L20: MLPs, RBFs and SPR

Bayes discriminants and MLPs

The role of MLP hidden units

Bayes discriminants and RBFs

Comparison between MLPs and RBFs

Bayes discriminants and MLPs

As we have seen throughout the course, the classifier that minimizes $p[error]$ is defined by the MAP rule

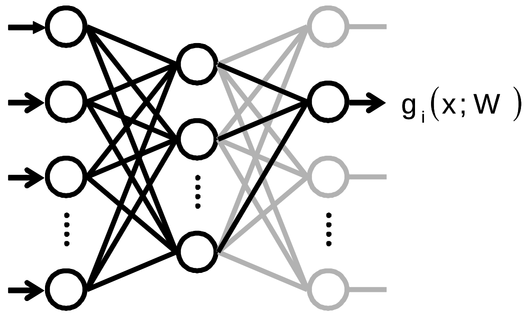
$$\omega^* = \arg\max_{\omega_i}\{g_i(x) = P(\omega_i|x)\}$$

How does the MLP relate to this optimal classifier?

– Assume a MLP with a one-of-C target encoding

$$t_i(x) = \begin{cases} 1 & x \in \omega_i \\ 0 & o.w. \end{cases}$$

– The contribution to the error of the i^{th} output is



$$J(W) = \sum_{\forall x}(g_i(x;W) - t_i)^2 =$$

$$\sum_{x\in\omega_i}(g_i(x;W) - 1)^2 + \sum_{x\notin\omega_i}(g_i(x;W) - 0)^2$$

$$= N\left[\frac{N_i}{N}\frac{1}{N_i}\sum_{x\in\omega_i}(g_i(x;W) - 1)^2 + \frac{N-N_i}{N}\frac{1}{N-N_i}\sum_{x\notin\omega_i}(g_i(x;W) - 0)^2\right]$$

– where $g_i(x;W)$ is the discriminant function computed by the MLP for the i^{th} class and the set of weights W

– Assuming infinite number of examples, the previous function becomes

$$\lim_{N\to\infty}\frac{1}{N}J(W)=$$

$$\lim_{N\to\infty}\left[\frac{N_i}{N}\frac{1}{N_i}\sum_{\mathrm{x}\in\omega_i}(g_i(x;W)-1)^2+\frac{N-N_i}{N}\frac{1}{N-N_i}\sum_{\mathrm{x}\notin\omega_i}(g_i(x;W)-0)^2\right]$$

$$=\lim_{N\to\infty}\left[\frac{N_i}{N}\right]\lim_{N\to\infty}\left[\frac{1}{N_i}\sum_{\mathrm{x}\in\omega_i}(g_i(x;W)-1)^2\right]+\lim_{N\to\infty}\left[\frac{N-N_i}{N}\right]\lim_{N\to\infty}\left[\frac{1}{N-N_i}\sum_{\mathrm{x}\notin\omega_i}g_i(x;W)^2\right]$$

$$=P(\omega_i)\int_x(g_i(x;W)-1)^2p(x|\omega_i)dx+P(\omega_{k\neq i})\int_x(g_i(x;W)-0)^2p(x|\omega_{k\neq i})dx$$

– Expanding the quadratic terms and rearranging

$$\lim_{N\to\infty}\frac{1}{N}J(W) =$$

$$= \int_x \left(g_i^2 - 2g_i + 1\right)p(x,\omega_i)dx + \int_x g_i^2 p(x,\omega_{k\neq i})dx =$$

$$\int_x g_i^2\left(p(x,\omega_i) + p(x,\omega_{k\neq i})\right)dx - \int_x 2g_i p(x,\omega_i)dx + \int_x p(x,\omega_i)dx =$$

$$\int_x g_i^2 p(x)dx - \int_x 2g_i P(\omega_i|x)p(x)dx + \int_x P^2(\omega_i|x)p(x)dx$$

$$- \int_x P^2(\omega_i|x)p(x)dx + \int_x p(x,\omega_i)dx =$$

$$\int_x (g_i - P(\omega_i|x))^2 p(x)dx \underbrace{- \int_x P^2(\omega_i|x)p(x)dx + \int_x p(x,\omega_i)dx}_{\text{independent of } W}$$

- Therefore, by minimizing J(W) we also minimize

$$\int_x (g_i(x;W) - P(\omega_i|x))^2 p(x)dx$$

- Summing over all classes (output neurons), we can then conclude that back-prop attempts to minimize the following expression

$$\sum_{i=1}^{N_C} \int_x (g_i(x;W) - P(\omega_i|x))^2 p(x)dx$$

- Therefore, in the large sample limit , the outputs of the MLP will approximate (in a least-squares sense) the posteriors

$$g_i(x;W) \cong P(\omega_i|x)$$

- Notice that nothing said here is specific to MLPs
    - Any discriminant function with adaptive parameters trained to minimize the sum squared error at the output of a 1-of-C encoding will approximate the a posteriori probabilities

This result will be true if and only if

- We use a 1-of-C encoding with [0,1] outputs, and
- The MLP has enough hidden units to represent the posteriors, and
- We have an infinite number of examples, and
- The MLP does not get trapped in a local minima

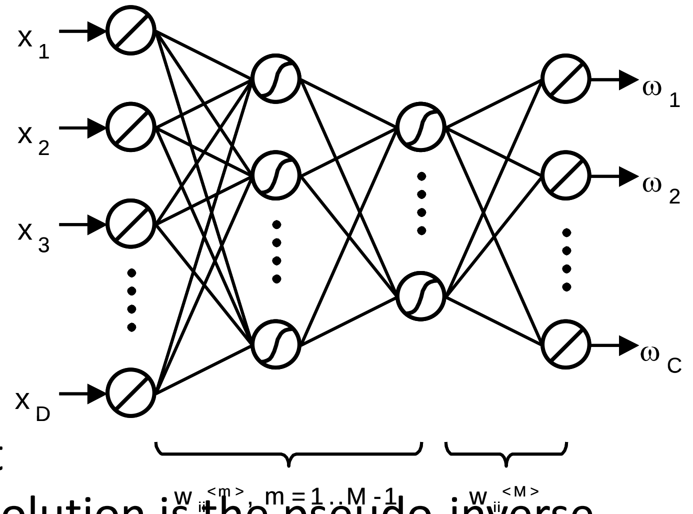
In practice we will have a limited number of examples

- The outputs will not always represent probabilities
  - For instance, there is no guarantee that they will sum up to 1
- We can use this result to determine if the network has trained properly
  - If the sum of the outputs differs significantly from 1, it will be an indication that the MLP is not modeling the posteriors properly and that we may have to change the MLP (topology, number of hidden units, etc.)
- If we still want to interpret MLP outputs as probabilities, we must then enforce that they add up to 1
  - This can be achieved by choosing the output's non-linearity to be exponential, and normalizing across the output layer

$$y_k^{\langle 2 \rangle} = \frac{\exp\left(net_k^{\langle 2 \rangle}\right)}{\sum_{k=1}^{N_O} \exp\left(net_k^{\langle 2 \rangle}\right)}$$

  - This is known as the **soft-max** method, which receives its name from the fact that it can be interpreted as a smoothed version of the *winner-take-all rule*

The role of MLP hidden units



Assume

- MLP with non-linear activation functions for the hidden layer(s) and linear activation function for the output layer
- Let us also hold constant the set of hidden weights $w_{ij}^{\langle m\rangle}; m = 1..M-1$
- In this case, minimizing $J(W)$ w.r.t. output weights $w_{ij}^{\langle M\rangle}$ is a linear problem, and its solution is the pseudo-inverse

$$W^{\langle M\rangle} = \underset{W}{\operatorname{argmin}} \left\{ \frac{1}{2} \sum_{n=1}^{N} \sum_{k=1}^{N_O} \left(y_k^{\langle M\rangle (n} - t_k^{(n} \right)^2 \right\}$$

$$= \underset{W}{\operatorname{argmin}} \left\{ \frac{1}{2} \left\| Y^{\langle M-1\rangle} W^{\langle M\rangle} - T \right\|^2 \right\}$$

$$\Rightarrow W^{\langle M\rangle} = \left(Y^{\langle M-1\rangle'} Y^{\langle M-1\rangle} \right)^{-1} Y^{\langle M-1\rangle'} T = Y^{\langle M-1\rangle^{\dagger}} T$$

- $W^{\langle M\rangle}$ denotes the weights of the (output) linear layer and
- $Y^{\langle M-1\rangle}$ denotes the outputs of the last hidden layer and
- T denotes the targets

[Bishop, 1995]

- It can be shown that the role of the output biases is to compensate for the difference between the averages of the targets and that of the hidden unit outputs

$$w_{0k}^{\langle M \rangle} = E[t_k] - \sum_{j=1}^{N_{H_{M-1}}} w_{jk}^{\langle M \rangle} E\left[y_j^{\langle M-1 \rangle}\right]$$

  - Knowing that the output biases can be computed in this manner, we will assume for convenience that both T and $Y^{\langle M-1 \rangle}$ are zero-mean
    - As we will see, this will allow us to interpret certain terms as scatter matrices
- Plugging the pseudo-inverse into the objective function we obtain

$$J(W) = \frac{1}{2}\left\|Y^{\langle M-1 \rangle} W^{\langle M \rangle} - T\right\|^2 = \frac{1}{2}\left\|Y^{\langle M-1 \rangle} Y^{\langle M-1 \rangle^{\dagger}} T - T\right\|^2$$

- And realizing that the SSE is the sum of ONLY the diagonal terms in $\|\cdot\|^2$

$$J(W) = \frac{1}{2} Tr\left\{\left(Y^{\langle M-1 \rangle} Y^{\langle M-1 \rangle^{\dagger}} T - T\right)\left(Y^{\langle M-1 \rangle} Y^{\langle M-1 \rangle^{\dagger}} T - T\right)'\right\}$$

- which, after some manipulation [Bishop, 1995], can be expressed as

$$J(W) = \frac{1}{2} Tr\{T'T - S_B S_{TOT}^{-1}\} \; where \begin{cases} S_{TOT} = Y^{\langle M \rangle'} Y^{\langle M \rangle} \\ S_B = Y^{\langle M \rangle'} T T' Y^{\langle M \rangle'} \end{cases}$$

- Since $T'T$ is independent of W, minimizing $J(W)$ is equivalent to maximizing $J'(W)$

$$J'(W) = \frac{1}{2} Tr\,[S_B S_{TOT}^{-1}]$$

- Since we are using 1-of-C encoding in the output layer, it can be shown that S_B becomes

$$S_B = \sum_{k=1}^{C} N_k^2 \left(\bar{y}_k{}^{\langle M-1\rangle} - \bar{y}^{\langle M-1\rangle}\right)\left(\bar{y}_k{}^{\langle M-1\rangle} - \bar{y}^{\langle M-1\rangle}\right)'$$

$$with \;\; \begin{cases} \bar{y}_k{}^{\langle M-1\rangle} = E\left[y_k{}^{\langle M-1\rangle}\right]_{x\in\omega_k} \\ \bar{y}^{\langle M-1\rangle} = E\left[y^{\langle M-1\rangle}\right]_{\forall x} \end{cases}$$

  - where $E\left[y_k{}^{\langle M-1\rangle}\right]_{x\in\omega_k}$ is the mean activation vector at the last hidden layer for all examples of class ω_k, and $E\left[y^{\langle M-1\rangle}\right]_{\forall x}$ is the mean activation vector regardless of class

- Notice that this S_B only differs from the conventional between-class covariance matrix by having N_k^2 instead of N_k
  - This means that this scatter matrix will have a strong bias in favor of classes that have a large number of examples
- and S_{TOT} is the total covariance matrix at the output of the final hidden layer

Conclusion

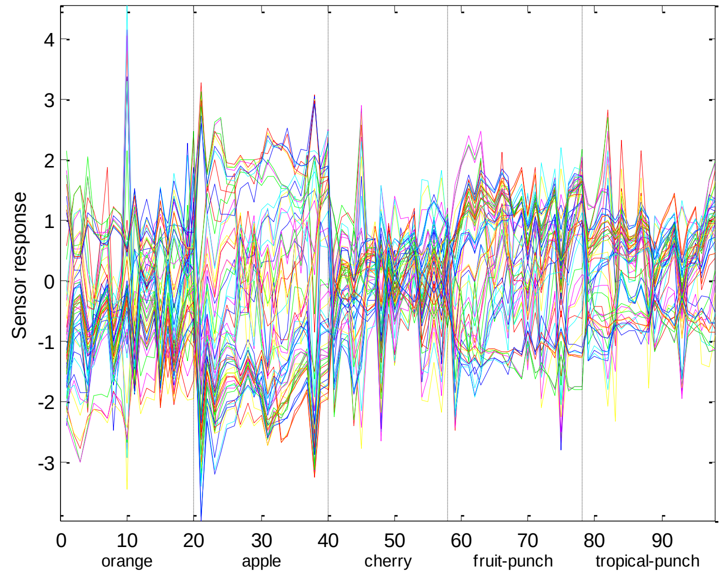
Minimization of the sum squared error between the desired targets and the outputs of an MLP with linear output neurons forces the hidden layer(s) to perform a non-linear transformation of the inputs that maximizes the discriminant function $\boldsymbol{Tr}\left[\boldsymbol{S_B S_{TOT}^{-1}}\right]$ measured at the output of the (last) hidden layer

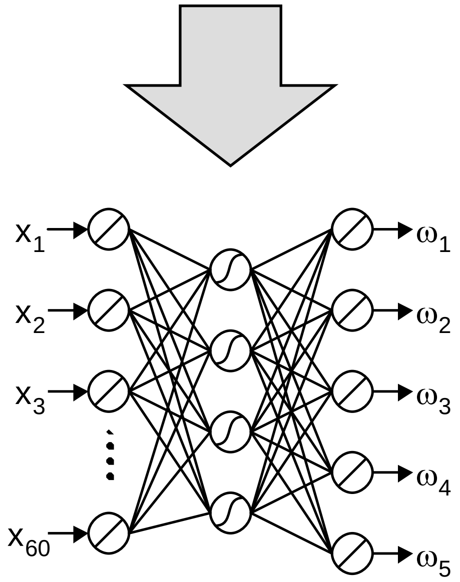
- In other words, the hidden layers of an MLP perform a non-linear version of Fisher's discriminant analysis (L10)
- <u>This is precisely why MLPs have been demonstrated to perform so well in pattern classification tasks</u>

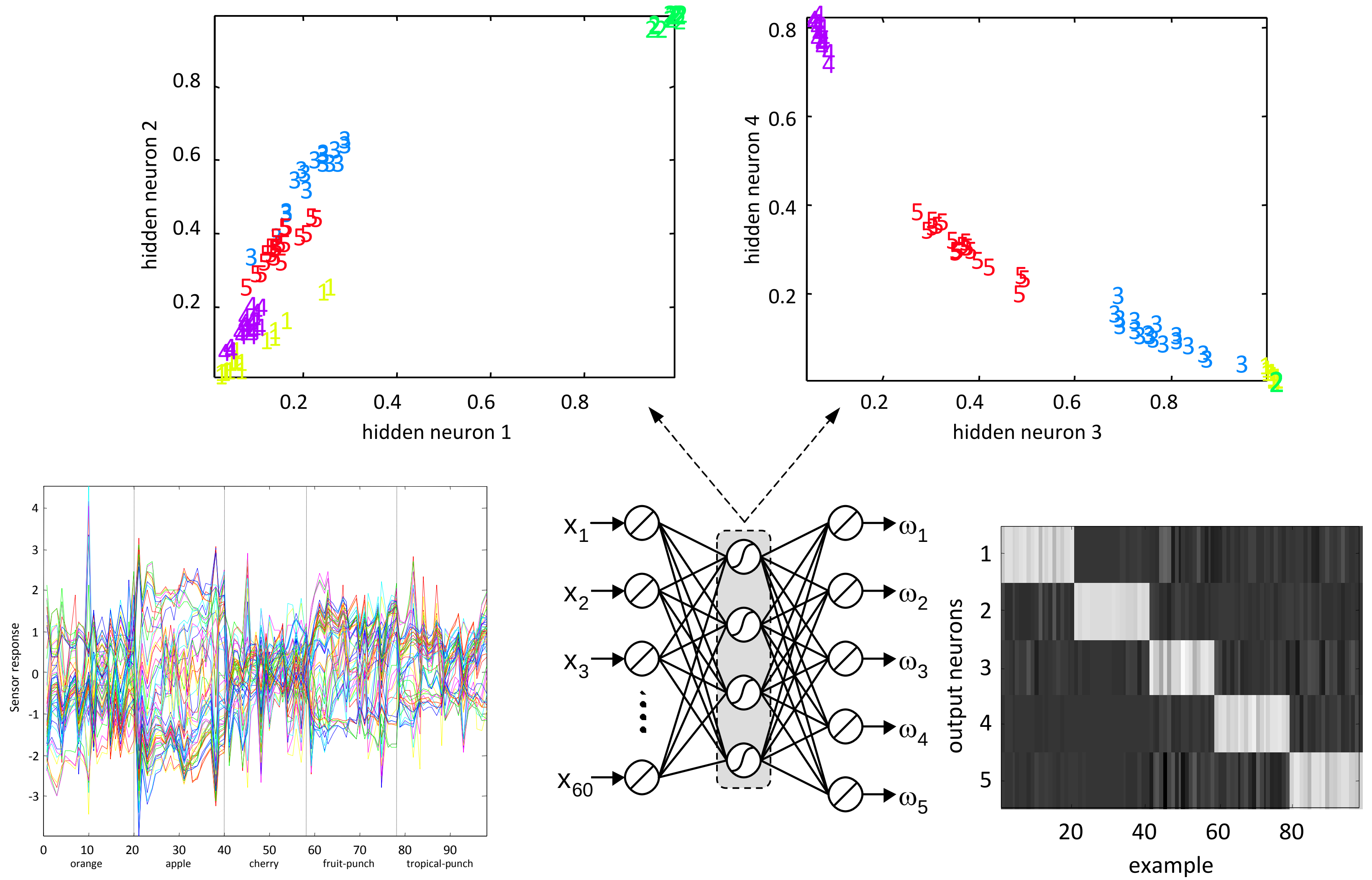
An example

Train an MLP to classify five odors with a 60-sensor array

- MLP has 60 inputs, one per sensor
- There are 5 outputs, one per odor
  - Output neurons use 1-of-C encoding
  - Output layer has linear activation function
- Four hidden neurons are used (as many as LDA projections)
  - Hidden layer has the logistic sigmoidal activation function
- Training
  - Hidden weights and biases trained with steepest descent rule
  - Output weights and biases trained with the pseudo-inverse rule






hidden neuron 2
0.2
0.4
0.6
0.8
hidden neuron 1
hidden neuron 4
hidden neuron 3
Sensor response
0
10
20
30
40
50
60
70
80
90
orange
apple
cherry
fruit-punch
tropical-punch
x_1
x_2
x_3
x_{60}
ω_1
ω_2
ω_3
ω_4
ω_5
output neurons
example

Bayes discriminants and RBFs

Recall from previous lectures that the posterior can be expressed as

$$p(\omega_k|x) = \frac{p(x|\omega_k)p(\omega_k)}{\sum_{k'=1}^{C} p(x|\omega_{k'})p(\omega_{k'})}$$

– If we model each class with a single kernel $p(x|\omega_k)$, this can be viewed as a simple network with normalized basis functions given by

$$\varphi_k(x) = \frac{p(x|\omega_k)}{\sum_{k'=1}^{C} p(x|\omega_{k'})p(\omega_{k'})}$$

– and hidden-to-output weights defined by

$$w_k = p(\omega_k)$$

– This provides a very simple interpretation of RBFs as Bayesian classifiers and vice versa

In some situations, however, a single kernel per class may not be sufficient to model the input space density

- Let's then use M different basis functions, labeled by index j
  - The class conditional density is then given by

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

- and the unconditional density is given by

$$p(x) = \sum_{k=1}^{C} p(x|\omega_k)p(\omega_k) = \sum_{j=1}^{M} p(x|j)p(j)$$

- where the priors $p(j)$ can be defined by

$$p(j) = \sum_{k=1}^{C} p(j|\omega_k)p(\omega_k)$$

- With these expressions in mind, the posterior probabilities become

$$p(\omega_k|x) = \frac{p(x|\omega_k)p(\omega_k)}{p(x)} = \frac{\sum_{j=1}^{M} p(x|j)p(j|\omega_k)p(\omega_k)}{\sum_{j'=1}^{M} p(x|j')p(j')}$$

- Moving the denom. inside the sum, and adding the term $p(j)/p(j) = 1$

$$p(\omega_k|x) = \sum_{j=1}^{M} \frac{p(x|j)p(j|\omega_k)p(\omega_k)}{\sum_{j'=1}^{M} p(x|j')p(j')} \frac{p(j)}{p(j)}$$

- This operation allows us to regroup the expression as

$$p(\omega_k|x) = \sum_{j=1}^{M} w_{jk}\varphi_j(x)$$

- where

$$\varphi_j(x) = \frac{p(x|j)p(j)}{\sum_{j'=1}^{M} p(x|j')p(j')} = p(j|x) \quad \text{and} \quad w_{jk} = \frac{p(j|\omega_k)p(\omega_k)}{p(j)} = p(\omega_k|j)$$

INTERPRETATION

- The activation of a basis function can be interpreted as the posterior probability of the presence of its prototype pattern in the input space
- H-O weights can be interpreted as the posterior probabilities of the classes given the prototype pattern of the basis function, and
  - The output of the RBF can also be interpreted as the posterior probability of class membership

Comparison between MLPs and RBFs

Similarities

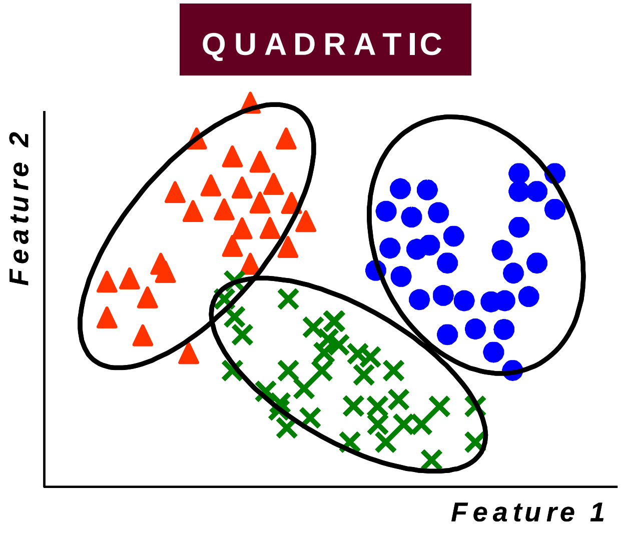
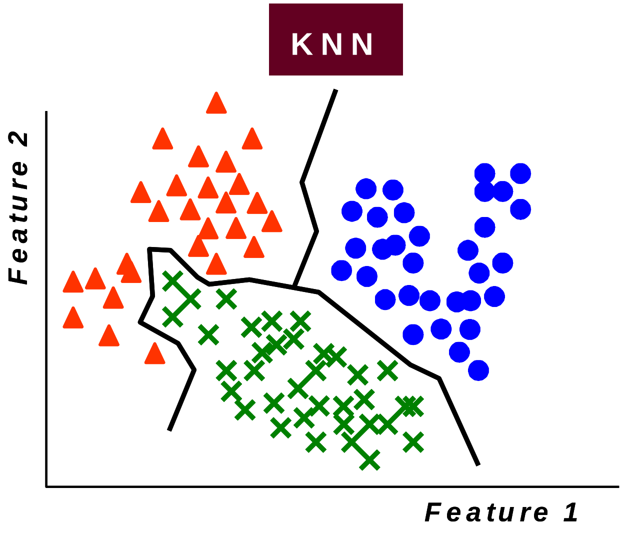
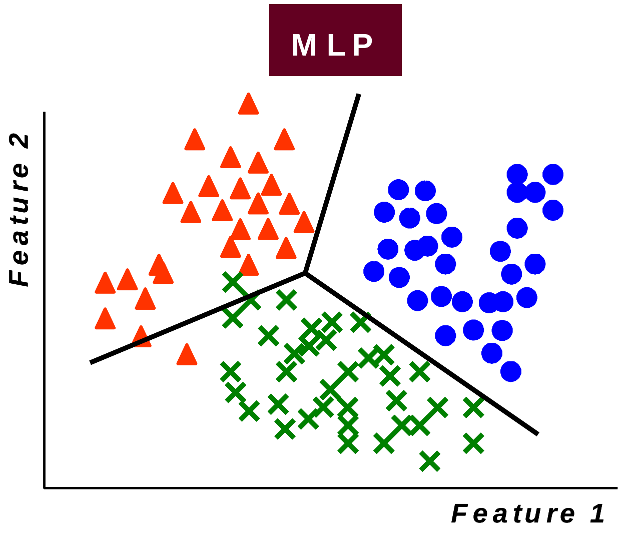
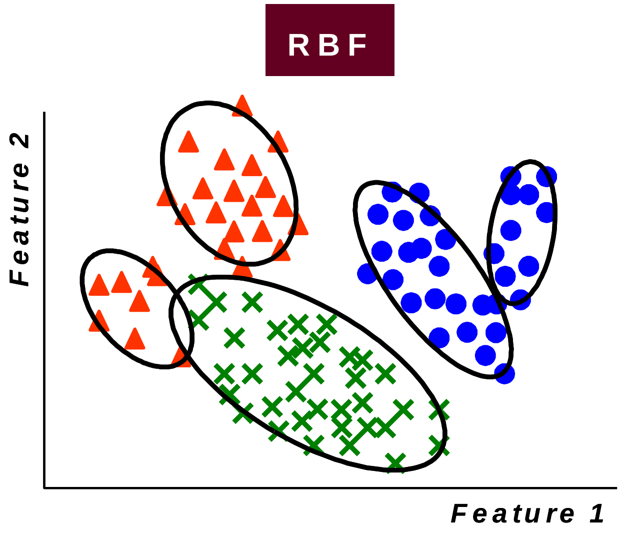
- Either model can function as an “universal approximator”
  - *MLPs and RBFs can approximate any functional continuous mapping with arbitrary accuracy, provided that the number of hidden units is sufficiently large*

Differences

- MLPs perform a global and distributed approximation of the target function, whereas RBF perform a local approximation
- MLP partition feature space with hyper-planes; RBF decision boundaries are hyper-ellipsoids
- The distributed representation of MLPs causes the error surface to have multiple local minima and nearly flat regions with very slow convergence. As a result training times for MLPs are usually larger than those for RBFs
- MLPs generalizes better than RBFs in regions of feature space outside of the local neighborhoods defined by the training set. On the other hand, extrapolation far from training data is oftentimes unjustified and dangerous
- MLPs typically require fewer parameters than RBFs to approximate a non-linear function with the same accuracy

Differences (cont.)

- All the parameters in an MLP are trained simultaneously; parameters in the hidden and output layers of an RBF network are typically trained separately using an efficient, faster hybrid algorithm
- MLPs may have multiple hidden layers with complex connectivity, whereas RBFs typically have only one hidden layer and full connectivity
- The hidden neurons of an MLP compute the inner product between an input vector and their weight vector; RBFs compute the Euclidean distance between an input vector and the radial basis centers
- The hidden layer of an RBF is non-linear, whereas the output layer is linear. In an MLP classifier, all layers are typically non-linear. Only when an MLP is used for non-linear regression, the output layer is typically linear


QUADRATIC
Feature 2
Feature 1

KNN
Feature 2
Feature 1

MLP
Feature 2
Feature 1

RBF
Feature 2
Feature 1