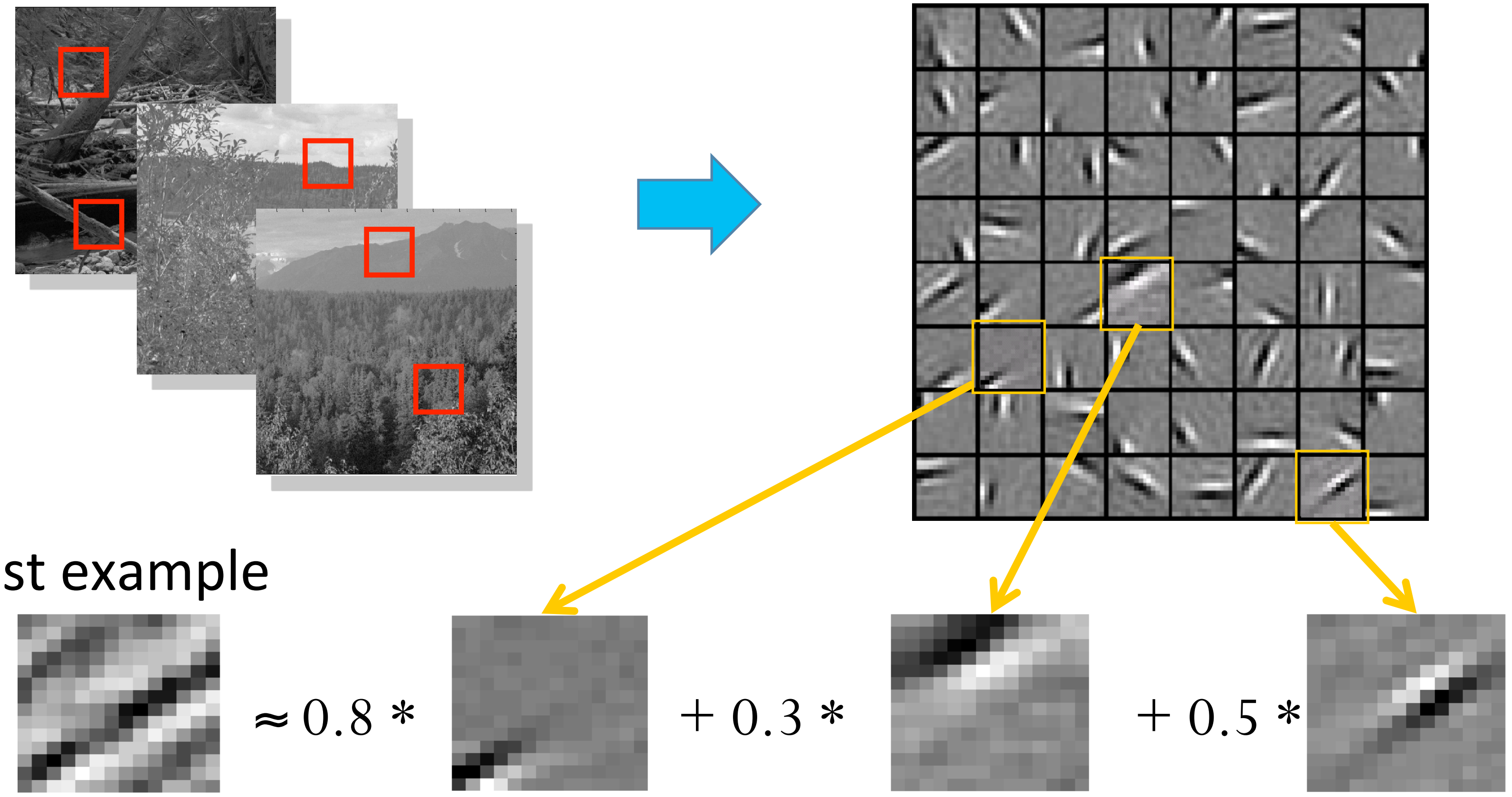
$$x \approx \sum a_j \phi_j$$

s.t. a_j's are mostly zero ("sparse")

Sparse coding illustration

Natural Images

Learned bases ($\phi_1, \ldots, \phi_{64}$): "Edges"



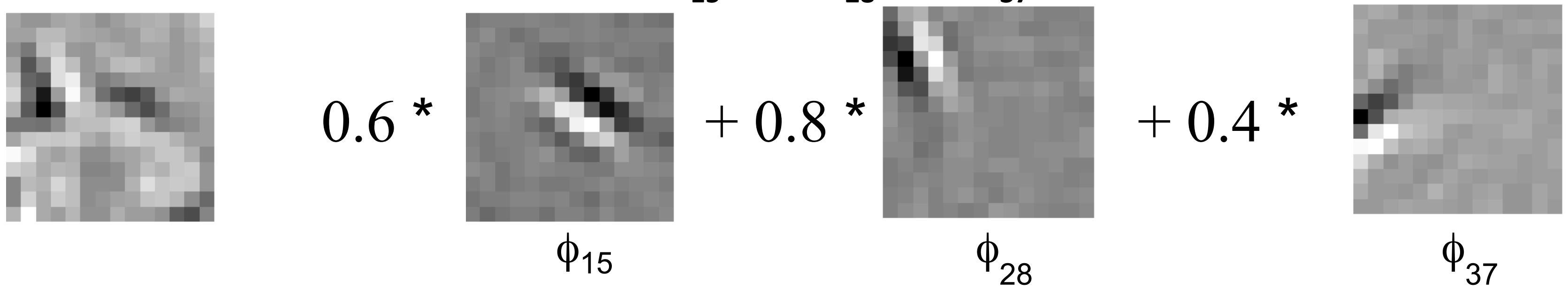
Test example

$$x \approx 0.8 * \phi_{36} + 0.3 * \phi_{42} + 0.5 * \phi_{63}$$

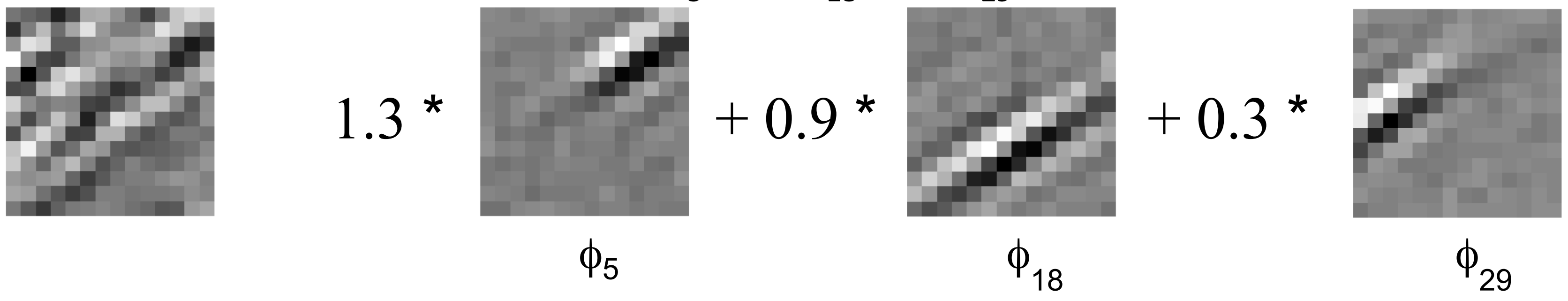
[$a_1, \ldots, a_{64}$] = [0, 0, …, 0, **0.8**, 0, …, 0, **0.3**, 0, …, 0, **0.5**, 0]
(feature representation)

Sparse coding illustration

Represent as: [a_{15}=0.6, a_{28}=0.8, a_{37} = 0.4]



Represent as: [a_5=1.3, a_{18}=0.9, a_{29} = 0.3]

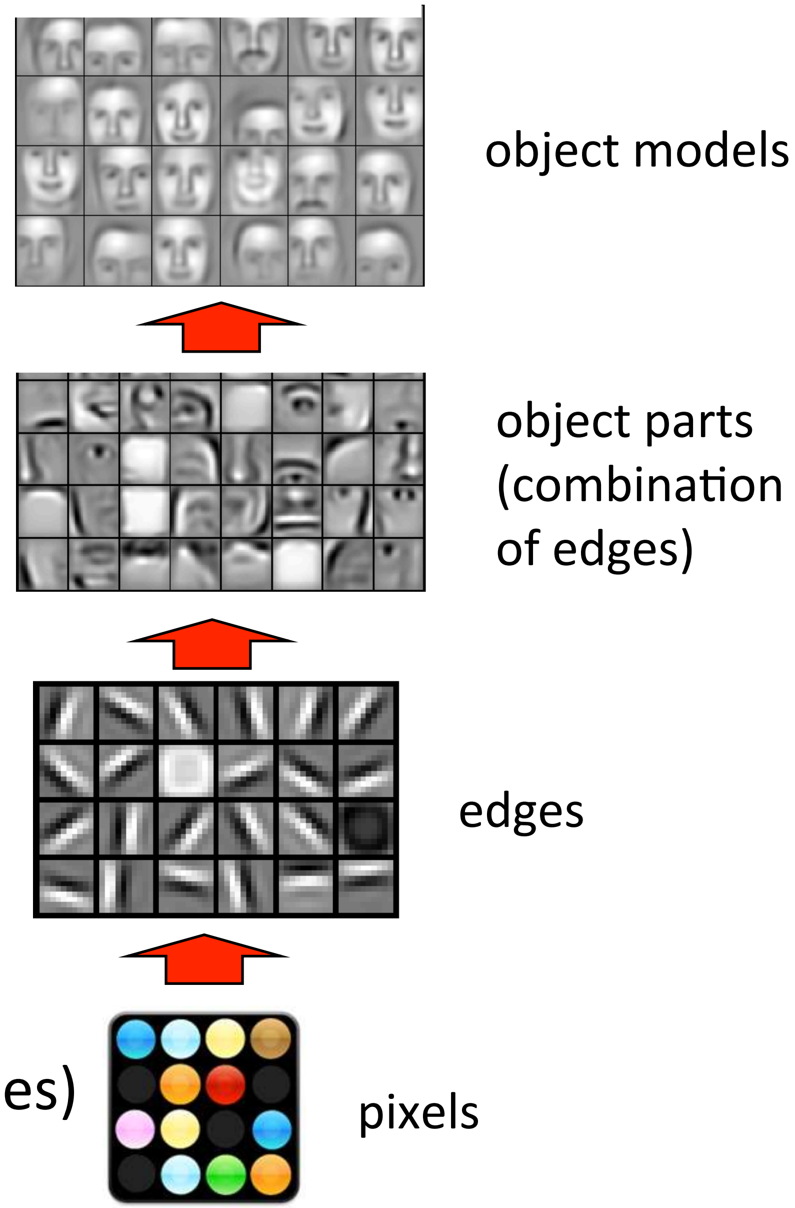


•**Method "invents" edge detection**

- Automatically learns to represent an image in terms of the edges that appear in it. Gives a more succinct, higher-level representation than the raw pixels.
- Quantitatively similar to primary visual cortex (area V1) in brain.

Going deep

Training set: Aligned images of faces.



Early work:
Uhr and students (recognition cones)
Fukushima (neocognitron)

Why deep learning?

Task: video activity recognition

Method	Accuracy
Hessian + ESURF [Williems et al 2008]	38%
Harris3D + HOG/HOF [Laptev et al 2003, 2004]	45%
Cuboids + HOG/HOF [Dollar et al 2005, Laptev 2004]	46%
Hessian + HOG/HOF [Laptev 2004, Williems et al 2008]	46%
Dense + HOG / HOF [Laptev 2004]	47%
Cuboids + HOG3D [Klaser 2008, Dollar et al 2005]	46%
Unsupervised feature learning (our method)	**52%**

$\leftarrow$

[Le, Zhou & Ng, 2011]

Audio

TIMIT Phone classification	Accuracy
Prior art (Clarkson et al.,1999)	79.6%
Feature learning	**80.3%**

TIMIT Speaker identification	Accuracy
Prior art (Reynolds, 1995)	99.7%
Feature learning	**100.0%**

Images

CIFAR Object classification	Accuracy
Prior art (Ciresan et al., 2011)	80.5%
Feature learning	**82.0%**

NORB Object classification	Accuracy
Prior art (Scherer et al., 2010)	94.4%
Feature learning	**95.0%**

Video

Hollywood2 Classification	Accuracy
Prior art (Laptev et al., 2004)	48%
Feature learning	**53%**

YouTube	Accuracy
Prior art (Liu et al., 2009)	71.2%
Feature learning	**75.8%**

KTH	Accuracy
Prior art (Wang et al., 2010)	92.1%
Feature learning	**93.9%**

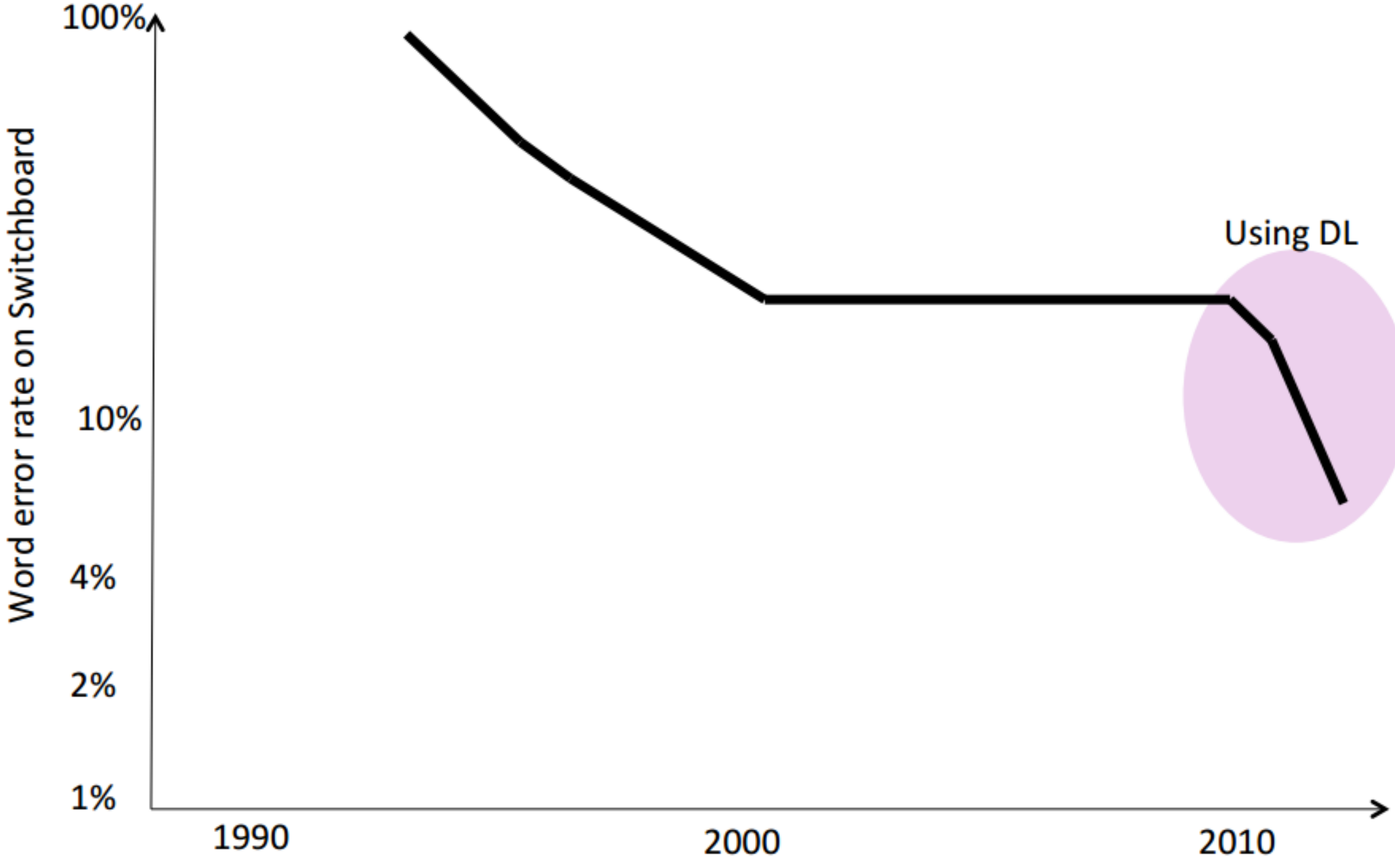
UCF	Accuracy
Prior art (Wang et al., 2010)	85.6%
Feature learning	**86.5%**

Text/NLP

Paraphrase detection	Accuracy
Prior art (Das & Smith, 2009)	76.1%
Feature learning	**76.4%**

Sentiment (MR/MPQA data)	Accuracy
Prior art (Nakagawa et al., 2010)	77.3%
Feature learning	**77.7%**

Impact on speech recognition



Application to Google Streetview

ImageNet classification: 22,000 classes

…
smoothhound, smoothhound shark, Mustelus mustelus
American smooth dogfish, Mustelus canis
Florida smoothhound, Mustelus norrisi
whitetip shark, reef whitetip shark, Triaenodon obseus
Atlantic spiny dogfish, Squalus acanthias
Pacific spiny dogfish, Squalus suckleyi
hammerhead, hammerhead shark
smooth hammerhead, Sphyrna zygaena
smalleye hammerhead, Sphyrna tudes
shovelhead, bonnethead, bonnet shark, Sphyrna tiburo
angel shark, angelfish, Squatina squatina, monkfish
electric ray, crampfish, numbfish, torpedo
smalltooth sawfish, Pristis pectinatus
guitarfish
roughtail stingray, Dasyatis centroura
butterfly ray
eagle ray
spotted eagle ray, spotted ray, Aetobatus narinari
cownose ray, cow-nosed ray, Rhinoptera bonasus
manta, manta ray, devilfish
Atlantic manta, Manta birostris
devil ray, Mobula hypostoma
grey skate, gray skate, Raja batis
little skate, Raja erinacea
…

Stingray

Mantaray

ImageNet Classification: 14M images, 22k categories

0.005%	9.5%	?
Random guess	State-of-the-art (Weston, Bengio '11)	Feature learning From raw pixels

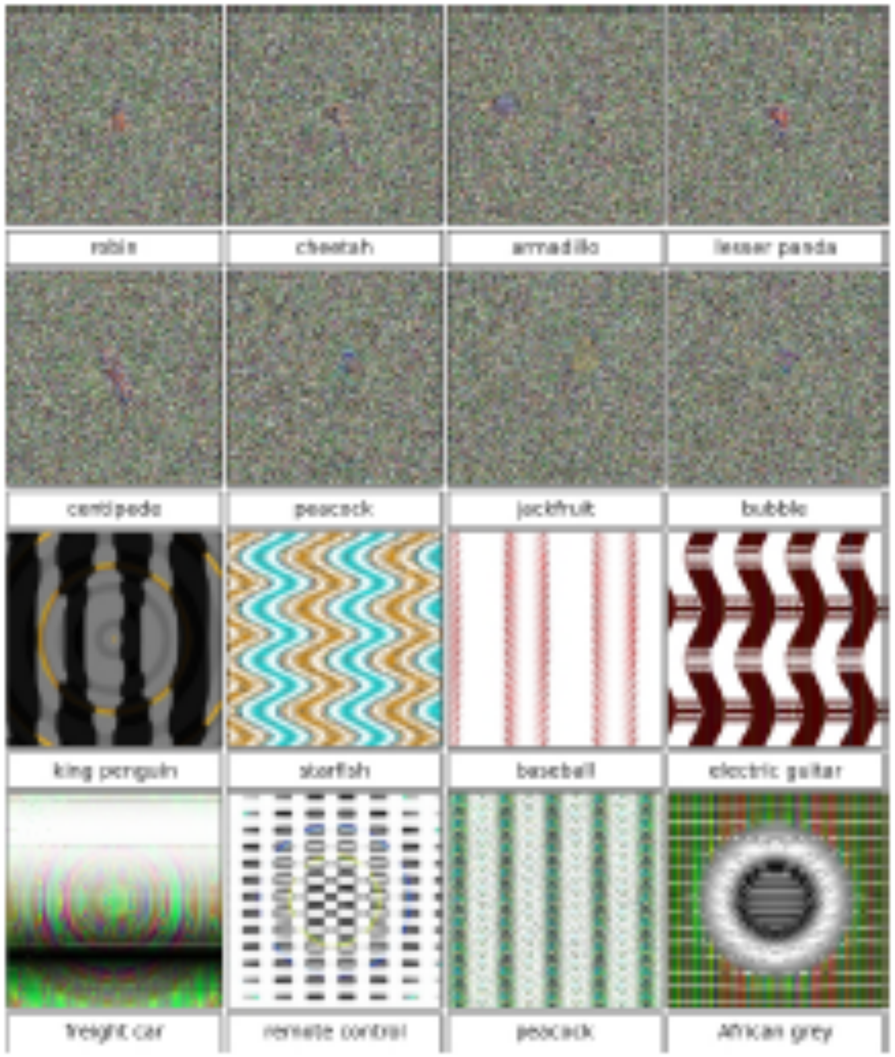
Le, et al., *Building high-level features using large-scale unsupervised learning*. ICML 2012

ImageNet Classification: 14M images, 22k categories

0.005%	9.5%	21.3%
Random guess	State-of-the-art (Weston, Bengio '11)	Feature learning From raw pixels

Le, et al., *Building high-level features using large-scale unsupervised learning*. ICML 2012

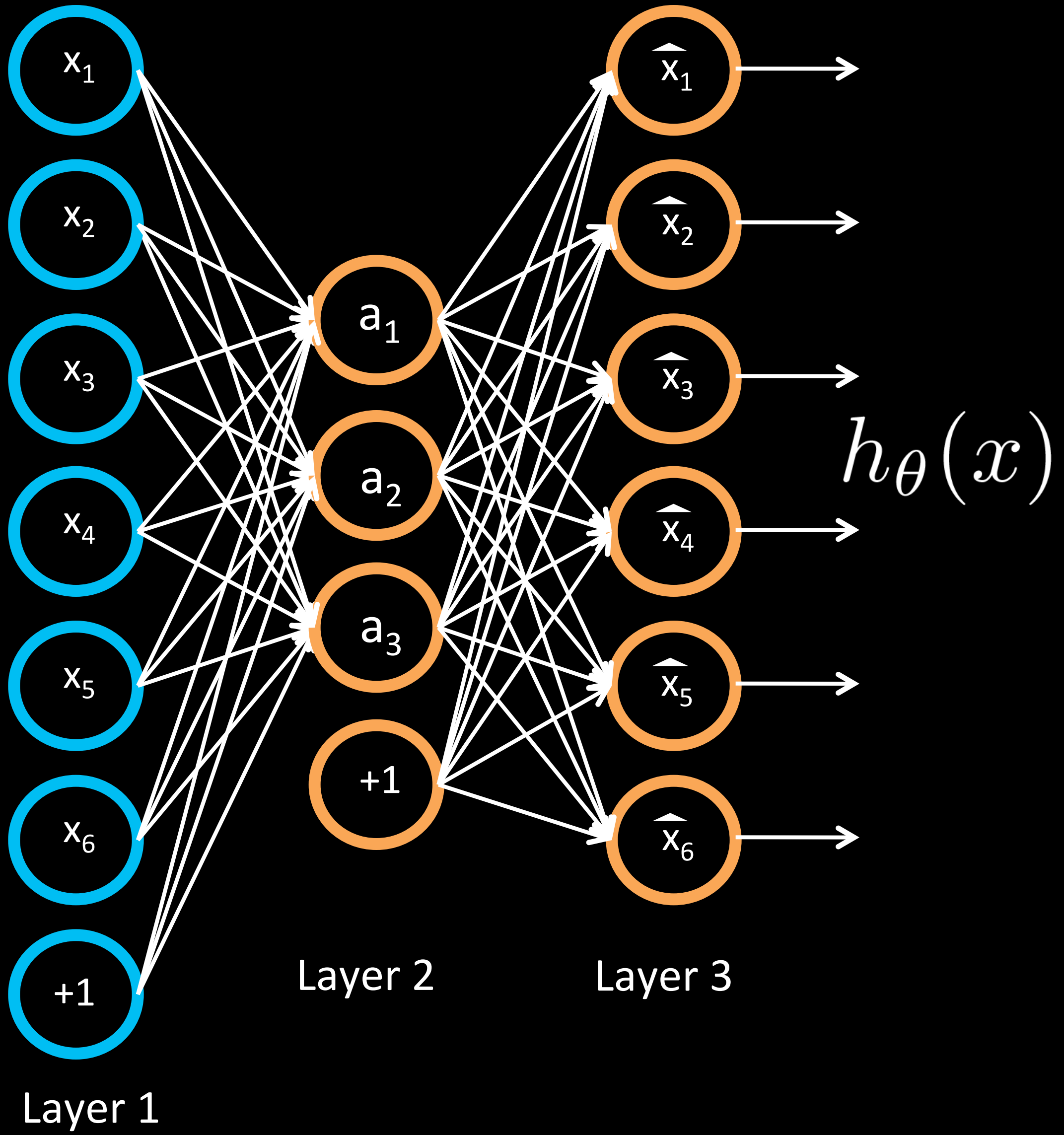
But… deep neural networks can be easily fooled



Some common deep architectures

- Autoencoders
- Deep belief networks (DBNs)
- Convolutional variants
- Sparse coding

Unsupervised feature learning with a neural network



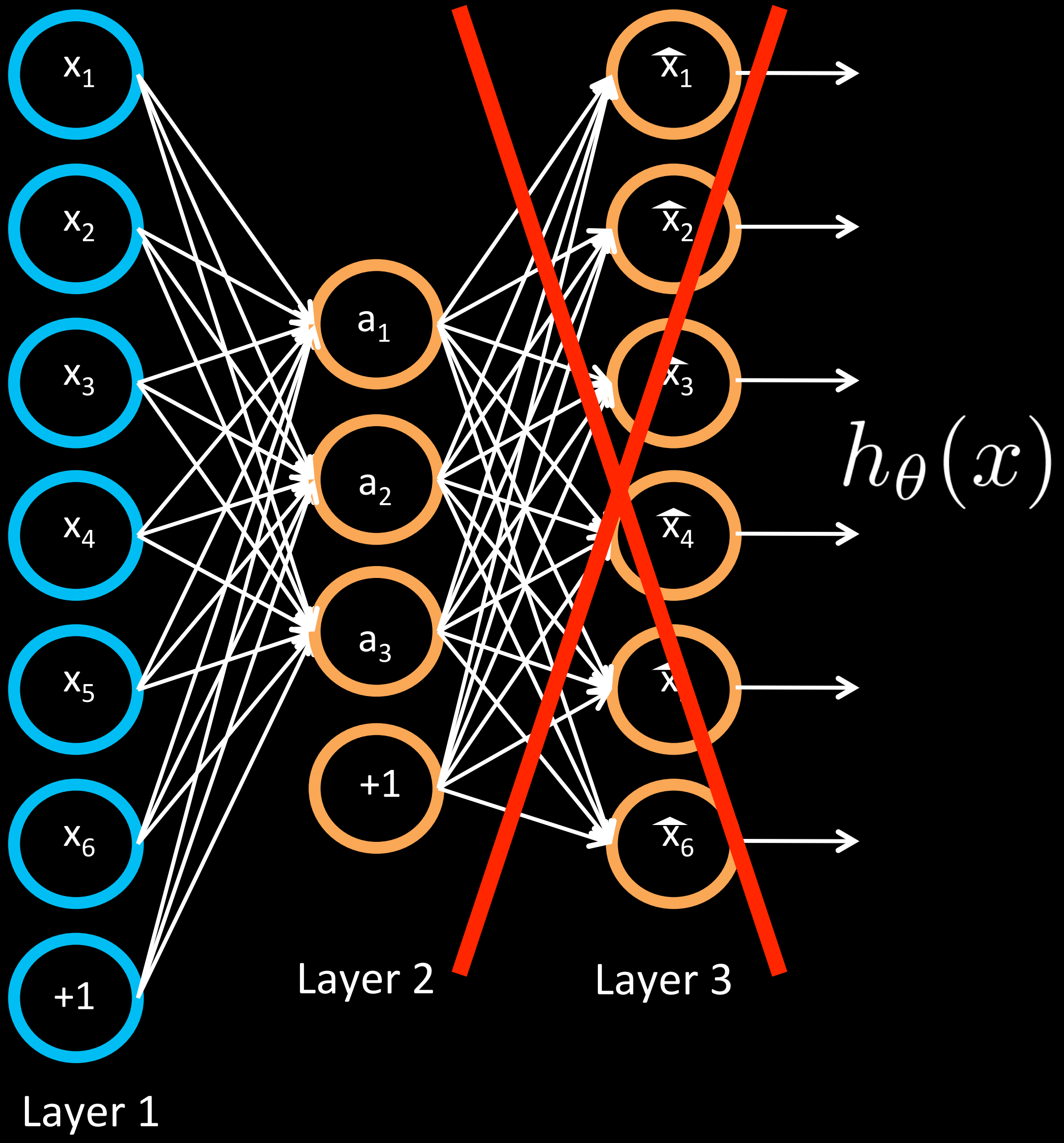
Autoencoder.

Network is trained to output the input (learn identify function).

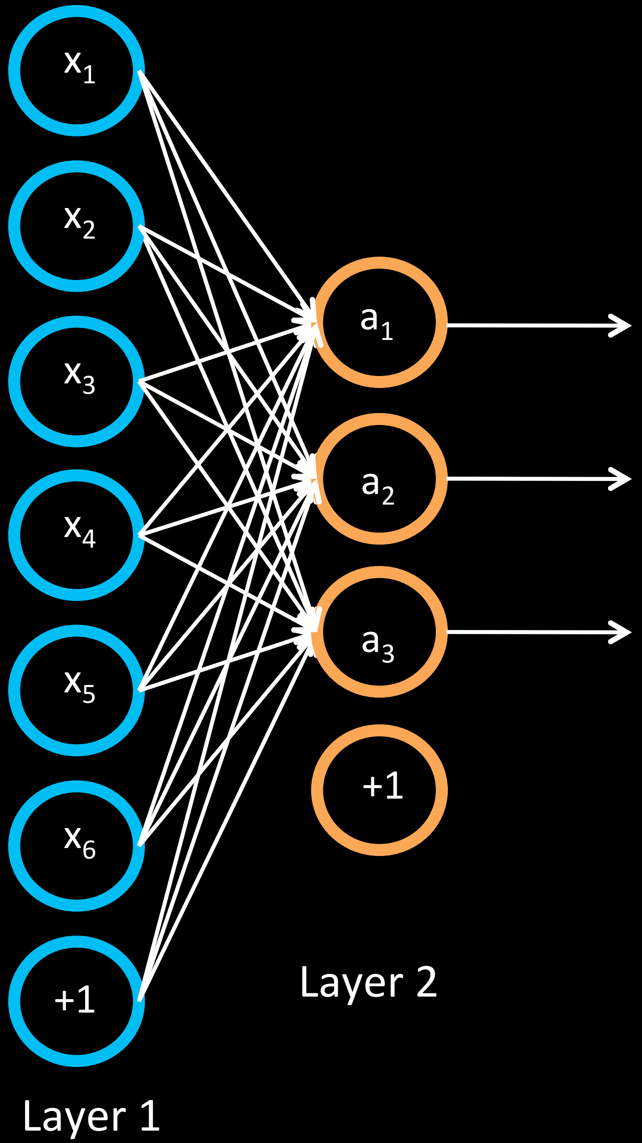
$$h_\theta(x) \approx x$$

Trivial solution unless:
- Constrain number of units in Layer 2 (learn compressed representation), or
- Constrain Layer 2 to be **sparse**.

Unsupervised feature learning with a neural network



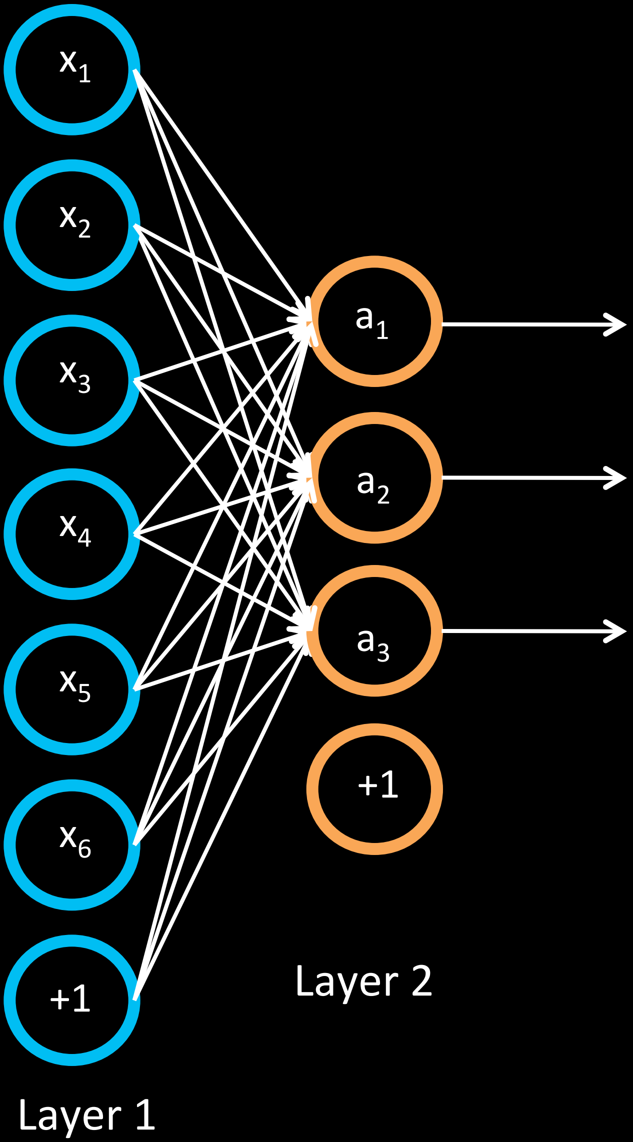
Unsupervised feature learning with a neural network



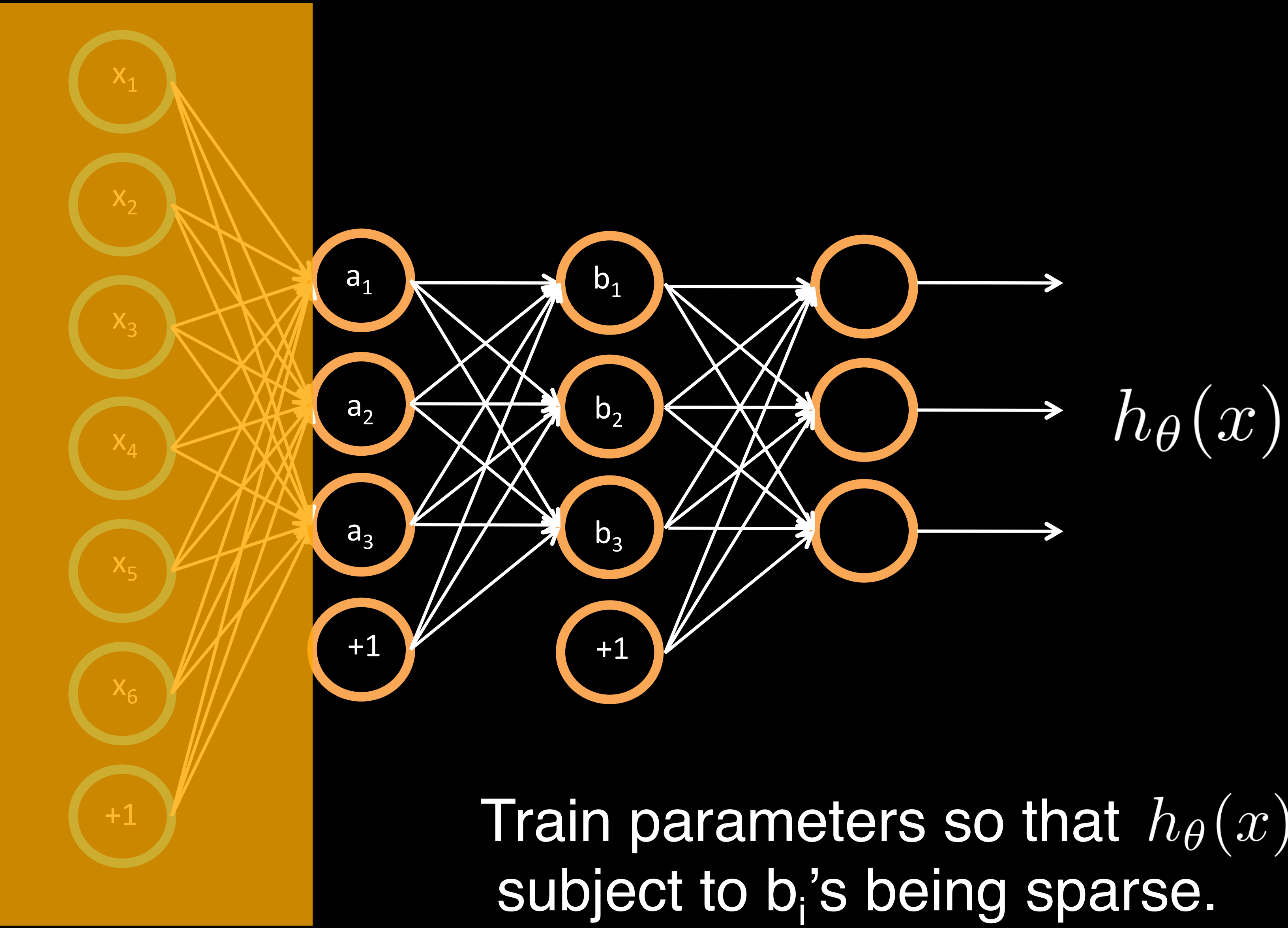
$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

New representation for input.

Unsupervised feature learning with a neural network

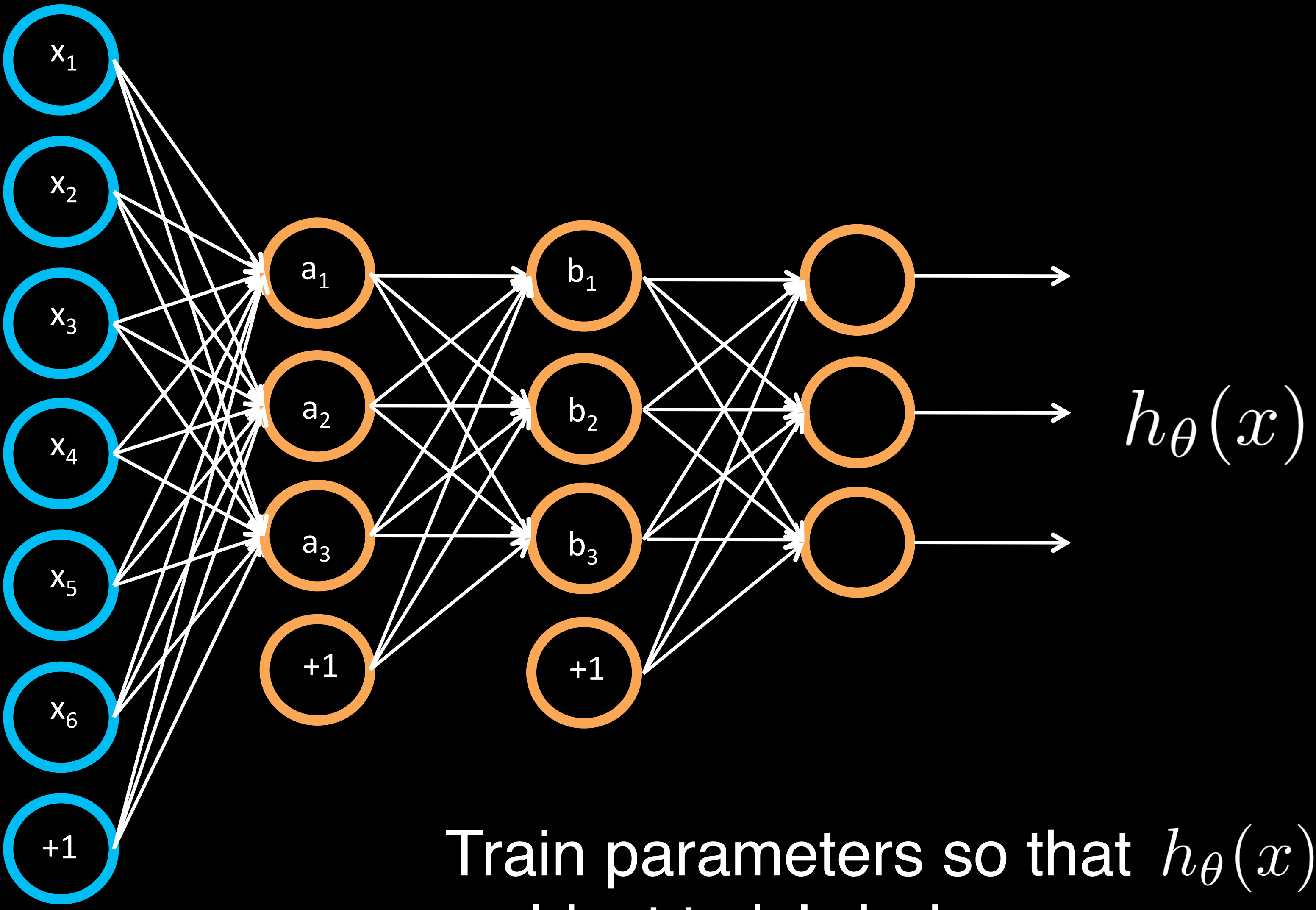


Unsupervised feature learning with a neural network



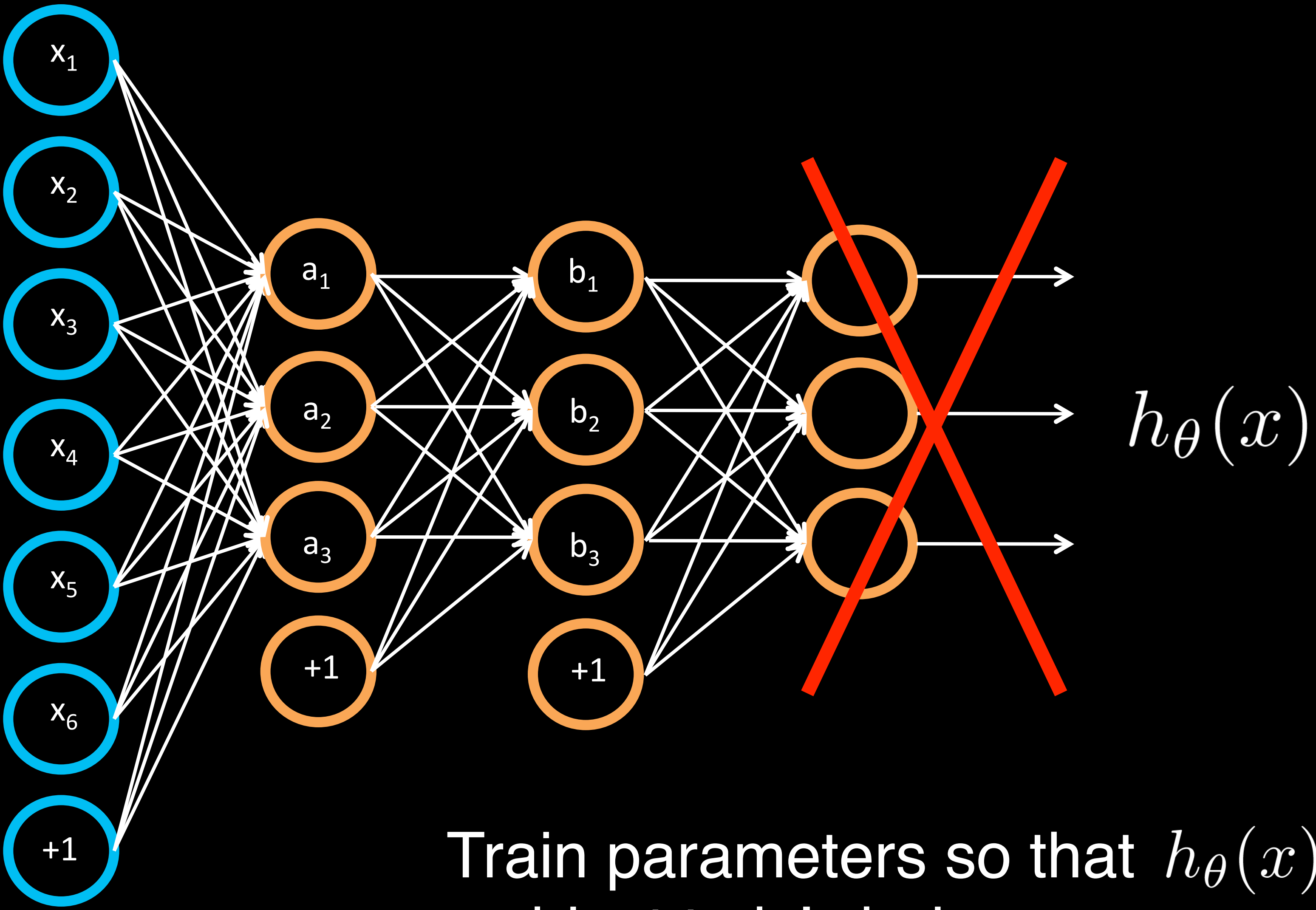
Train parameters so that $h_\theta(x) \approx a$, subject to b_i's being sparse.

Unsupervised feature learning with a neural network



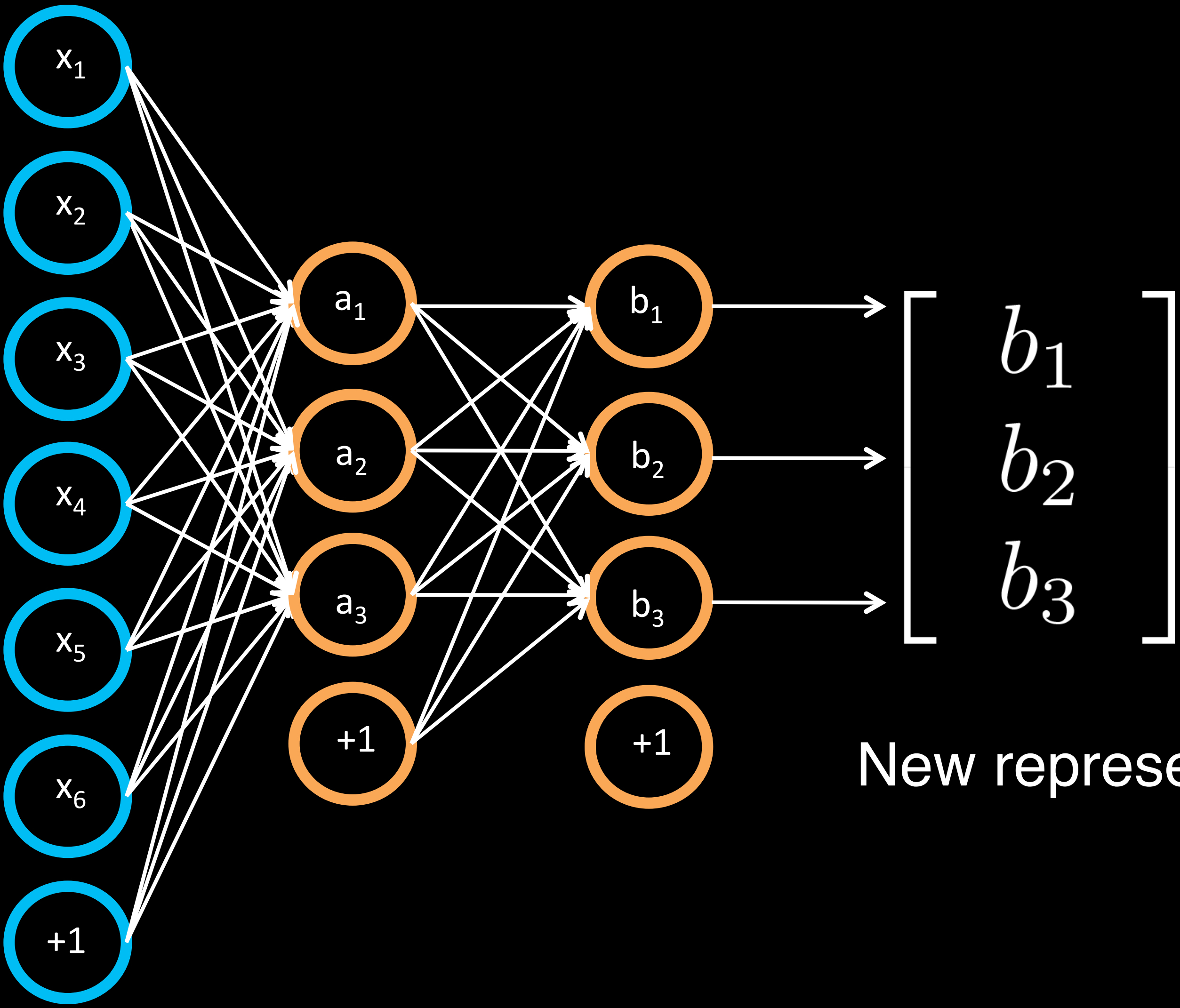
Train parameters so that $h_\theta(x) \approx a$, subject to b_i's being sparse.

Unsupervised feature learning with a neural network



Train parameters so that $h_\theta(x) \approx a$, subject to b_i's being sparse.

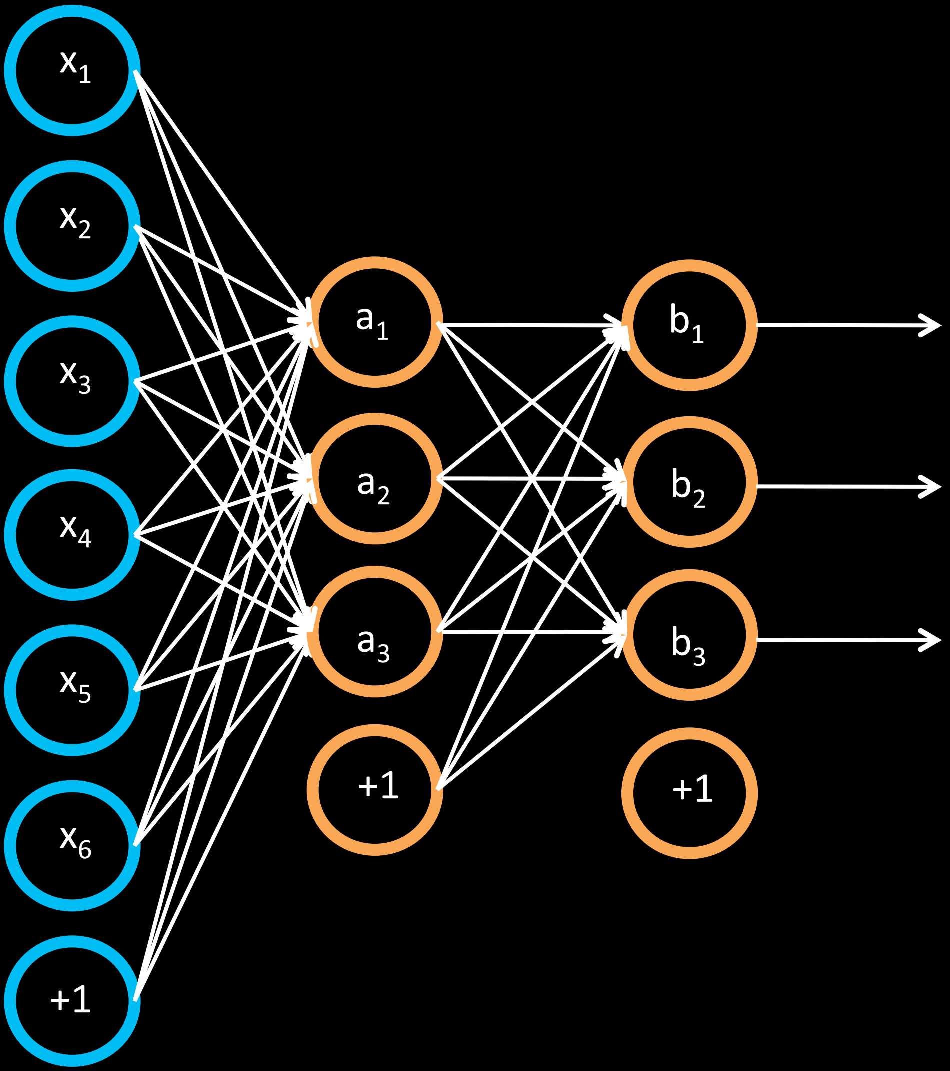
Unsupervised feature learning with a neural network



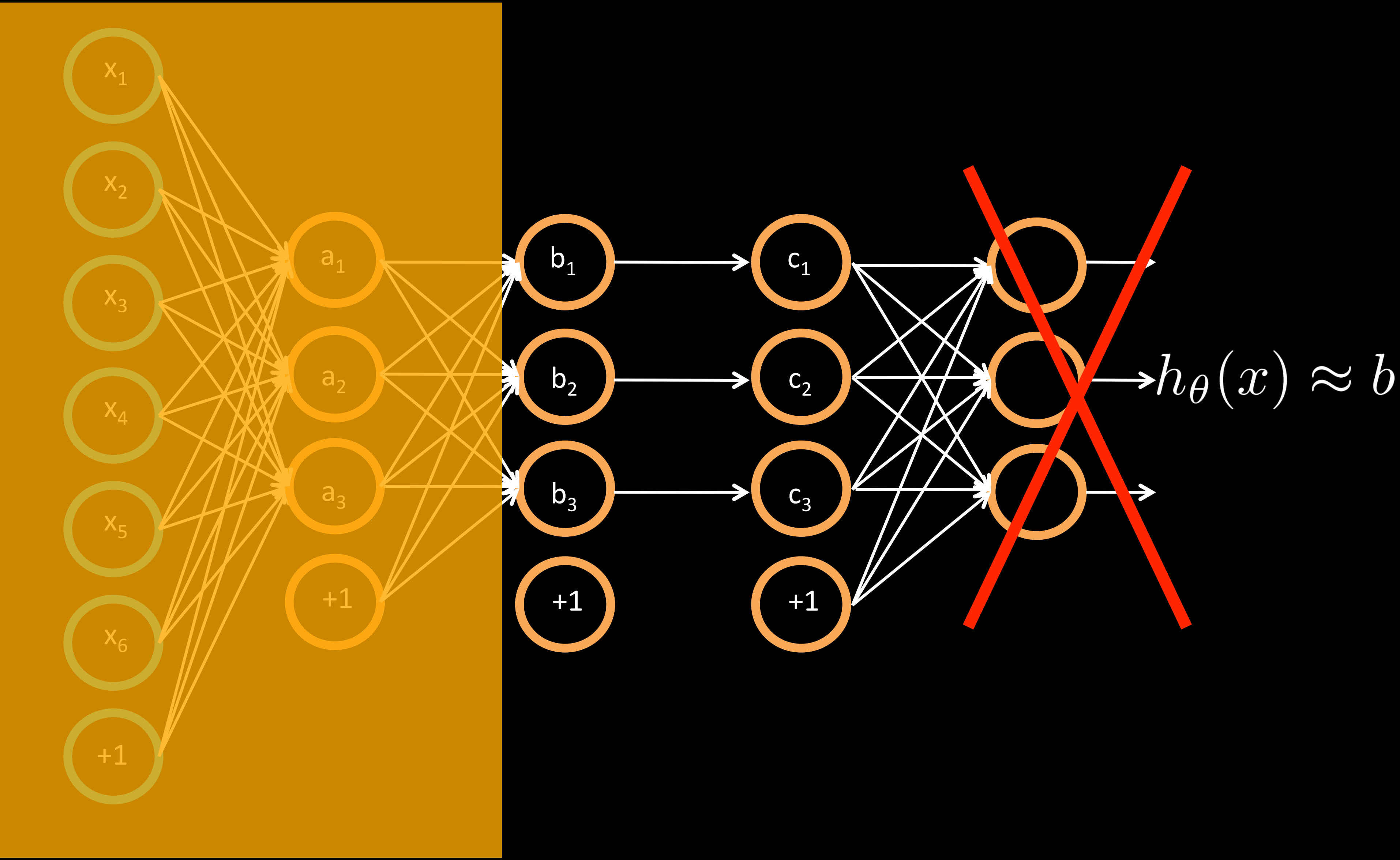
$$\begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

New representation for input.

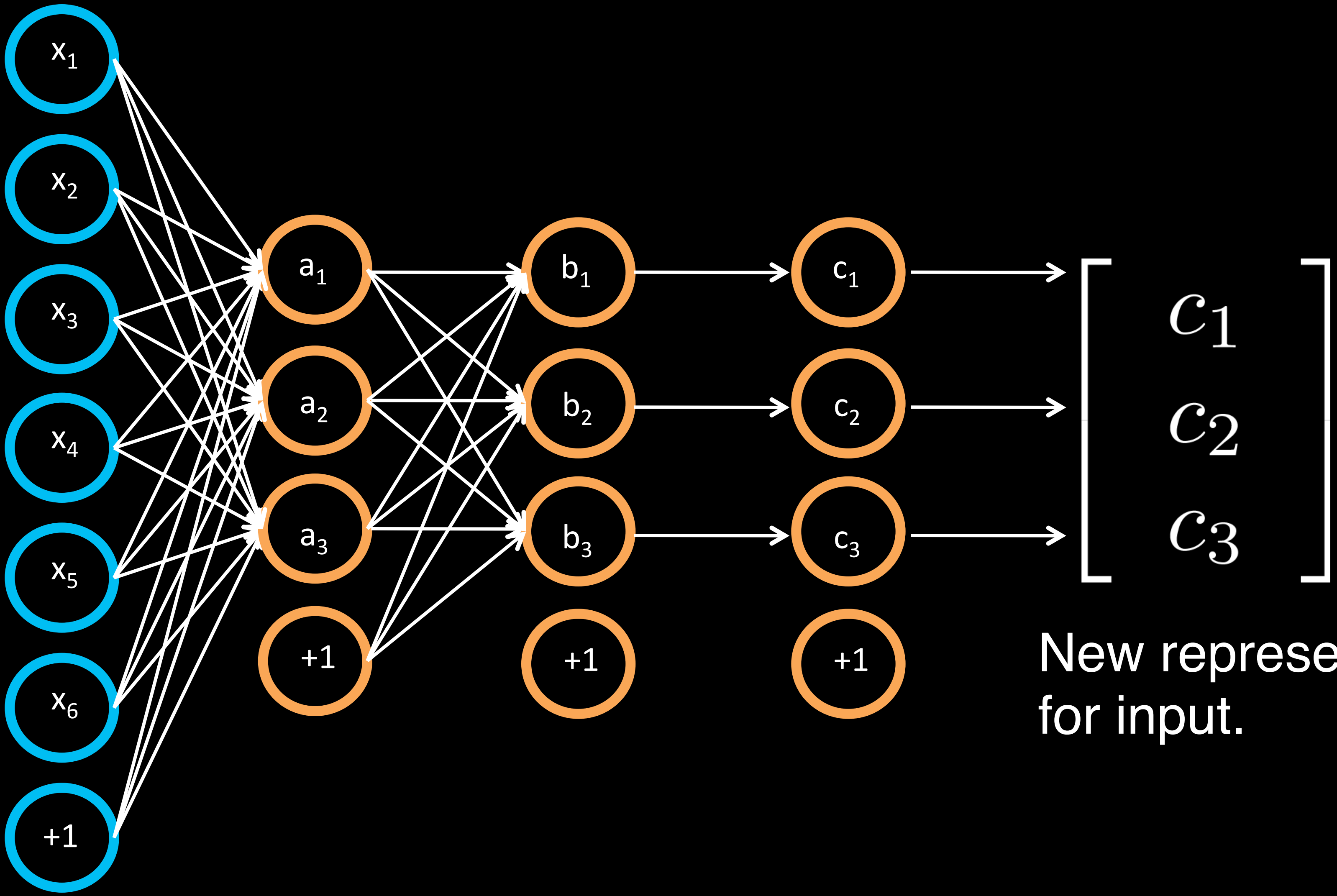
Unsupervised feature learning with a neural network



Unsupervised feature learning with a neural network



Unsupervised feature learning with a neural network



$$\begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

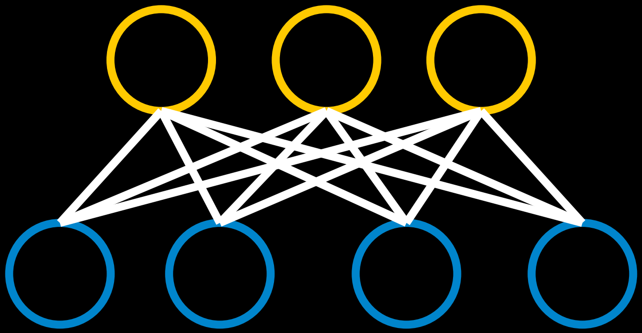
New representation for input.

Use [c_1, c_3, c_3] as representation to feed to learning algorithm.

Deep Belief Net

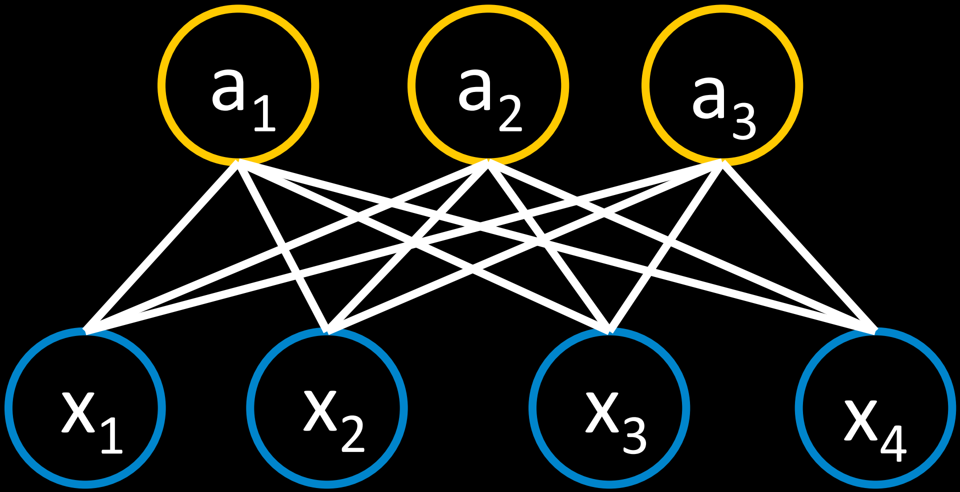
Deep Belief Net (DBN) is another algorithm for learning a feature hierarchy.

Building block: 2-layer graphical model (Restricted Boltzmann Machine).



Can then learn additional layers one at a time.

Restricted Boltzmann machine (RBM)



Layer 2. $[a_1, a_2, a_3]$ (binary-valued)

Input $[x_1, x_2, x_3, x_4]$

MRF with joint distribution:

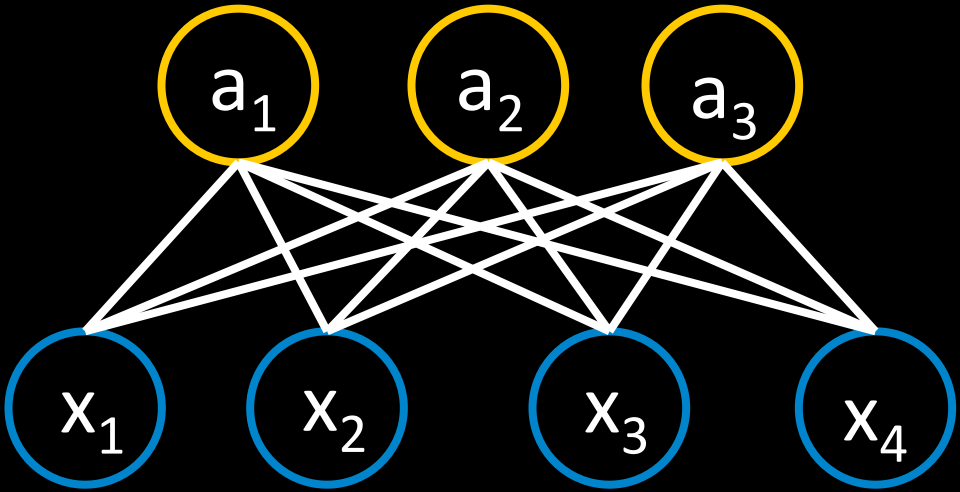
$$P(x, a) \propto \exp\left(-\sum_{i,j} x_i a_j W_{ij}\right)$$

Use Gibbs sampling for inference.

Given observed inputs x, want maximum likelihood estimation:

$$\max_W P(x) = \max_W \sum_a P(x, a)$$

Restricted Boltzmann machine (RBM)



Layer 2. $[a_1, a_2, a_3]$
(binary-valued)

Input $[x_1, x_2, x_3, x_4]$

Gradient ascent on log P(x) :

$$\Delta W_{ij} = \alpha \left([x_i a_j]_{\mathrm{obs}} - [x_i a_j]_{\mathrm{prior}} \right)$$

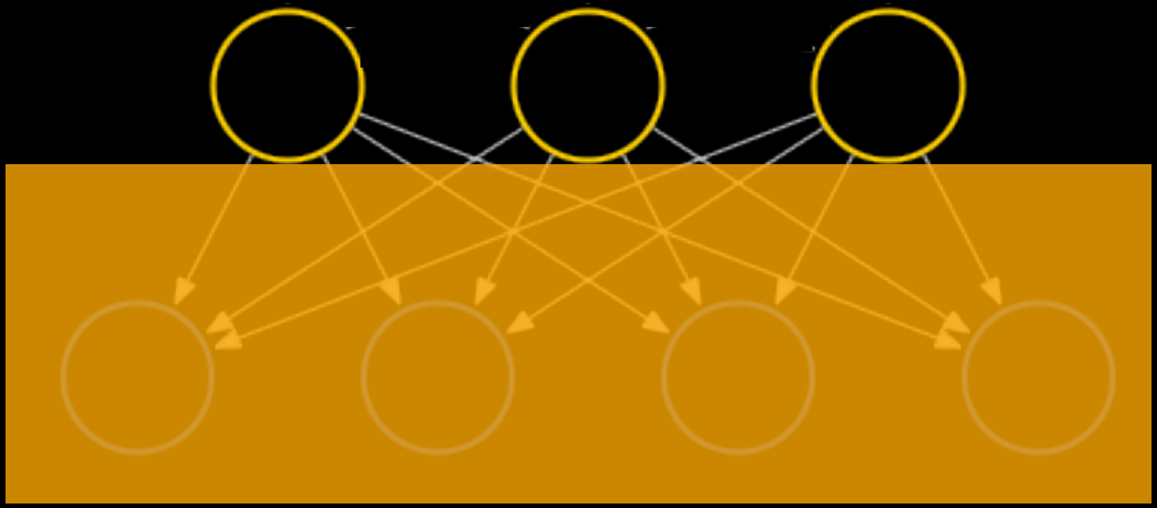
$[x_i a_j]_{obs}$ from fixing x to observed value, and sampling a from P(a|x).

$[x_i a_j]_{prior}$ from running Gibbs sampling to convergence.

Adding sparsity constraint on a_i's usually improves results.

Deep Belief Network

Similar to a sparse autoencoder in many ways.
Stack RBMs on top of each other to get DBN.

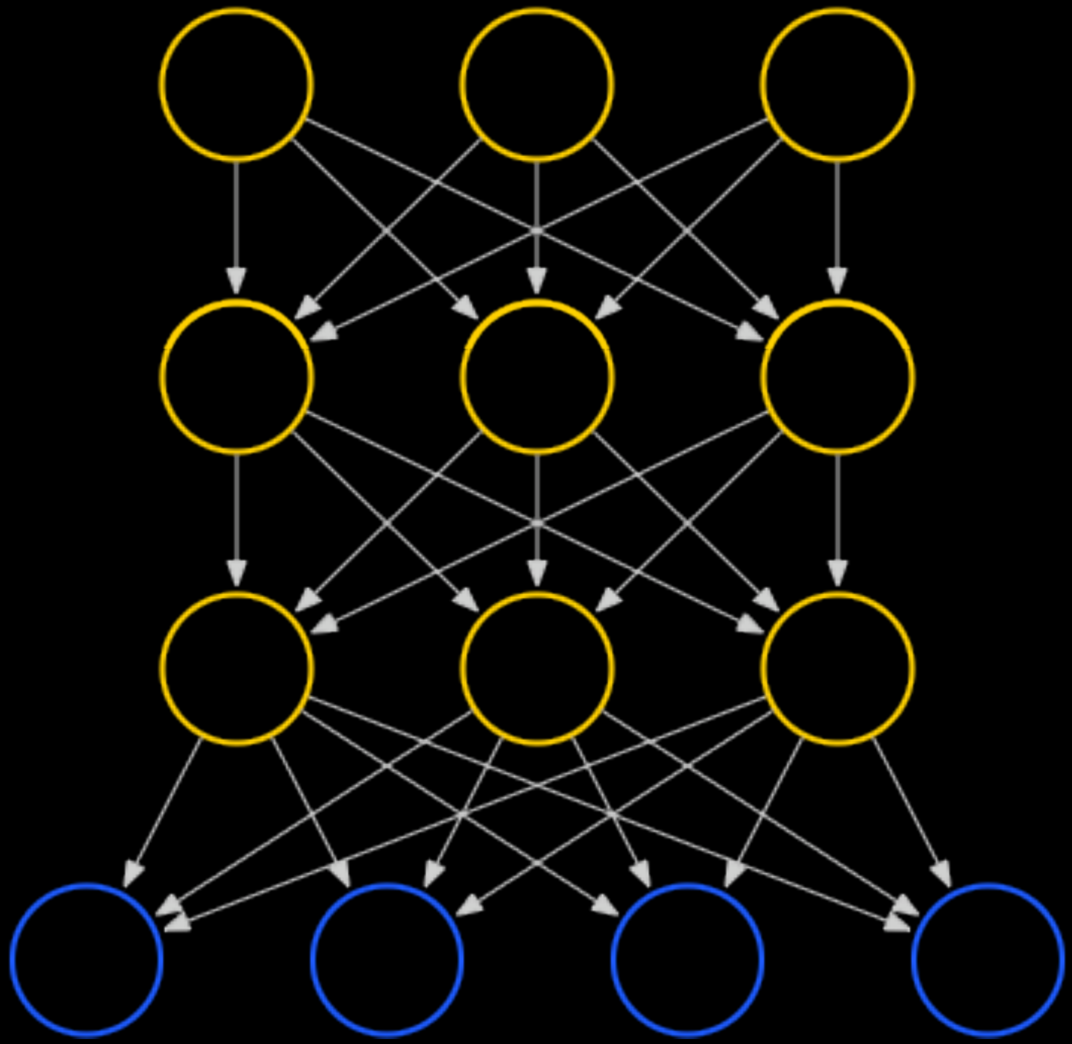


Layer 3. [$b_{1,}$ b_2, b_3]

Layer 2. [$a_{1,}$ a_2, a_3]

Input [$x_{1,}$ x_2, x_3, x_4]

Deep Belief Network



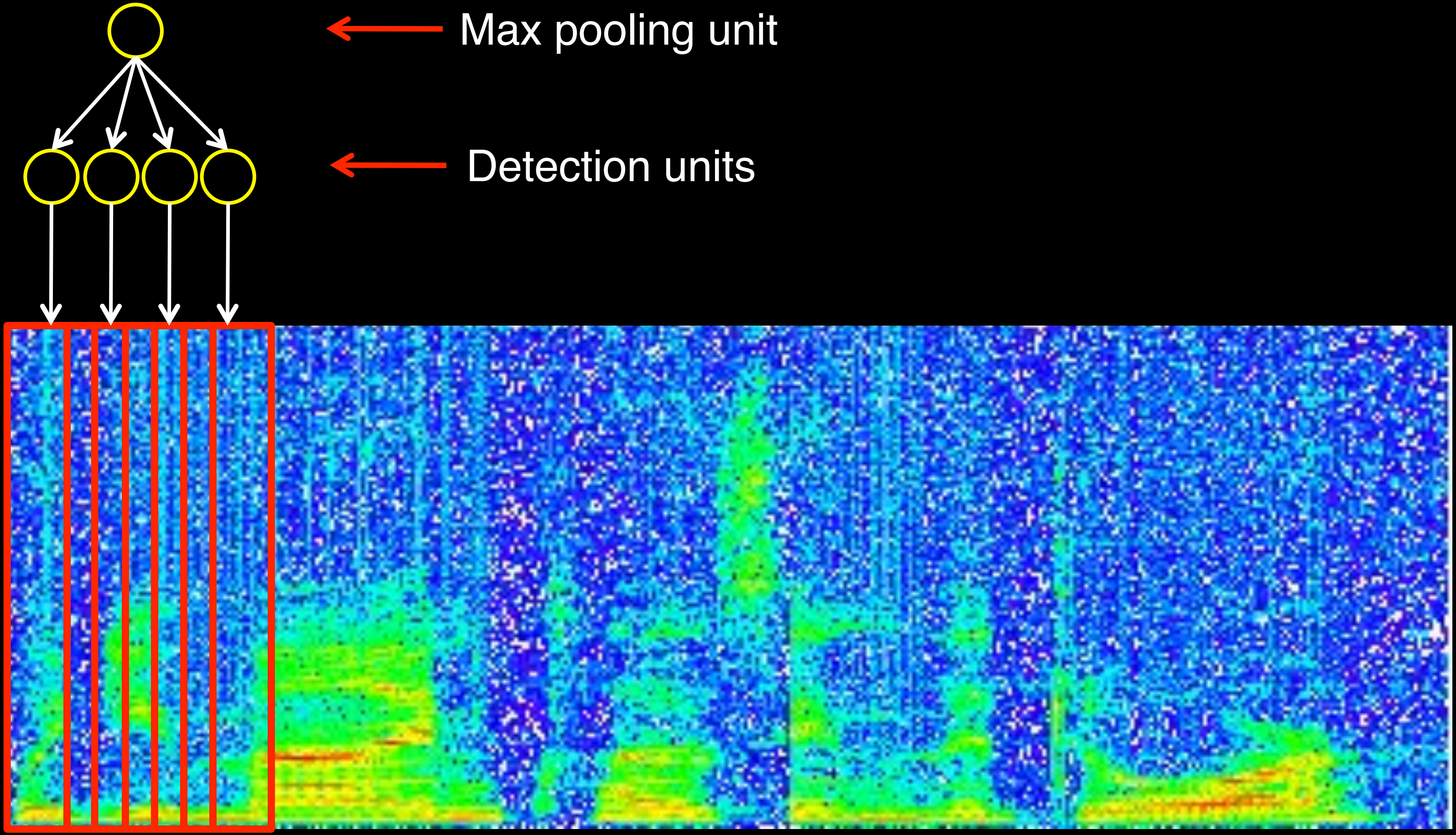
Layer 4. [$c_{1,}$ c_2, c_3]

Layer 3. [$b_{1,}$ b_2, b_3]

Layer 2. [$a_{1,}$ a_2, a_3]

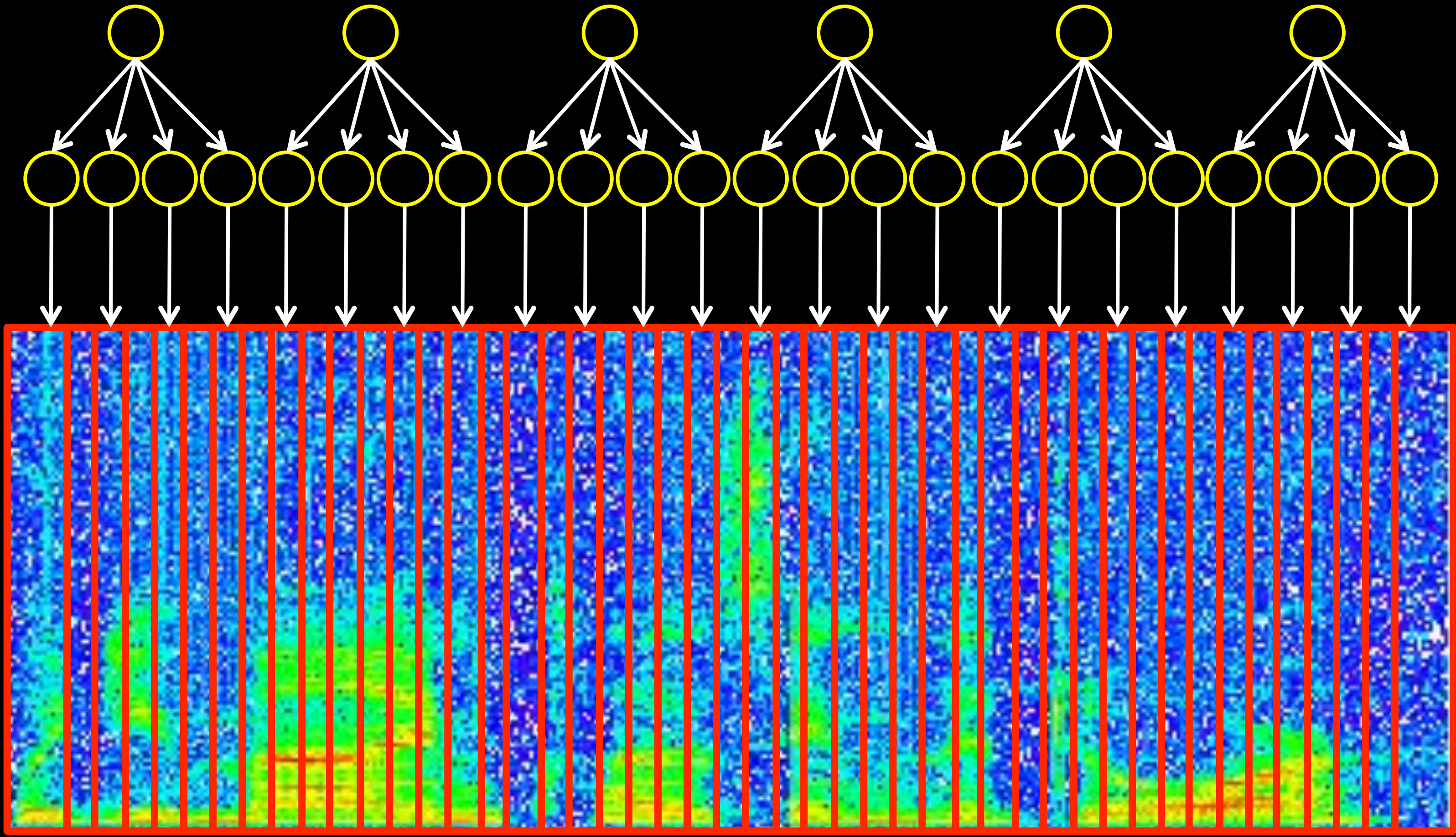
Input [$x_{1,}$ x_2, x_3, x_4]

Convolutional DBN for audio



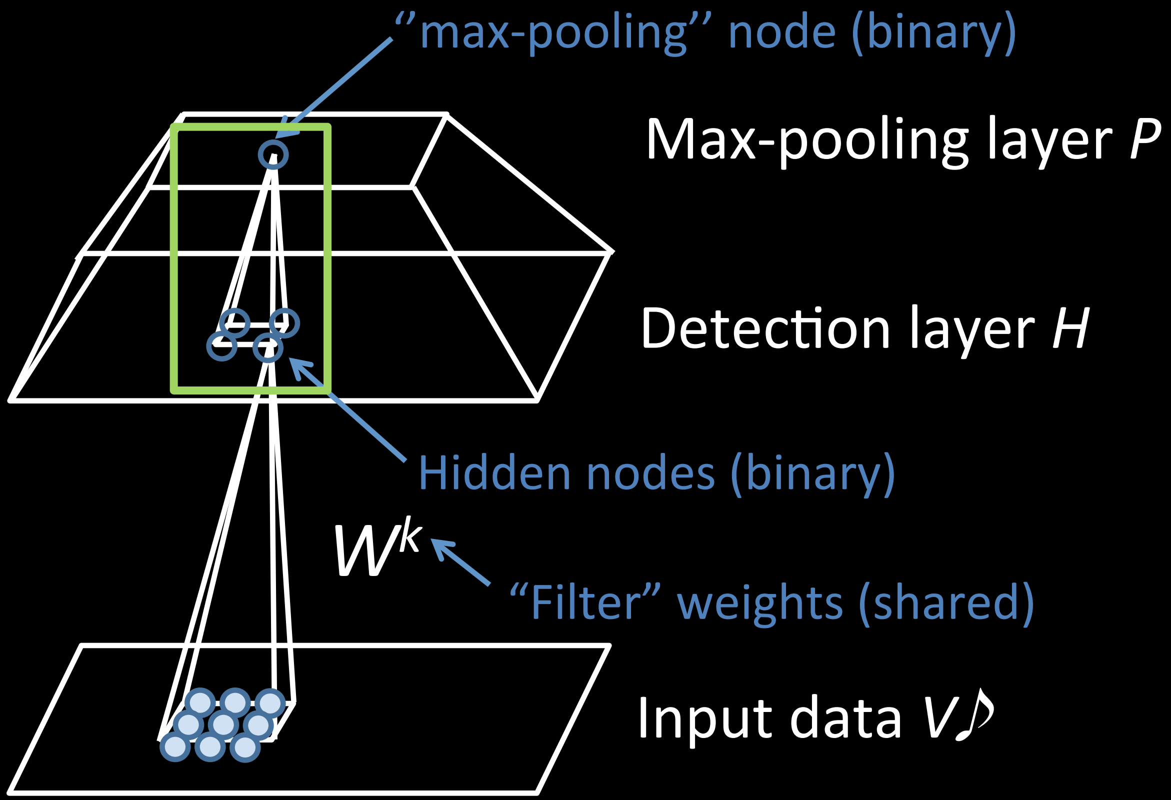
Spectrogram

Convolutional DBN for audio



Spectrogram

Convolutional DBN for Images



"max-pooling" node (binary)
Max-pooling layer *P*
Detection layer *H*
Hidden nodes (binary)
W^k
"Filter" weights (shared)
Input data *V*♪