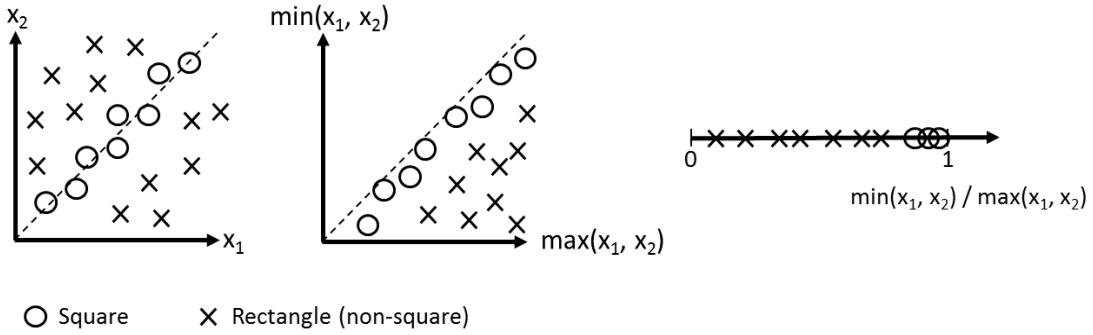
Solutions to the exam in Neural Networks and Learning Systems - TBMI26 Exam 2012-03-08

Part 1

1.
   - k-Nearest Neighbors - (S)
   - Support Vector Machines - (S)
   - AdaBoost - (S)
   - Principal Component Analysis (U)
   - Multi-layer Perceptron (Neural Network) (S)
   - Mixture of Gaussian Clustering (U)
2. A learning method generalizes well if it performs well on previously unseen data, for example, that it can generalize what it has learned on training data to new data.
3. The cost function $\sum_{k=1}^{N} I(z_i \neq y_i)$ is piecewise flat and not differentiable at all points, so that we do not get any information from the gradient, which is required in gradient descent.
4. For example, if a 1-point cross-over takes place after the second letter 'AB|CBC' and 'BB|CAC', the children will be 'ABCAC' and 'BBCBC'.
5. k-NN: k is the number of the stored data vectors that vote for the classification decision. k-Means: k is the number of clusters.
6. The perceptron models the neuron using the function $f(\mathbf{x}; \mathbf{w}) = sign(\mathbf{w}^T\mathbf{x})$, where $\mathbf{x}$ is the input at the dendrites which is weighted by $\mathbf{w}$ and summed to model the output at the axon.
7. Expectation Maximization
8. If the covariance matrix is diagonal, the features are uncorrelated and are therefore likely to carry independent information (but there could be non-linear dependencies!).
9. The discount factor controls the trade-off between optimizing for immediate rewards and long term rewards.
10. The maximum margin principle says that the decision boundary between two classes should be placed so that the distance between the boundary and the training data should be as large as possible, i.e., the error-margin should be maximized.

Part 2

11. The figure shows three different ways this can be done.

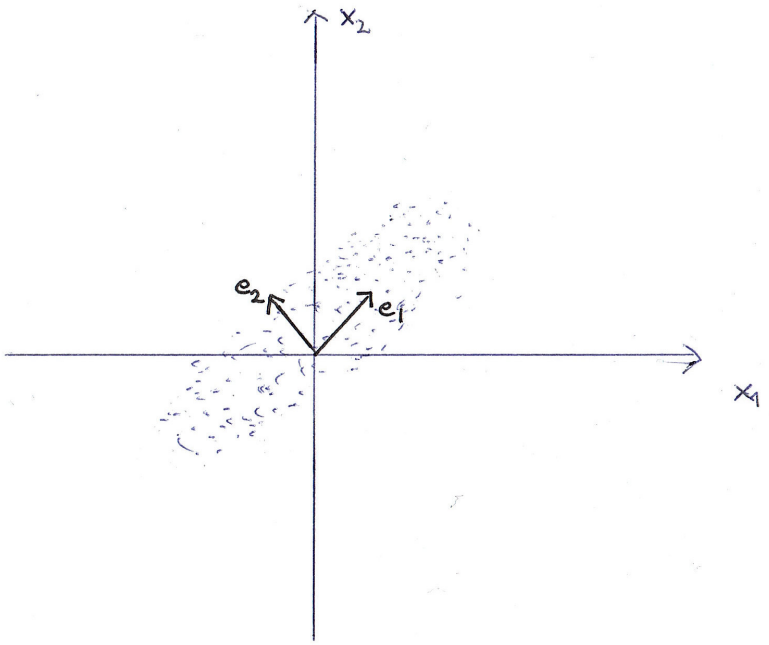


- One can keep x_1 and x_2 as they are as features. The squares will the be located along the diagonal line because $x_1 \approx x_2$, while the non-square rectangles will be located away from the diagonal in the feature space. A non-linear classifier is required for this case, e.g., a neural network, SVM, ...
- One can create two new features $y_1 = \max(x_1, x_2)$ and $y_2 = \min(x_1, x_2)$. The feature space will then look like the middle plot, and a linear classifier will be enough.
- Another option is to calculate a new 1D-feature as the ratio $y = \frac{\min(x_1,x_2)}{\max(x_1,x_2)}$, shown in the right figure. Again, a simple linear classifier will do (i.e., a single threshold in this case).

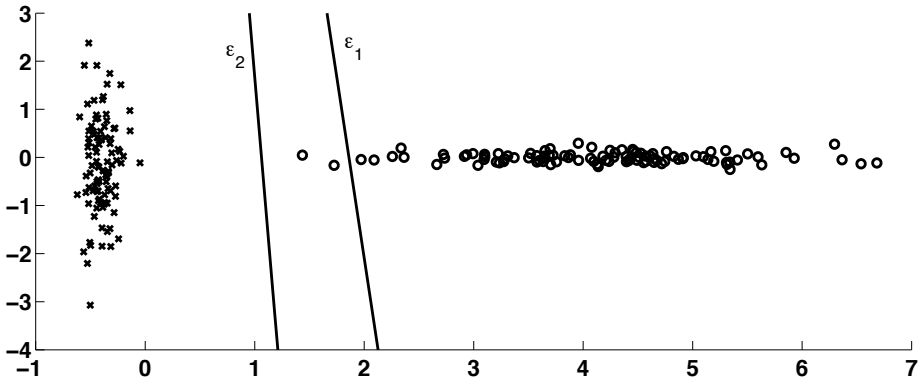
12.
- $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \langle \boldsymbol{\varphi}(\mathbf{x}_i), \boldsymbol{\varphi}(\mathbf{x}_j) \rangle$, i.e. it implicitly calculates the standard scalar product in feature space, expressed in the input space.
- $\kappa(\mathbf{x}, \mathbf{y}) = \varphi(\mathbf{x})^T \varphi(\mathbf{y}) = x_1^2 y_1^2 + x_2^2 y_2^2 + 2x_1 x_2 y_1 y_2 = (x_1 y_1 + x_2 y_2)^2 = \left(\mathbf{x^T y}\right)^2$

13. Divide the data set into equal parts P1, P2 and P3 with 300 samples each. Then train and evaluate 3 times as follows:
Train using P1 and P2 and evaluate using P3.
Train using P1 and P3 and evaluate using P2.
Train using P2 and P3 and evaluate using P1.
The total performance is the average classification accuracy of the 3 runs.

14. One example on a distribution that gives these eigenvalues are an elliptic symmetric distribution that is centered around origin where the intersections between the level curves of the distributions and one of the axes are a factor $\sqrt{5/2}$ longer away from origin than the intersections between the other axis. Observe that the eigenvalues correspond to the variance, and that a scaling of the samples affect the variance quadratically.



15. The problem here is that we have two linearly separable classes whom can be classified using a simple threshold on the horizontal axis. They do however have different distributions along this axis, a factor that can confuse our training if we are not careful. The first error function ϵ_1 is sensitive to the distance of the samples. The effect is that the right most cluster will have a higher impact in the total error, and the boundary will shift closer to the right class. This is somewhat fixed with ϵ_2 were tanh quickly saturates to about ± 1, thus limiting the effect of the different distributions. The figure below shows the result after training a single neuron using the two functions. Here we can see that the theory holds true.



Part 3

16. a) In figure 1 the classification problem is sketched along with the initial weights. All vertical/horizontal decision stumps within the square of the data samples give the error $\epsilon = \frac{1}{4}$. In figure 1 we have chosen one of these lines as an solution example. We get $\alpha_1 = \frac{1}{2}\ln\frac{1-\epsilon_1}{\epsilon_1} = \frac{1}{2}\ln\frac{3/4}{1/4} = \frac{\ln 3}{2}$. Now we update (decreasing) the weights of the correctly classified samples with $e^{-\alpha_1} = \frac{1}{\sqrt{3}}$. The weight of the erroneous classified sample (upper right in the solution example) is instead increased with $e^{\alpha_1} = \sqrt{3}$. The sum of the weights are now $3\frac{1}{4\sqrt{3}} + \frac{\sqrt{3}}{4} = \frac{\sqrt{3}}{2}$. After normalizing the weights (dividing with the sum) we get the resulting weights as shown in the right part of figure 1. As we can see, AdaBoost works well here.

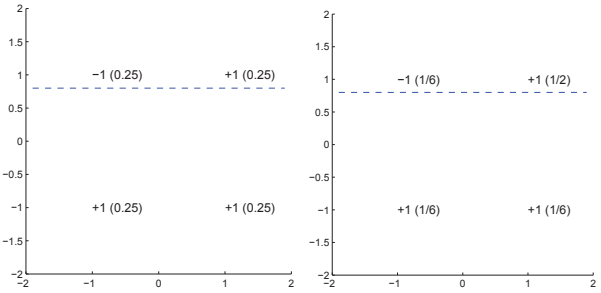


Figure 1: Classification problem 16a, weights before/after first iteration

b) In figure 2 the classification problem is sketched along with the initial weights. All vertical/horizontal decision stumps within the square of the data samples give the error $\epsilon = \frac{1}{2}$. In figure 2 we have chosen one of these lines as an solution example. We get $\alpha_1 = \frac{1}{2}\ln\frac{1-\epsilon_1}{\epsilon_1} = \frac{1}{2}\ln\frac{1/2}{1/2} = \frac{\ln 1}{2} = 0$. If we try to update the weights, we get $e^{\pm\alpha_1} = 1$, i.e. the weights don't change. This means that the weak classifier in iteration two will face the same problem as in iteration 1. As we can see, AdaBoost does not work in this setting. One solution is to allow other types of weak classifiers.

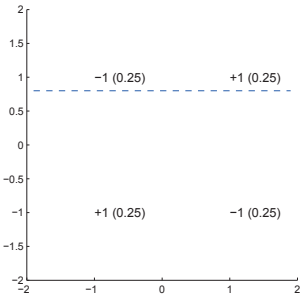


Figure 2: Classification problem 16b

c) Outliers gain weight exponentially which will eventually result in bad weak classifiers. This may ruin the composite strong classifier. Solutions to this problem include various forms of weight trimming and alternative weight update approaches.

17. A)

The problem is not linearly separable so we need at least two layers in the design. The activation function σ shall exist after every node. If the bias is missing in figure or not mentioned in the text 0.5 points are withdrawn.

B)

We use a network design with a output layer and one hidden layer in our calculations. The bias has been added as an extra feature to the input and as an additional input to the output layer. We denote the output of the hidden layer $\mathbf{u}$ whom is calculated using the weights $\mathbf{W}$, input $\mathbf{x}$ and activation function tanh. Thus we get $\mathbf{u} = \tanh(\mathbf{s}) = \tanh(\mathbf{Wx})$. Since the output layer uses the same activation function we get $\mathbf{y} = \tanh(\mathbf{t}) = \tanh(\mathbf{Vu})$, where $\mathbf{V}$ stores the weights for the output neurons. Output neurons are idexed using i, hidden using j and input features k.

To train the output layer we want to take a step $\Delta V_{ij} = -\eta \frac{\partial \epsilon}{\partial V_{ij}}$ were:
$\frac{\partial \epsilon}{\partial V_{ij}} = \frac{\partial \epsilon}{\partial y_i} \frac{\partial y_i}{\partial t_i} \frac{\partial t_i}{\partial V_{ij}} = -2(d_i - y_i)(1 - {y_i}^2)u_j$

To update the hidden layer we take a step $\Delta W_{jk} = -\eta \frac{\partial \epsilon}{\partial W_{jk}}$ were:
$\frac{\partial \epsilon}{\partial W_{jk}} = \sum_i \frac{\partial \epsilon}{\partial y_i} \frac{\partial y_i}{\partial t_i} \frac{\partial t_i}{\partial u_j} \frac{\partial u_j}{\partial s_j} \frac{\partial s_j}{\partial W_{jk}} = \sum_i -2(d_i - y_i)(1 - {y_i}^2)V_{ij}(1 - u_j^2)x_k$

18. Initially, we have the dataset and the two prototypes (k=2) as presented in figure 3.

a) k-Means has two alternating phases, *sample assignment* and prototype update. During the first phase, we assign each data sample to the closest (Euclidean) prototype. In the second phase, we update the prototypes using the assigned samples: $\mathbf{p}_j = \frac{1}{|S_j|}\sum_{k\in S_j}\mathbf{x}_k$, where S_j is the set of data samples belonging to prototype j. In the first iteration, the left hand samples are closest to $\mathbf{p}_1$ and the right hand samples are closest to $\mathbf{p}_2$. We then update the prototypes to $\mathbf{p}_1 = \begin{bmatrix}-1\\0\end{bmatrix}$ and $\mathbf{p}_2 = \begin{bmatrix}+1\\0\end{bmatrix}$. During the second iteration, the assigned sets do not change. This implies no change in the prototypes either. The k-Means method has converged.

b) Mixture of Gaussians (MoG) has the same alternating phases, *sample assignment* and *prototype update*. The 'closeness' measure during the first phase is, however, here replaced with finding the Gaussian prototype which gives the highest likelihood for each sample $\mathbf{x}_i$, i.e. finding the prototype j which maximizes

$$p(\mathbf{x}_i;\mathbf{m}_j,\mathbf{C}_j) = \frac{1}{(2\pi)^{N/2}\sqrt{\det\mathbf{C}_j}}\exp\left[-\frac{1}{2}(\mathbf{x}_i-\mathbf{m}_j)^T\mathbf{C}_j^{-1}(\mathbf{x}_i-\mathbf{m}_j)\right]$$

Since $\mathbf{C}_1 = \mathbf{C}_2$ and they are isotropic ($= \mathbf{I}$), it is enough to compare the euclidean distance from the sample to the prototype, i.e. we have the same situation as in a): the left hand samples are closest to $\mathbf{p}_1$ and the right hand samples are closest to $\mathbf{p}_2$. We then update the prototypes to $\mathbf{p}_1 = \begin{bmatrix}-1\\0\end{bmatrix}$ and $\mathbf{p}_2 = \begin{bmatrix}+1\\0\end{bmatrix}$. The covariance matrices are updated as $\mathbf{C}_j = \frac{1}{|S_j|}\sum_{k\in S_j}(\mathbf{x}_k-\mathbf{p}_j)(\mathbf{x}_k-\mathbf{p}_j)^T$, where S_j is the set of data samples assigned to prototype j as before. We get $\mathbf{C}_1 = \mathbf{C}_2 = \frac{1}{3}\begin{bmatrix}0&0\\0&2\end{bmatrix}$. This corresponds to Gaussians which have no extension/width/variance in the first dimension, a variance of $\frac{2}{3}$ in the second dimension and no covariance between the dimensions $\rightarrow$ 'a very thin vertical ellipse' as a general shape.

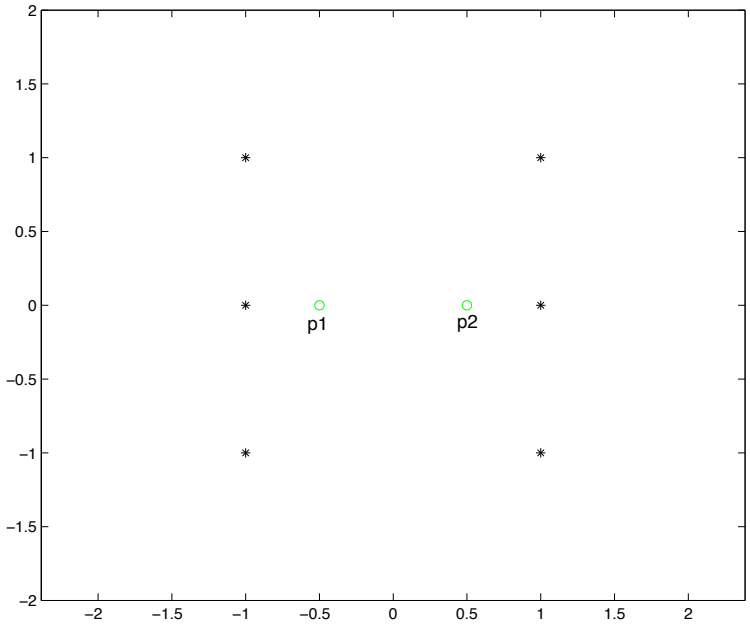


Figure 3: Initial data set and prototypes (problem 18)

19. The formula for the optimal Q-function is given by:

$$\begin{aligned} Q^*(x,a) &= r(x,a) + \gamma V^*(g(x,\mu^*(x))) \\ &= r(x,a) + \gamma \max_b Q^*(g(x,\mu^*(x)),b) \end{aligned}$$

A)

All paths in the system are of length 3. Thus the optimal path become $1 \to 2, 2 \to 5, 5 \to 6$, independent of γ.

B)

Counting backwards we get:

$$\begin{aligned} V^*(6) &= 0 \\ V^*(3) = Q^*(3,right) &= 0.5 + \gamma V^*(6) = 0.5 \\ V^*(5) = Q^*(5,up) &= 1.5 + \gamma V^*(6) = 1.5 \\ Q^*(2,up) &= 0.5 + \gamma V^*(3) = 0.5(1+\gamma) \\ Q^*(2,right) &= 1.5 + \gamma V^*(5) = 1.5(1+\gamma) \\ V^*(2) &= 1.5(1+\gamma) \\ V^*(4) = Q^*(4,up) &= 0.5 + \gamma V^*(5) = 0.5 + 1.5\gamma \\ Q^*(1,up) &= 1.5 + \gamma V^*(2) = 1.5(1+\gamma+\gamma^2) \\ Q^*(1,right) &= 0.5 + \gamma V^*(4) = 0.5 + 0.5\gamma + 1.5\gamma^2 \\ V^*(1) &= 1.5(1+\gamma+\gamma^2) \end{aligned}$$

C)

The sequence corresponds to how we calculated the optimal policy in B. And since the learning rate $\alpha = 1$ all we need to do is to enter $\gamma = 0.9$ when we calculate Q(x,a).

$$\begin{aligned} Q^*(3,right) &= 0.5 + 0.9V^*(6) = 0.5 \\ Q^*(5,up) &= 1.5 + 0.9V^*(6) = 1.5 \\ Q^*(2,up) &= 0.5 + 0.9V^*(3) = 0.5(1+0.9) = 0.95 \\ Q^*(2,right) &= 1.5 + 0.9V^*(5) = 1.5(1+0.9) = 2.85 \\ Q^*(4,up) &= 0.5 + 0.9V^*(5) = 0.5 + 1.5 \cdot 0.9 = 1.85 \\ Q^*(1,up) &= 1.5 + 0.9V^*(2) = 1.5(1+0.9+0.9^2) = 4.065 \\ Q^*(1,right) &= 0.5 + 0.9V^*(4) = 0.5 + 0.5 \cdot 0.9 + 1.5 \cdot 0.9^2 = 2.165 \end{aligned}$$