Introduction to Neural Networks

Objectives

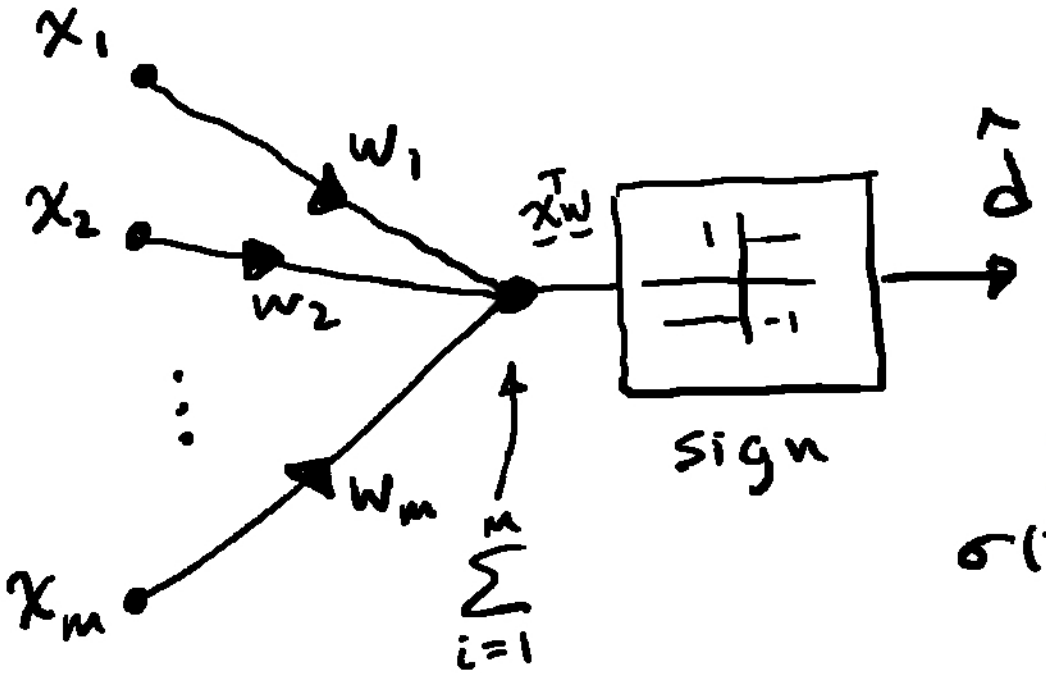
- relate neural networks to linear classifiers
- define structure of multilayer neural network
- overview procedure for training neural networks

The "neuron" generalizes a linear classifier 2

Feature: $\underline{x}^T = [x_1 \; x_2 \; \cdots \; x_m]$ Weights: $\underline{w}^T = [w_1 \; w_2 \cdots w_m]$

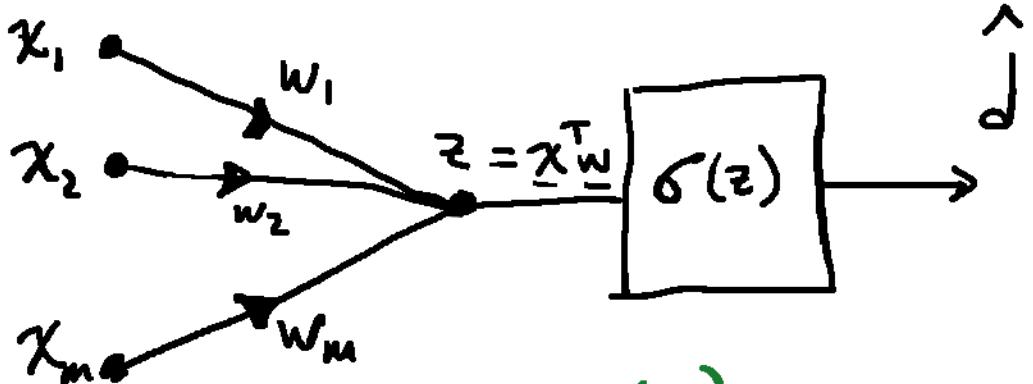
Linear classifier

$$\hat{d} = \text{sign}(\underline{x}^T \underline{w})$$



Neuron

$$\hat{d} = \sigma(\underline{x}^T \underline{w})$$



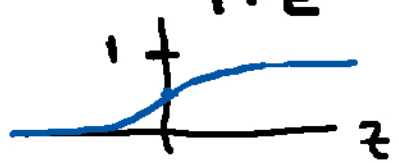
Common $\sigma(z)$

ReLU

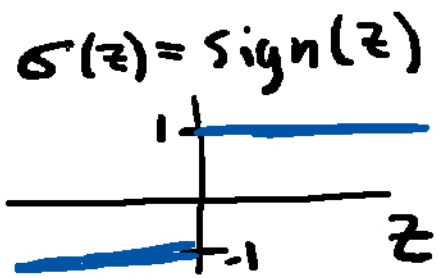
$\sigma(z) = \max\{0, z\}$

Logistic

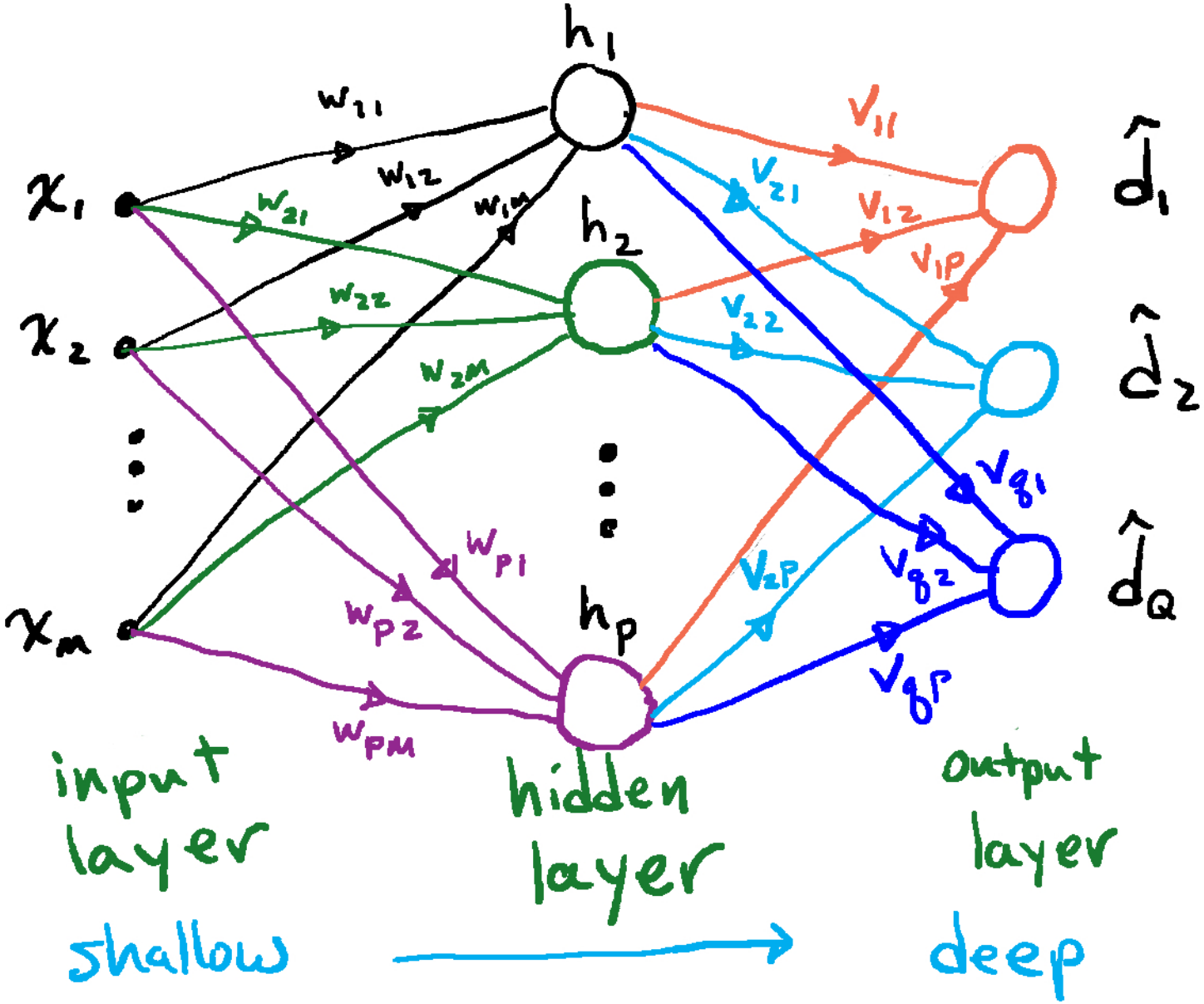
$\sigma(z) = \frac{1}{1+e^{-z}}$



$\sigma(z) = \text{sign}(z)$



A neural network is a network of neurons



multiple outputs → solve multiple problems

$$h_l = \sigma\left(\sum_{j=1}^{m} w_{lj} x_j\right)$$

$$\hat{d}_m = \sigma\left(\sum_{n=1}^{P} V_{mn} h_n\right)$$

$$= \sigma\left(\sum_{n=1}^{P} V_{m,n} \sigma\left(\sum_{j=1}^{m} w_{nj} x_j\right)\right)$$

"deep" learning → many hidden layers

Two issues must be addressed to use NNs

1) Network structure: number of layers, number of hidden nodes in each layer

Open question. Universal approximation theorem (1991): Three layer network can approximate any function arbitrarily well given enough hidden nodes and the right weights

2) Choosing the weights

Stochastic gradient descent and backpropagation

Nonconvex → local minima

Backpropagation updates each layer in sequence

- work back from deep (output) to shallow (input)

$$\min_{W_{lj}, V_{lj}, \ldots} \sum_{i=1}^{N} \sum_{q=1}^{Q} \frac{1}{2} \left(\hat{d}_{i,q} - d_{i,q} \right)^2$$

N training samples, Q outputs

1) Initial guess on W_{lj}, V_{lj} (etc)

2) Randomly choose training sample i

3) Calculate $h_{i,p}, \hat{d}_{i,q}$

4) Gradient descent update V_{lj}, then W_{lj} (deep to shallow)

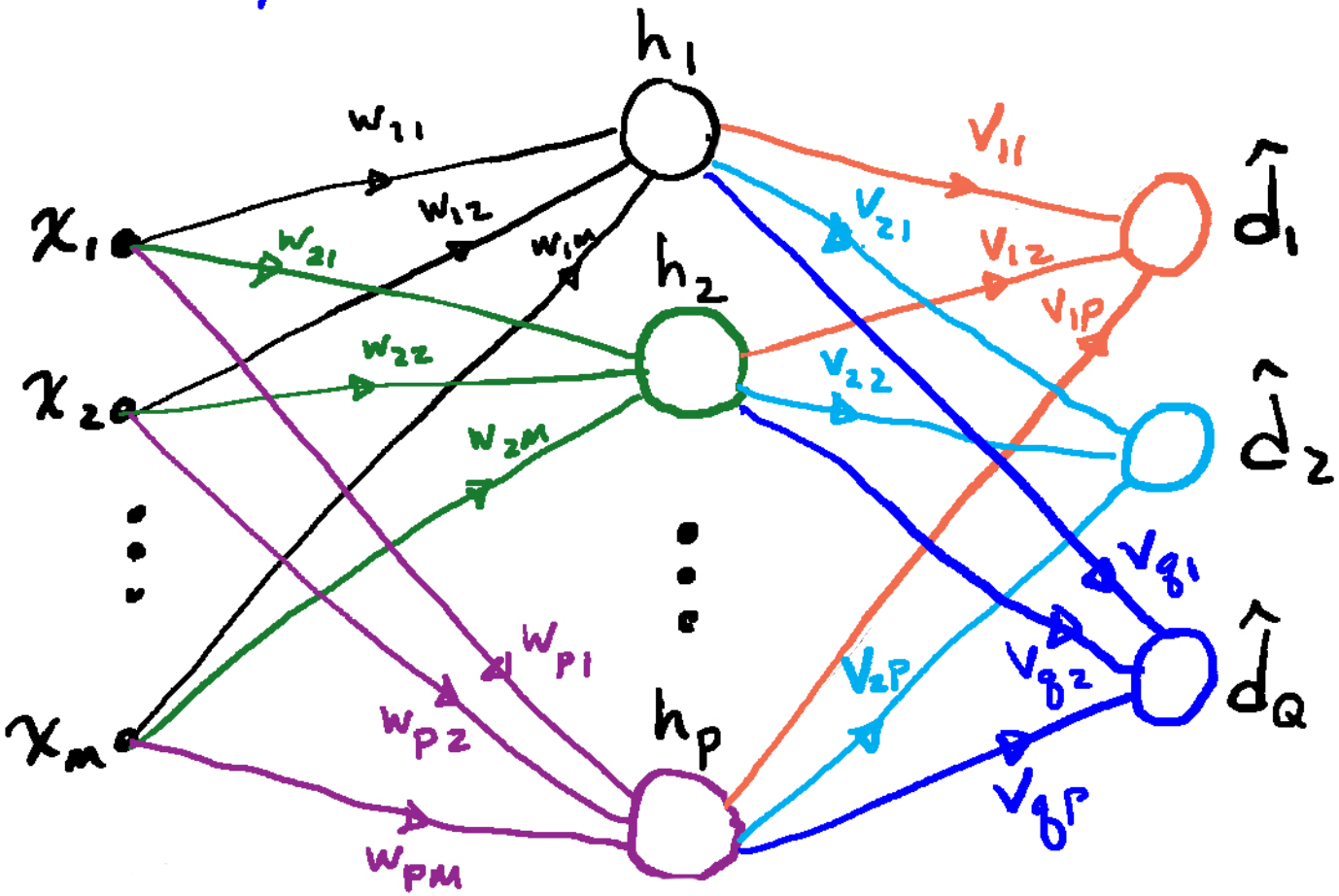
Chain rule is key for deriving gradients

The Backpropagation Algorithm for Training Neural Networks

Objectives

- Method for learning weights in a neural network
- Apply stochastic gradient descent (SGD)
- Find gradients of squared error loss with respect to weights

Backpropagation uses SGD to train the weights 2



N training samples

$(x_1^i, x_2^i \cdots x_m^i, d^i,\ i = 1, 2, \cdots N)$

1) initialize $w_{n,l}$, $v_{q,n}$

2) for $t = 1, 2, 3, \cdots$

→ Pick $i_t \in \{1, 2, \cdots, N\}$ randomly

Compute $h_j^{i_t}$, $\hat{d}_k^{i_t}$

$$v_{k,l}^{t+1} = v_{k,l}^t - \alpha_t \frac{d f^{i_t}}{d v_{k,l}}$$

$$w_{m,j}^{t+1} = w_{m,j}^t - \alpha_t \frac{d f^{i_t}}{d w_{m,j}}$$

— Converged?

$$\hat{d}_q = \sigma\left(\sum_{n=1}^{p} v_{q,n} h_n\right)$$

$\sigma(z)$: activation function

$$h_n = \sigma\left(\sum_{l=1}^{m} w_{n,l} x_l\right) \quad \left(= \frac{1}{1+e^{-z}} \text{ here}\right)$$

Gradients are obtained using the chain rule

Squared error loss: $f(\underline{V},\underline{W}) = \sum_{i=1}^{N} \underbrace{\frac{1}{2}\sum_{q=1}^{Q}(\hat{d}_q^i - d_q^i)^2}_{f^i(\underline{V},\underline{W})}$

$$\frac{\partial f^i}{\partial V_{k,\ell}} = \left(\frac{\partial f^i}{\partial \hat{d}_k^i}\right)\frac{\partial \hat{d}_k^i}{\partial V_{k,\ell}}$$

$\frac{1}{1+e^{-z}} = \sigma(z)$

$\frac{\partial}{\partial z}\sigma(z) = \sigma'(z)$

$$\frac{\partial f^i}{\partial V_{k,\ell}} = (\hat{d}_k^i - d_k^i)\cdot\sigma'\left(\sum_{n=1}^{P} h_n^i V_{k,n}\right)\cdot \frac{\partial}{\partial V_{k,\ell}}\left(\sum_{n=1}^{P} h_n^i V_{k,n}\right)$$

$$\sigma'(z) = \frac{1}{(1+e^{-z})^2}e^{-z} = \frac{1}{1+e^{-z}}\left(1 - \frac{1}{1+e^{-z}}\right)$$

$$= \sigma(z)(1-\sigma(z))$$

But $\sigma\left(\sum_{n=1}^{P} h_n^i V_{k,n}\right) = \hat{d}_k^i$

$$\left(1 - \sigma\left(\sum_{n=1}^{P} h_n^i V_{k,n}\right)\right) = \left(1 - \hat{d}_k^i\right)$$

$$\frac{\partial f^i}{\partial V_{k,\ell}} = \underbrace{(\hat{d}_k^i - d_k^i)\,\hat{d}_k^i(1-\hat{d}_k^i)}_{\delta_k^i} h_\ell^i$$

$$= \delta_k^i h_\ell^i$$

$$V_{k,\ell}^{t+1} = V_{k,\ell}^t - \alpha_t \delta_k^i h_\ell^i$$

h_ℓ^i: layer input

δ_k^i: "error" at output

Use a similar approach for $w_{m,j}$

$$\frac{\partial f^i}{\partial w_{m,j}} = \sum_{q=1}^{Q} \frac{\partial f^i}{\partial \hat{d}_q^i} \cdot \frac{\partial \hat{d}_q^i}{\partial h_m^i} \frac{\partial h_m^i}{\partial w_{m,j}} \qquad \frac{\partial f^i}{\partial \hat{d}_q^i} = (\hat{d}_q^i - d_q^i)$$

loss/output output/hidden hidden/prev. layer weights

$$\frac{\partial \hat{d}_q^i}{\partial h_m^i} = \frac{\partial}{\partial h_m^i} \sigma\left(\sum_{n=1}^{P} h_n^i v_{q,n}^i\right) = \sigma'\left(\sum_{n=1}^{P} h_n^i v_{q,n}^i\right) \cdot v_{q,m}^i = \hat{d}_q^i (1-\hat{d}_q^i) v_{q,m}^i$$

$$\frac{\partial h_m^i}{\partial w_{m,j}} = \frac{\partial}{\partial w_{m,j}} \sigma\left(\sum_{l=1}^{M} w_{m,l} x_l^i\right) = \sigma'\left(\sum_{l=1}^{M} w_{m,l} x_l^i\right) x_j^i = h_m^i (1-h_m^i) x_j^i$$

$$\frac{\partial f^i}{\partial w_{m,j}} = \underbrace{\sum_{q=1}^{Q} (\hat{d}_q^i - d_q^i) \hat{d}_q^i (1-\hat{d}_q^i) v_{q,m}^i h_m^i (1-h_m^i)}_{\gamma_m^i} x_j^i = \gamma_m^i x_j^i$$

$$w_{m,j}^{t+1} = w_{m,j}^t - \alpha^t \gamma_m^i x_j^i$$

x_j^i: layer input

γ_m^i: backpropagated "error"

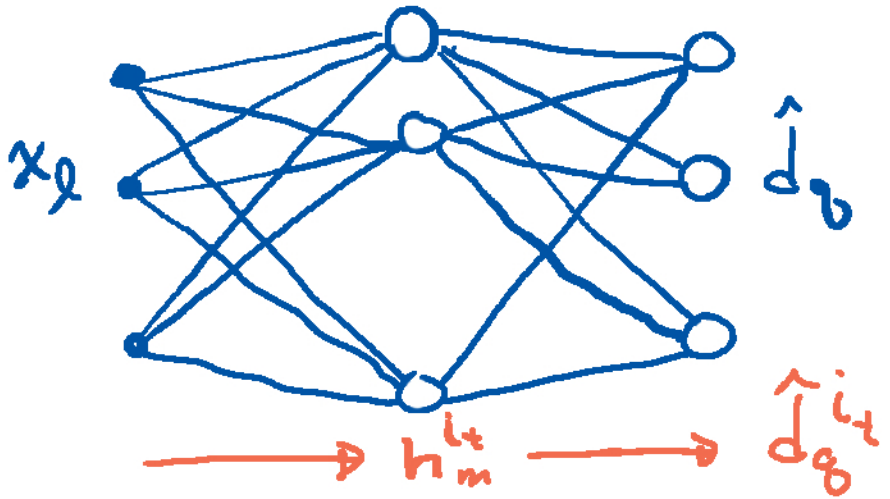
Backpropagation involves simple computations

Initialize $w^0_{m,j}$, $v^0_{k,\ell}$

Iterate for $t = 0, 1, 2, \ldots$

SGD — choose $i_t \in \{1, 2, \ldots, N\}$ at random

Forward Net — compute $h^{i_t}_m$, $\hat{d}^{i_t}_q$ from $x^{i_t}_\ell$, $w^t_{m,j}$, $v^t_{k,\ell}$



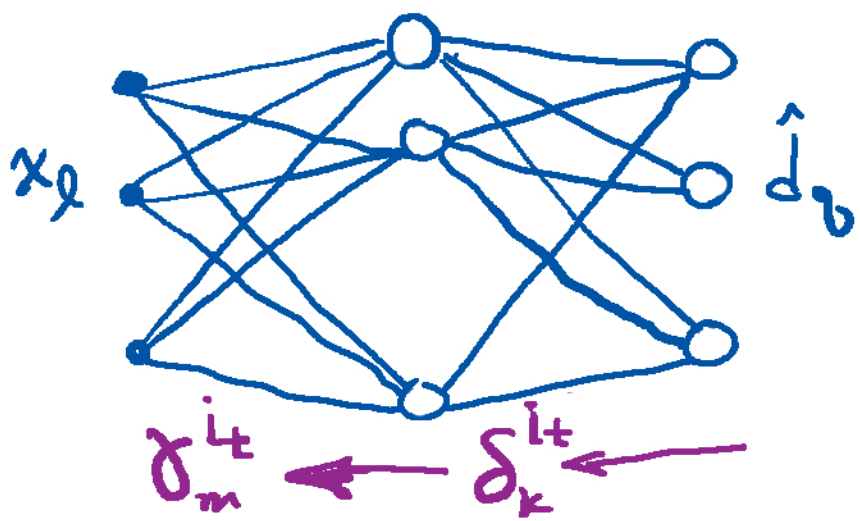
Backward Updates

$$\delta^{i_t}_k = \left(\hat{d}^{i_t}_k - d^{i_t}_k\right) \hat{d}^{i_t}_k \left(1 - \hat{d}^{i_t}_k\right)$$

$$v^{t+1}_{k,\ell} = v^t_{k,\ell} - \alpha_t \delta^{i_t}_k h^{i_t}_\ell$$

$$\gamma^{i_t}_m = \sum_{q=1}^{Q} \delta^{i_t}_q v^t_{q,m} h^{i_t}_m \left(1 - h^{i_t}_m\right)$$

$$w^{t+1}_{m,j} = w^t_{m,j} - \alpha_t \gamma^{i_t}_m x^{i_t}_j$$



Variations follow the same pattern

- other activation functions
- deeper networks

Cost is a non convex function of weights

- converge to local minima
- empirical "tricks" e.g. batch SGD, normalization to accelerate convergence

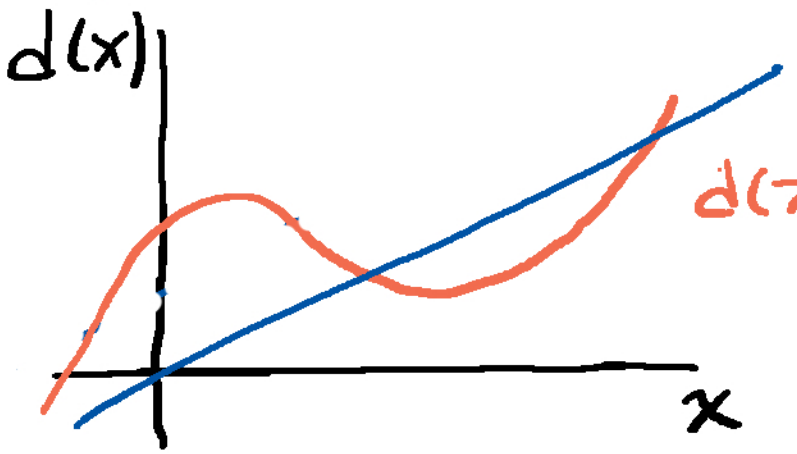
Can add regularization

Kernel Regression

Objectives

- Why use higher-dimensional feature spaces
- Reformulate regression in terms of kernels
- Popular kernels
- Cautions and considerations

Higher dimensional feature spaces extend regression



$d(x) = w_1 x$

$d(x) = w_3 x^3 + w_2 x^2 + w_1 x + w_0$

Let $\underline{x} = [x_1\ x_2 \cdots x_M]^T \in \mathbb{R}^M$

Consider $d(\underline{x}) = \underline{\phi}^T(\underline{x})\,\underline{w}$, $\underline{\phi}(\underline{x}) \in \mathbb{R}^P$, $P > M$

Example: $\underline{x} = [x_1, x_2]^T$, $\underline{\phi}^T(\underline{x}) = [x_1^2\ x_2^2\ \sqrt{2}x_1x_2\ x_1\ x_2\ 1]$

Finding $\underline{w}$: "training" data $\underline{x}^i, d^i$, $i = 1, 2, \cdots N$

$$\min_{\underline{w}} \sum_{i=1}^{N} (d^i - \underline{\phi}^T(\underline{x}^i)\,\underline{w})^2 + \lambda \|\underline{w}\|_2^2 \quad \text{(Ridge)}$$

$$\underline{d} = [d^1\ d^2 \cdots d^N]^T$$

$$\min_{\underline{w}} \|\underline{d} - \underline{\Phi}\,\underline{w}\|_2^2 + \lambda \|\underline{w}\|_2^2$$

$$\underline{\Phi} = [\underline{\phi}(\underline{x}^1)\ \underline{\phi}(\underline{x}^2) \cdots \underline{\phi}(\underline{x}^N)]^T \quad (N \times P) \Rightarrow \underline{w} = (\underline{\Phi}^T\underline{\Phi} + \lambda \underline{I})^{-1}\underline{\Phi}^T\underline{d}$$

Regression is a weighted sum of "kernels"

$$d(\underline{x}) = \underline{\phi}^T(\underline{x})\,\underline{w} = \underline{\phi}^T(\underline{x})\left(\underline{\Phi}^T\underline{\Phi} + \lambda \underline{I}\right)^{-1}\underline{\Phi}^T\underline{d}$$

$\underline{\Phi}^T\underline{\Phi} + \lambda \underline{I}$: P x P

Matrix identity: $(\underline{\Phi}^T\underline{\Phi}+\lambda\underline{I})^{-1}\underline{\Phi}^T = \underline{\Phi}^T(\underline{\Phi}\,\underline{\Phi}^T+\lambda\underline{I})^{-1}$ (activity)

Thus $d(\underline{x}) = \underline{\phi}^T(\underline{x})\,\underline{\Phi}^T(\underline{\Phi}\,\underline{\Phi}^T+\lambda\underline{I})^{-1}\underline{d}$

$\underline{\Phi}\,\underline{\Phi}^T+\lambda\underline{I}$: N x N

Note: $\left[\underline{\Phi}\,\underline{\Phi}^T\right]_{i,j} = \underline{\phi}^T(\underline{x}^i)\,\underline{\phi}(\underline{x}^j)$

$\left[\underline{\phi}^T(\underline{x})\,\underline{\Phi}^T\right]_j = \underline{\phi}^T(\underline{x})\,\underline{\phi}(\underline{x}^j)$

Define "Kernel" $K(\underline{u},\underline{v}) = \underline{\phi}^T(\underline{u})\,\underline{\phi}(\underline{v})$

Let $\underline{\alpha} = [\alpha_1 \cdots \alpha_N]^T = (\underline{\Phi}\,\underline{\Phi}^T+\lambda\underline{I})^{-1}\underline{d}$

$$d(\underline{x}) = \sum_{i=1}^{N} \alpha_i\,\underline{\phi}^T(\underline{x})\,\underline{\phi}(\underline{x}^i) = \sum_{i=1}^{N} \alpha_i K(\underline{x},\underline{x}^i)$$

Kernel methods find $d(\underline{x})$ without computing $\underline{\phi}(\underline{x})$ 4

$$d(\underline{x}) = \sum_{i=1}^{N} \alpha_i K(\underline{x}, \underline{x}^i) \qquad \underline{\alpha} = \left(\underline{\Phi}\,\underline{\Phi}^T + \lambda \underline{I}\right)^{-1} \underline{d} \rightarrow \underline{K} = \underline{\Phi}\,\underline{\Phi}^T$$

$$= \left(\underline{K} + \lambda \underline{I}\right)^{-1} \underline{d}$$

$$[\underline{K}]_{ij} = \underline{\phi}^T(\underline{x}^i)\,\underline{\phi}(\underline{x}^j) = K(\underline{x}^i, \underline{x}^j)$$

$\underline{K}$ can be computed efficiently!

Ex: Monomials of degree q $\quad \underline{\phi}(\underline{x}) \rightarrow x_1^q,\ x_1^{q-1}x_2,\ \cdots\ x_3^{q-5}x_6^2x_8^3 \cdots$

$$K(\underline{u}, \underline{v}) = \underbrace{\underline{\phi}^T(\underline{u})\,\underline{\phi}(\underline{v})}_{O(P)} = \underbrace{(\underline{u}^T\underline{v})^q}_{O(M)} \quad \text{(activity)} \qquad P = \frac{(q+M-1)!}{q!\,(M-1)!} \text{ terms}$$

Suppose $M = 10, q = 5 \rightarrow P \sim 2000$ computing $O(P)$ vs $O(M)$

$M = 100, q = 5 \rightarrow P \sim 10^8$ memory $O(NP)$ vs $O(N^2)$

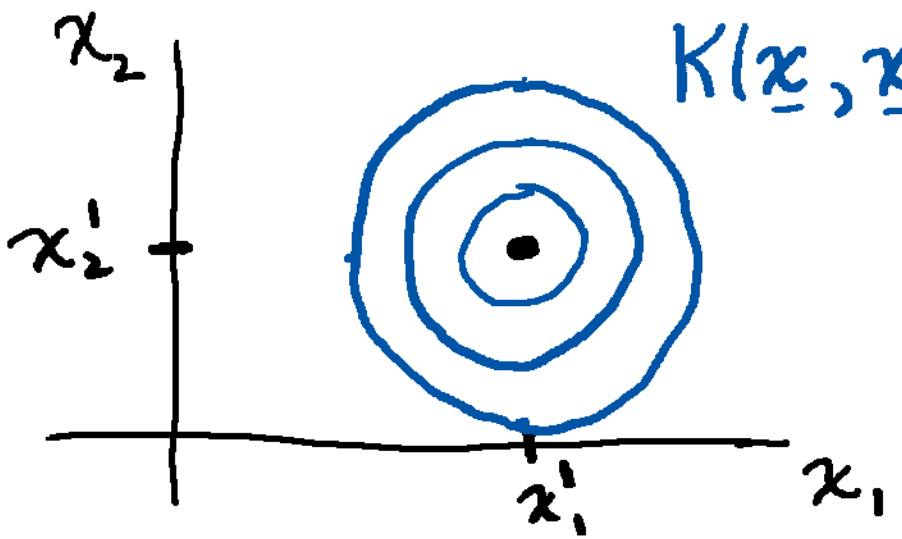
Popular kernels depend on similarity of $\underline{u}, \underline{v}$

$$\underline{u}^T \underline{v} = \|\underline{u}\|_2 \|\underline{v}\|_2 \cos\theta$$

Monomials of degree q: $K(\underline{u}, \underline{v}) = (\underline{u}^T \underline{v})^q$

Polynomials up to degree q: $K(\underline{u}, \underline{v}) = (\underline{u}^T \underline{v} + 1)^q$

Gaussian/radial kernel: $K(\underline{u}, \underline{v}) = \exp\left\{ - \frac{\|\underline{u} - \underline{v}\|_2^2}{2\sigma^2} \right\}$

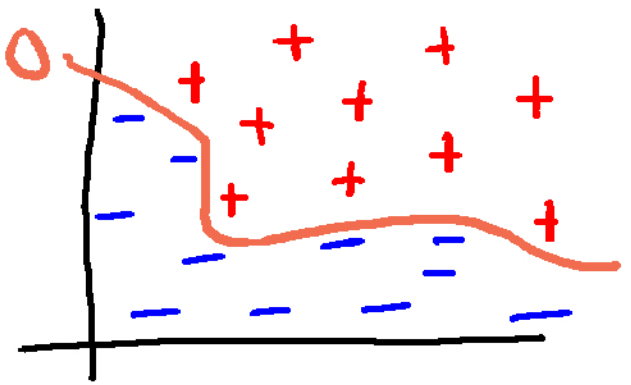


- No explicit $\underline{\phi}(\underline{x})$
- All polynomial orders
- smoothness controlled by σ

Kernel regression considerations

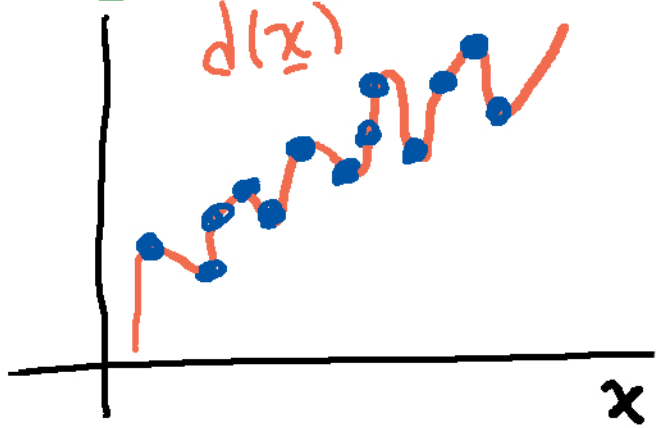
$$d(\underline{x}) = \underline{\phi}^T(\underline{x})\,\underline{w} \quad \text{vs} \quad d(\underline{x}) = \sum_{i=1}^{N} \alpha_i\, k(\underline{x}, \underline{x}^i)$$

- Store and compute $\underline{\alpha}$ $(N \times 1)$ vs $\underline{w}$ $(P \times 1)$
- Binary classification $\text{sign}\{d(\underline{x})\}$



- Avoid "overfitting" with high-D feature spaces

(cross-validation)



Kernel Based Support Vector Machines

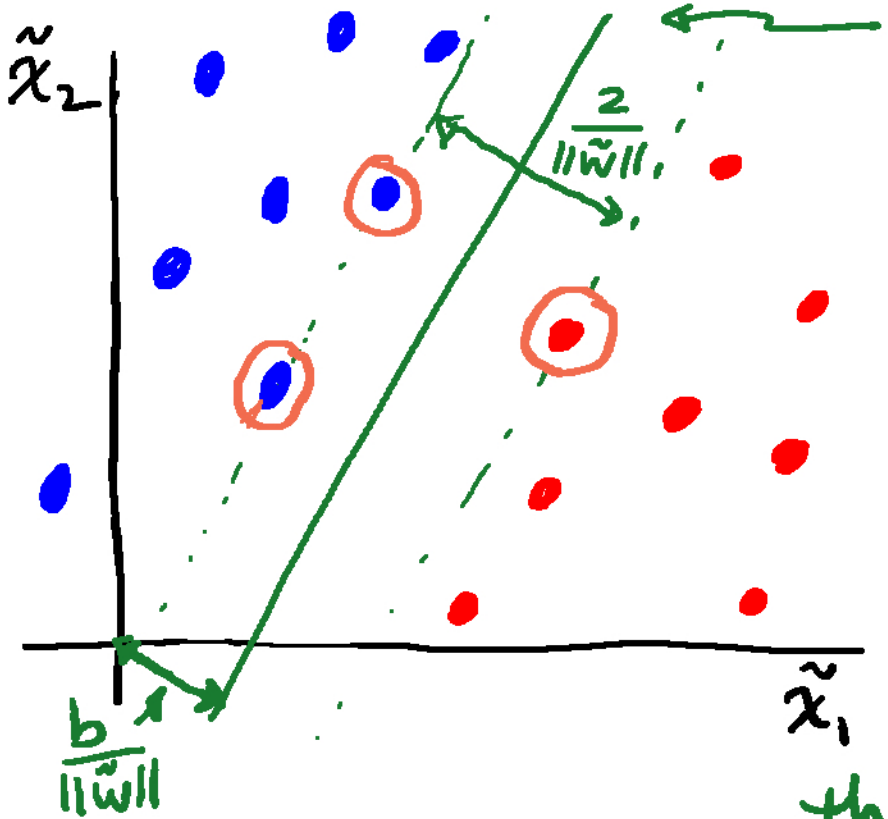
Objectives

- reformulate linear max margin classifier in terms of support vectors

- derive kernel version of hinge loss with ridge regression

- summarize features of support vector machines

Support vectors define max-margin classifier



$\tilde{x}^T\tilde{w} + b = 0$

$x^T w = 0$

$w = [\tilde{w}^T \; b]^T$ depends only on the support vectors

Recall kernel regression

$$d(x) = \phi^T(x) w = \sum_{j=1}^{N} \alpha_j K(x, x^j)$$

Let $\phi(x) = x \Rightarrow K(x, x^j) = x^T x^j$

then $d(x) = \sum_{j=1}^{N} \alpha_j x^T x^j = x^T \underbrace{\sum_{j=1}^{N} \alpha_j x^j}_{w}$

So $w = \sum_{j=1}^{N} \alpha_j x^j$

All $\alpha_j = 0$ except Support vectors!

Use kernels for nonlinear decision boundaries

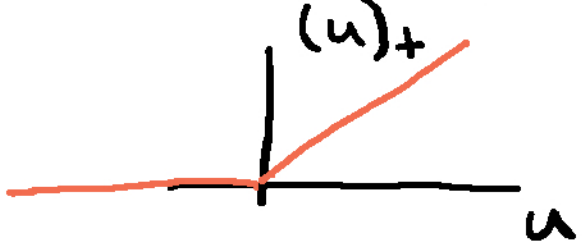
High-dimensional feature space: $\underline{x} \rightarrow \underline{\phi}(\underline{x})$

e.g., $\underline{\phi}(\underline{x}) = [\, x_1^2 \;\; x_2^2 \cdots x_2 x_4 \cdots x_{m-1} \;\; x_m \;\; 1 \,]$

$$\hat{d}(\underline{x}) = \text{sign}\left(\underline{\phi}^T(\underline{x})\,\underline{w}\right)$$

Hinge loss with ridge regression

$$\min_{\underline{w}} \sum_{i=1}^{N} \left(1 - d^i \underline{\phi}^T(\underline{x}^i)\,\underline{w}\right)_+ + \lambda \|\underline{w}\|_2^2$$



Claim: $\underline{w} = \sum_{j=1}^{N} \underline{\phi}(\underline{x}^j)\,\alpha_j$ (proof in notes)

Kernel "trick" replaces $\underline{\phi}^T(\underline{x}^i)\underline{\phi}(\underline{x}^i)$ with $K(\underline{x}^j, \underline{x}^i)$

Restate: $\underline{w} = \sum_{j=1}^{N} \alpha_j \underline{\phi}(\underline{x}^j)$

Hinge loss with ridge regression

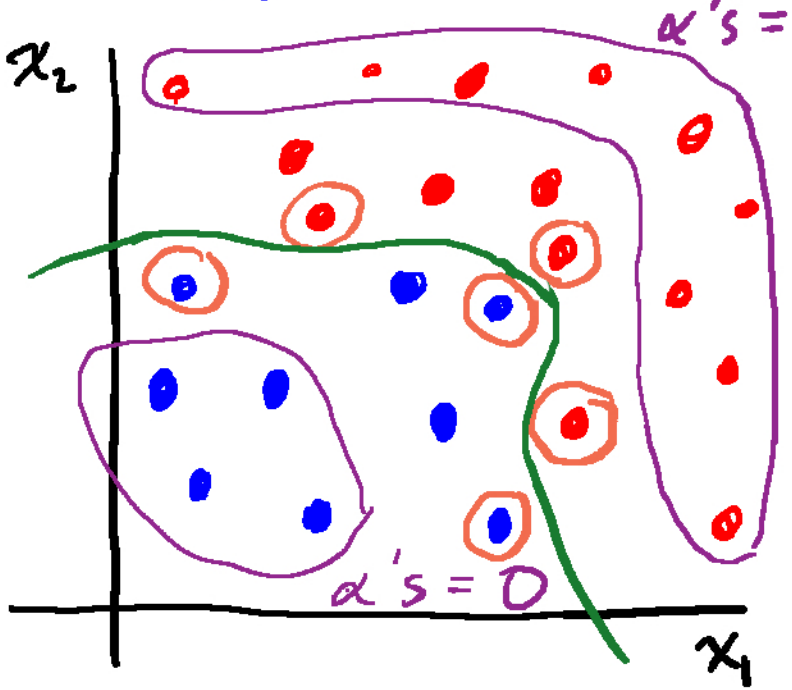
$$\min_{\underline{\alpha}} \sum_{i=1}^{N} \left(1 - d^i \underline{\phi}^T(\underline{x}^i) \underbrace{\sum_{j=1}^{N} \alpha_j \underline{\phi}(\underline{x}^j)}_{\underline{w}}\right)_+ + \lambda \underbrace{\sum_{i=1}^{N} \alpha_i \underline{\phi}^T(\underline{x}^i)}_{\underline{w}^T} \underbrace{\sum_{j=1}^{N} \alpha_j \underline{\phi}(\underline{x}^j)}_{\underline{w}}$$

$$\min_{\underline{\alpha}} \sum_{i=1}^{N} \left(1 - d^i \sum_{j=1}^{N} \alpha_j \underbrace{\underline{\phi}^T(\underline{x}^i)\underline{\phi}(\underline{x}^j)}_{K(\underline{x}^i, \underline{x}^j)}\right)_+ + \lambda \sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j \underbrace{\underline{\phi}^T(\underline{x}^i)\underline{\phi}(\underline{x}^j)}_{K(\underline{x}^i, \underline{x}^j)}$$

Kernel "trick"

SVM $$\min_{\underline{\alpha}} \sum_{i=1}^{N} \left(1 - d^i \sum_{j=1}^{N} \alpha_j K(\underline{x}^i, \underline{x}^j)\right)_+ + \lambda \sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j K(\underline{x}^i, \underline{x}^j)$$

Support vector machines have sparse $\underline{\alpha}$



decision boundary

$$d(\underline{x}) = 0 = \underline{\phi}^T(\underline{x})\underline{w} = \sum_{j=1}^{N} \alpha_j K(\underline{x}, \underline{x}^j)$$

Boundary (hinge loss) depends only on the support vectors

$K(\underline{u},\underline{v})$ measures similarity/alignment of $\underline{u}, \underline{v}$

Ex: Gaussian kernels

$$K(\underline{u},\underline{v}) = \exp\left\{-\frac{\|\underline{u}-\underline{v}\|_2^2}{2\sigma^2}\right\}$$

Solve for $\underline{\alpha}$ using gradient descent