Copyright Cambridge University Press 2003. On-screen viewing permitted. Printing not permitted. http://www.cambridge.org/0521642981
You can buy this book for 30 pounds or $50. See http://www.inference.phy.cam.ac.uk/mackay/itila/ for links.

44

Supervised Learning in Multilayer Networks

▶ 44.1 Multilayer perceptrons

No course on neural networks could be complete without a discussion of supervised multilayer networks, also known as backpropagation networks.

The *multilayer perceptron* is a feedforward network. It has input neurons, hidden neurons and output neurons. The hidden neurons may be arranged in a sequence of layers. The most common multilayer perceptrons have a single hidden layer, and are known as 'two-layer' networks, the number 'two' counting the number of layers of neurons not including the inputs.

Such a feedforward network defines a nonlinear parameterized mapping from an input $\mathbf{x}$ to an output $\mathbf{y} = \mathbf{y}(\mathbf{x}; \mathbf{w}, \mathcal{A})$. The output is a continuous function of the input and of the parameters $\mathbf{w}$; the architecture of the net, i.e., the functional form of the mapping, is denoted by $\mathcal{A}$. Feedforward networks can be 'trained' to perform regression and classification tasks.

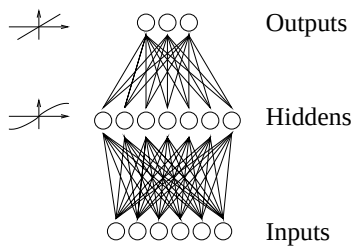

Figure 44.1. A typical two-layer network, with six inputs, seven hidden units, and three outputs. Each line represents one weight.

Regression networks

In the case of a regression problem, the mapping for a network with one hidden layer may have the form:

$$\text{Hidden layer:} \quad a_j^{(1)} = \sum_l w_{jl}^{(1)} x_l + \theta_j^{(1)}; \quad h_j = f^{(1)}(a_j^{(1)}) \tag{44.1}$$

$$\text{Output layer:} \quad a_i^{(2)} = \sum_j w_{ij}^{(2)} h_j + \theta_i^{(2)}; \quad y_i = f^{(2)}(a_i^{(2)}) \tag{44.2}$$

where, for example, $f^{(1)}(a) = \tanh(a)$, and $f^{(2)}(a) = a$. Here l runs over the inputs $x_1, \ldots, x_L$, j runs over the hidden units, and i runs over the outputs. The 'weights' w and 'biases' θ together make up the parameter vector $\mathbf{w}$. The nonlinear sigmoid function $f^{(1)}$ at the hidden layer gives the neural network greater computational flexibility than a standard linear regression model. Graphically, we can represent the neural network as a set of layers of connected neurons (figure 44.1).

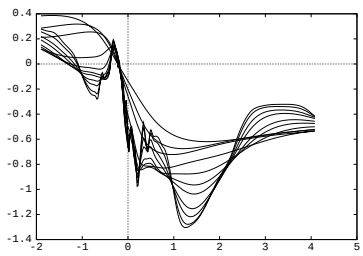

Figure 44.2. Samples from the prior over functions of a one-input network. For each of a sequence of values of $\sigma_{\text{bias}} = 8, 6, 4, 3, 2, 1.6, 1.2, 0.8, 0.4, 0.3, 0.2$, and $\sigma_{\text{in}} = 5\sigma_{\text{bias}}^{w}$, one random function is shown. The other hyperparameters of the network were $H = 400$, $\sigma_{\text{out}}^{w} = 0.05$.

What sorts of functions can these networks implement?

Just as we explored the weight space of the single neuron in Chapter 39, examining the functions it could produce, let us explore the weight space of a multilayer network. In figures 44.2 and 44.3 I take a network with one input and one output and a large number H of hidden units, set the biases

Copyright Cambridge University Press 2003. On-screen viewing permitted. Printing not permitted. http://www.cambridge.org/0521642981
You can buy this book for 30 pounds or $50. See http://www.inference.phy.cam.ac.uk/mackay/itila/ for links.

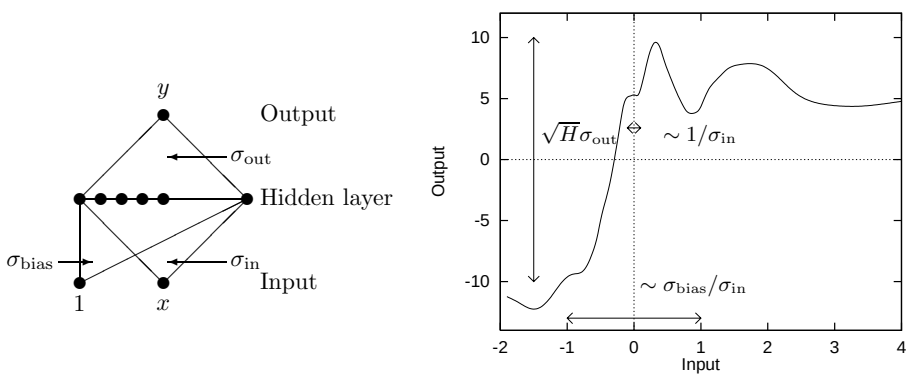

Figure 44.3. Properties of a function produced by a random network. The vertical scale of a typical function produced by the network with random weights is of order $\sqrt{H}\sigma_{\rm out}$; the horizontal range in which the function varies significantly is of order $\sigma_{\rm bias}/\sigma_{\rm in}$; and the shortest horizontal length scale is of order $1/\sigma_{\rm in}$. The function shown was produced by making a random network with $H = 400$ hidden units, and Gaussian weights with $\sigma_{\rm bias} = 4$, $\sigma_{\rm in} = 8$, and $\sigma_{\rm out} = 0.5$.

and weights $\theta_j^{(1)}$, $w_{jl}^{(1)}$, $\theta_i^{(2)}$ and $w_{ij}^{(2)}$ to random values, and plot the resulting function $y(x)$. I set the hidden units' biases $\theta_j^{(1)}$ to random values from a Gaussian with zero mean and standard deviation $\sigma_{\rm bias}$; the input-to-hidden weights $w_{jl}^{(1)}$ to random values with standard deviation $\sigma_{\rm in}$; and the bias and output weights $\theta_i^{(2)}$ and $w_{ij}^{(2)}$ to random values with standard deviation $\sigma_{\rm out}$.

The sort of functions that we obtain depend on the values of $\sigma_{\rm bias}$, $\sigma_{\rm in}$ and $\sigma_{\rm out}$. As the weights and biases are made bigger we obtain more complex functions with more features and a greater sensitivity to the input variable. The vertical scale of a typical function produced by the network with random weights is of order $\sqrt{H}\sigma_{\rm out}$; the horizontal range in which the function varies significantly is of order $\sigma_{\rm bias}/\sigma_{\rm in}$; and the shortest horizontal length scale is of order $1/\sigma_{\rm in}$.

Radford Neal (1996) has also shown that in the limit as $H \to \infty$ the statistical properties of the functions generated by randomizing the weights are independent of the number of hidden units; so, interestingly, the complexity of the functions becomes independent of the number of parameters in the model. What determines the complexity of the typical functions is the characteristic magnitude of the weights. Thus we anticipate that when we fit these models to real data, an important way of controlling the complexity of the fitted function will be to control the characteristic magnitude of the weights.

Figure 44.4 shows one typical function produced by a network with two inputs and one output. This should be contrasted with the function produced by a traditional linear regression model, which is a flat plane. Neural networks can create functions with more complexity than a linear regression.

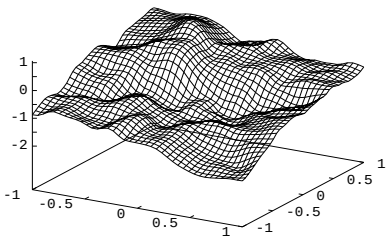

Figure 44.4. One sample from the prior of a two-input network with $\{H, \sigma_{\rm in}^w, \sigma_{\rm bias}^w, \sigma_{\rm out}^w\} = \{400, 8.0, 8.0, 0.05\}$.

▶ 44.2 How a regression network is traditionally trained

This network is trained using a data set $D = \{\mathbf{x}^{(n)}, \mathbf{t}^{(n)}\}$ by adjusting $\mathbf{w}$ so as to minimize an error function, e.g.,

$$E_D(\mathbf{w}) = \frac{1}{2} \sum_n \sum_i \left(t_i^{(n)} - y_i(\mathbf{x}^{(n)}; \mathbf{w}) \right)^2 . \tag{44.3}$$

This objective function is a sum of terms, one for each input/target pair $\{\mathbf{x}, \mathbf{t}\}$, measuring how close the output $\mathbf{y}(\mathbf{x}; \mathbf{w})$ is to the target $\mathbf{t}$.

This minimization is based on repeated evaluation of the gradient of E_D. This gradient can be efficiently computed using the *backpropagation* algorithm (Rumelhart *et al.*, 1986), which uses the chain rule to find the derivatives.

Copyright Cambridge University Press 2003. On-screen viewing permitted. Printing not permitted. http://www.cambridge.org/0521642981
You can buy this book for 30 pounds or $50. See http://www.inference.phy.cam.ac.uk/mackay/itila/ for links.

Often, regularization (also known as weight decay) is included, modifying the objective function to:

$$M(\mathbf{w}) = \beta E_D + \alpha E_W \tag{44.4}$$

where, for example, $E_W = \frac{1}{2}\sum_i w_i^2$. This additional term favours small values of $\mathbf{w}$ and decreases the tendency of a model to overfit noise in the training data.

Rumelhart *et al.* (1986) showed that multilayer perceptrons can be trained, by gradient descent on $M(\mathbf{w})$, to discover solutions to non-trivial problems such as deciding whether an image is symmetric or not. These networks have been successfully applied to real-world tasks as varied as pronouncing English text (Sejnowski and Rosenberg, 1987) and focussing multiple-mirror telescopes (Angel *et al.*, 1990).

▶ 44.3 Neural network learning as inference

The neural network learning process above can be given the following probabilistic interpretation. [Here we repeat and generalize the discussion of Chapter 41.]

The error function is interpreted as defining a noise model. βE_D is the negative log likelihood:

$$P(D \mid \mathbf{w}, \beta, \mathcal{H}) = \frac{1}{Z_D(\beta)} \exp(-\beta E_D). \tag{44.5}$$

Thus, the use of the sum-squared error E_D (44.3) corresponds to an assumption of Gaussian noise on the target variables, and the parameter β defines a noise level $\sigma_\nu^2 = 1/\beta$.

Similarly the regularizer is interpreted in terms of a log prior probability distribution over the parameters:

$$P(\mathbf{w} \mid \alpha, \mathcal{H}) = \frac{1}{Z_W(\alpha)} \exp(-\alpha E_W). \tag{44.6}$$

If E_W is quadratic as defined above, then the corresponding prior distribution is a Gaussian with variance $\sigma_W^2 = 1/\alpha$. The probabilistic model $\mathcal{H}$ specifies the architecture $\mathcal{A}$ of the network, the likelihood (44.5), and the prior (44.6).

The objective function $M(\mathbf{w})$ then corresponds to the *inference* of the parameters $\mathbf{w}$, given the data:

$$P(\mathbf{w} \mid D, \alpha, \beta, \mathcal{H}) = \frac{P(D \mid \mathbf{w}, \beta, \mathcal{H}) P(\mathbf{w} \mid \alpha, \mathcal{H})}{P(D \mid \alpha, \beta, \mathcal{H})} \tag{44.7}$$

$$= \frac{1}{Z_M} \exp(-M(\mathbf{w})). \tag{44.8}$$

The $\mathbf{w}$ found by (locally) minimizing $M(\mathbf{w})$ is then interpreted as the (locally) most probable parameter vector, $\mathbf{w}_{\text{MP}}$.

The interpretation of $M(\mathbf{w})$ as a log probability adds little new at this stage. But new tools will emerge when we proceed to other inferences. First, though, let us establish the probabilistic interpretation of classification networks, to which the same tools apply.

Copyright Cambridge University Press 2003. On-screen viewing permitted. Printing not permitted. http://www.cambridge.org/0521642981
You can buy this book for 30 pounds or $50. See http://www.inference.phy.cam.ac.uk/mackay/itila/ for links.

Binary classification networks

If the targets t in a data set are binary classification labels $(0, 1)$, it is natural to use a neural network whose output $y(\mathbf{x}; \mathbf{w}, \mathcal{A})$ is bounded between 0 and 1, and is interpreted as a probability $P(t=1 \,|\, \mathbf{x}, \mathbf{w}, \mathcal{A})$. For example, a network with one hidden layer could be described by the feedforward equations (44.1) and (44.2), with $f^{(2)}(a) = 1/(1+e^{-a})$. The error function βE_D is replaced by the negative log likelihood:

$$G(\mathbf{w}) = -\left[\sum_n t^{(n)} \ln y(\mathbf{x}^{(n)}; \mathbf{w}) + (1 - t^{(n)}) \ln(1 - y(\mathbf{x}^{(n)}; \mathbf{w}))\right]. \tag{44.9}$$

The total objective function is then $M = G + \alpha E_W$. Note that this includes no parameter β (because there is no Gaussian noise).

Multi-class classification networks

For a multi-class classification problem, we can represent the targets by a vector, $\mathbf{t}$, in which a single element is set to 1, indicating the correct class, and all other elements are set to 0. In this case it is appropriate to use a 'softmax' network having coupled outputs which sum to one and are interpreted as class probabilities $y_i = P(t_i=1 \,|\, \mathbf{x}, \mathbf{w}, \mathcal{A})$. The last part of equation (44.2) is replaced by:

$$y_i = \frac{e^{a_i}}{\sum_{i'} e^{a_{i'}}}. \tag{44.10}$$

The negative log likelihood in this case is

$$G(\mathbf{w}) = -\sum_n \sum_i t_i^{(n)} \ln y_i(\mathbf{x}^{(n)}; \mathbf{w}). \tag{44.11}$$

As in the case of the regression network, the minimization of the objective function $M(\mathbf{w}) = G + \alpha E_W$ corresponds to an inference of the form (44.8). A variety of useful results can be built on this interpretation.

▶ 44.4 Benefits of the Bayesian approach to supervised feedforward neural networks

From the statistical perspective, supervised neural networks are nothing more than nonlinear curve-fitting devices. Curve fitting is not a trivial task however. The effective complexity of an interpolating model is of crucial importance, as illustrated in figure 44.5. Consider a control parameter that influences the complexity of a model, for example a regularization constant α (weight decay parameter). As the control parameter is varied to increase the complexity of the model (descending from figure 44.5a–c and going from left to right across figure 44.5d), the best fit to the *training* data that the model can achieve becomes increasingly good. However, the empirical performance of the model, the *test error*, first decreases then increases again. *An over-complex model overfits the data and generalizes poorly.* This problem may also complicate the choice of architecture in a multilayer perceptron, the radius of the basis functions in a radial basis function network, and the choice of the input variables themselves in any multidimensional regression problem. Finding values for model control parameters that are appropriate for the data is therefore an important and non-trivial problem.

Copyright Cambridge University Press 2003. On-screen viewing permitted. Printing not permitted. http://www.cambridge.org/0521642981
You can buy this book for 30 pounds or $50. See http://www.inference.phy.cam.ac.uk/mackay/itila/ for links.

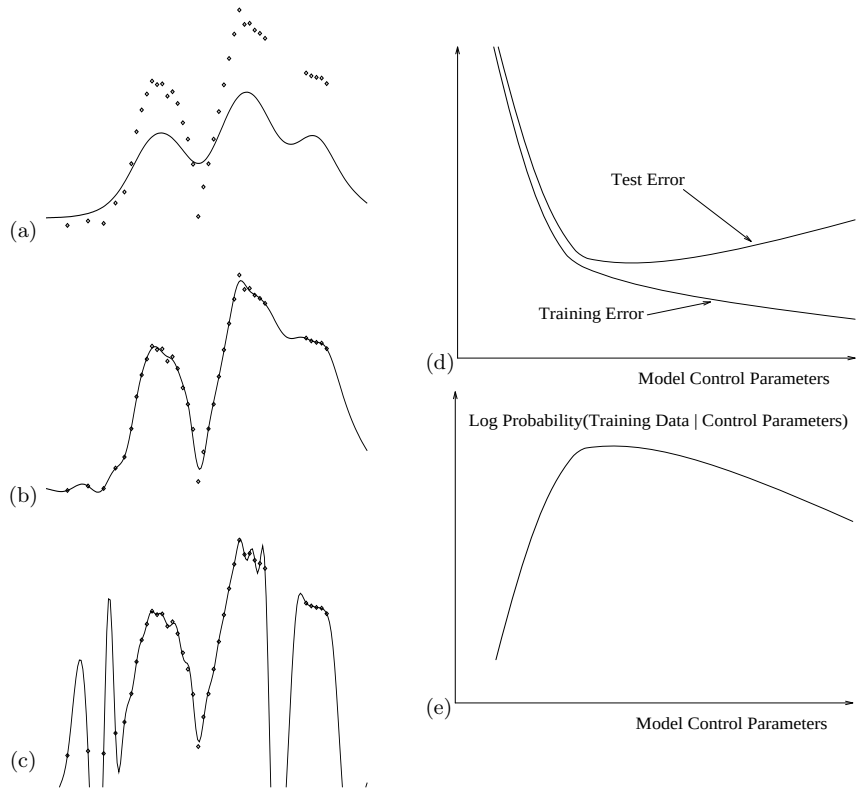

Figure 44.5. Optimization of model complexity. Panels (a–c) show a radial basis function model interpolating a simple data set with one input variable and one output variable. As the regularization constant is varied to increase the complexity of the model (from (a) to (c)), the interpolant is able to fit the training data increasingly well, but beyond a certain point the generalization ability (test error) of the model deteriorates. Probability theory allows us to optimize the control parameters without needing a test set.

The *overfitting problem* can be solved by using a Bayesian approach to control model complexity.

If we give a probabilistic interpretation to the model, then we can evaluate the evidence for alternative values of the control parameters. As was explained in Chapter 28, over-complex models turn out to be less probable, and the evidence $P(\text{Data} \mid \text{Control Parameters})$ can be used as an objective function for optimization of model control parameters (figure 44.5e). The setting of α that maximizes the evidence is displayed in figure 44.5b.

Bayesian optimization of model control parameters has four important advantages. (1) No 'test set' or 'validation set' is involved, so all available training data can be devoted to both model fitting and model comparison. (2) Regularization constants can be optimized on-line, i.e., simultaneously with the optimization of ordinary model parameters. (3) The Bayesian objective function is not noisy, in contrast to a cross-validation measure. (4) The gradient of the evidence with respect to the control parameters can be evaluated, making it possible to simultaneously optimize a large number of control parameters.

Probabilistic modelling also handles *uncertainty* in a natural manner. It offers a unique prescription, *marginalization*, for incorporating uncertainty about parameters into predictions; this procedure yields better predictions, as we saw in Chapter 41. Figure 44.6 shows error bars on the predictions of a trained neural network.

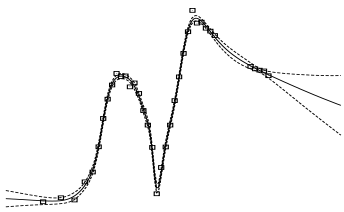

Figure 44.6. Error bars on the predictions of a trained regression network. The solid line gives the predictions of the best-fit parameters of a multilayer perceptron trained on the data points. The error bars (dotted lines) are those produced by the uncertainty of the parameters $\mathbf{w}$. Notice that the error bars become larger where the data are sparse.

Implementation of Bayesian inference

As was mentioned in Chapter 41, Bayesian inference for multilayer networks may be implemented by Monte Carlo sampling, or by deterministic methods employing Gaussian approximations (Neal, 1996; MacKay, 1992c).

Copyright Cambridge University Press 2003. On-screen viewing permitted. Printing not permitted. http://www.cambridge.org/0521642981
You can buy this book for 30 pounds or $50. See http://www.inference.phy.cam.ac.uk/mackay/itila/ for links.

Within the Bayesian framework for data modelling, it is easy to improve our probabilistic models. For example, if we believe that some input variables in a problem may be irrelevant to the predicted quantity, but we don't know which, we can define a new model with multiple hyperparameters that captures the idea of uncertain input variable relevance (MacKay, 1994b; Neal, 1996; MacKay, 1995b); these models then infer automatically from the data which are the relevant input variables for a problem.

▶ 44.5 Exercises

Exercise 44.1.[4] How to measure a classifier's quality. You've just written a new classification algorithm and want to measure how well it performs on a test set, and compare it with other classifiers. What performance measure should you use? There are several standard answers. Let's assume the classifier gives an output $y(\mathbf{x})$, where $\mathbf{x}$ is the input, which we won't discuss further, and that the true target value is t. In the simplest discussions of classifiers, both y and t are binary variables, but you might care to consider cases where y and t are more general objects also.

The most widely used measure of performance on a test set is the *error rate* – the fraction of *misclassifications* made by the classifier. This measure forces the classifier to give a 0/1 output and ignores any additional information that the classifier might be able to offer – for example, an indication of the firmness of a prediction. Unfortunately, the error rate does not necessarily measure how *informative* a classifier's output is. Consider frequency tables showing the joint frequency of the 0/1 output of a classifier (horizontal axis), and the true 0/1 variable (vertical axis). The numbers that we'll show are percentages. The error rate e is the sum of the two off-diagonal numbers, which we could call the false positive rate e_+ and the false negative rate e_-.

Of the following three classifiers, A and B have the same error rate of 10% and C has a greater error rate of 12%.

Classifier A

t \ y	0	1
0	90	0
1	10	0

Classifier B

t \ y	0	1
0	80	10
1	0	10

Classifier C

t \ y	0	1
0	78	12
1	0	10

But clearly classifier A, which simply guesses that the outcome is 0 for all cases, is conveying no information at all about t; whereas classifier B has an informative output: if $y = 0$ then we are sure that t really is zero; and if $y = 1$ then there is a 50% chance that $t = 1$, as compared to the prior probability $P(t = 1) = 0.1$. Classifier C is slightly less informative than B, but it is still much more useful than the information-free classifier A.

How common sense ranks the classifiers:

(best) $B > C > A$ (worst).

How error rate ranks the classifiers:

(best) $A = B > C$ (worst).

One way to improve on the error rate as a performance measure is to report the pair (e_+, e_-), the false positive error rate and the false negative error rate, which are $(0, 0.1)$ and $(0.1, 0)$ for classifiers A and B. It is especially important to distinguish between these two error probabilities in applications where the two sorts of error have different associated costs. However, there are a couple of problems with the 'error rate pair':

- First, if I simply told you that classifier A has error rates $(0, 0.1)$ and B has error rates $(0.1, 0)$, it would not be immediately evident that classifier A is actually utterly worthless. Surely we should have a performance measure that gives the worst possible score to A!

Copyright Cambridge University Press 2003. On-screen viewing permitted. Printing not permitted. http://www.cambridge.org/0521642981
You can buy this book for 30 pounds or $50. See http://www.inference.phy.cam.ac.uk/mackay/itila/ for links.

- Second, if we turn to a multiple-class classification problem such as digit recognition, then the number of types of error increases from two to $10 \times 9 = 90$ – one for each possible confusion of class t with t'. It would be nice to have some sensible way of collapsing these 90 numbers into a single rankable number that makes more sense than the error rate.

Another reason for not liking the error rate is that it doesn't give a classifier credit for accurately specifying its uncertainty. Consider classifiers that have three outputs available, '0', '1' and a *rejection* class, '?', which indicates that the classifier is not sure. Consider classifiers D and E with the following frequency tables, in percentages:

Classifier D

t \ y	0	?	1
0	74	10	6
1	0	1	9

Classifier E

t \ y	0	?	1
0	78	6	6
1	0	5	5

Both of these classifiers have $(e_+, e_-, r) = (6\%, 0\%, 11\%)$. But are they equally good classifiers? Compare classifier E with C. The two classifiers are equivalent. E is just C in disguise – we could make E by taking the output of C and tossing a coin when C says '1' in order to decide whether to give output '1' or '?'. So E is equal to C and thus inferior to B. Now compare D with B. Can you justify the suggestion that D is a more informative classifier than B, and thus is superior to E? Yet D and E have the same (e_+, e_-, r) scores.

People often plot *error-reject curves* (also known as ROC curves; ROC stands for 'receiver operating characteristic') which show the total $e = (e_+ + e_-)$ versus r as r is allowed to vary from 0 to 1, and use these curves to compare classifiers (figure 44.7). [In the special case of binary classification problems, e_+ may be plotted versus e_- instead.] But as we have seen, error rates can be undiscerning performance measures. Does plotting one error rate as a function of another make this weakness of error rates go away?

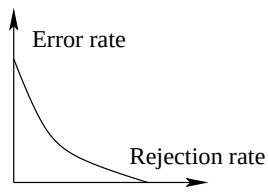

Figure 44.7. An error-reject curve. Some people use the area under this curve as a measure of classifier quality.

For this exercise, either construct an explicit example demonstrating that the error-reject curve, and the area under it, are not necessarily good ways to compare classifiers; or prove that they *are*.

As a suggested alternative method for comparing classifiers, consider the *mutual information* between the output and the target,

$$I(T;Y) \equiv H(T) - H(T\,|\,Y) = \sum_{y,t} P(y)P(t\,|\,y) \log \frac{P(t)}{P(t\,|\,y)}, \tag{44.12}$$

which measures how many *bits* the classifier's output conveys about the target.

Evaluate the mutual information for classifiers A–E above.

Investigate this performance measure and discuss whether it is a useful one. Does it have practical drawbacks?