

Rules of Probability

Sum Rule : $p(X) = \sum_Y p(X, Y)$

Product Rule : $p(X, Y) = p(Y|X) \cdot p(X)$

Bayes' Theorem : $p(Y|X) = \frac{p(X, Y)}{p(X)}$ where

$$p(X) = \sum_Y p(X|Y) p(Y)$$

Conditional Expectation : $E_X[f(y)] = \sum_x p(x|y) f(x)$

Bayes Decision Theory

1. Priors :

A priori
 $p(C_k)$

what we can tell about the probability before seeing the data

2. Conditional probabilities :

$p(x|C_k)$

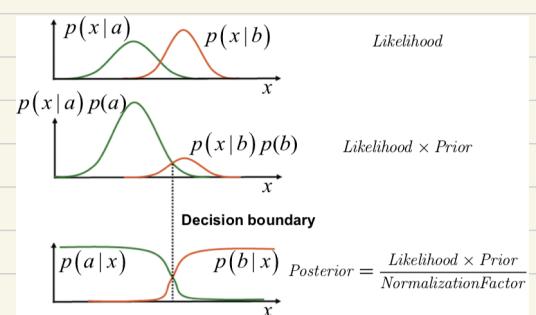
Likelihood for class C_k

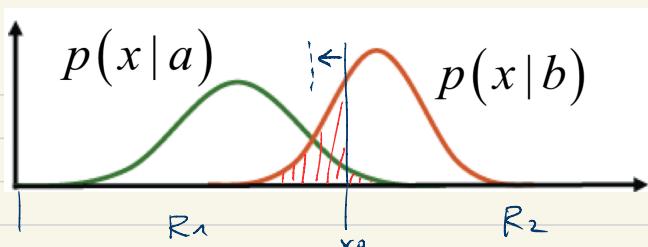
feature vector

3. Posterior $p(C_k|x)$: the probability of C_k given the x .

$$p(C_k|x) = \frac{p(x|C_k) \cdot p(C_k)}{p(x)} = \frac{p(x|C_k) \cdot p(C_k)}{\sum_i p(x|C_i) \cdot p(C_i)}$$

posterior = $\frac{\text{likelihood} \times \text{prior}}{\text{Normalisation Factor}}$





$$p(\text{mistake}) = p(x \in R_1, b) + p(x \in R_2, a)$$

= Area

Move $x_0 \rightarrow$ to the optimised

Optimal decision rule: $p(C_1|x) > p(C_2|x)$

$$\Leftrightarrow \frac{p(x|C_1)}{p(x|C_2)} > \frac{p(C_2)}{p(C_1)} = \theta \quad (\text{decision threshold})$$

[Likelihood ratio test]

Minimise the expected loss: $E[L] = \sum_k \sum_j \int_{R_j} L_{kj} p(x|C_k) dx$

$$\sim = \sum_k L_{kj} \underbrace{p(C_k|x)}_{\text{posterior}}$$

RISK $\left\{ \begin{array}{l} R(\alpha_1|x) = L_{11} p(C_1|x) + L_{21} p(C_2|x) \\ R(\alpha_2|x) = L_{12} p(C_1|x) + L_{22} p(C_2|x) \end{array} \right.$

$$R(\alpha_2|x) > R(\alpha_1|x) \Leftrightarrow \frac{p(x|C_1)}{p(x|C_2)} > \frac{(L_{21} - L_{22}) p(C_2)}{(L_{12} - L_{11}) p(C_1)}$$

Reject Option: the largest posterior probability $p(C_k|x)$
is significantly less than 1.

Discriminant Function

- $y_k(x) \propto p(x|C_k) \cdot p(C_k)$: Generative Methods

First determine the class-conditional densities for each class individually and separately infer the prior class probabilities. Then use Bayes' Theorem to determine class.

- $y_k(x) = p(C_k|x)$: Discriminative Methods

First solve the inference problem of determining the posterior class prob.

Then use decision theory to assign each new x to its class.

To solve: $x \rightarrow C_k : p(C_k|x)$

{ Parametric Representation

Non-Parametric Representation

Mixture Models.

posterior

Gaussian

$$N(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} \exp \left\{ -\frac{(x-\mu)^2}{2\sigma^2} \right\}$$

$$\{ N(x|\mu, \Sigma) = \frac{1}{(2\pi)^{D/2} |\Sigma|^{1/2}} \exp \left\{ -\frac{1}{2} (x-\mu)^T \Sigma^{-1} (x-\mu) \right\}$$

$$\Delta^2 = (x-\mu)^T \Sigma^{-1} (x-\mu) \quad : \quad \Delta: \text{Mahalanobis Distance},$$

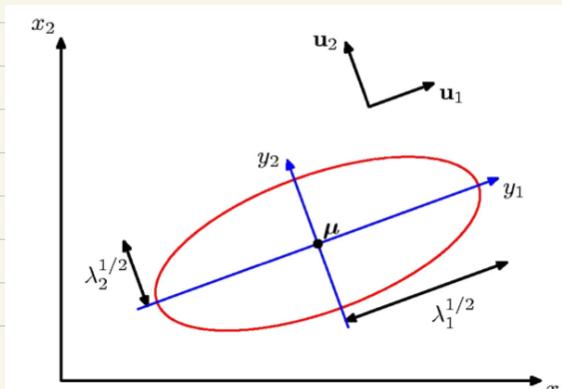
$$\Sigma = \sum_{i=1}^D \lambda_i u_i u_i^T, \quad \Sigma^{-1} = \sum_{i=1}^D \frac{1}{\lambda_i} u_i u_i^T$$

$$\Rightarrow \Delta^2 = \sum_{i=1}^D \frac{y_i^2}{\lambda_i^2} \quad \text{with } y_i = u_i^T (x-\mu)$$

$$= \sum_{i=1}^D \frac{(u_i^T (x-\mu))^T \cdot (u_i^T (x-\mu))}{\lambda_i^2} = \sum_{i=1}^D \frac{(x-\mu)^T u_i u_i^T (x-\mu)}{\lambda_i^2} = (x-\mu)^T \Sigma^{-1} (x-\mu)$$

\rightsquigarrow Constant density on ellipsoids with main directions along the eigenvectors u_i and scaling factors $\sqrt{\lambda_i}$

Σ diag: Axis-aligned / $\Sigma = \sigma^2 I$: Hypersphere.



Parametric Methods

$L(X|\theta)$: Likelihood of θ : Probability that X have indeed been generated from pd with θ .

MLE Estimation: $E(\theta) = -\ln L(\theta) = -\sum_{n=1}^N \ln p(x_n|\theta)$

$$(1D) \quad E(\theta) = -\sum_{n=1}^N \ln p(x_n|\mu, \sigma)$$

$$= -\sum_{n=1}^N \ln \left(\frac{1}{\sqrt{2\pi}\sigma} \exp \left\{ -\frac{(x_n - \mu)^2}{2\sigma^2} \right\} \right)$$

$$\frac{\partial E(\theta)}{\partial \mu} = -\sum_{n=1}^N \frac{\frac{\partial}{\partial \mu} p(x_n|\mu, \sigma)}{p(x_n|\mu, \sigma)} = -\sum_{n=1}^N -\frac{\sigma^2}{2\sigma^2} = \sum_{n=1}^N \frac{(x_n - \mu)}{2\sigma^2}$$

$$= \frac{1}{\sigma^2} \sum_{n=1}^N (x_n - \mu) \stackrel{!}{=} 0 \Rightarrow \hat{\mu} = \frac{1}{N} \sum_{n=1}^N x_n / \hat{\sigma}^2 = \frac{1}{N} \sum_{n=1}^N (x_n - \bar{x})^2$$

Biased! To fix: $\tilde{\sigma}^2 = \frac{N}{N-1} \hat{\sigma}_{ML}^2 = \frac{1}{N-1} \sum_{n=1}^N (x_n - \hat{\mu})^2$

- Limitations:
- It systematically underestimates σ of distribution.
 - Overfits to the observed data.

Non-Parametric Methods: Estimate probability density

Histogram $p_i = \frac{n_i}{N \Delta_i}$

General: In the limit $N \rightarrow \infty$ every pd can be represented

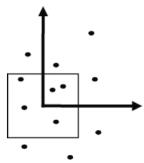
- No need to store data once hist is computed
- Rather brute-force

- Curse of dimensionality: $D \cdot M \text{ bins}^D = M^D \text{ bins}$ (exponential)

- Discontinuities at bin edges.

- Bin size?

too large: too much smoothing
too small: too much noise



$$P(x) \approx \frac{K}{NV}$$

local region sufficiently small.
 $p(x)$ in R roughly constant
 N sufficiently large

fixed K , determine V :

KNN

Kernel Methods

Parzen Window

$$k(u) = \begin{cases} 1, & |u| \leq \frac{1}{2}h \\ 0, & \text{else.} \end{cases}$$

$$K = \sum_{n=1}^N k(x - x_n), \quad V = \int k(u) du = h^D$$

$$p(x) = \frac{K}{NV} = \frac{1}{Nh^D} \sum_{n=1}^N k(x - x_n)$$

Gaussian Kernel

$$k(u) = \frac{1}{(2\pi h^2)^{D/2}} \exp \left\{ -\frac{u^2}{2h^2} \right\}$$

$$K = \sum_{n=1}^N k(x - x_n), \quad V = \int k(u) du = 1$$

$$p(x) \approx \frac{K}{NV} = \frac{1}{N} \sum_{n=1}^N \frac{1}{(2\pi)^{D/2} h} \cdot \exp \left\{ -\frac{\|x - x_n\|^2}{2h^2} \right\}$$

In general: $k(u) \geq 0, \int k(u) du = 1$.

$$K = \sum_{n=1}^N k(x - x_n), \quad p(x) \approx \frac{K}{NV} = \frac{1}{N} \sum_{n=1}^N k(x - x_n)$$

Kernel / KNN:

- General. In the limit $N \rightarrow \infty$, every probability density can be represented.
- No computation involved in the training phase.
(simply storage)
- Requires storing & computing with entire dataset.
- Computational cost: linear \rightarrow # data (\rightsquigarrow Tree)

h { too large \rightarrow too smooth
 K { too small \rightarrow not smooth enough

MoG

prior of component j

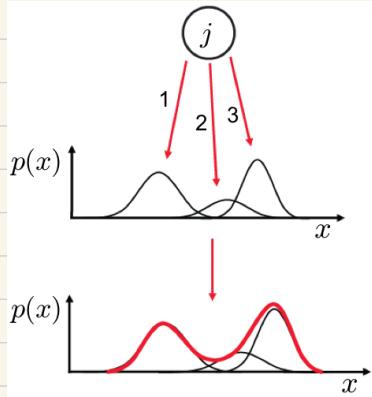
$$p(x|\theta) = p(x|\theta_j) p(j) \rightarrow p(j) = \pi_j \text{ with } \pi_j \in (0,1) \text{ & } \sum_{j=1}^M \pi_j = 1$$

$$p(x|\theta_j) = N(x|\mu_j, \sigma_j^2) = \frac{1}{\sqrt{2\pi}\sigma_j} \cdot \exp\left\{-\frac{1}{2\sigma_j^2}(x-\mu_j)^2\right\}$$

likelihood of x given mixture component j

Note: $\int p(x) dx = 1$.

Mixture parameter: $\Theta = (\pi_1, \mu_1, \sigma_1, \dots, \pi_M, \mu_M, \sigma_M)$



Generative model:

$$p(x|\theta) = \sum_{j=1}^M p(x|\theta_j) p(j)$$

Mixture density
Weight of mixture component

Mixture component

Estimation Attempt:

$$\text{Minimize } E = -\ln L(\theta) = -\sum_{n=1}^N \ln p(x_n|\theta)$$

$$\text{with } \ln p(x|\pi, \mu, \Sigma) = \sum_{n=1}^N \ln \left\{ \sum_{k=1}^K \pi_k N(x_n|\mu_k, \Sigma_k) \right\}$$

$$\begin{aligned} \frac{\partial E}{\partial \mu_j} &= -\sum_{n=1}^N \frac{\frac{\partial}{\partial \mu_j} p(x_n|\theta_j)}{\sum_{k=1}^K p(x_n|\theta_k)} = -\sum_{n=1}^N \left(\sum_{k=1}^K (x_n - \mu_j) \frac{p(x_n|\theta_j)}{\sum_{k=1}^K p(x_n|\theta_k)} \right) \\ &\stackrel{\partial}{\rightarrow} \frac{\partial}{\partial \mu_j} N(x_n|\mu_k, \Sigma_k) = \sum_{k=1}^K (x_n - \mu_j) N(x_n|\mu_k, \Sigma_k) \end{aligned}$$

$$= -\sum_{n=1}^N \sum_{k=1}^K (x_n - \mu_j) \cdot \frac{\pi_k N(x_n|\mu_j, \Sigma_j)}{\sum_{k=1}^K \pi_k N(x_n|\mu_k, \Sigma_k)} \stackrel{!}{=} 0 \quad : = \gamma_j(x_n)$$

responsibility

$$\Rightarrow \sum_{n=1}^N (x_n - \mu_j) \cdot \gamma_j(x_n) = 0, \quad \mu_j = \frac{\sum_{n=1}^N \gamma_j(x_n) \cdot x_n}{\sum_{n=1}^N \gamma_j(x_n)}$$

↓
No direct analytical solution!

EM (Expectation Maximization)

K-Means - Clustering:

- Initialisation: Pick K arbitrary centroids (cluster means)

(E)

- Assign each sample to the closest centroid.

(M)

- Adjust the centroids to be the means of the samples assigned to them.
- Goto , until no change.

$$J = \sum_{n=1}^N \sum_{k=1}^K r_{nk} \|x_n - \mu_k\|^2$$

$$\text{where } r_{nk} = \begin{cases} 1 & \text{if } k = \arg \min_j \|x_n - \mu_j\|^2 \\ 0 & \text{otherwise} \end{cases}$$

(Indicator to check whether μ_k is the nearest cluster center to point x_n)

- + Simple, fast to compute.
- + Converges to local optimum of within-cluster squared error
- Speedups through search structures
- K-Medoids (member of data).
- Setting k ?
- sensitive to initial centers
- sensitive to outliers
- Detects spherical clusters only

Credit Assignment Problem

$$P(j=1 | x, \theta) = \frac{P(j=1, x | \theta)}{P(x | \theta)}$$

Evaluate posterior probability that an observed x was generated from 1. mixture component.

$$P(j=1 | x, \theta) = P(x | j=1, \theta) P(j=1) = P(x | \theta_1) P(j=1)$$

$$P(j=1 | x, \theta) = \frac{P(x | \theta_1) P(j=1)}{\sum_{j=1}^2 P(x | \theta_j) P(j)} = y_j(x)$$

E-Step: softly assign samples to mixture components.

$$\gamma_j(x_n) \leftarrow \frac{\pi_j N(x_n | \mu_j, \Sigma_j)}{\sum_{k=1}^K \pi_k N(x_n | \mu_k, \Sigma_k)}, \quad \forall j = 1 \dots K, n = 1 \dots N$$

M-Step: re-estimate the parameters $\boldsymbol{\theta}$ separately for each mixture component based on the soft assignments.

$$\hat{N}_j \leftarrow \sum_{n=1}^N \gamma_j(x_n) = \text{soft number of samples labeled } j$$

$$\hat{\mu}_j^{\text{new}} \leftarrow \frac{\hat{N}_j}{N}, \quad \hat{\mu}_j^{\text{new}} \leftarrow \frac{1}{\hat{N}_j} \sum_{n=1}^N \gamma_j(x_n) \cdot x_n$$

$$\hat{\Sigma}_j^{\text{new}} \leftarrow \frac{1}{\hat{N}_j} \sum_{n=1}^N \gamma_j(x_n) (x_n - \hat{\mu}_j^{\text{new}})(x_n - \hat{\mu}_j^{\text{new}})^T$$

Techniques:

- use $(\Sigma + \sigma_{\min} I)^{-1}$ instead of Σ^{-1} to avoid singularity
- Initialise with k-Means!
(Arbitrary, but convergence much faster)
- Run k-Means M times (10~100)
 - Pick the best result (lowest J)
 - Use it as initial:
 - Set μ_j to the corresponding cluster mean from k-Means
 - Initialise Σ_j to the sample covariance of the associated data.

MoG Summary:

- + General, can represent any continuous distribution.
- + Once trained, fast to evaluate
- + Can be updated online.
- Avoid singularity
- Computationally expensive (high dim, More overhead & slow converge vs. K-Means
Sensitive to initial)
- Select K.

Discriminant Function

- Bayesian:
- Model conditional pd: $p(x|C_k)$
 - prior prob: $p(C_k)$
 - Compute $p(C_k|x)$
 - Min. Misclassification by max. $p(C|x)$

- New:
- Directly encode decision boundary
 - Without explicit modeling of pd.
 - Minimise misclass prob. directly.

Linear discriminant function: $y(x) = \tilde{w}^\top \tilde{x} + w_0$

$$y(x) = \tilde{w}^\top \tilde{x}$$

\swarrow
weight vec

\searrow
bias
threshold

$$= \sum_{i=0}^D w_i x_i \text{ with } x_0 = 1$$

To solve: $\hat{Y}(\tilde{x}) = \tilde{x}^\top \tilde{w} , \tilde{x}^\top \tilde{w} - T \underset{\min}{\text{min}}$

Least-Square : sum of square error

$$E(w) = \frac{1}{2} \sum_{n=1}^N \sum_{k=1}^K (y_k(x_n; w) - t_{kn})^2$$

$$= \frac{1}{2} \sum_{n=1}^N \sum_{k=1}^K (w_k^\top x_n - t_{kn})^2$$

Matrix

$$E_D(\tilde{w}) = \frac{1}{2} \text{Tr} \{ (\tilde{x} \tilde{w} - T)^\top (\tilde{x} \tilde{w} - T) \}$$

$$\sum_{ij} a_{ij}^2 = \text{Tr}(A^\top A)$$

$$\frac{\partial}{\partial \tilde{w}} E_D(\tilde{w}) = \frac{1}{2} \frac{\partial}{\partial \tilde{w}} \text{Tr} \{ (\tilde{x} \tilde{w} - T)^\top (\tilde{x} \tilde{w} - T) \}$$

$$= \frac{1}{2} \frac{\partial}{\partial (\tilde{x} \tilde{w} - T)^\top (\tilde{x} \tilde{w} - T)} \text{Tr} \{ (\tilde{x} \tilde{w} - T)^\top (\tilde{x} \tilde{w} - T) \} \cdot \frac{\partial}{\partial \tilde{w}} (\tilde{x} \tilde{w} - T)^\top (\tilde{x} \tilde{w} - T)$$

$$= \tilde{x}^\top (\tilde{x} \tilde{w} - T) \stackrel{!}{=} 0$$

$$\boxed{\frac{\partial}{\partial A} \text{Tr}\{A\} = I}$$

$$\Rightarrow \tilde{w} = \tilde{x}^\top T , y(x) = \tilde{w}^\top \tilde{x} = T^\top (\tilde{x}^\top)^\top \tilde{x}$$

- Least-Square: LS corresponds to ML under the assumption of Gaussian.

~ Generalised Linear Models

Activation function: $g(\cdot)$: $y(x) = g(w^T x + w_0)$
 Monotonic \uparrow

$$P(C_i|x) = \frac{P(x|C_i) P(C_i)}{P(x|C_1) P(C_1) + P(x|C_2) P(C_2)} = \frac{1}{1 + \frac{P(x|C_2) P(C_2)}{P(x|C_1) P(C_1)}} = \frac{1}{1 + e^{-a}}$$

Logistic Sigmoid: $g(a) \equiv \frac{1}{1 + e^{-a}}$

Normalised Exponential / Softmax :

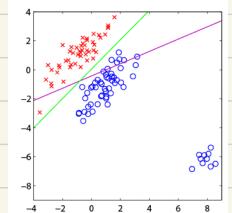
\leadsto perceptron

$$P(C_k|x) = \frac{P(x|C_k) P(C_k)}{\sum_j P(x|C_j) P(C_j)} = \frac{e^{a_k}}{\sum_j e^{a_j}}$$

\leadsto posterior

Motivation of Nonlinearity

- limit the influences of "too correct" data points
- Not linear separable



Generalized Linear Models

- + Nonlinearity \rightarrow more flexibility
- + to limit the effect of outliers
- + Choice of Sigmoid leads to a nice probabilistic interpretation
- Least Square Min. in general not analytically solved
- \Rightarrow iterative
- \Rightarrow Gradient Descent

Basis Function

$$y_k(x) = \sum_{j=1}^M w_{kj} \phi_j(x) + w_{k0} = \sum_{j=0}^M w_{kj} \phi_j(x) \quad \text{with } \phi_0(x) = 1$$

\hookrightarrow basis function

- Allow non-linear decision boundaries
- By choosing the right ϕ_j , every continuous func. can be approximated with arbitrary accuracy.

Gradient Descent

$$w_{kj}^{(T+1)} = w_{kj}^{(T)} - y \cdot \frac{\partial E(w)}{\partial w_{kj}} \Big|_{w^{(T)}}$$

Batch learning: Compute gradient based on all training data

$$\frac{\partial E(w)}{\partial w_{kj}}$$

Sequential updating: Compute grad based on a single data point at a time

$$\frac{\partial E_n(w)}{\partial w_{kj}}$$

Error Function:

$$E(w) = \sum_{n=1}^N E_n(w) = \frac{1}{2} \sum_{n=1}^N \sum_{k=1}^K \left(\sum_{j=1}^M w_{kj} \phi_j(x_n) - t_{kn} \right)^2$$

$$\Rightarrow E_n(w) = \frac{1}{2} \sum_{k=1}^K \left(\sum_{j=1}^M w_{kj} \phi_j(x_n) - t_{kn} \right)^2$$

$$\frac{\partial E_n(w)}{\partial w_{kj}} = \left(\sum_{j=1}^M w_{kj} \phi_j(x_n) - t_{kn} \right) \phi_j(x_n)$$

$$= (y_k(x_n; w) - t_{kn}) \phi_j(x_n)$$

Delta Rule:

$$w_{kj}^{(T+1)} = w_{kj}^{(T)} - y (y_k(x_n; w) - t_{kn}) \phi_j(x_n)$$

$$= w_{kj}^{(T)} - y \cdot \delta_{kn} \phi_j(x_n)$$

Activation function diff'bar, non linear: $\delta_{kn} = \frac{\partial g(a_k)}{\partial w_{kj}} (y_k(x_n; w) - t_{kn})$

$$y_k(x) = g(a_k) = g \left(\sum_{j=0}^M w_{kj} \phi_j(x_n) \right)$$

+ General class of decision function

+ $g(\cdot) + \phi_j$ allow non-separable.

+ GD or 2. order GD (Newton-Raphson)

- curse of dim: $g(\cdot) + \phi_j$ introduce additional para.

- Overfitting

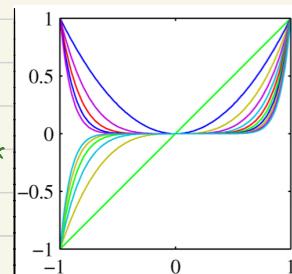
Linear basis function ϕ_j

Polynomial
 $\phi_j(x) = x^j$

Global: A small change in x affects all basis functions.

The decision boundary: polynomial func of x
 (Non linear)

Still solve a linear problem in $\phi(x)$

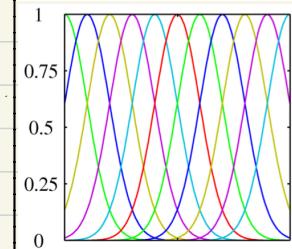


Gaussian

$$\phi_j(x) = \exp\left\{-\frac{(x-\mu_j)^2}{2s^2}\right\}$$

Local: A small change in x affects only nearby basis functions

μ_j and s control location and scale (width)



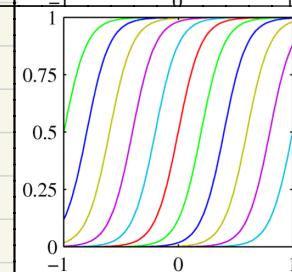
Sigmoid

$$\phi_j(x) = \sigma\left(\frac{x-\mu_j}{s}\right)$$

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

Local: A small change in x affects only nearby basis functions

μ_j and s control location and scale (slope)



Probabilistic Discriminative Models

$$p(C_1|x) = \sigma(a) = \frac{1}{1+e^{-a}}, \quad a = \ln \frac{p(x|C_1)p(C_1)}{p(x|C_2)p(C_2)}, \quad a = w^T \phi(x)$$

Logistic Regression

$$p(C_1|\phi) = y(\phi) = \sigma(w^T \phi)$$

$$p(C_2|\phi) = 1 - p(C_1|\phi)$$

$$p(C_1|\phi) = \frac{p(\phi|C_1)p(C_1)}{p(\phi|C_1)p(C_1) + p(\phi|C_2)p(C_2)}$$

Parameter: Mean: $2M$ $\dim(\bar{\Phi}_1) + \dim(\bar{\Phi}_2)$

2 classes

$M - \dim \phi$

$$\text{Covariance: } \frac{M \cdot (M+1)}{2} \text{ cov}(\Delta)$$

class: 1

$$\left. \begin{array}{l} \frac{M(M+5)}{2} + 1 \end{array} \right\}$$

$$p(C_1|\phi) = y(\phi) = \sigma(w^T \phi) = M$$

↑

Logistic Sigmoid $\sigma(a) = \frac{1}{1+e^{-a}}$, $\Rightarrow a = \ln\left(\frac{\sigma}{1-\sigma}\right)$

Symmetry: $\sigma(-a) = 1 - \sigma(a)$

Derivative: $\frac{d\sigma}{da} = \sigma(1-\sigma)$

With $y_n = p(a|\phi_n)$: $p(t|w) = \prod_{n=1}^N y_n^{t_n} \{1-y_n\}^{1-t_n}$ (Likelihood)

Negative Log-likelihood $E(w) = -\ln p(t|w)$

Cross-Entropy

$$= - \sum_{n=1}^N \{ t_n \ln y_n + (1-t_n) \ln (1-y_n) \}$$

$$\nabla E(w) = - \sum_{n=1}^N \left\{ t_n \frac{\frac{d}{dw} y_n}{y_n} + (1-t_n) \frac{\frac{d}{dw} (1-y_n)}{1-y_n} \right\}$$

$$= - \sum_{n=1}^N \left\{ t_n \frac{y_n(1-y_n)}{y_n} \phi_n - (1-t_n) \frac{y_n(1-y_n)}{1-y_n} \phi_n \right\}$$

$$\sim = \sum_{n=1}^N (y_n - t_n) \phi_n$$

$$\begin{cases} y_n = \sigma(w^T \phi_n) \\ \frac{d}{dw} y_n = y_n(1-y_n)\phi_n \end{cases}$$

Delta Rule: $w_{kj}^{(T+1)} = w_{kj}^{(T)} - y_j (y_k(x_n; w) - t_n) \phi_j(x_n)$

\Rightarrow A more efficient iterative method:

2. order Newton-Raphson Gradient descent :

$$w^{(T+1)} = w^{(T)} - H^{-1} \nabla E(w) \quad \text{with } H = \nabla \nabla E(w)$$

(Hessian)

- Local quadratic approximation to the log-likelihood.
- Faster convergence

Newton-Raphson for LSE (Least-Squares Estimation)

$$E(w) = \frac{1}{2} \sum_{n=1}^N (w^\top \phi_n - t_n)^2$$

$$\nabla E(w) = \sum_{n=1}^N (w^\top \phi_n - t_n) \phi_n = \Phi^\top \Phi w - \Phi^\top t$$

$$\nabla \nabla E(w) = \sum_{n=1}^N \phi_n \phi_n^\top = \Phi^\top \Phi$$

$$\Rightarrow w^{(T+1)} = w^{(T)} - (\Phi^\top \Phi)^{-1} (\Phi^\top \Phi w^{(T)} - \Phi^\top t)$$

$$= (\Phi^\top \Phi)^{-1} \Phi^\top t \quad \text{Closed-form!}$$

Newton-Raphson for cross-entropy:

$$E(w) = - \sum_{n=1}^N \{ t_n \ln y_n + (1-t_n) \ln (1-y_n) \}$$

$$\nabla E(w) = \sum_{n=1}^N (y_n - t_n) \phi_n = \Phi^\top (y - t)$$

$$\nabla \nabla E(w) = \sum_{n=1}^N y_n (1-y_n) \phi_n \phi_n^\top = \Phi^\top R \Phi$$

↑ weighting
Matrix

$$\frac{dy_n}{dw} = y_n (1-y_n) \phi_n$$

$$R_{nn} = y_n (1-y_n) .$$

R : diagonal

$$\Rightarrow w^{(T+1)} = w^{(T)} - (\Phi^\top R \Phi)^{-1} \cdot \Phi^\top (y - t)$$

$$= (\Phi^\top R \Phi)^{-1} \cdot (\Phi^\top R \Phi w^{(T)} - \Phi^\top (y - t))$$

$$= (\Phi^\top R \Phi)^{-1} \cdot \Phi^\top R z$$

$$\text{with } z = \Phi w^{(T)} - R^{-1} (y - t)$$

- R non-constant (c depends on w)
- Need to apply normal equations iteratively

Iteratively Reweighted Least-Squares

Logistic Regression Summary

- | | |
|--|--|
| <ul style="list-style-type: none"> • Directly represent posterior $P(\phi C_k)$ • Cross-entropy is concave: Unique optim., no closed-form, IRLS • Both online & Batch optimisations exist. - Tend to systematically overestimate odds ratios when sample size $< \sim 500$. | <ul style="list-style-type: none"> • Fewer para than likelihood + prior |
|--|--|

Softmax Regression

- Multiclass generalisation of logistic regression.

In logistic regression : $t_n \in \{0, 1\}$

Softmax : K values in 1-of- K notation

$$y(x; w) = \begin{bmatrix} P(y=1|x; w) \\ P(y=2|x; w) \\ \vdots \\ P(y=K|x; w) \end{bmatrix} = \frac{1}{\sum_{j=1}^K \exp(w_j^T x)} \cdot \begin{bmatrix} \exp(w_1^T x) \\ \exp(w_2^T x) \\ \vdots \\ \exp(w_K^T x) \end{bmatrix}$$

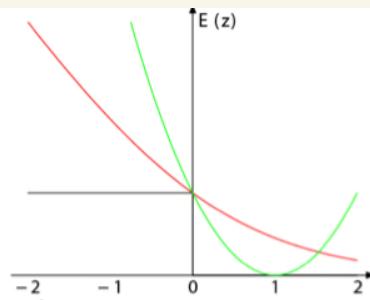
$\leadsto \frac{\exp(ax)}{\sum \exp(ax)}$ (normalised)

Logistic : $E(w) = -\sum_{n=1}^N \sum_{k=0}^{K-1} \{ \mathbb{I}(t_n=k) \ln P(y_n=k|x_n; w) \}$

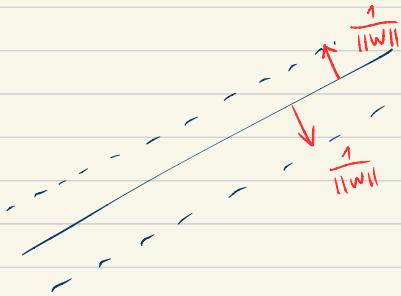
Softmax : $E(w) = -\sum_{n=1}^N \sum_{k=1}^K \left\{ \mathbb{I}(t_n=k) \ln \frac{\exp(w_k^T x)}{\sum_{j=1}^K \exp(w_j^T x)} \right\}$

GD : $\nabla_{w_k} E(w) = -\sum_{n=1}^N [\mathbb{I}(t_n=k) \ln P(y_n=k|x_n; w)]$

- Ideal Misclassification Error
 - This is what we would like to optimize.
 - But cannot compute gradients here.
- Quadratic Error
 - Easy to optimize, closed-form solutions exist.
 - But not robust to outliers.
- Cross-Entropy Error
 - Minimizer of this error is given by posterior class probabilities.
 - Concave error function, unique minimum exists.
 - But no closed-form solution, requires iterative estimation.



SVM



$$t_n (w^T x_n + b) \geq 1 \quad \forall n$$

maximising margin $\Leftrightarrow \min \|w\|^2$

Original QP :

$$\underset{w, b}{\operatorname{argmin}} \frac{1}{2} \|w\|^2, \text{ s.t. } t_n (w^T x_n + b) \geq 1 \quad \forall n$$

SGD

Primal : $L_p = \frac{1}{2} \|w\|^2 - \sum_{n=1}^N a_n (t_n y(x_n) - 1)$

KKT : $a_n \geq 0, t_n y(x_n) - 1 \geq 0, a_n (t_n y(x_n) - 1) = 0$

KKT:
 $\lambda \geq 0$
 $f(x) \geq 0$
 $\lambda f(x) = 0$

d+1 variables

Dual : $L_d(a) = \sum_{n=1}^N a_n - \frac{1}{2} \sum_{n=1}^N \sum_{m=1}^N a_n a_m t_n t_m K(x_m, x_n)$

$$\left. \begin{array}{l} 0 \leq a_n \leq C \\ \sum_{n=1}^N a_n t_n = 0 \end{array} \right\}$$

$$w = \sum_{n=1}^N a_n t_n x_n \quad \text{support vectors}$$

n+1 variables

Comparison : L_d only depends on a_n

Primal Dual $L_p : \mathcal{O}(D^3) \text{ vs. } L_d : \mathcal{O}(N^3)$

$$\rightarrow \mathcal{O}(N) - \mathcal{O}(N^3)$$

AdaBoost & Ensemble

↓

- Assume we have K classifiers.
 - They're independent (errors uncorrelated)
 - error $p_{c,s}$
- Bagging**
- (Bootstrap Aggregation)** ↵
- A simple majority vote of all classifiers should have a lower error than each individual classifier.

Bayesian Model Averaging

$$p(X) = \sum_{h=1}^H p(X|h) p(h)$$

- Just one model is responsible for generating the entire data set.
- The probability distribution over h just reflects our uncertainty which model that is.
- As the size of the data set increases, this uncertainty reduces. $\Rightarrow p(X|h)$ becomes focused on just one of the models.

MoG (Model Combination)

- Different data points generated by different model components
- Uncertainty is about which component created which data point.
 \Rightarrow One latent z_n for each point:

$$p(X) = \prod_{n=1}^N p(x_n) = \prod_{n=1}^N \sum_{z_n} p(x_n, z_n)$$

Bayesian Model Averaging

- Whole data generated by a single model
- Uncertainty is about which model was responsible
 \Rightarrow One latent z for entire data

$$p(X) = \sum_z p(X, z)$$

Expected Error

Combine M predictors $y_m(x)$ for target output $h(x)$

$$y_{\text{com}}(x) = \frac{1}{M} \sum_{m=1}^M y_m(x) : \text{committee}$$

$$y(x) = h(x) + e(x)$$

$$E_x = \left[\{y_m(x) - h(x)\}^2 \right] = E_x [e_m^2(x)]$$

Average error of individual models: $E_{\text{AV}} = \frac{1}{M} \sum_{m=1}^M E_x [e_m^2(x)]$

Average error of committee: $E_{\text{com}} = E_x \left[\left(\frac{1}{M} \sum_{m=1}^M y_m(x) - h(x) \right)^2 \right]$

with assumptions:

$$= E_x \left[\left(\frac{1}{M} \sum_{m=1}^M e_m(x) \right)^2 \right]$$

$$\left. \begin{array}{l} E_x(e_m(x)) = 0 \quad \text{zero mean} \\ E_x(e_m(x) e_j(x)) = 0 \quad \text{uncorrelated} \end{array} \right.$$

$$\Rightarrow E_{\text{com}} = \frac{1}{M} E_{\text{AV}}$$

↳ In practice: usually highly correlated!

Adaptive Boosting

- Iteratively select an ensemble of component classifiers
- After each iteration, reweight misclassified training examples.
 - increase the chance of being selected in a sampled training set
 - Increase the misclassification cost when training on the full set

$h_m(x)$: weak classifier / base classifier

$H(x)$: strong classifier / final classifier.

$$H(x) = \text{sign} \left(\sum_{m=1}^M \alpha_m h_m(x) \right)$$

AdaBoost

1. Initialisation : Set $w_n^{(1)} = \frac{1}{N} \quad \forall n=1 \dots N$

2. For $m=1 \dots M$ iterations

a) Train a new weak classifier $h_m(x)$ using the current weighting coefficients $W^{(m)}$ by minimising the weighted error function :

$$J_m = \sum_{n=1}^N w_n^{(m)} I(h_m(x) \neq t_n)$$

b) Estimate the weighted error of this classifier on X :

$$\epsilon_m = \frac{\sum_{n=1}^N w_n^{(m)} I(h_m(x) \neq t_n)}{\sum_{n=1}^N w_n^{(m)}}$$

c) Calculate a weighting coefficient for $h_m(x)$

$$\alpha_m = \ln \left\{ \frac{1 - \epsilon_m}{\epsilon_m} \right\}$$

d) Update the weighting coefficients :

$$w_n^{(m+1)} = w_n^{(m)} \exp \left\{ \alpha_m I(h_m(x) \neq t_n) \right\}$$

AdaBoost Summary

- Properties

- Simple combination of multiple classifiers.
- Easy to implement.
- Can be used with many different types of classifiers.
 - None of them needs to be too good on its own.
 - In fact, they only have to be slightly better than chance.
- Commonly used in many areas.
- Empirically good generalization capabilities.

- Limitations

- Original AdaBoost sensitive to misclassified training data points.
 - Because of exponential error function.
 - Improvement by GentleBoost
- Single-class classifier
 - Multiclass extensions available

Minimising Exponential Error :

$$E = \sum_{n=1}^N \exp \left\{ -t_n f_m(x_n) \right\}$$

$$= \sum_{n=1}^N \exp \left\{ -t_n f_{m-1}(x_n) - \frac{1}{2} t_n \alpha_m h_m(x_n) \right\}$$

$= \text{const.}$

$$= \sum_{n=1}^N w_n^{(m)} \exp \left\{ -\frac{1}{2} t_n \alpha_m h_m(x_n) \right\}$$

$$f_m(x) = \frac{1}{2} \sum_{l=1}^m \alpha_l h_l(x)$$

$$f_{m-1}(x) = \frac{1}{2} \sum_{l=1}^{m-1} \alpha_l h_l(x)$$

$$\Rightarrow f_m(x) = f_{m-1}(x) + \frac{1}{2} \alpha_m h_m(x)$$

Sequential Minimisation : Suppose $h_1(x) \dots h_{m-1}(x)$ & $\alpha_1 \dots \alpha_{m-1}$ are fixed \Rightarrow only minimise w.r.t. α_m h_m(x)

$$E = e^{-\alpha_m/2} \sum_{n \in T_m} w_n^{(m)} + e^{\alpha_m/2} \sum_{n \in F_m} w_n^{(m)}$$

$$= (e^{\alpha_m/2} - e^{-\alpha_m/2}) \sum_{n=1}^N w_n^{(m)} I(h_m(x_n) \neq t_n) + e^{-\alpha_m/2} \sum_{n=1}^N w_n^{(m)}$$

$\underbrace{e^{\alpha_m/2} - e^{-\alpha_m/2}}_{\text{const.}}$ $\frac{\partial E}{\partial h_m(x_n)} = 0$ $\underbrace{e^{-\alpha_m/2} \sum_{n=1}^N w_n^{(m)}}_{\text{const}}$

True: $t_n h_m(x_n) = +1$

False: $t_n h_m(x_n) = -1$

$$\Rightarrow \text{equivalent to : } J_m = \sum_{n=1}^N w_n^{(m)} I(h_m(x) \neq t_n)$$

$$\frac{\partial E}{\partial \alpha_m} = 0 \Rightarrow \left(\frac{1}{2} e^{\alpha_m/2} + \frac{1}{2} e^{-\alpha_m/2} \right) \sum_{n=1}^N w_n^{(m)} I(h_m(x_n) \neq t_n) = \frac{1}{2} e^{-\alpha_m/2} \sum_{n=1}^N w_n^{(m)}$$

$$E_m := \frac{\sum_{n=1}^N w_n^{(m)} I(h_m(x) \neq t_n)}{\sum_{n=1}^N w_n^{(m)}} = \frac{e^{-\alpha_m/2}}{e^{\alpha_m/2} + e^{-\alpha_m/2}} = \frac{1}{e^{\alpha_m} + 1}$$

$$\Rightarrow \alpha_m = \ln \left\{ \frac{1 - E_m}{E_m} \right\}$$

$$w_n^{(m+1)} = w_n^{(m)} \cdot \exp \left\{ -\frac{1}{2} t_n \alpha_m h_m(x_n) \right\} = w_n^{(m)} \exp \left\{ \alpha_m I(h_m(x_n) \neq t_n) \right\}$$

Perception Learning - Learn ϕ

Linear Output: $y(x) = w^T x + w_0$

Logistic Output: $y(x) = \sigma(w^T x + w_0)$

~ Generalised Multiclass (non-linear):

$$y_k(x) = \sum_{i=0}^d w_{ki} \phi(x_i) \quad y_k(x) = \sigma \left(\sum_{i=0}^d w_{ki} \phi(x_i) \right)$$

\uparrow activation \uparrow basis

- > If the output unit is correct, leave the weights alone.
- > If the output unit incorrectly outputs a zero, add the input vector to the weight vector.
- > If the output unit incorrectly outputs a one, subtract the input vector from the weight vector.

$$\Rightarrow w_{kj}^{(T+1)} = w_{kj}^{(T)} - \gamma (y_k(x_n; w) - t_{kn}) \phi_j(x_n)$$

(Delta Rule! LMS Rule!)

Limitation: - Perceptrons with fixed, hand-coded input features can model any separable function perfectly, given right input features.
 - Exponential number of input features
 - Once features designed, strong limit on what it can learn

⇒ Learn features!

➤ L2 loss	⇒ Least-squares regression
$L(t, y(\mathbf{x})) = \sum_n (y(\mathbf{x}_n) - t_n)^2$	
➤ L1 loss:	⇒ Median regression
$L(t, y(\mathbf{x})) = \sum_n y(\mathbf{x}_n) - t_n $	
➤ Cross-entropy loss	⇒ Logistic regression
$L(t, y(\mathbf{x})) = -\sum_n \{t_n \ln y_n + (1 - t_n) \ln(1 - y_n)\}$	
➤ Hinge loss	⇒ SVM classification
$L(t, y(\mathbf{x})) = \sum_n [1 - t_n y(\mathbf{x}_n)]_+$	
➤ Softmax loss	⇒ Multi-class probabilistic classification
$L(t, y(\mathbf{x})) = -\sum_n \sum_k \left\{ \mathbb{I}(t_n = k) \ln \frac{\exp(y_k(\mathbf{x}))}{\sum_j \exp(y_j(\mathbf{x}))} \right\}$	

Learning with hidden Units using GD.

$$E(W) = \sum_n L(t_n, y(x_n; W)) + \lambda \lVert W \rVert^2$$

$$L(t_n, y(x_n; W)) = \sum_n (y(x_n; W) - t_n)^2 \quad \text{L}_2 \text{ Loss}$$

$$\lVert W \rVert^2 = \|W\|^2_F$$

L₂ Regn.

Update each weight $W_{ij}^{(k)}$ in the direction of $\frac{\partial E}{\partial w_{ij}^{(k)}}$ (weight decay)

Gradient

1. Naive Analytical Differentiation

Analytical computation + Chain Rule :

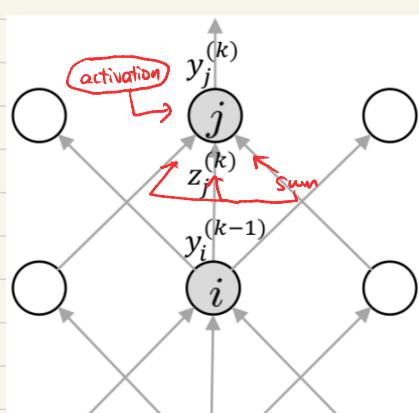
- with increasing depth, exponential paths! \Rightarrow Infeasible

2. Numerical Differentiation

Inefficient

3. Incremental Analytical Differentiation (Backpropagation)

- ① Convert discrepancy between each output & its target value into an error derivative
- ② Compute error derivatives in each hidden layer from error derivatives in the layer above.
- ③ Use error derivatives w.r.t. activities to get error derivatives w.r.t. the incoming weights.



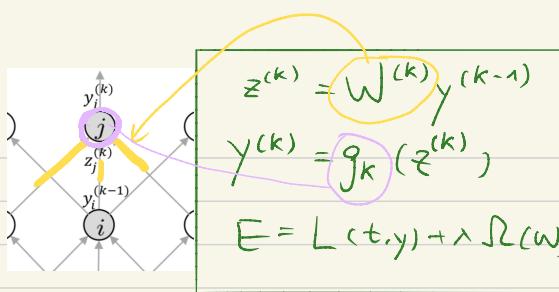
$$\frac{\partial E}{\partial z_j^{(k)}} = \frac{\partial E}{\partial y_j^{(k)}} \cdot \frac{\partial y_j^{(k)}}{\partial z_j^{(k)}} = \frac{\partial E}{\partial y_j^{(k)}} \cdot \frac{\partial g(z_j^{(k)})}{\partial z_j^{(k)}}$$

$$\frac{\partial E}{\partial y_i^{(k-1)}} = \sum_j \frac{\partial z_j^{(k)}}{\partial y_i^{(k-1)}} \cdot \frac{\partial E}{\partial z_j^{(k)}} = \sum_j w_{ji}^{(k-1)} \frac{\partial E}{\partial z_j^{(k)}}$$

$$\frac{\partial E}{\partial w_{ji}^{(k-1)}} = \frac{\partial z_j^{(k)}}{\partial w_{ji}^{(k-1)}} \cdot \frac{\partial E}{\partial z_j^{(k)}} = y_i^{(k-1)} \cdot \frac{\partial E}{\partial z_j^{(k)}}$$

Forward Pass

$$1. \frac{\partial E}{\partial y^{(l)}} = \frac{\partial}{\partial y^{(l)}} L(t, y) + \lambda \frac{\partial}{\partial y^{(l)}} \mathcal{R}$$



$$2. \frac{\partial E}{\partial z^{(l)}} = \frac{\partial E}{\partial y^{(l)}} \cdot \frac{\partial y^{(l)}}{\partial z^{(l)}} = \frac{\partial E}{\partial y^{(l)}} \cdot g'(z^{(l)})$$

$$3. \frac{\partial E}{\partial w^{(l)}} = \frac{\partial E}{\partial z^{(l)}} \cdot y^{(l-1)\top} + \lambda \cdot \frac{\partial \mathcal{R}}{\partial w^{(l)}}$$

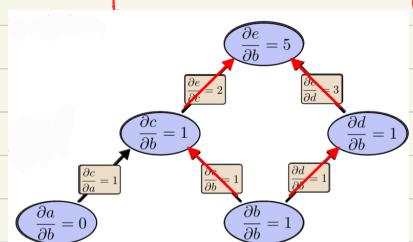
$$4. \frac{\partial E}{\partial y^{(l-1)}} = w^{(l)\top} \cdot \frac{\partial E}{\partial z^{(l)}}$$

Forward

Backward

4. Automatic Differentiation

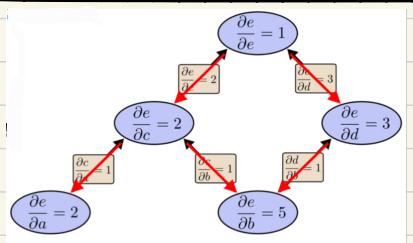
Computational Graph



Forward Mode : $\mathcal{O}(c \# \text{edges})$

Derivative of every node w.r.t. b

One Pass / Node.



Reverse Mode : $\mathcal{O}(\# \text{edges})$

Derivative of e w.r.t. every node

Single Pass : Speedup in $\mathcal{O}(\# \text{input})$

$$1. y = \text{module}.fprop(x)$$

- Practical issue

Exponentials get very big and can have vastly different magnitudes.

Trick 1: Do not compute first softmax, then log, but instead directly evaluate log-exp in the nominator and log-sum-exp in the denominator.

Trick 2: Softmax has the property that for a fixed vector b $\text{softmax}(a + b) = \text{softmax}(a)$

\Rightarrow Subtract the largest weight vector w_j from the others.

$$E(w) = - \sum_{n=1}^N \sum_{k=1}^K \left\{ \mathbb{I}_{(t_n=k)} \ln \frac{\exp(w_k^\top x)}{\sum_{j=1}^K \exp(w_j^\top x)} \right\}$$

$$2. \frac{\partial E}{\partial x} = \text{module}.bprop\left(\frac{\partial E}{\partial y}\right)$$

Gradient Descent

- ← Compute gradients for each weight
- Adjust weights in the direction of gradient

Batch

$$w_{kj}^{(t+1)} = w_{kj}^{(t)} - \gamma \frac{\partial E(w)}{\partial w_{kj}} \Big|_{w^{(t)}}$$

- Process the full dataset at once to compute gradients

- + Conditions of convergence are well understood.
- + Many acceleration techniques (conjugate gradients) only operate in batch learning
- + Theoretical analysis of weight dynamics & convergence rates are simpler

Error function should be normalized by Minibatch size.
s.t. keep same learning rate between minibatches

$$E(w) = \frac{1}{N} \sum_n L(x_n, y_n; w) + \frac{\lambda}{2} \|w\|^2$$

Stochastic

$$w_{kj}^{(t+1)} = w_{kj}^{(t)} - \gamma \frac{\partial E_n(w)}{\partial w_{kj}} \Big|_{w^{(t)}}$$

- choose a single example from the training set
- Compute gradient based on this example.
- Noisy, which has some advantages

- + Usually much faster than batch
- + often results in better solutions [?]
- + can be used for tracking changes

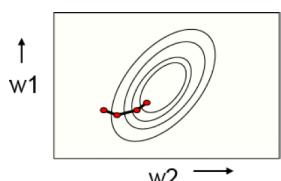
Minibatch

- * Process only a small batch of training examples together
- * Start with a small batch size and increase it as training proceeds

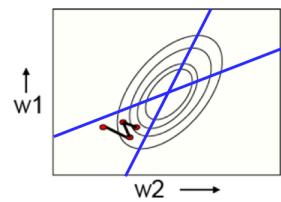
- + Gradients will be more stable than SGD
still faster than BGD
- + Take advantage of redundancies in the training set
- + Matrix operations are more efficient than vector operations

Learning Rate

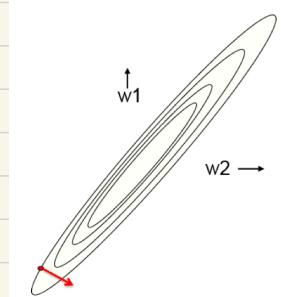
$$\eta_{\text{opt}} = \left(\frac{\partial^2 E(W^{(t)})}{\partial W^2} \right)^{-1}$$



Batch : updates perpendicular to contour lines



Stochastic : updates perpendicular to constraints
from training examples.



Inputs are correlated :

The ellipse very elongated :

The direction of steepest descent is almost perpendicular to the dir towards the minimum.

→ Momentum

$$\Delta W = V(t) = \alpha V(t-1) - \varepsilon \frac{\partial E}{\partial W}(t)$$

$$= \alpha \Delta W(t-1) - \varepsilon \frac{\partial E}{\partial W}(t)$$

Techniques : At the beginning of learning:

- use small $\alpha = 0.5$
- Once the large gradients have disappeared raise momentum smoothly to its final values.

$$\text{Plane surface error : } V(\infty) = \frac{1}{1-\alpha} \left(-\varepsilon \frac{\partial E}{\partial W} \right)$$

for $\alpha \rightarrow 1$, much faster than simple gradient descent.

Global learning rate , multiplied by a local gain per weight:

Gradients can get very small in the early layers of pixels

Fan-in (often varies widely between layers) of a unit determines the size of **Overshooting** when changing multi. weights simultaneously to correct the same error,

RMSProp

- Motivation:
- The magnitude of gradient can be very different for different weights & can change during learning.
 - Hard to choose a single global learning rate
 - For batch, use the sign of the gradient ?

Idea: Divide the gradient by a running average of its recent magnitude

$$\text{MeanSq}(w_{ij}, t) = 0.9 \text{ MeanSq}(w_{ij}, t-1) + 0.1 \left(\frac{\partial E}{\partial w_{ij}}(t) \right)^2$$

Divide the gradient by $\sqrt{\text{MeanSq}(w_{ij}, t)}$

AdaGrad / AdaDelta / Adam: less sensitive to parameter setting
quasi-standard.

Tricks of the Trade

- * Shuffling
- * Augmentation : reduce overfitting

- Cropping
- Zooming
- Flipping
- Colour PCA

- * Normalisation

MLP only

- Motivation

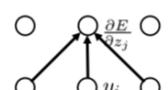
- Consider the Gradient Descent update steps

$$w_{kj}^{(\tau+1)} = w_{kj}^{(\tau)} - \eta \frac{\partial E(\mathbf{w})}{\partial w_{kj}} \Big|_{\mathbf{w}^{(\tau)}}$$



- From backpropagation, we know that

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial z_j}{\partial w_{ij}} \frac{\partial E}{\partial z_j} = y_i \frac{\partial E}{\partial z_j}$$

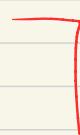


- When all of the components of the input vector y_i are positive, all of the updates of weights that feed into a node will be of the same sign.
⇒ Weights can only all increase or decrease together.
- Slow convergence

- Normalise all inputs that an input unit sees to zero-mean, unit covariance
- if possible, decorrelate them using PCA

Fast convergence:

- zero-mean
- same covariance
- uncorrelated



Nonlinearities

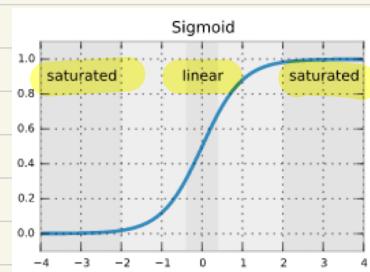
- Symmetric sigmoids: such as \tanh often converge faster than the standard logistic sigmoid: $f = 1.7159 \tanh(\frac{2}{3}x)$
- ⇒ When used with transformed inputs, the variance of the output will be close to 1.

Output nodes

- Sigmoid for nice probabilistic interpretation (range [0,1])
- \tanh for regression tasks

Internal nodes

- \tanh (historically)
 - ↳ better than sigmoid due to symmetry.
- ReLU =

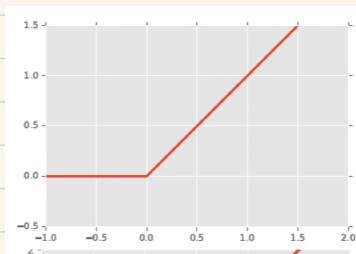


All parameters too small:

- Variance of activations will drop in each layer
- Sigmoids are approx. linear close to 0
- Good for passing gradients through.
 - ↳ But Gradient loss of the nonlinearity.
 - ↳ No benefit of having multiple layers

All parameters become larger:

- They will saturate and gradient $\rightarrow 0$



ReLU

de-facto
standard

$$g(a) = \max\{0, a\}, \quad \frac{\partial g(a)}{\partial a} = \begin{cases} 1, & a > 0 \\ 0, & \text{else} \end{cases}$$

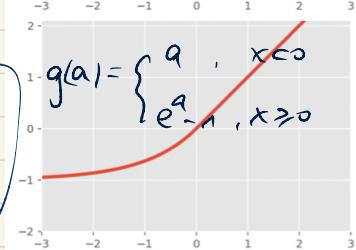
- + Much easier to propagate gradients through deep networks
- + No need store ReLU Output separately.

avoid
stuck at 0
weaker
offset bias

No offset bias
Need store
activations

Leaky

ReLU



ELU

ReLU

- A certain fraction of units stuck at zero
- + if initial weights are chosen s.t. ReLU output is 0 for the entire training set, the unit will never pass through a gradient to change those weights

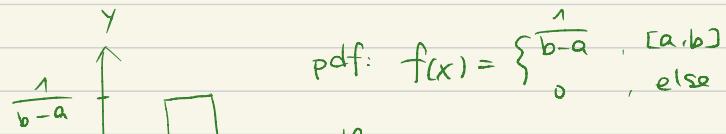
- ReLU has an offset bias, since its outputs will always be positive!

Initialisation of Weights

- Assuming that the training set has been normalised
 $f(x) = \text{ReLU}(\tanh(\frac{2}{3}x))$ is used

- The initial weights should be randomly drawn from distribution e.g. uniform or normal with mean zero & variance $\sigma_w^2 = \frac{1}{n_{in}}$

fan-in (# connections into ↑
the node)



cdf: $F(x) = \begin{cases} 0, & x < a \\ \frac{x-a}{b-a}, & a \leq x < b \\ 1, & x \geq b \end{cases}$

$E(X) = \int x f(x) dx$

$$E(X) = \int_a^b x \frac{1}{b-a} dx = \frac{1}{b-a} \cdot \frac{1}{2} x^2 \Big|_a^b = \frac{1}{2} (b-a)$$

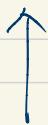
$$E(X^2) = \int_a^b x^2 \frac{1}{b-a} dx = \frac{1}{b-a} \cdot \frac{1}{3} x^3 \Big|_a^b = \frac{1}{3} (b-a)^2$$

$$\text{Var}(X) = \frac{1}{3} (b-a)^2 - \left(\frac{1}{2} (b-a) \right)^2 = \left(\frac{1}{3} - \frac{1}{4} \right) (b-a)^2 = \frac{1}{12} (b-a)^2$$

$$\text{SD}(X) = \sqrt{\frac{1}{12} (b-a)^2}$$

$$\frac{1}{12} (b-a)^2 = \frac{1}{n_{in}} \Rightarrow W \sim U \left[-\frac{\sqrt{3}}{\sqrt{n_{in}}}, \frac{\sqrt{3}}{\sqrt{n_{in}}} \right]$$

$$\text{or wrt. Fan-in \& Fan-out: } W \sim U \left[-\frac{\sqrt{6}}{\sqrt{n_{in+n_{out}}}}, \frac{\sqrt{6}}{\sqrt{n_{in+n_{out}}}} \right]$$



Based on tanh units: linearity assumption around zero.

He et al.: for ReLU: $\text{Var}(w) = \frac{2}{n_{in}}$

Batch Normalisation

Motivation: Optim. works best if all inputs of a layer are normed.

Idea: perform transformations on all activations

& undo them when backproping gradients.

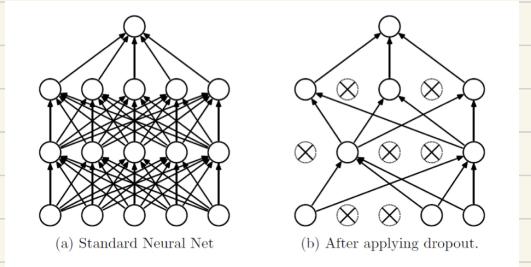
Complication: centering + normalisation also needs to be done at test time, but minibatch no longer available

- Learning normalisation parameters to compensate for the expected bias of the previous layer
(moving average)

Effect:

- Much improved convergence (but para important)
- widely used

Dropout



Randomly switch off units during training (sort of Regularisation)

- change network architecture of each minibatch
effectively training many different variants of the network.

[- When applying the trained network, multiply activations with the probability that the unit was set to zero during training.

⇒ Greatly improved performance.

CNN

• Hierarchical Multi-Layer ?

- Visual scenes are hierarchically organized.
- Biological vision is hierarchical too.
- Shallow architectures are inefficient at representing complex functions

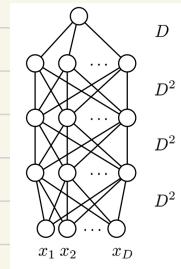
PARA
FCN

$$|\theta| = 3D^2 + D$$

for 32×32 image:

$$D = 32^2$$

$$\Rightarrow |\theta| = 3 \cdot 32^4 + 32^2 \approx 3 \cdot 10^6$$

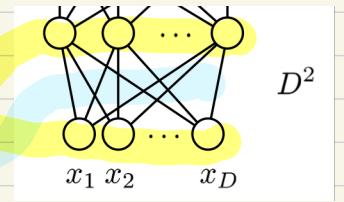


↪ Hard to train, Need to initialise carefully.

For 1000×1000 image:

1M Hidden units per layer

1T Para per layer

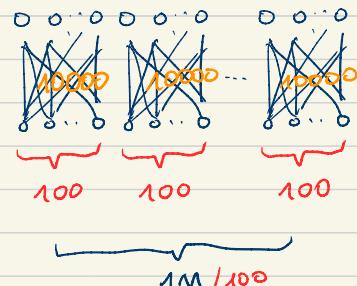


Local FCN

10x10 Receptive fields

100M Para

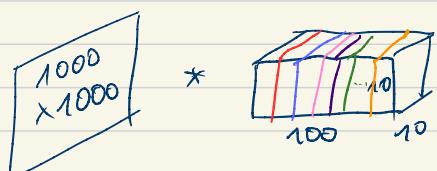
$$= \frac{1M}{100} \cdot 100^2$$



ConvN

100 Filters 10×10

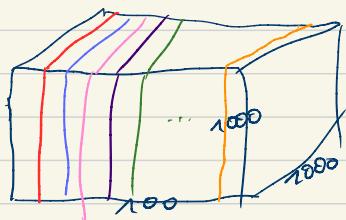
$$100 \times 10 \times 10 = 10K \text{ Para}$$



Response map:

$1000 \times 1000 \times 100$

Only memory, not para.



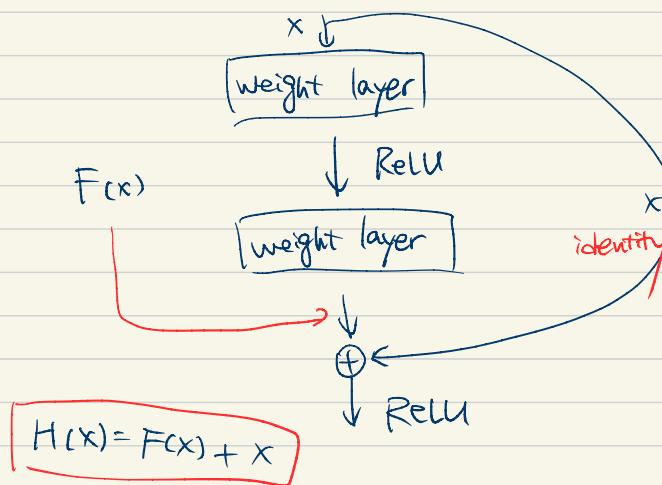
Depth , Parameters , Stride , Max Pooling ...

Architecture

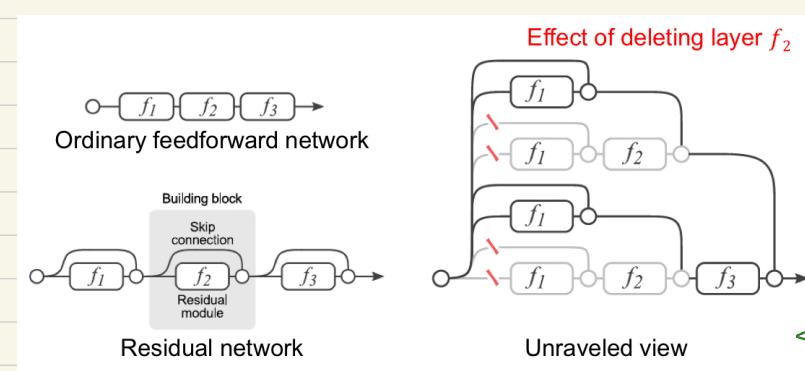
Le Net	Alex Net	VGG Net	Google Net	Residual
	$1 \times 1 \times 1$, stride 4	3×3 stride 1		

- stack 3×3 on top of another $3 \times 3 \Rightarrow 5 \times 5$ receptive field
- 3×3 three $\Rightarrow 7 \times 7$
- But much fewer parameters $3 \cdot 3^2 = 27$, $7^2 = 49$
- In addition: non-linearity in-between 3×3 layers for additional discriminativity.

Res Net



- + if identity optimal: easy to set weights as 0
- + if optimal mapping closer to identity, easy to find small fluctuations
- + direct path for the gradient to flow to the previous stages



Summary ResNet :

- + The effective path in ResNets are relatively shallow. (5-17)

- + Resilience to deletion :

← Deleting any single layer only affects a subset of paths, & the shorter ones less than the longer

- New Interpretation :
- ResNets work by creating an ensemble of relatively shallow paths
 - Deeper \rightarrow longer ensemble

- excluding longer paths doesn't negatively affect results

Word Embedding

$$h = W^T \cdot x$$

embeddings ↗
 LUT ↑
 one-hot vec ↘
 $[dx1]$ $[Vxd]$ $[Vx1] \sim nm$
 ~ $(50, 300)$

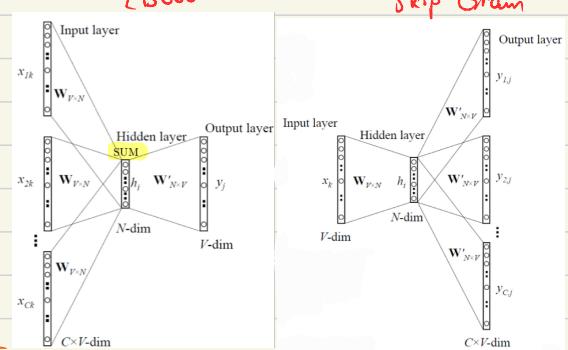
~ Full Connect.
↳ HuGu Para.

word2vec CBoW Skip-gram → fewer parameters

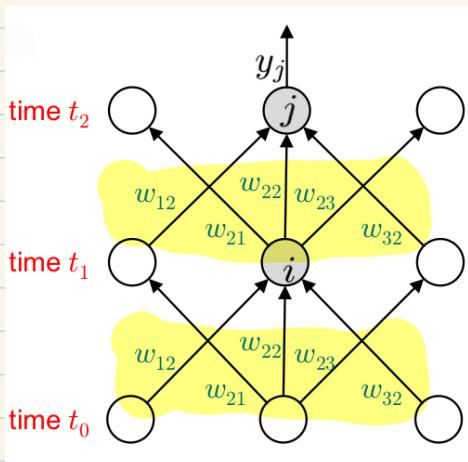
predict current word.
predict words within a certain range

→ Embedding preserves linear regularity

↳ Hierarchical Softmax



RNN

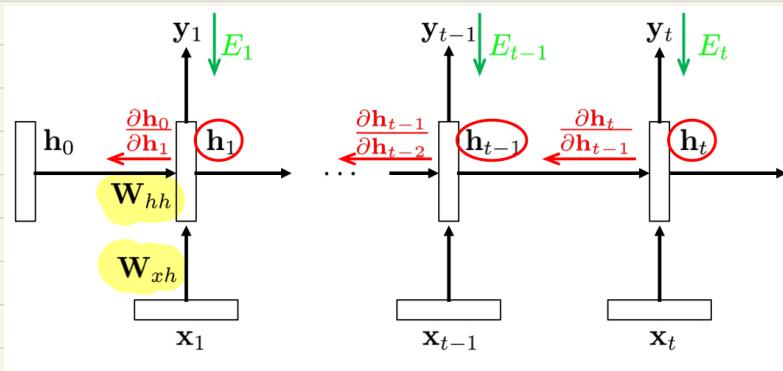


RNN : just a feedforward net
that keeps reusing the same weight

Backprop : $w_1 = w_2 \leftarrow \nabla w_1 = \nabla w_2$

$$\frac{\partial E}{\partial w_1} + \frac{\partial E}{\partial w_2} \quad \text{for both } w_1 \text{ & } w_2.$$

BPTT Backpropagation Through Time



Inputs: x_t

Outputs: y_t

Hidden Units: h_t

Initial State: h_0

Connection:

$$\begin{cases} W_{xh} \\ W_{yh} \\ W_{hh} \end{cases}$$

$$\left\{ \begin{array}{l} h_t = \sigma(W_{xh}x_t + W_{hh}h_{t-1} + b) \\ \hat{y}_t = \text{softmax}(W_{yh}h_t) \end{array} \right.$$

$$E = \sum_t E_t : \text{error sum over time.}$$

For some weight: $\frac{\partial E_t}{\partial w_{ij}} = \frac{\partial E_t}{\partial h_t} \frac{\partial h_t}{\partial w_{ij}} + \frac{\partial E_t}{\partial h_t} \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial w_{ij}} + \dots$

$$= \sum_{1 \leq k \leq t} \left(\frac{\partial E_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial w_{ij}} \right)$$

immediate
p's: h_{t-1}
const.

$$= \prod_{i>k} \frac{\partial h_i}{\partial h_{i-1}} = \prod_{i>k} W_{hh}^T \text{diag}(S'(h_{i-1}))$$

Exploding/Vanishing Gradient:

For W_{hh} : largest eigenvalue $\begin{cases} > 1 & \text{exploding} \\ < 1 & \text{vanishing} \end{cases}$

$$= (W_{hh}^T)^L, \text{ for } t \rightarrow \infty, L=t-k.$$

Gradient Clipping: if $\|\hat{g}\| \geq \text{threshold}$: $\hat{g} \leftarrow \frac{\text{threshold}}{\|\hat{g}\|} \cdot \hat{g}$

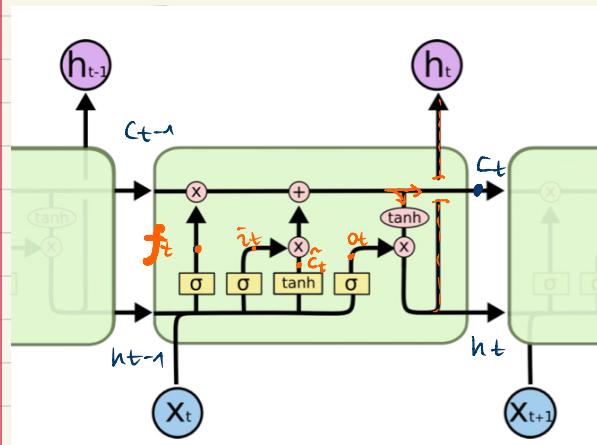
- Glorot / He

- ReLU

- LSTM, GRU

BPTT : Dependency between long distance :

↪ LSTM



C : Cell State

h : hidden Units

σ : sigmoid $[0, 1]$

tanh : $[-1, 1]$

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \quad | \text{forget}$$

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \quad | \text{update}$$

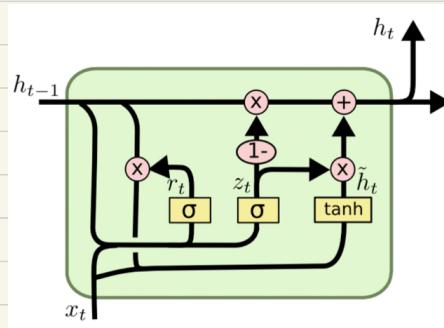
$$\tilde{c}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \quad | \text{update}$$

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{c}_t$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \quad | \text{output}$$

$$h_t = \tanh(C_t) \cdot o_t$$

Gated Recurrent Units



$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t]) \quad | \text{update}$$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t]) \quad | \text{reset}$$

$$\tilde{h}_t = \tanh(W_t \cdot [r_t * h_{t-1}, x_t])$$

$$h_t = h_{t-1} \cdot (1 - z_t) + z_t \cdot \tilde{h}_t$$

Repetition

1. Bayes Decision Theory

- Prior, Likelihood, Posterior = $\frac{\text{Likelihood} \times \text{Prior}}{\text{Normalised Factor}}$
- | | | |
|--------|-------------------|---------------------------------|
| $p(a)$ | $p(x a)$ | $p(a x)$ |
| $p(b)$ | $p(x b)$ | $p(b x)$ |
| Result | Given results | Given feature architecture data |
| | Predict feature | Predict results / labels. |
| | Architecture data | |

- Optimal decision Rule \rightarrow Likelihood-Ratio-Test

Decide for C_1 , if $p(C_1|x) > p(C_2|x)$

$$\Leftrightarrow p(x|C_1)p(C_1) > p(x|C_2)p(C_2) \Leftrightarrow \frac{p(x|C_1)}{p(x|C_2)} > \frac{p(C_2)}{p(C_1)}$$

- Minimising Expected Loss

$$E(L) = \sum_k \sum_j \int_{R_j} L_{kj} p(x, C_k) dx$$

Choosing Regions R_j such that $E(L) = \sum_k L_{kj} \underbrace{p(C_k|x)}_{\substack{\uparrow \\ \text{Expected Loss}}} \underbrace{\sum_j}_{\substack{\uparrow \\ \text{Loss}}} \underbrace{R_j}_{\substack{\curvearrowleft \\ \text{Posterior}}}$

2. Probability Density Estimation

Gaussian: $N(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$

$$N(x|\mu, \Sigma) = \frac{1}{(2\pi)^{\frac{D}{2}} |\Sigma|^{\frac{1}{2}}} \exp \left\{ -\frac{1}{2} (x-\mu)^T \Sigma^{-1} (x-\mu) \right\}$$

- Maximum Likelihood Approach.

$$L(\theta) = p(X|\theta) = \prod_{n=1}^N p(x_n|\theta)$$

Log-Likelihood

$$E(\theta) = -\ln L(\theta) = -\sum_{n=1}^N \ln p(x_n|\theta)$$

$$\frac{\partial}{\partial \theta} E(\theta) = - \sum_{n=1}^N \frac{\frac{\partial}{\partial \theta} p(x_n|\theta)}{p(x_n|\theta)} = 0$$

Non-parametric Solutions: Histogram: $p_i = \frac{n_i}{N \Delta x}$

kernel Density Estimation: $p(x) \approx \frac{K}{NV}$

fixed V
determine K

fixed K , determine V

Kernel

(window, Gaussian...)

3. Mixture Models (MoG)

$$p(j) = \pi_j \cdot p(x|\theta_j) \quad p(x|D) = \sum_{j=1}^M p(x|\theta_j) \underbrace{p(j)}_{\text{weights}}$$

→ Iterative Strategy (Prior knowledge missing)

K-Means Clustering: - Local, - results initialisation dependent

1. Initialisation: pick K arbitrary centroids: cluster means
2. Assign each sample to the closest centroid.
3. Adjust the centroids to be the means of the samples assigned to them
4. Go to 2 until no change.

↪ EM (Expectation-Maximisation) - Local

E

Softly assign samples to mixture components.

$$\gamma_j(x_n) \leftarrow \frac{\pi_j N(x_n | \mu_j, \Sigma_j)}{\sum_{k=1}^K \pi_k N(x_n | \mu_k, \Sigma_k)} \quad \forall j=1, \dots, K, n=1, \dots, N$$

M

Re-estimate the parameters (separately for each mixture component) based on the soft assignments

$$\hat{N}_j \leftarrow \sum_{n=1}^N \gamma_j(x_n) = \text{soft number of samples labeled } j$$

$$\hat{\pi}_j^{\text{new}} \leftarrow \frac{\hat{N}_j}{N}$$

$$\hat{\mu}_j^{\text{new}} \leftarrow \frac{1}{\hat{N}_j} \sum_{n=1}^N y_j(x_n) x_n$$

$$\hat{\Sigma}_j^{\text{new}} \leftarrow \frac{1}{\hat{N}_j} \sum_{n=1}^N y_j(x_n) (x_n - \hat{\mu}_j^{\text{new}}) (x_n - \hat{\mu}_j^{\text{new}})^T$$

4. Linear Discriminant Functions (Decision !)

$$y(x) = w^T x + w_0$$

Least-Squares (sum-of-Squares Error: Closed form!)

$$E(w) = \sum_{n=1}^N (y(x_n) - t_n)^2$$

$$\Rightarrow E(\tilde{w}) = \frac{1}{2} \text{Tr} \{ (\tilde{X}\tilde{w} - T)^T (\tilde{X}\tilde{w} - T) \} \Rightarrow \tilde{w} = (\tilde{X}^T \tilde{X})^{-1} \tilde{X}^T T = \tilde{X}^+ T$$

$$\Rightarrow y(x) = \tilde{w}^T \tilde{x} = T^T (\tilde{X}^+)^T \tilde{x}$$

→ sensitive to outliers

↳ Generalized Linear Models. $y(x) = g(w^T x + w_0)$

If g is monotonous, the resulting decision boundaries are still linear function of x

↓ activation function (nonlinear)

↳ With non-linear Basis Functions

$$y(x) = \sum_{j=1}^M w_j \phi_j(x) + w_0$$

+ Nonlinear decision boundaries

+ choosing right ϕ_j , every continuous func. can be approx.

- No closed form \rightarrow GD

Probabilistic Discriminative Models (logistic regression)

- + Probabilistic interpretation but discriminative method:
only focus on decision hp
- + Advantageous for high-dim

$$p(t|w) = \prod_{n=1}^N y_n^{t_n} \{1 - y_n\}^{1-t_n} \quad \text{with } y_n = P(C_1 | \phi_n)$$

$$E(w) = -\ln p(t|w) = -\sum_{n=1}^N \{t_n (\ln y_n + (1-t_n) \ln (1-y_n)\}$$

cross-entropy error

$$= -\sum_{n=1}^N \sum_{k=0}^2 \{ \mathbb{I}(t_n = k) \ln P(y_n = k | x_n; w) \}$$

multi class

Softmax

$$\frac{\exp(a_k)}{\sum_i \exp(a_i)}$$

$$E(w) = -\sum_{n=1}^N \sum_{k=1}^K \left\{ \mathbb{I}(t_n = k) \ln \frac{\exp(u_k^\top x)}{\sum_{j=1}^K \exp(u_j^\top x)} \right\}$$

$$\nabla_{w_k} E(w) = -\sum_{n=1}^N [\mathbb{I}(t_n = k) \ln P(y_n = k | x_n; w)]$$

5. SVM

$$\text{Primal: } L_p = \frac{1}{2} \|w\|^2 - \sum_{n=1}^N a_n \{ t_n y(x_n) - 1 \}$$

$$\text{KKT: } a_n \geq 0$$

$$t_n y(x_n) - 1 \geq 0$$

$$a_n \{ t_n y(x_n) - 1 \} = 0$$

$$\boxed{\begin{array}{l} \lambda \geq 0 \\ f(x) \geq 0 \\ \lambda f(x) = 0 \end{array}}$$

$\forall a_n > 0 : t_n y(x_n) - 1 \geq 0 \Rightarrow$ Supporting Vectors

$$\text{Dual: } L_d(a) = \sum_{n=1}^N a_n - \frac{1}{2} \sum_{n=1}^N \sum_{m=1}^N a_n a_m t_n t_m (x_m^\top x_n)$$

$$a_n \geq 0 \quad \forall n$$

$$\sum_{n=1}^N a_n t_n = 0$$

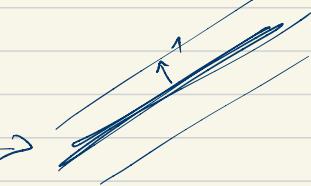
Comparison: L_d only depends on a_n

$$L_p : \mathcal{O}(D^3) \text{ vs. } L_d : \mathcal{O}(N^3) \rightarrow \mathcal{O}(N) - \mathcal{O}(N^2)$$

Slack Variable for Non-Separable

$$\xi_n = \begin{cases} 0, & \text{correct} \\ |t_n - y(x_n)|, & \text{misclassified} \end{cases}$$

Points on decision boundary: $\xi_n = 1$



$$\Rightarrow L_d(a) = \sum_{n=1}^N a_n - \frac{1}{2} \sum_{n=1}^N \sum_{m=1}^N a_n a_m t_m (x_m^T x_n)$$

$$\left\{ \begin{array}{l} 0 \leq a_n \leq C \\ \sum_{n=1}^N a_n = 0 \end{array} \right.$$

Kernel Trick: $y(x) = w^T \phi(x) + b = \sum_{n=1}^N a_n t_n \phi(x_n)^T \phi(x) + b$

$$K(x, y) = \phi(x)^T \phi(y)$$

$$L_d(a) = \sum_{n=1}^N a_n - \frac{1}{2} \sum_{n=1}^N \sum_{m=1}^N a_n a_m t_m K(x_m, x_n)$$

$$\left\{ \begin{array}{l} 0 \leq a_n \leq C \\ \sum_{n=1}^N a_n = 0 \end{array} \right.$$