Homework #1, EE556, Fall 2018

Due: 9/7/18; hand in #3,#4

Problem #1

Consider the case of 2 multivariate Gaussian classes with distinct means, equal class prior probabilities, and common covariance matrix $\Sigma = \sigma^2 I$. Derive the equation that specifies the locus of points representing the decision boundary between the two classes (the result for this case was given in lecture).

Problem #2

For the distributions given in problem 1, derive an expression for the probability of error, leaving the final result in a simple integral form. *Hint:* as mentioned in lecture, the decision rule amounts to applying a threshold to a scalar random variable Y that is a linear combination of Gaussian random variables. Hence, the probability of error expression can be written in a simple form, once the mean and variance of Y, conditioned on $c = 1, 2$ is known. There is a fair amount of calculation involved in this problem.

Problem #3

Consider the case of 2 multivariate Gaussian classes with distinct means, equal class prior probabilities, and common covariance matrix Σ that is *not* necessarily a diagonal matrix. Derive the equation that specifies the locus of points representing the decision boundary between the two classes (the result for this case was given in lecture). For $d = 2$, make a particular choice of Σ and accurately sketch the decision boundary in the plane.

Problem #4

Three categories with equal prior probabilities must be distinguished by observing a two-dimensional feature vector. The class-conditional pdfs are each Gaussian with uncorrelated components. The class 1 feature vector has mean (2, 0) and unit variances. The class 3 feature vector has mean (0, 2) and unit variances. The class 2 feature vector has mean (0,0) and variances (1, 2). Find the equations describing the decision boundaries between the classes and (roughly) sketch the decision regions in the plane.

HW #1 Solns., EE556, Fa

#1) $g(\underline{x}) = \ln\left(\frac{f(\underline{x}/w_1)}{f(\underline{x}/w_2)}\right) = 0 \Rightarrow$

$$-\frac{1}{2\sigma^2}\|\underline{x} - \underline{\mu}_1\|^2 + \frac{1}{2\sigma^2}\|\underline{x} - \underline{\mu}_2\|^2 = 0$$

or

$$\|\underline{x} - \underline{\mu}_1\|^2 = \|\underline{x} - \underline{\mu}_2\|^2$$

or

$$-2\underline{x}^T\underline{\mu}_1 + \|\underline{\mu}_1\|^2 + 2\underline{x}^T\underline{\mu}_2 - \|\underline{\mu}_2\|^2 = 0$$

or

$$\underline{x}^T(\underline{\mu}_2 - \underline{\mu}_1) + \underbrace{\frac{\|\underline{\mu}_1\|^2 - \|\underline{\mu}_2\|^2}{2}}_{-\frac{(\underline{\mu}_2 + \underline{\mu}_1)^T(\underline{\mu}_2 - \underline{\mu}_1)}{2}} = 0$$

or

$$\left(\underline{x} - \left(\frac{\underline{\mu}_1 + \underline{\mu}_2}{2}\right)\right)^T(\underline{\mu}_2 - \underline{\mu}_1) = 0$$

#2) $y = \left(\underline{x} - \frac{(\underline{\mu}_1 + \underline{\mu}_2)}{2}\right)^T (\underline{\mu}_2 - \underline{\mu}_1)$

$y \geq 0 \Rightarrow$ "decide w_2"

else "decide w_1"

$$y = x^T(\underline{\mu}_2 - \underline{\mu}_1) - \left(\frac{\|\underline{\mu}_2\|^2 - \|\underline{\mu}_1\|^2}{2}\right)$$

Q: How is y distributed ?

A: consider y given $\underline{x}$ is from class w_2 :

$$f_{y/w_2}(y/w_2) \sim N(\mu_{y/2}, \sigma^2_{y/2})$$

$$\mu_{y/2} = E[y/w_2] = E[\underline{x}^T(\underline{\mu}_2 - \underline{\mu}_1)/w_2] - \frac{(\|\underline{\mu}_2\|^2 - \|\underline{\mu}_1\|^2)}{2}$$

$$= \underline{\mu}_2^T(\underline{\mu}_2 - \underline{\mu}_1) - \frac{(\underline{\mu}_2 + \underline{\mu}_1)^T(\underline{\mu}_2 - \underline{\mu}_1)}{2}$$

$$= \frac{1}{2}(\underline{\mu}_2 - \underline{\mu}_1)^T(\underline{\mu}_2 - \underline{\mu}_1) = \frac{1}{2}\|\underline{\mu}_2 - \underline{\mu}_1\|^2$$

Also, $E[(y - \mu_{y/2})^2/w_2] =$

$$E\left[\left(\underline{x}^T(\underline{\mu}_2 - \underline{\mu}_1) - (\underline{\mu}_2 - \underline{\mu}_1)^T\frac{(\underline{\mu}_1 + \underline{\mu}_2)}{2} - \underbrace{\frac{(\underline{\mu}_2 - \underline{\mu}_1)^T(\underline{\mu}_2 - \underline{\mu}_1)}{2}}_{\mu_{y/2}}\right)^2/w_2\right]$$

$$= E\left[\left(\underline{x}^T(\underline{\mu}_2 - \underline{\mu}_1) - \frac{(\underline{\mu}_2 - \underline{\mu}_1)^T}{2}(2\underline{\mu}_2)\right)^2/w_2\right]$$

$$= E\left[\left((\underline{\mu}_2 - \underline{\mu}_1)^T(\underline{x} - \underline{\mu}_2)\right)^2/w_2\right]$$

$$= (\underline{\mu}_2 - \underline{\mu}_1)^T E\left[(\underline{x} - \underline{\mu}_2)(\underline{x} - \underline{\mu}_2)^T/w_2\right](\underline{\mu}_2 - \underline{\mu}_1)$$

$$= \|\underline{\mu}_2 - \underline{\mu}_1\|^2 \cdot \sigma^2$$

So:

$$\text{Prob}[\text{error}] = \frac{1}{2} \cdot \frac{1}{\sqrt{2\pi}\,\sigma_{Y/2}} \int_{-\infty}^{0} e^{-\frac{(Y-\mu_{Y/2})^2}{2\sigma_{Y/2}^2}}\, dy$$

$$+$$

$$\frac{1}{2} \cdot \frac{1}{\sqrt{2\pi}\,\sigma_{Y/1}} \int_{0}^{\infty} e^{-\frac{(Y-\mu_{Y/1})^2}{2\sigma_{Y/1}^2}}\, dy$$

by symmetry

$$= \frac{1}{\sqrt{2\pi}\,\sigma_{Y/2}} \int_{-\infty}^{0} e^{-(Y-\mu_{Y/2})^2/2\sigma_{Y/2}^2}\, dy$$

Let $r = \dfrac{Y - \mu_{Y/2}}{\sigma_{Y/2}} \Rightarrow \sigma_{Y/2}\, dr = dy$

Can then rewrite:

$$\text{Prob}[\text{error}] = \frac{1}{\sqrt{2\pi}\,\sigma_{Y/2}} \int_{-\infty}^{-\mu_{Y/2}/\sigma_{Y/2}} e^{-\frac{r^2}{2}}\, \sigma_{Y/2}\, dr$$

$$= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\|\underline{\mu}_2 - \underline{\mu}_1\|/2\sigma} e^{-r^2/2}\, dr$$

$$= \underbrace{\frac{1}{\sqrt{2\pi}} \int_{\|\underline{\mu}_2 - \underline{\mu}_1\|/2\sigma}^{\infty} e^{-r^2/2}\, dr}_{\text{can calculate via standard tables for normal c.d.f.}}$$

#3) $g_i(\underline{x}) = \ln\left(f(\underline{x}/w_i)\,P[w_i]\right)$

$$= -\frac{1}{2}(\underline{x}-\underline{\mu}_i)^T \Sigma^{-1}(\underline{x}-\underline{\mu}_i) + \left\{\begin{array}{l}\text{nonessential} \\ \text{terms (in} \\ \text{this case)}\end{array}\right\}$$

$g_1(\underline{x}) - g_2(\underline{x}) =$

$$-\frac{1}{2}\underline{x}^T\Sigma^{-1}\underline{x} + \frac{1}{2}\underline{\mu}_1^T\Sigma^{-1}\underline{x} + \frac{1}{2}\underline{x}^T\Sigma^{-1}\underline{\mu}_1 - \frac{1}{2}\underline{\mu}_1^T\Sigma^{-1}\underline{\mu}_1$$

$$+\frac{1}{2}\underline{x}^T\Sigma^{-1}\underline{x} - \frac{1}{2}\underline{\mu}_2^T\Sigma^{-1}\underline{x} - \frac{1}{2}\underline{x}^T\Sigma^{-1}\underline{\mu}_2 + \frac{1}{2}\underline{\mu}_2^T\Sigma^{-1}\underline{\mu}_2$$

cancel

same

$\underline{\mu}_1^T\Sigma^{-1}\underline{x} = \underline{x}^T\Sigma^{-1}\underline{\mu}_1$ -- why?

A: $(\underline{\mu}_1^T\Sigma^{-1}\underline{x})^T = (\Sigma^{-1}\underline{x})^T(\underline{\mu}_1^T)^T$

$$= \underline{x}^T\underbrace{(\Sigma^{-1})^T}_{\text{symmetric}}\underline{\mu}_1 = x^T\Sigma^{-1}\mu_1$$

$\Rightarrow$

$$g_1(\underline{x}) - g_2(\underline{x}) = \underline{x}^T\Sigma^{-1}\underline{\mu}_1 - \frac{1}{2}\underline{\mu}_1^T\Sigma^{-1}\underline{\mu}_1 - \underline{x}^T\Sigma^{-1}\underline{\mu}_2 + \frac{1}{2}\underline{\mu}_2^T\Sigma^{-1}\underline{\mu}_2$$

$$= \underbrace{\underline{x}^T(\Sigma^{-1}\underline{\mu}_1 - \Sigma^{-1}\underline{\mu}_2)}_{x^T\Sigma^{-1}(\underline{\mu}_1-\underline{\mu}_2)} \underbrace{- \frac{1}{2}\underline{\mu}_1^T\Sigma^{-1}\underline{\mu}_1 + \frac{1}{2}\underline{\mu}_2^T\Sigma^{-1}\underline{\mu}_2}_{=\ \frac{1}{2}(\underline{\mu}_2-\underline{\mu}_1)^T\Sigma^{-1}(\underline{\mu}_1+\underline{\mu}_2)}$$

$$= (\underline{\mu}_1-\underline{\mu}_2)^T\Sigma^{-1}\left(\underline{x} - \left(\frac{\underline{\mu}_1+\underline{\mu}_2}{2}\right)\right) = 0$$

#4)

$$P[x/w_1] \sim N\left(\begin{pmatrix}2\\0\end{pmatrix}, \begin{pmatrix}1&0\\0&1\end{pmatrix}\right)$$

$$P[x/w_3] \sim N\left(\begin{pmatrix}0\\2\end{pmatrix}, \begin{pmatrix}1&0\\0&1\end{pmatrix}\right)$$

$$P[x/w_2] \sim N\left(\begin{pmatrix}0\\0\end{pmatrix}, \begin{pmatrix}1&0\\0&2\end{pmatrix}\right)$$

$$P[w_1] = P[w_2] = P[w_3] = 1/3$$

Let's find boundaries between all pairs of classes:

Between classes 1 & 3, we know (from class) that the decision boundary is the line:

$$(\mu_1 - \mu_2)^T(\underline{x} - \underline{x}_0) = 0.$$

We can choose $\underline{x}_0 = \begin{pmatrix}0\\0\end{pmatrix}$ in this case $\Rightarrow$

$$B_{13} = \left\{ \underline{x} = \begin{pmatrix}x_1\\x_2\end{pmatrix} \text{ s.t. } (1\ \ -1)\begin{pmatrix}x_1\\x_2\end{pmatrix} = 0 \right\}$$

The boundary between classes 1 & 2 is given by:

$$-\frac{1}{2}(\underline{x} - \underline{\mu}_1)^T \Sigma^{-1}(\underline{x} - \mu_1) - \frac{1}{2}\log|\Sigma_1| =$$

$$-\frac{1}{2}(\underline{x} - \underline{\mu}_2)^T \Sigma^{-1}(\underline{x} - \mu_2) - \frac{1}{2}\log|\Sigma_2|$$

or

$$(x_1 - 2)^2 + x_2^2 + \frac{1}{2}\underbrace{\log 1}_{0} = x_1^2 + \frac{1}{2}x_2^2 + \log 2$$

or

$$\frac{1}{2}x_2^2 - 4x_1 + 4 - \log 2 = 0$$

$$B_{12} = \left\{ \underline{x} = \begin{pmatrix}x_1\\x_2\end{pmatrix} \text{ s.t. } \frac{1}{8}x_2^2 = \left(x_1 - 1 + \frac{\log 2}{4}\right)\right.$$

The boundary between classes 2 and 3 is given by:

$$x_1^2 + (x_2-2)^2 = x_1^2 + \frac{1}{2}x_2^2 + \log 2$$

or

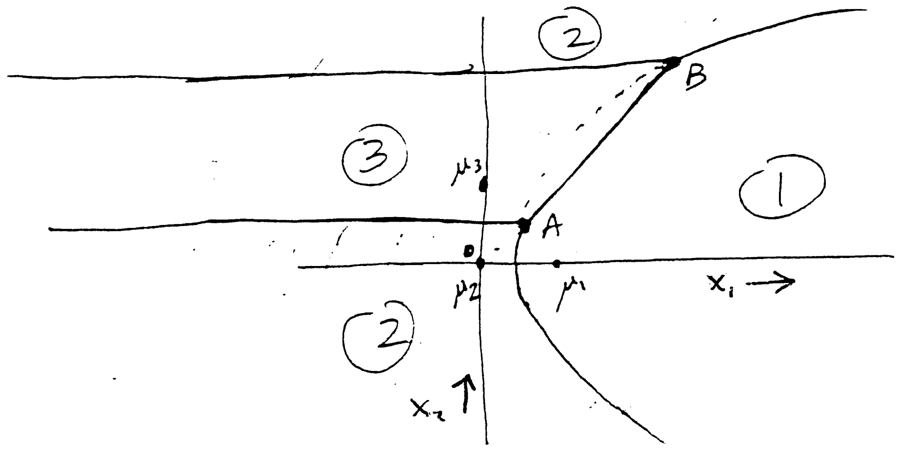
$$\frac{1}{8}x_2^2 = x_2 - 1 + \frac{\log 2}{4}$$

$$B_{23} = \{\underline{x} : x_2 = 0.96 \text{ or } x_2 = 7.04\}$$

<u>Rough sketch</u> :

$A = (.94, .94)$

$B = (7.06, 7.06)$



Homework #2, EE556, Fall 2018

Due: 9/13/18; Please hand in Problems 1,3, and 6.

Problem #1

Prove that the decision regions of a linear machine are convex by showing that if $\underline{x_1} \in R_i$ and $\underline{x_2} \in R_i$, then $\lambda \underline{x_1} + (1-\lambda)\underline{x_2} \in R_i$.

Problem #2

i) Using the method of Lagrange multipliers (for constrained optimization), show that the distance from a point $\underline{x_1}$ to the closest point on the hyperplane $\underline{w}^T \underline{x} + w_0 = 0$ is $|g(\underline{x_1})|/||\underline{w}||$.

ii) Determine an equation specifying the location of this nearest point on the hyperplane.

Problem #3

Consider the multiclass pairwise linear discriminant approach where discriminants separate one class from all others. Give an example in two dimensions where this approach fails to separate classes, even though all classes are pairwise linearly separable.

Problem #4

Consider a d-dimensional binary vector $\underline{x}$ with entries that are either 0 or 1. Suppose we assign $\underline{x}$ to class 1 if the number of ones in the vector is odd, and to class 2 otherwise (this is the d-bit parity problem).

a) Show that these classes are not linearly separable if $d > 1$.

b) Come up with a method that uses *multiple* linear discriminants and solves this problem. For each class, the largest discriminant function output, amongst the set of discriminants belonging to that class, is taken as the class's discriminant function value. The values from each of the two classes are then compared to select the winning class. This is called a "piecewise linear" discriminant function. Show that this approach can be used to solve the parity problem and specify all the discriminant functions and how they are used in the decisionmaking.

Problem #5

Give a 1-D example (depicted in 2-D weight space) involving 2 classes and 3 data points where the patterns are not linearly separable.

Problem #6

Consider the problem of maximum likelihood estimation for the mean vector, μ, and covariance matrix, Σ, of a multivariate Gaussian density. Using the gradient results given in class, derive the ML estimate for μ and Σ.

Problem #7

In class we derived the ML estimates for the mean and variance of a univariate Gaussian density. Verify that this solution is, indeed, a maximum of the likelihood function. To do this: i) compute the Hessian matrix of second order partial derivatives; 2) evaluate the Hessian at the ML solution; 3) show that the resulting matrix is negative definite.

HW#2 Solutions, EE556

1) Recall: a linear machine chooses

$$\hat{C}(\underline{x}) = \arg\max_j g_j(\underline{x}), \qquad (*)$$

where $g_j(\underline{x}) = \underline{w}_j^T \underline{x} + w_{j0}$.

Now, suppose two points $\underline{x}_0$ and $\underline{x}_1$ both get assigned, via $(*)$, to the same class, e.g. class i. A linear machine produces convex decision regions if $\lambda \underline{x}_0 + (1-\lambda)\underline{x}_1$ also gets assigned to class i, for all $0 \le \lambda \le 1$. (Note: the class, i, was arbitrarily chosen...).

Now, from the assumption, we have:

$$\max_j g_j(\underline{x}_0) = g_i(\underline{x}_0) = \underline{w}_i^T \underline{x}_0 + w_{i0}$$

$$\max_j g_j(\underline{x}_1) = g_i(\underline{x}_1) = \underline{w}_i^T \underline{x}_1 + w_{i0}$$

Next, consider

$$g_j(\lambda \underline{x}_0 + (1-\lambda)\underline{x}_1) = \underline{w}_j^T(\lambda \underline{x}_0 + (1-\lambda)\underline{x}_1) + w_{j0}$$
$$= \lambda(\underline{w}_j^T \underline{x}_0 + w_{j0}) + (1-\lambda)(\underline{w}_j^T \underline{x}_1 + w_{j0}).$$

Clearly,

$$\max_j g_j(\lambda \underline{x}_0 + (1-\lambda)\underline{x}_1) \le \lambda \max_j g_j(\underline{x}_0) + (1-\lambda)\max_j g_j(\underline{x}_1) \qquad (\Delta)$$

But

$$g_i(\lambda \underline{x}_0 + (1-\lambda)\underline{x}_1) = \lambda(\underline{w}_i^T \underline{x}_0 + w_{i0}) + (1-\lambda)(\underline{w}_i^T \underline{x}_1 + w_{i0})$$
$$= \lambda \max_j g_j(\underline{x}_0) + (1-\lambda)\max_j g_j(\underline{x}_1)$$

$\Rightarrow$ $\lambda \underline{x}_0 + (1-\lambda)\underline{x}_1$ is also assigned to class i by the linear machine.

$\therefore$, linear machines produce convex decision regions.

2) i) Let's solve $\min_{\underline{x}_q} \|\underline{x} - \underline{x}_q\|^2$ s.t. $g(\underline{x}_q) = 0$,

where $g(\underline{x}) = \underline{w}^T \underline{x} + w_0$.

We'll use the <u>method of Lagrange multipliers</u> from calculus:

We first form the Lagrangian cost function:

$$L = \|\underline{x} - \underline{x}_q\|^2 + \lambda(\underline{w}^T \underline{x}_q + w_0 - 0).$$

Then,

$$(1) \quad \nabla_{\underline{x}_q} L = 2(\underline{x}_q - \underline{x}) + \lambda \underline{w} = \underline{0}$$

and

$$(2) \quad \frac{\partial L}{\partial \lambda} = \underline{w}^T \underline{x}_q + w_0 = 0$$

@ an extremum of the Lagrangian, L.

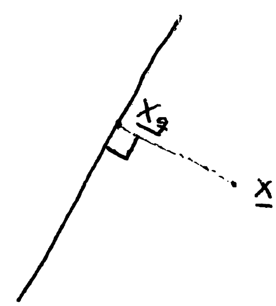
(1) $\Rightarrow$ $2\underline{x}_q = 2\underline{x} - \lambda \underline{w}$. Plug this into (2) $\Rightarrow$

$$\underline{w}^T\left(\frac{2\underline{x} - \lambda \underline{w}}{2}\right) + w_0 = 0 \Rightarrow$$

$$\underline{w}^T \underline{x} - \frac{\lambda}{2}\|\underline{w}\|^2 + w_0 = 0 \quad \text{or}$$

$$g(\underline{x}) - \frac{\lambda}{2}\|\underline{w}\|^2 = 0 \Rightarrow \lambda = \frac{2g(\underline{x})}{\|\underline{w}\|^2}$$

$$\text{or,} \quad \underline{x}_q = \underline{x} - \frac{g(\underline{x})\underline{w}}{\|\underline{w}\|^2} \Rightarrow$$



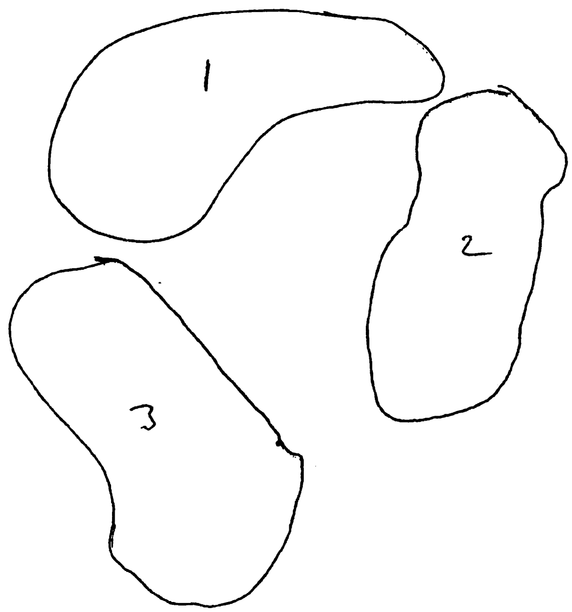
$$\|\underline{x}_q - \underline{x}\|^2 = \frac{|g(\underline{x})|^2 \underline{w}^T \underline{w}}{\|\underline{w}\|^4}$$

$$\sqrt{} \qquad = \frac{|g(\underline{x})|}{\|\underline{w}\|}.$$

---

ii) We already found the point on the decision boundary in part i): $\underline{x}_q = \underline{x} - \dfrac{g(\underline{x})\underline{w}}{\|\underline{w}\|^2}$

3) Pretty easy to illustrate this graphically. Consider 3 classes in the plane:



These classes <u>are</u> pairwise linearly separable, but you cannot, e.g., linearly separate class 1 from classes 2 and 3.

4) Let $b = \underline{x}^T \underline{1}$, where $\underline{1} = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix}$ (all ones vector).

Then, $W_1 = \{ \underline{x} : b \text{ is odd}$

$W_2 = \{ \underline{x} : b \text{ is even}$

a) Let's show these classes are not lin. sep. by showing a contradiction if we assume it to be true.

Spse. $\exists (\underline{w}, w_0)$ s.t.

$$\underline{w}^T \underline{x} + w_0 \geq 0 \quad \text{for } \underline{x} \in W_1$$

and

$$\underline{w}^T \underline{x} + w_0 < 0 \quad \text{for } \underline{x} \in W_2$$

$\underline{x} = \underline{0} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \Rightarrow b = 0$ + belongs to W_1; $\therefore$,

$\underline{x} = \underline{0}$ must satisfy $w_0 \geq 0$ (1)

$\underline{x} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$, i.e. a vector with one '1', $\Rightarrow b = 1$ + belongs to W_2.

$\therefore$, it must satisfy $\underbrace{w_i + w_0 < 0}_{(2)}$, i the position of the '1'.

$\underline{x} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$, i.e. a vector with two '1''s $\Rightarrow b = 2$ + belongs to W_1.

Suppose the ones are in positions i and j, i.e.

(3) $w_i + w_j + w_0 \geq 0$, $i \neq j$.

Consider eqn. (2) for positions i and j: $w_i + w_0 < 0$
$w_j + w_0 < 0$

Also, rewrite (1) as $-w_0 \leq 0$

Adding these 3 eqns together, we get.

$$\underbrace{(w_i + w_0)}_{<0} + \underbrace{(w_j + w_0)}_{<0} + \underbrace{(-w_0)}_{\leq 0} < 0 \,!$$

But this contradicts (3).

b) For simplicity, let's assume d is even.
First, we observe (recall) that $\underline{w}' = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}$ can be used to count the number of ones in $\underline{x}$, via $K_0 = \underline{w}'^T \underline{x}$.

This immediately suggests the following as a possible solution strategy:

1) Choose $\ell^* = \underset{\ell \in \{0,1,\dots d\}}{\arg\min} (\underline{w}'^T \underline{x} - \ell)^2$

2) Take the parity of ℓ^* as the decision result.

This method certainly works, but unfortunately is not based on linear discriminants, but rather quadratic discriminants (expand the square above to see this clearly).
However, something related that is based on linear discriminants will in fact work.

First, just consider the even integers $0, 2, 4, \dots d$.

$$|\underline{w}'^T \underline{x} - 2m|^2 = |K_0 - 2m|^2 = K_0^2 - 4mK_0 + 4m^2$$

$$\text{Clearly, } \underset{m \in \{0,1,\dots d/2\}}{\arg\min} |K_0 - 2m|^2 = \underset{m \in \{0,1,\dots d/2\}}{\arg\min} -4mK_0 + 4m^2$$

$$= \underset{m \in \{0,1,\dots d/2\}}{\arg\max} 4mK_0 - 4m^2$$

$$= \underset{m \in \{0,1,\dots d/2\}}{\arg\max} 4m\underline{w}'^T \underline{x} - 4m^2 \triangleq m_e^* \quad (\square)$$

(obvious from above, but you can also take derivatives w.r.t. m, set to zero, + solve for m_e^*).

It's easy to verify in several ways that $m_e^* = \frac{K_0}{2}$, i.e. the even integer result is $2m_e^*$.

Further recognize that $(\square)$ specifies a set of $d/2+1$ linear discriminant functions, and a way of selecting the nearest even integer to K_0.

Next, consider the odd integers:

$$(\underline{w}'^T\underline{x} - (2m+1))^2 = (k_0 - (2m+1))^2 = k_0^2 - (4m+2)k_0 + 4m^2 + 4m + 1$$

$$\underset{m \in \{0,1,\dots \frac{d}{2}-1\}}{\arg\min} (k_0 - (2m+1))^2 = \underset{m}{\arg\min} -(4m+2)k_0 + 4m^2 + 4m + 1$$

$$= \underset{m}{\arg\max} (4m+2)k_0 - 4m^2 - 4m - 1$$

(again, ~~take~~ derivatives + solve for maximum)

$$= \underset{m}{\arg\max} (4m+2)\underline{w}'^T\underline{x} - 4m^2 - 4m - 1 \quad (\Delta)$$

The last quantity is maximized @ $m_0^* = \frac{k_0}{2} - \frac{1}{2}$ -- this is integer when k_0 is odd, and non-integer otherwise. In the odd case, note that we get the desired equation $2m_0^* + 1 = k_0$.

In the even case, (Δ) still gives a result (the nearest odd integer to the even k_0...).

→ class 1 eq., and the the null set.

OK, so $(\square)$ defines a set of LDFs that are compared to pick the best even integer.

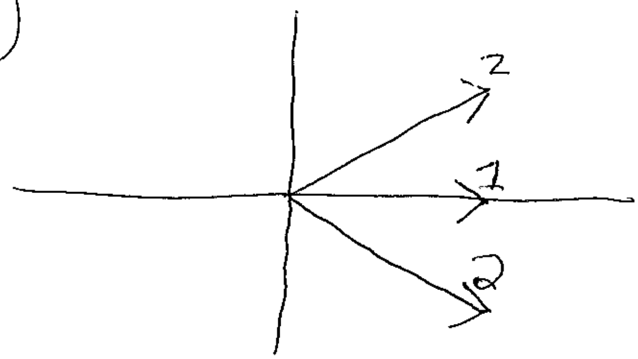
(Δ) defines a set of LDFs compared to pick the best odd integer.

Now, we need to choose between the best even + odd selections. For this, simply recognize that in deriving LDFs in both the even and odd cases, we started from the same sqd. distance expression, and we made the function linear by ignoring the same constant term k_0^2. Thus, in both the even and odd cases, the LDFs amount to the sqd. distance minus the same quantity, k_0^2. Thus, clearly, the (best) even + odd discriminants can simply be compared, with the (even/odd) argument of the largest of the two giving the parity, i.e.

Choose even if + only if

$$4m_e^* \underline{w}'^T\underline{x} - 4m^2 \geq (4m_o^*+2)\underline{w}'^T\underline{x} - 4m_o^{*2} - 4m_o^* - 1$$

#5)



Easy to verify that the solution planes for these 3 points have a null intersection (no solution cone).

#5) $P[x_1, x_2, \ldots x_T / \mu, \Sigma] =$

$$\frac{1}{(2\pi)^{Td/2} |\Sigma|^{T/2}} \cdot e^{\left[-\frac{1}{2}\sum_{k=1}^{T}(x_k-\mu)^T\Sigma^{-1}(x_k-\mu)\right]}$$

$$\ell(\mu, \Sigma) = \log P[\cdot]$$

$$= -\frac{Td}{2}\ln(2\pi) - \frac{T}{2}\ln|\Sigma| - \frac{1}{2}\sum_{k=1}^{T}\left(x_k^T \Sigma^{-1} x_k - 2\mu^T\Sigma^{-1}x_k + \mu^T\Sigma^{-1}\mu\right)$$

$$= -\frac{Td}{2}\ln(2\pi) - \frac{T}{2}\ln|\Sigma| - \frac{1}{2}\sum_{k=1}^{T} x_k^T x_k + \mu^T\Sigma^{-1}\cdot\sum_{k=1}^{T}x_k - \frac{T}{2}\mu^T\Sigma^{-1}\mu$$

To calculate $\nabla_\mu \ell(\cdot)$ and $\nabla_\Sigma \ell(\cdot)$, we'll use the following ∇ properties (for vectors $\underline{r}, \underline{v}$ and matrix A)

1) $\nabla_{\underline{r}}(\underline{r}^T\underline{v}) = \underline{v}$
2) $\nabla_{\underline{r}}(\underline{r}^T A \underline{r}) = (A + A^T)\underline{r}$
3) $\nabla_A(\underline{r}^T A \underline{r}) = \underline{r}\,\underline{r}^T$
4) $\nabla_A(\ln|A|) = A^{-1}$

$$\nabla_\mu(\ell(\mu,\Sigma)) = \nabla_\mu\Big(\mu^T\underbrace{\Big(\Sigma^{-1}\cdot\sum_{k=1}^{T}x_k\Big)}_{\underline{v}}\Big) - \frac{T}{2}\nabla_\mu(\mu^T\Sigma^{-1}\mu)$$

$\boxed{\text{from 1), this equals } \left(\Sigma^{-1}\cdot\sum_{k=1}^{T}x_k\right)}$

$\boxed{\text{From 2), this equals } (\Sigma^{-1} + (\Sigma^{-1})^T)\mu = 2\Sigma^{-1}\mu}$

So: $\nabla_\mu \ell(\hat{\mu}, \hat{\Sigma}) = \hat{\Sigma}^{-1}\left(\sum_{k=1}^{T} x_k\right) - T\hat{\Sigma}^{-1}\hat{\mu} = 0$

multiply both sides by $\hat{\Sigma}$
+ divide by $T \Rightarrow$

$$\boxed{\hat{\mu} = \frac{1}{T}\sum_{k=1}^{T} x_k}$$

$\nabla_\Sigma (\ell(\mu, \Sigma)) = ?$

To simplify this part, let $B = \Sigma^{-1}$.

Then, $\ell(\mu, B) = -\frac{Td}{2}\ln(2\pi) - \frac{T}{2}\ln|B^{-1}|$
$- \frac{1}{2}\sum_{k=1}^{T}(x_k - \mu)^T B (x_k - \mu)$

$\ln|B^{-1}| = -\ln|B|$

Also, from 4), $\nabla_B \ln|B| = B^{-1} \Rightarrow$

$\nabla_B\left(-\frac{T}{2}\ln|B^{-1}|\right) = \frac{T}{2}B^{-1}$

Next, $\nabla_B\left(-\frac{1}{2}\sum_{k=1}^{T}(x_k-\mu)^T B (x_k - \mu)\right) =$

$-\frac{1}{2}\sum_{k=1}^{T}\nabla_B\left[(x_k-\mu)^T B (x_k-\mu)\right] \overset{\text{From 3)}}{=}$

$-\frac{1}{2}\sum_{k=1}^{T}(x_k-\mu)(x_k-\mu)^T$

So: $\nabla_B(\ell(\hat{\mu}, \hat{B})) = \frac{T}{2}\hat{B}^{-1} - \frac{1}{2}\sum_{k=1}^{T}(x_k-\hat{\mu})(x_k-\hat{\mu})^T = 0$

$\hat{B}^{-1} = \hat{\Sigma} \Rightarrow$

$$\boxed{\hat{\Sigma} = \frac{1}{T}\sum_{k=1}^{T}(x_k-\hat{\mu})(x_k-\hat{\mu})^T}$$

#7) Using the notation from class:

$$\frac{\partial}{\partial\theta_1}(l_k) = \frac{1}{\theta_2}(x_k - \theta_1)$$

$$\frac{\partial}{\partial\theta_2}(l_k) = -\frac{1}{2\theta_2} + \frac{1}{2\theta_2^2}(x_k - \theta_1)^2$$

$$\frac{\partial^2(l_k)}{\partial\theta_1^2} = -\frac{1}{\theta_2}$$

$$\frac{\partial^2(l_k)}{\partial\theta_2^2} = \frac{1}{2\theta_2^2} - \frac{1}{\theta_2^3}(x_k - \theta_1)^2$$

$$\frac{\partial^2(l_k)}{\partial\theta_1\partial\theta_2} = -\frac{(x_k - \theta_1)}{\theta_2^2}$$

$$\frac{\partial^2(l_k)}{\partial\theta_2\partial\theta_1} = -\frac{1}{\theta_2^2}(x_k - \theta_1)$$

Now, $$\frac{\partial^2 l(\underline{\theta})}{\partial\theta_i\partial\theta_j} = \sum_{k=1}^{T} \frac{\partial^2(l_k)}{\partial\theta_i\partial\theta_j}, \quad \forall i, j$$

$$H = \left.\begin{pmatrix} \frac{\partial^2 l(\underline{\theta})}{\partial\theta_1^2} & \frac{\partial^2 l(\underline{\theta})}{\partial\theta_2\partial\theta_1} \\ \frac{\partial^2 l(\underline{\theta})}{\partial\theta_1\partial\theta_2} & \frac{\partial^2 l(\underline{\theta})}{\partial\theta_2^2} \end{pmatrix}\right|_{\underline{\theta} = \hat{\theta}_{ML}}$$

It is easy to see that, in our case,

$$H = \begin{pmatrix} -\frac{N}{\hat{\sigma}^2} & 0 \\ 0 & \underbrace{\frac{N}{2\hat{\sigma}^4} - \frac{N}{\hat{\sigma}^4}}_{-\frac{N}{2\hat{\sigma}^4}} \end{pmatrix}$$

Let $\underline{a} = \begin{pmatrix} a_0 \\ a_1 \end{pmatrix}$ be an arbitrary 2-vector.

Then, $$\underline{a}^T H \underline{a} = -\frac{N}{\hat{\sigma}^2}a_0^2 - \frac{N}{2\hat{\sigma}^4}a_1^2 < 0$$

H negative definite $\Longrightarrow$ ML solution is a maximum...

Homework #3, EE556, Fall 2018

Due: 9/27/18; hand in Problems 1, 2, and 6

Problem #1

A classifier uses four linear discriminant functions in the plane: $g_1 = x_1$, $g_2 = x_2$, $g_3 = x_2 + x_1 - 3$, and $g_4 = x_2 - x_1 + 1$.

i) Sketch the line boundaries in pattern space and label each region with a 4-bit binary codeword.

ii) Make a logical table, identifying for each codeword whether or not there is an associated region.

iii) Sketch the **four**-dimensional hypercube in state space. (**Don't panic !** Use two 3-dimensional projections, one yielding a 3D cube for the case $T_1 = 0$ and the other giving a 3D cube for $T_1 = 1$.). Label each vertex with its corresponding binary codeword. Draw a curved line joining each vertex in the first cube with the corresponding vertex in the second cube.

ii) Consider the discriminant function $y = \mathrm{sgn}(\sum_{i=1}^{4} T_i - 2.5)$. Sketch the decision region induced by this rule in the (2D) feature space.

Problem #2

Construct a multilayer perceptron that solves the N-bit parity problem. (Recall this problem from homework 2).

Problem #3

i) Consider an MLP with I inputs, J hidden units, and K output units (a single hidden layer). What is the space complexity of the network ? (Include the the storage required for MLP parameters, training data, and and any additional storage needed during training).

ii) What is the computational complexity of backpropagation training in batch gradient descent mode ?

Problem #4

Consider a standard multilayer perceptron. Show that if the sign on every weight is flipped the operation of the network remains unchanged (a type of "polar symmetry") – does this property extend to changing the signs of the inputs to the network ?

Problem #5

Derive the learning rule for updating an input-to-hidden unit weight for a single-hidden layer MLP that uses a "softmax function" in the output layer and the cross entropy criterion for training (both "softmax" and "cross entropy" are discussed in lecture).

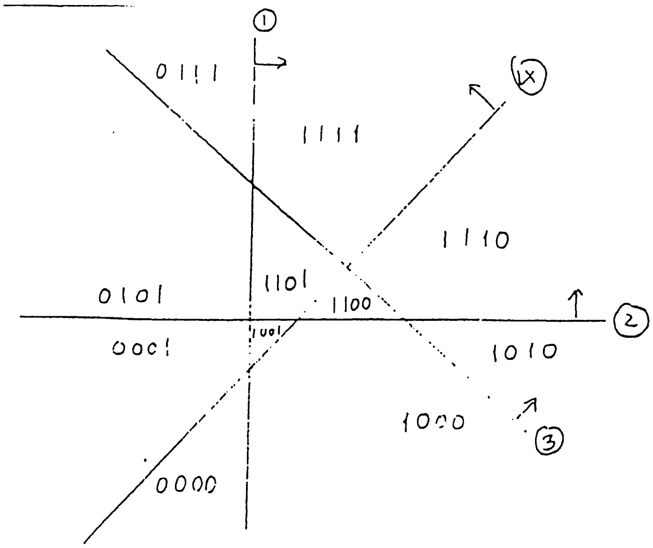
Problem #6

Consider the problem of neural network *inversion*, wherein, given a fixed network and target output values, the objective is to learn the associated *input* patterns which, when forward-propagated through the network, produce outputs that well-approximate the targets. Sketch a method (an optimization technique ?) that approximately achieves NN inversion. Also suggest some possible applications for NN inversion.

#1)

HW #3 Solns., EE556

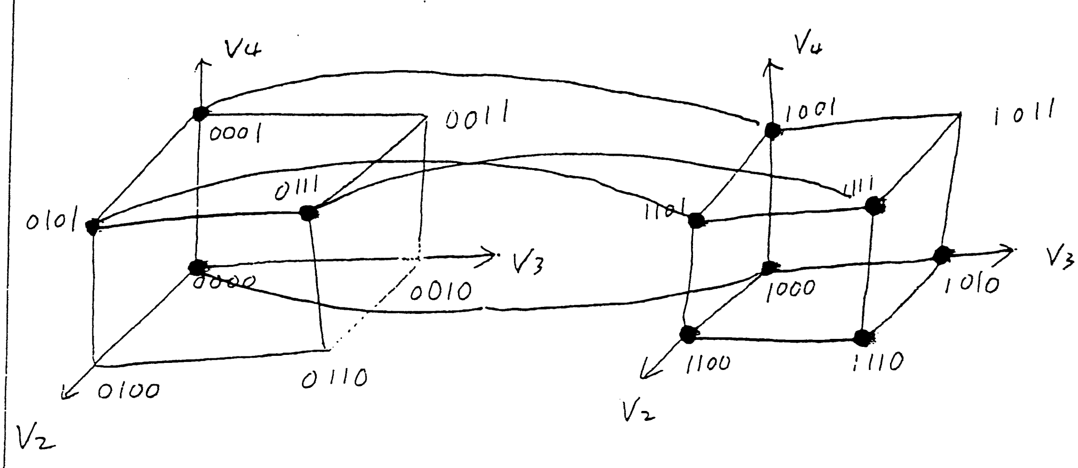
(a)



(b)

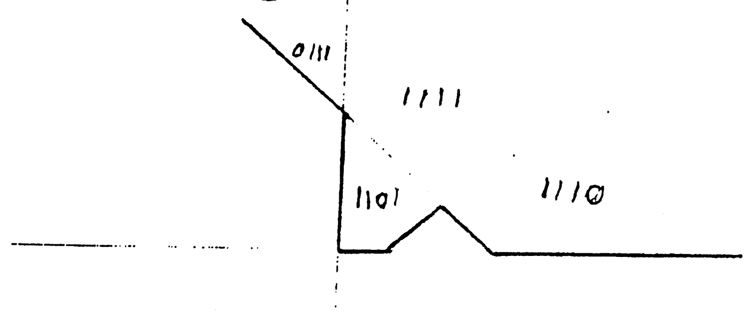
V_1	V_2	V_3	V_4	Class
0	0	0	0	✓
0	0	0	1	✓
0	0	1	0	IMP
0	0	1	1	IMP
0	1	0	0	IMP
0	1	0	1	✓
0	1	1	0	IMP
0	1	1	1	✓
1	0	0	0	✓
1	0	0	1	✓
1	0	1	0	✓
1	0	1	1	IMP
1	1	0	0	✓
1	1	0	1	✓
1	1	1	0	✓
1	1	1	1	✓

(c) $V_1 = 0$ $V_1 = 1$



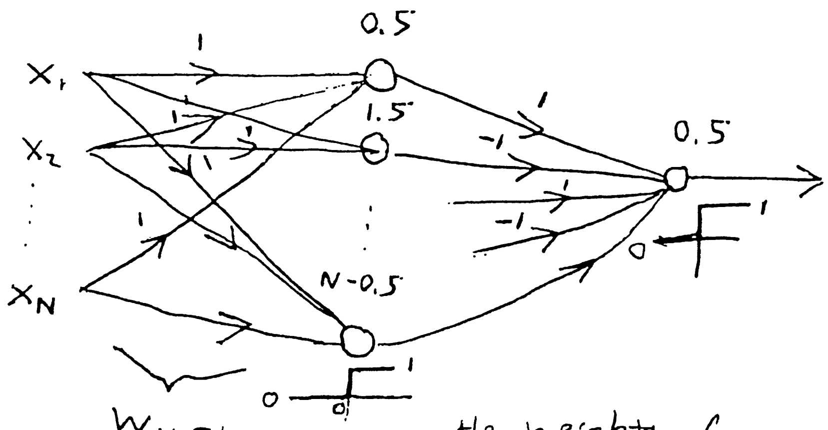
(d) Let $w_1 = w_2 = w_3 = w_4 = 1$, $w_5 = -2.5$

then $y = U(v_1 + v_2 + v_3 + v_4 - 2.5)$ classifies a region that is not convex nor is it the compliment of a convex region as shown below.



#2)

Consider the following network:



- the weights from input to each hidden neuron are all '1' ⇒ each hidden neuron counts the # of ones; neuron i then compares the count to the threshold $i - 0.5$ and uses a $0-1$ threshold activation.
- "odd" neurons use "1" weights to the output; "even" neurons use "-1" weights to the output neuron.
- The output is "1" if the parity is odd & "0" if even.

#3) i) Space (storage) complexity :

# of adjustable weights from input to hidden layer is : $I \cdot J$

# of adjustable weights from hidden to output layer is : JK

Assuming thresholds are all at zero, total # weights =

$$I \cdot J + J \cdot K$$

During learning, however, one also has to store the gradients $\Rightarrow$ total storage for weights + their gradients is : $2(IJ + JK)$.

We also need to store the training patterns $\Rightarrow$

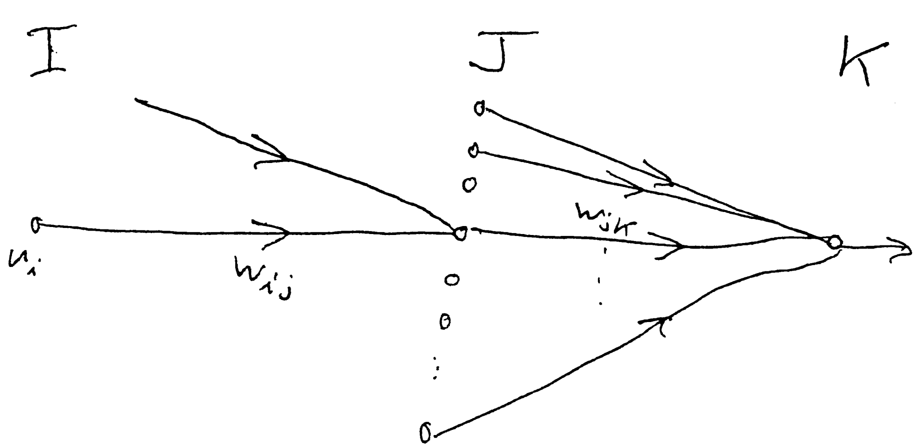
$(I + 1)T$

input — output dim. — # of training examples

$\Rightarrow$ space complexity $= 2(IJ + JK) + (I+1)T$

ii) <u>Computational complexity of BP</u> :

Refer to the BP derivation from lecture :



(assume single output, y)

#3) i) Space (storage) complexity :

# of adjustable weights from input to hidden layer is : $I \cdot J$

# of adjustable weights from hidden to output layer is : JK

Assuming thresholds are all at zero, total # weights =

$$I \cdot J + J \cdot K$$

During learning, however, one also has to store the gradients $\Rightarrow$ total storage for weights + their gradients is : $2(IJ + JK)$.

We also need to store the training patterns $\Rightarrow$

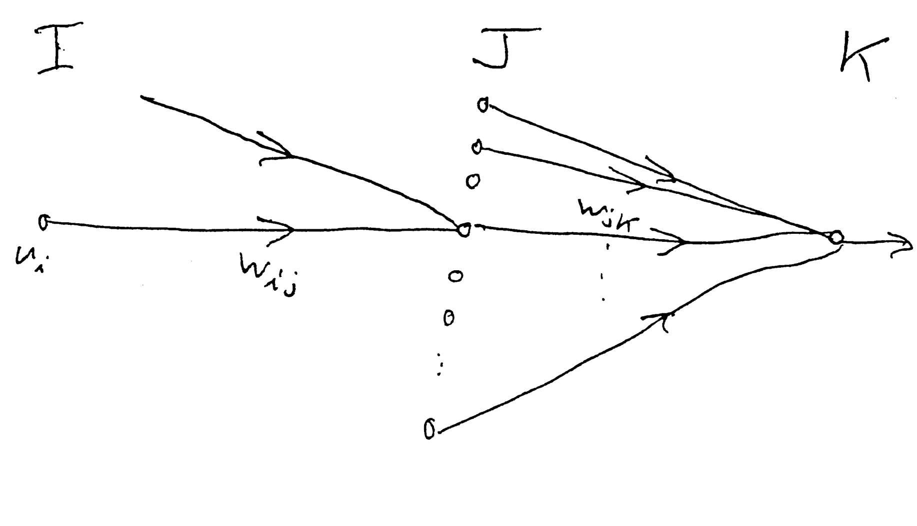
$(I + 1)T$

input output dim. # of training examples

$\Rightarrow$ space complexity $= 2(IJ + JK) + (I+1)T$

ii) <u>Computational complexity of BP</u> :

Refer to the BP derivation from lecture :



(assume single output, y)

$$V_j = \sum_{i=1}^{I} w_{ij} u_i, \quad j = 1, \ldots J \Rightarrow O(IJ) \text{ operations,}$$

per sample

$$Y = g\left(\sum_{j=1}^{J} W_{jo}\, g(V_j)\right) \Rightarrow O(J) \text{ operations,}$$

per sample

(mults, & adds).

Forward operations

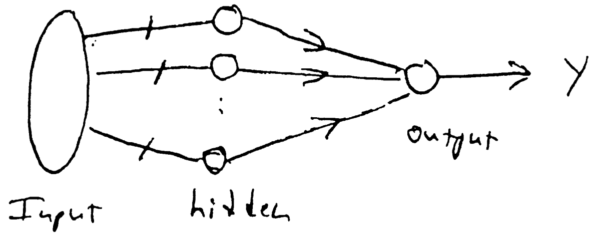
$$\delta_j = g'(V_j)\,\delta_o\, W_{jo}, \quad j = 1, \ldots J \Rightarrow O(J) \text{ operations}$$

$$\Delta W_{ij} \propto \delta_j \cdot u_i, \quad i = 1, \ldots I, \; j = 1, \ldots J \Rightarrow O(IJ) \text{ operations}$$

$$\Delta W_{jo} \propto -\frac{\partial E}{\partial V_o} \cdot \frac{\partial V_o}{\partial W_{jo}}, \quad j = 1, \ldots J \Rightarrow O(J) \text{ operations}$$

overall complexity is $O(I \cdot J \cdot T)$ operations

#4) Consider the MLP:



Suppose each neuron uses an <u>antisymmetric</u> activation function, i.e. $f(-x) = -f(x)$ (sigmoid, tanh, or $_{-1}$ $^{+1}$).

The output can be written

$$y = f\Big(\sum_{j=1}^{N_h} \underbrace{w_{j0}}_{\text{output}} f\Big(\sum_{i=1}^{d} x_i w_{ij} + w_{j0}\Big) + w_0\Big)$$

Spse. we flip the sign on all the weights.

Then, neuron j's output becomes

$$f\Big(\sum_{i=1}^{d} x_i(-w_{ij}) + (-w_{j0})\Big) = -f\Big(\sum_{i=1}^{d} x_i w_{ij} + w_{j0}\Big)$$

The overall output becomes:

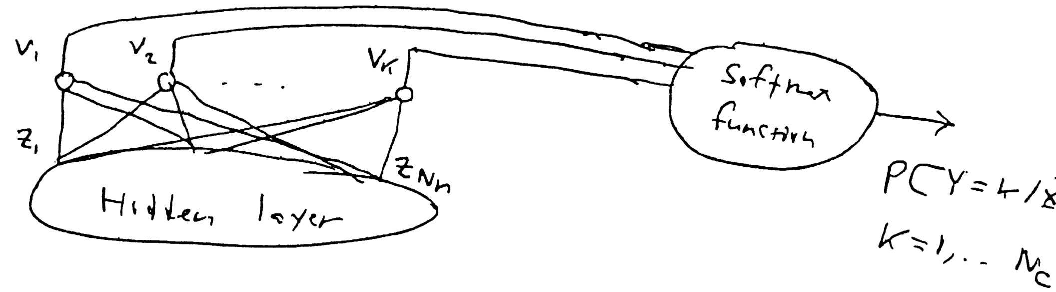
$$f\Big(\sum_{j=1}^{N_h} (-w_{j0})\Big(-f\Big(\sum_{i=1}^{d} x_i w_{ij} + w_{j0}\Big)\Big) - w_0\Big)$$

$$= f\Big(\sum_{j=1}^{N_h} w_{j0} f\Big(\sum_{i=1}^{d} x_i w_{ij} + w_{j0}\Big) - w_0\Big).$$

The output stays the same <u>if</u> $w_0 = 0$ -- otherwise the operation is in fact changed.

This property does NOT extend to flipping the sign on the input.

#5) Let's just consider the signals from the hidden layer to the softmax function:



Let's denote the outputs of the hidden layer by: z_j, $j=1,\dots N_h$.

Then, $v_K = \sum_{j=1}^{N_h} z_j w_{jK}$ and $P(Y=k/\underline{x}) = \dfrac{e^{v_K}}{\sum_{K'=1}^{N_c} e^{v_{K'}}}$

As derived in lecture, the cross entropy objective function can be written as:

$$E = -\sum_{c=1}^{N_c} \sum_{\underline{x}:\, c(\underline{x})=c} \log P(Y=c/\underline{x})$$

Now, $\Delta w_{jK} \propto \dfrac{\partial E}{\partial v_K} \cdot \underbrace{\dfrac{\partial v_K}{\partial w_{jK}}}_{z_j}$

$$\frac{\partial E}{\partial v_K} = -\sum_{\underline{x}:\, c(\underline{x})=k} \frac{\partial}{\partial v_K}\left(\log P(Y=k/\underline{x})\right) - \sum_{\substack{c=1,\\ c\neq k}}^{N_c} \sum_{\underline{x}:\, c(\underline{x})=c} \frac{\partial}{\partial v_K}\left(\log P(Y=c/\underline{x})\right),$$

where:

$$\frac{\partial}{\partial v_K}\left(\log P(Y=k/\underline{x})\right) = \frac{\partial}{\partial v_K}\left(v_K - \log\left(\sum_{K'=1}^{N_c} e^{v_{K'}}\right)\right)$$

$$= 1 - \frac{e^{v_K}}{\sum_{K'=1}^{N_c} e^{v_{K'}}} = 1 - P(Y=k/\underline{x})$$

#6) Neural Network Inversion

i) $\underline{X} \longrightarrow \boxed{\text{MLP}} \longrightarrow \underline{Y}$

$\underline{Y} = f(\underline{X})$, with $f(\cdot)$ defined by the MLP. Ideally, if we were given a vector $\underline{Y}$, we could find $\underline{X}$ via $\underline{X} = f^{-1}(\underline{Y})$. Unfortunately,

i) there are/may be many possible $\underline{X}$ that could lead to same $\underline{Y}$ $\Rightarrow$ non-unique inverse;

ii) there may be no $\underline{X}$ leading to a particular $\underline{Y}$.

iii) In general, there is no analytical form for $f^{-1}(\cdot)$.

So, how to find an approximate inverse?

Ans: Use backpropagation!

Let $E = \|\underline{Y} - f(\underline{X})\|^2$ + choose $\underline{X}$ to minimize E.

Algorithm:

$t=0$; $\underline{X}^t = \underline{X}_0$ (Init) $\longleftarrow$ Note: strong dependence on initialization...

loop

$$X_{t+1} = X_t - \mu \nabla_{\underline{X}} E(\underline{X})\Big|_{\underline{X} = \underline{X}_t}$$

$t \leftarrow t+1$

end loop

Two applications: 1) signal/image reconstruction from noisy, blurred output image,

2) Boundary-finding in classification -- given an MLP classifier, find $\underline{X}$ that cause output to be $\approx 1/2$ (points on the boundary ...).

Homework #4, EE556, Fall 2018

Due: 10/11; hand in problems 4 and 5

Problem #1

6.6, Haykin's text

Problem #2

Consider a single LDF that, for augmented patterns from two classes (and with all patterns from class 2 multiplied by negative one) achieves $\underline{a}^T \tilde{\underline{y}} > b, \forall \tilde{\underline{y}}$. Show that the solution vector with minimum length is unique (this relates to uniqueness of the SVM solution). *Hint: suppose that there are two such solutions, and take their average.*

Problem #3

Consider the linear support vector machine mathematical framework developed in lecture. Specialize this development for the case where all the training vectors are orthonormal – it should be relatively easy to derive the solution in this case. How many training points are support vectors in this case ?

Problem #4

Consider M linearly independent data patterns $\underline{x}_i, i = 1, \ldots, M$, each N-dimensional, where $M < N$. Prove that there is a *linear* separator, i.e. a vector $\underline{w}$ such that $\underline{w}^T \underline{x}_i > 0 \forall i$. *Hint:* try a linear algebraic approach.

Problem #5: Multilayer Perceptron Computer Assignment

i) In this assignment, you are asked to design a multilayer perceptron to solve the XOR problem. The objective of XOR is to assign the patterns (0,0) and (1,1) to class ω_0 and the patterns (0,1) and (1,0) to class ω_1. The design will consist of three steps: i) choose an architecture capable of solving this problem (choose the number of inputs, layers, number of hidden units in each layer, activation function type for each neuron); ii) write a computer program that implements the back propagation algorithm (you can use built-in matlab functions if you prefer); iii) Choose initial parameter values and train the network using your program to minimize a sum-of-squared errors cost function over a training set consisting of the four patterns. At convergence, save the learned parameters; iv) demonstrate that your network correctly classifies all four input patterns. You should hand in your code and a plot showing the training cost function versus the number of batch gradient steps (to indicate gradient descent progress on the cost surface).

Note: you can use the Neural Network toolbox in Matlab to perform the design and to classify the patterns. Alternatively, you can implement your own back propagation algorithm (which I highly recommend as a learning experience).

ii) Now we will also investigate several real data sets from the UC Irvine repository: *glass* and *Pima Indians*. You are asked to do the following:

a) Read the given descriptions of these data sets.

b) Download these data sets from the UC Irvine machine learning web site.

c) Split into equal-sized training and test sets (Note that for *glass* this cannot be done by choosing the first half as the training set and the second half as the test set – why ?).

d) Build MLPs with different numbers of hidden units and then evaluate the training set and test set classification accuracies. Plot these performances as a function of number of hidden units.

e) Note any distinctive experimental observations.

EE 556

HW #4 Solutions

#2)
(6.6, Haykin)

$K = [K(\underline{x}_i, \underline{x}_j)]$ is a square matrix.

Therefore, it can be written as:

$K = Q \Lambda Q^T$, where Λ is a diagonal matrix w/ the eigenvalues of K and Q is an orthogonal matrix whose columns are the associated eigenvectors (this is often called a "similarity transform").

Because $K(\underline{x}_i, \underline{x}_j)$ is positive definite, all the eigenvalues are non-negative.

We can write:

$$K(\underline{x}_i, \underline{x}_j) = (Q \Lambda Q^T)_{ij} = \sum_{l=1}^{m} (Q)_{il} (\Lambda)_{ll} (Q^T)_{lj}$$

$$= \sum_{l=1}^{m} Q_{il} \Lambda_{ll} Q_{jl}$$ (since for an orthogonal matrix, $Q = Q^T$).

Let $\underline{u}_i$ denote the i-th <u>row</u> of the matrix Q. (Note that $\underline{u}_i$ is <u>NOT</u> an eigenvector of K ...). (The columns are the eigenvectors...).

Then, $K(\underline{x}_i, \underline{x}_j) = \underline{u}_i^T \Lambda \underline{u}_j$

$$= (\Lambda^{1/2} \underline{u}_i)^T (\Lambda^{1/2} \underline{u}_j)$$

By definition, $K(\underline{x}_i, \underline{x}_j) = \underline{\Phi}^T(\underline{x}_i)\,\underline{\Phi}(\underline{x}_j)$.

Therefore, we have : $\underline{\Phi}(\underline{x}_i) = \Lambda^{1/2}\underline{u}_i$, i.e. the mapping from the input space to the feature space for a Kernel SVM is given by :

$$\Phi : \underline{x}_i \longrightarrow \Lambda^{1/2}\underline{u}_i \quad \left(\begin{array}{l}\text{the mapping for the} \\ \text{training vectors} \ldots\end{array}\right).$$

#3)

We employ proof by contradiction. Suppose there were two *distinct* minimum length solution vectors a_1 and a_2 with $a_1^t y > 0$ and $a_2^t y > 0$. Then necessarily we would have $\|a_1\| = \|a_2\|$ (otherwise the longer of the two vectors would not be a *minimum* length solution). Next consider the average vector $a_o = \frac{1}{2}(a_1 + a_2)$. We note that

$$a_o^t y_i = \frac{1}{2}(a_1 + a_2)^t y_i = \frac{1}{2}a_1^t y_i + \frac{1}{2}a_2^t y_i \geq 0,$$

and thus a_o is indeed a solution vector. Its length is

$$\|a_o\| = \|1/2(a_1 + a_2)\| = 1/2\|a_1 + a_2\| \leq 1/2(\|a_1\| + \|a_2\|) = \|a_1\| = \|a_2\|,$$

where we used the triangle inequality for the Euclidean metric. Thus a_o is a solution vector such that $\|a_o\| \leq \|a_1\| = \|a_2\|$. But by our hypothesis, a_1 and a_2 are minimum length solution vectors. Thus we must have $\|a_o\| = \|a_1\| = \|a_2\|$, and thus

$$\frac{1}{2}\|a_1 + a_2\| \quad = \quad \|a_1\| = \|a_2\|.$$

We square both sides of this equation and find

$$\frac{1}{4}\|a_1 + a_2\|^2 = \|a_1\|^2$$

or

$$\frac{1}{4}(\|a_1\|^2 + \|a_2\|^2 + 2a_1^t a_2) = \|a_1\|^2.$$

We regroup and find

$$\begin{aligned} 0 &= \|a_1\|^2 + \|a_2\|^2 - 2a_1^t a_2 \\ &= \|a_1 - a_2\|^2, \end{aligned}$$

and thus $a_1 = a_2$, contradicting our hypothesis. Therefore, the minimum-length solution vector is unique.

#4) Recall Wolfe-Dual problem:

$$\max_{\underline{\lambda}} \left\{ -\frac{1}{2} \sum_{x,x'} \lambda_x \lambda_{x'} t_x t_{x'} \underline{x}'^T \underline{x} + \sum_x \lambda_x \right\}$$

$$\text{s.t.} \quad \sum_x \lambda_x t_x = 0, \quad \underline{\lambda} \geq 0$$

If the data vectors are $\underbrace{\text{orthogonal, + unit norm}}_{\text{orthonormal}}$ $\underline{x}'^T \underline{x} = \delta_{x'x} \Rightarrow$

the function to be maximized reduces to:

$$-\frac{1}{2} \sum_x \lambda_x^2 + \sum_x \lambda_x$$

For now, ignore $\underline{\lambda} \geq 0$ + take the equality constraint into account via a Lagrange multiplier, μ.

I.e., form

$$L = -\frac{1}{2} \sum_x \lambda_x^2 + \sum_x \lambda_x - \mu \left(\sum_x \lambda_x t_x \right)$$

$$\frac{\partial L}{\partial \lambda_z} = -\lambda_z + 1 - \mu t_z = 0 \Rightarrow$$

$$\lambda_z = 1 - \mu t_z$$

Plugging into $\sum_x \lambda_x t_x = 0$ gives

$$\underbrace{\sum_x (1 - \mu t_x) t_x}_{= 0} \Rightarrow \mu = \frac{\sum_x t_x}{N}, \quad N$$

the # of data points.

Since $t_x \in \{-1, +1\}$, $-1 < \mu < 1 \Rightarrow \lambda_x > 0, \forall x$,

i.e. inequality constraint is automatically satisfied.

This also means that all training points are support vectors, in this case.

The solution thus has the form

$$\underline{w} = \sum_x \lambda_x t_x \underline{x} = \sum_x \left(1 - t_x \left(\frac{\sum_{x'} t_{x'}}{N} \right)\right) t_x \underline{x}$$

+

$$w_0 = t_x - \underline{w}^T \underline{x}, \quad \text{any } x.$$

$$= t_x - \lambda_x t_x = t_x - (1 - \mu t_x) t_x$$

$$= \mu = \boxed{\frac{\sum_{x} t_x}{N}}$$

Also, we can rewrite $\underline{w}$ as:

$$\underline{w} = \sum_x (1 - \mu t_x) t_x \underline{x}$$

$$\underbrace{\boxed{= \sum_x (t_x - \mu) \underline{x}}}$$

This is related to Hebbian learning...

#5) <u>Theorem</u>: Given M linearly independent vectors $\underline{x}_1, \ldots, \underline{x}_M$, $\underline{x}_i \in \mathbb{R}^N$, $M < N$, $\exists$ a vector $\underline{w}$ s.t. $\underline{w}^T \underline{x}_i > 0$, $i = 1, \ldots M$

<u>Proof</u>: The theorem statement is equivalent to the statement that

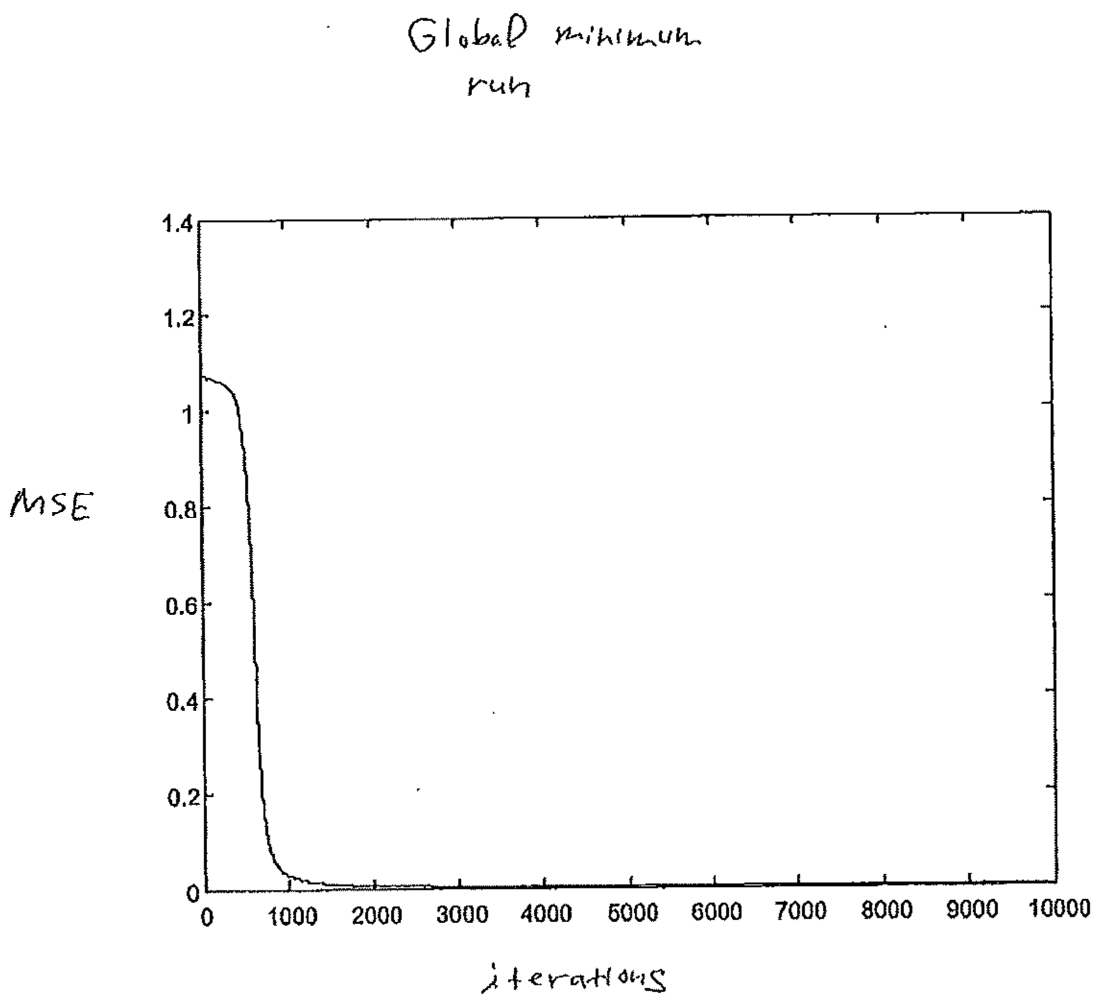
$$X^T \underline{w} = \underline{b} > \underline{0}$$

, that is, the right-hand side is a vector with strictly positive entries.

Here, $\underbrace{X}_{N \times M} = [\underline{x}_1, \ldots, \underline{x}_M]$.

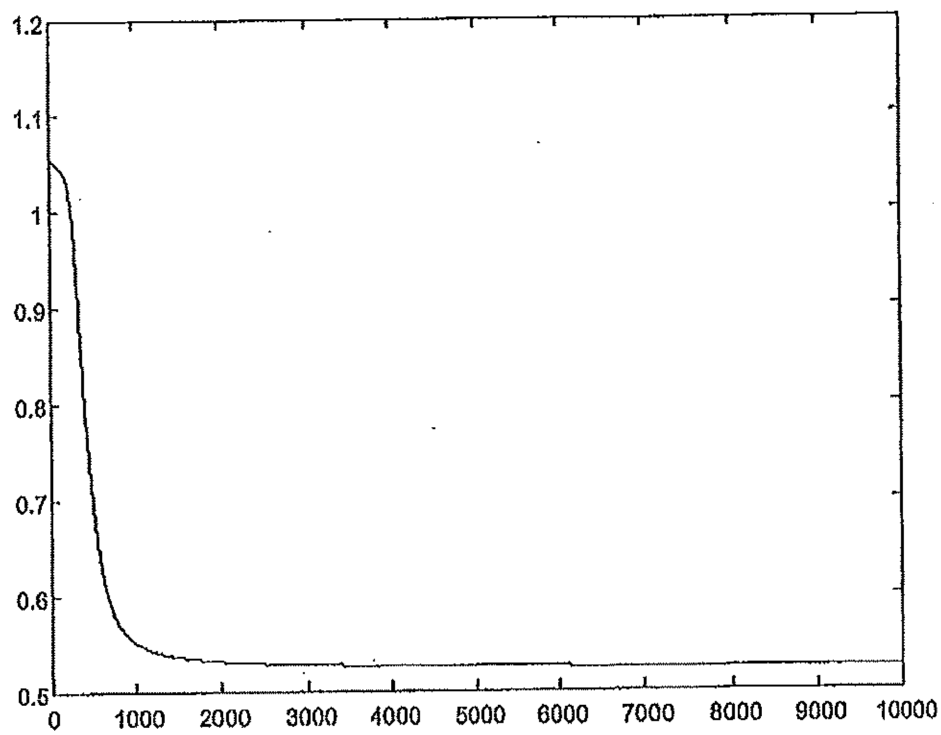
The matrix X has full column rank, M. Moreover, its <u>row</u> rank is also M. (see, e.g. $\boxed{\text{Strang}}$). This means that X^T has an M-dimensional column basis, However, note that the columns of X^T are M-dimensional vectors. This implies that the columns of X^T span $\mathbb{R}^M \Longrightarrow$ <u>any</u> vector $\underline{b} \in \mathbb{R}^M$ is in the column space of X^T, including $\underline{b}$ with all positive entries.

Thus, $\exists$ $\underline{w}$ s.t. $X^T \underline{w} = \underline{b} > \underline{0}$.


Global minimum run
MSE
1.4
1.2
1
0.8
0.6
0.4
0.2
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
iterations

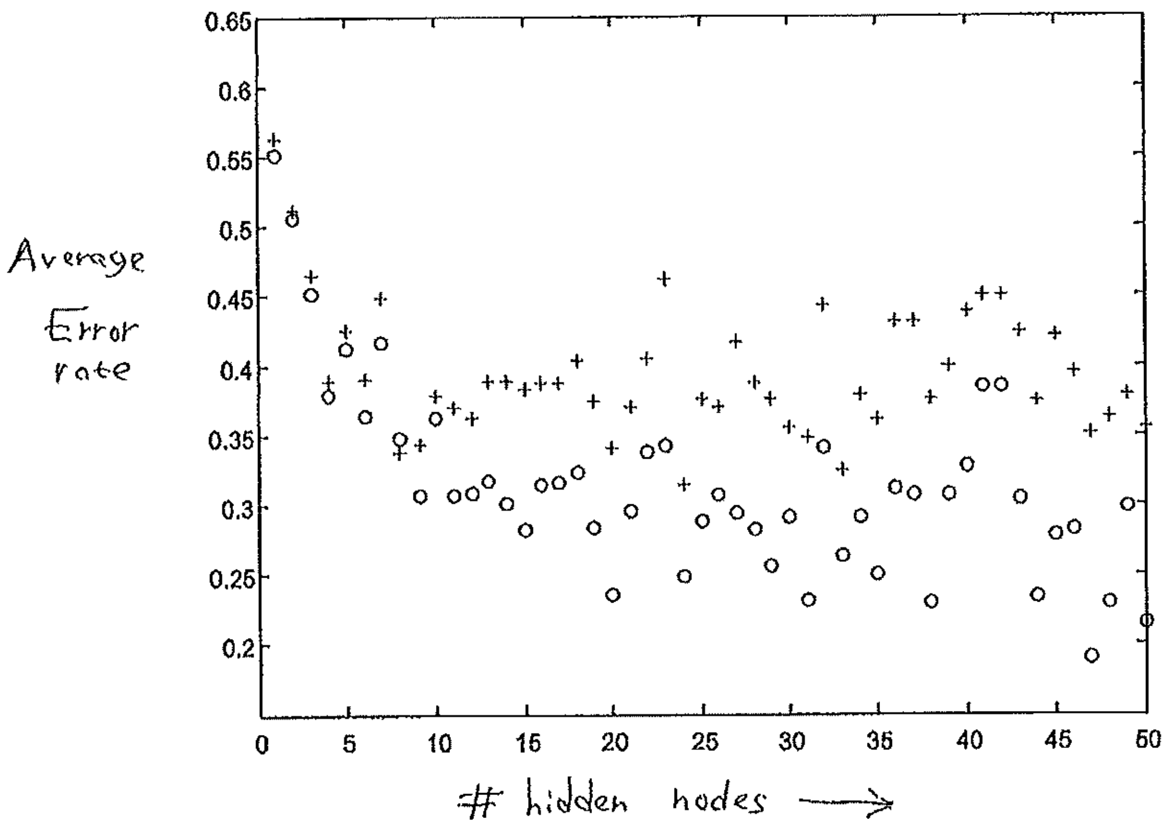
Local minimum run

MSE



iterations

Glass + = ~~training~~ test error rate
o = training error rate



Note: 1) training error rate lower than test error, generally.

2) Variance in performance grows with the number of hidden nodes

3) Best test error occurs with ~ 24 hidden units.

4) For this 6-class problem, test error rate is certainly much better than random guessing.