PATTERN RECOGNITION AND MACHINE LEARNING

CHAPTER 5: NEURAL NETWORKS

Feed-forward Network Functions

$$y(\mathbf{x}, \mathbf{w}) = f\left(\sum_{j=1}^{M} w_j \phi_j(\mathbf{x})\right)$$

f() is a nonlinear activation function

In linear/logistic regression, only coefficients w_j change values.
In neural networks, we extend this model by making the basis functions depend on parameters and then to allow these parameters to be adjusted, along with the coefficients.

First, we construct M linear combinations of the input variables $x_1, \ldots, x_D$ in the form

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}$$

where j = 1, . . . , M, and the superscript (1) indicates that the corresponding parameters are in the first ‘layer’ of the network

The quantities a_j are known as activations.

Feed-forward Network Functions

Each of them is then transformed using a differentiable, nonlinear activation function h(·) to give

$$z_j = h(a_j).$$

These quantities correspond to the outputs of the basis functions and in the context of neural networks, are called hidden units.

Next, these values are again linearly combined to give output unit activations

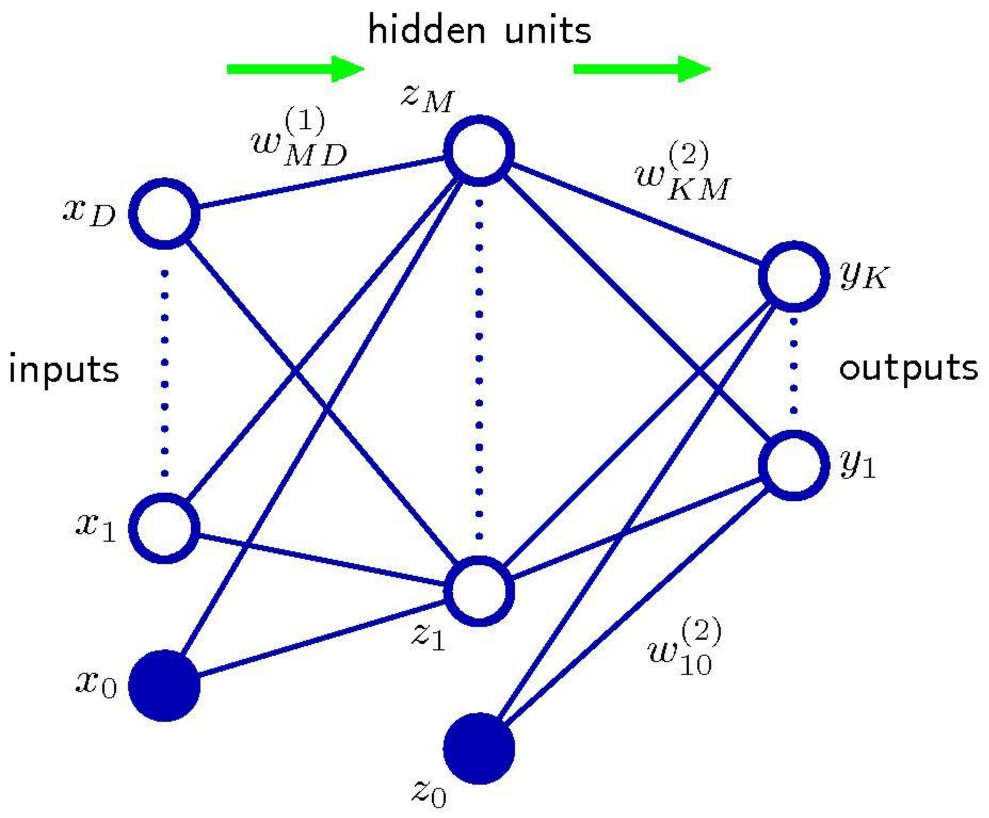
$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)}$$

where k = 1, . . . , K, and K is the total number of outputs. This transformation corresponds to the second layer of the network.

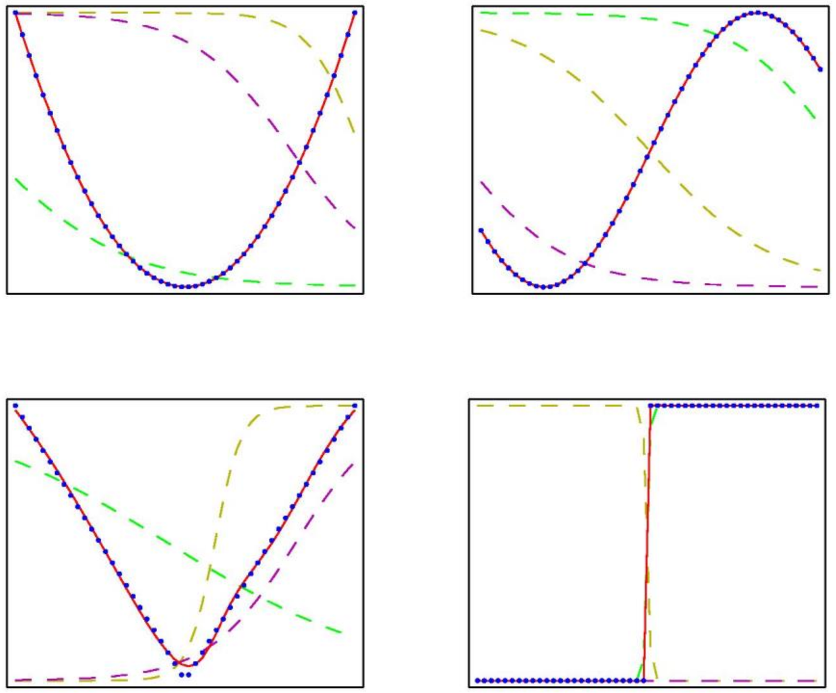
Finally, the output unit activations are transformed using an appropriate activation function to give a set of network outputs y_k.

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right)$$

Feed-forward Network Functions



Feed-forward Network Functions



Neural networks are therefore said to be universal approximators.

For example, a two-layer network with linear outputs can uniformly approximate any continuous function on a compact input domain to arbitrary accuracy provided the network has a sufficiently large number of hidden units.

Network Training

Given a training set comprising a set of input vectors $\{x_n\}$, where n = 1, . . . , N, together with a corresponding set of target vectors $\{t_n\}$, we minimize the error function

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N} \|\mathbf{y}(\mathbf{x}_n, \mathbf{w}) - \mathbf{t}_n\|^2.$$

The error E(**w**) is a smooth continuous function of **w**, its smallest value will occur at a point in weight space such that the gradient of the error function vanishes, so that

$$\nabla E(\mathbf{w}) = 0$$

The above equation cannot be solved analytically, hence, we use an iterative numerical procedure

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} + \Delta\mathbf{w}^{(\tau)}$$

There are different approaches to calculate the weight vector update, $\Delta w^{(\tau)}$

Network Training

There are different approaches to calculate the weight vector update, $\Delta w^{(\tau)}$

The simplest approach to using gradient information is to choose $\Delta w^{(\tau)}$ to comprise a small step in the direction of the negative gradient

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E(\mathbf{w}^{(\tau)})$$

where the parameter $\eta > 0$ is known as the learning rate.

This is a batch method, as the entire training set is used in every step.

At each step the weight vector is moved in the direction of the greatest rate of decrease of the error function, and so this approach is known as gradient descent or steepest descent.

Network Training

There is, however, an on-line version of gradient descent that has proved useful in practice for training neural networks on large data sets.
Error functions based on maximum likelihood for a set of independent observations comprise a sum of terms, one for each data point

$$E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w}).$$

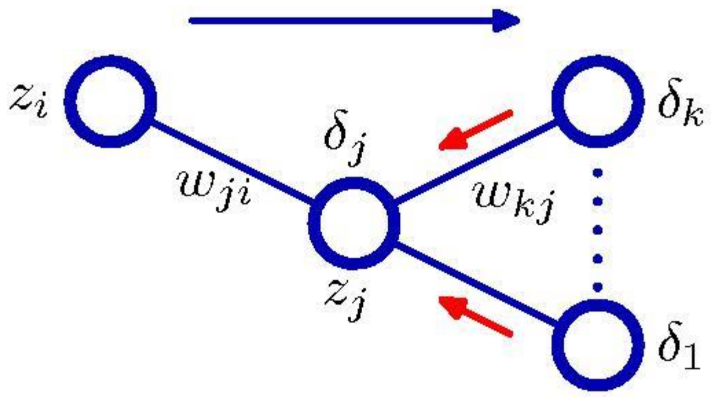
On-line gradient descent, also known as sequential gradient descent or stochastic gradient descent, makes an update to the weight vector based on one data point at a time:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E_n(\mathbf{w}^{(\tau)}).$$

Error Backpropagation

We need to find an efficient technique for evaluating the gradient of an error function E(w) for a feed-forward neural network.

This can be achieved using a local message passing scheme in which information is sent alternately forwards and backwards through the network and is known as error backpropagation.



Error Backpropagation

1. Apply an input vector x_n to the network and forward propagate through the network using

$$a_j = \sum_i w_{ji} z_i \qquad z_j = h(a_j).$$

to find the activations of all the hidden and output units.

2. Evaluate the δ_k for all the output units using

$$\delta_k = y_k - t_k$$

3. Backpropagate the δ's using

$$\delta_j = h'(a_j) \sum_k w_{kj} \delta_k$$

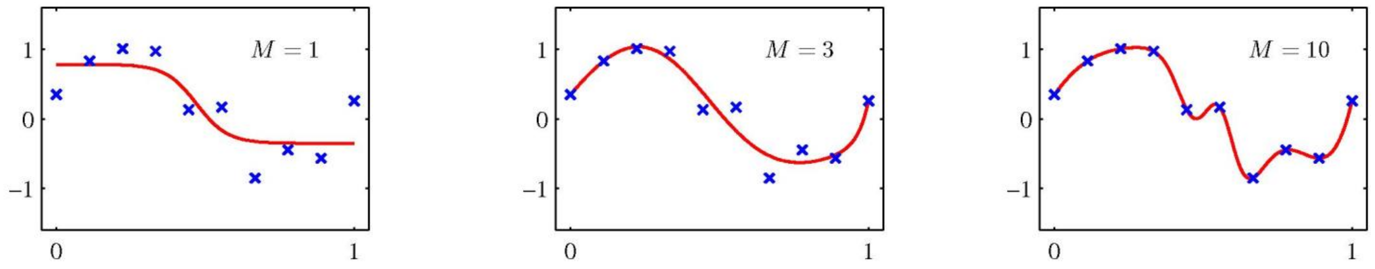
to obtain δ_j for each hidden unit in the network.

4. To evaluate the required derivatives, we can use

$$\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i.$$

Regularization in Neural Networks

The number of input and outputs units in a neural network is generally determined by the dimensionality of the data set, whereas the number M of hidden units is a free parameter that can be adjusted to give the best predictive performance.

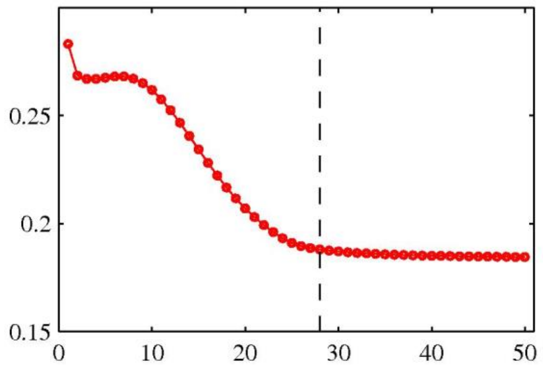


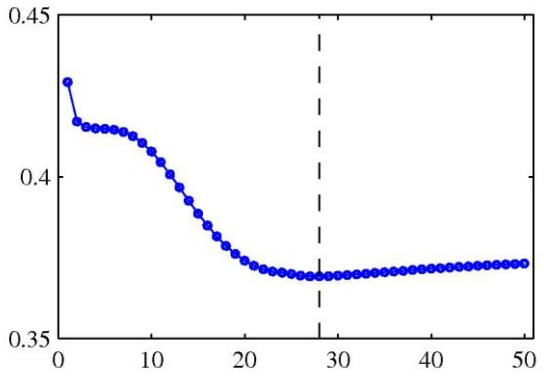
There are other ways to control the complexity of a neural network model in order to avoid over-fitting.
We see that an alternative approach is to choose a relatively large value for M and then to control complexity by the addition of a regularization term to the error function. The simplest regularizer is the quadratic, giving a regularized error

$$\widetilde{E}(\mathbf{w}) = E(\mathbf{w}) + \frac{\lambda}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w}.$$

Regularization in Neural Networks

An alternative to regularization as a way of controlling the effective complexity of a network is the procedure of early stopping.
The training of nonlinear network models corresponds to an iterative reduction of the error function defined with respect to a set of training data.





Training can therefore be stopped at the point of smallest error with respect to the validation data set

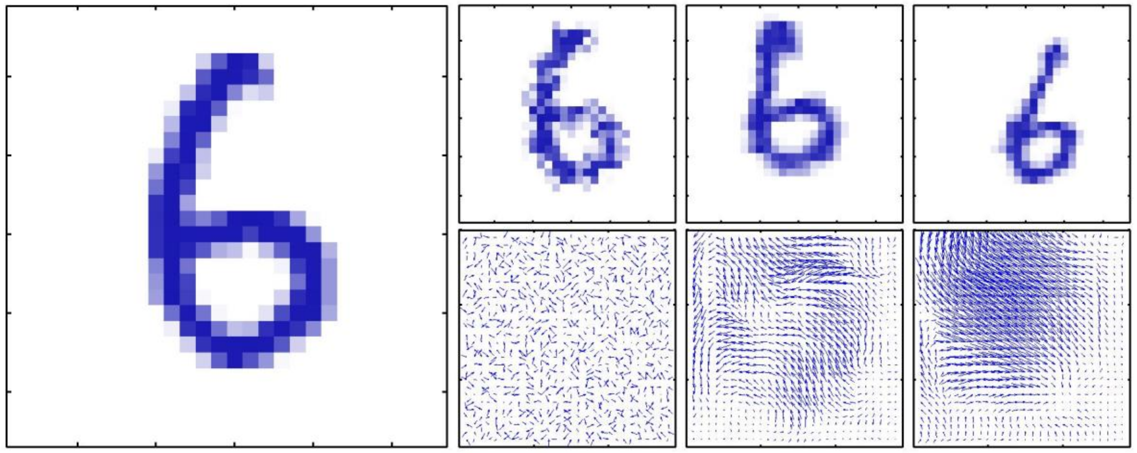
Regularization in Neural Networks

Predictions should be unchanged, or invariant (translation invariance and/or scale invariance), under one or more transformations of the input variables.
If sufficiently large numbers of training patterns are available, then an adaptive model such as a neural network can learn the invariance, at least approximately.

There are four approaches to ensure that neural networks exhibit
the required invariances:
Approach 1) The training set is augmented using replicas of the training patterns, transformed according to the desired invariances.
Approach 2) A regularization term is added to the error function that penalizes changes in the model output when the input is transformed (tangent propagation).
Approach 3) Invariance is built into the pre-processing by extracting features that are invariant under the required transformations.
Approach 4) Build the invariance properties into the structure of a neural network (or into the definition of a kernel function in the case of techniques such as the relevance vector machine) -> Convolutional neural networks.

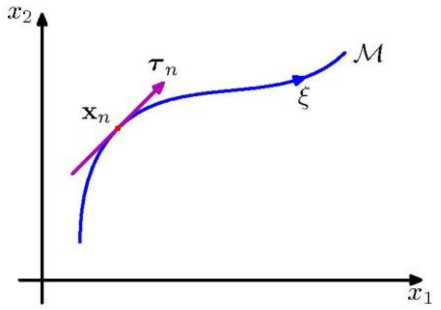
Regularization in Neural Networks

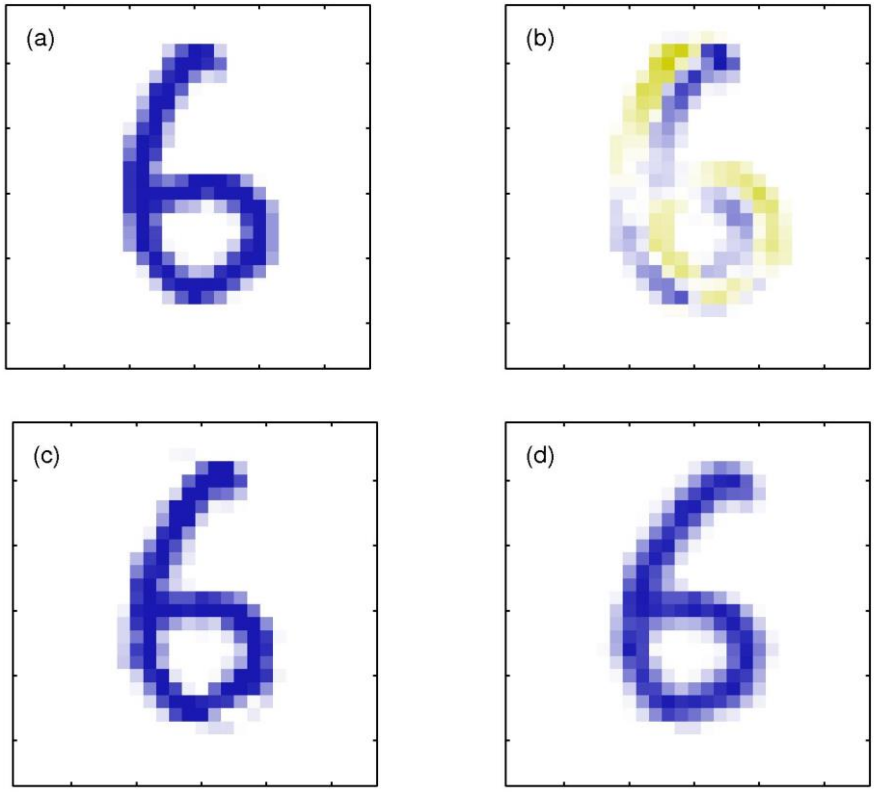
Approach 1 is often relatively easy to implement.
For sequential training algorithms, this can be done by transforming each input pattern before it is presented to the model so that, if the patterns are being recycled, a different transformation (drawn from an appropriate distribution) is added each time.
For batch methods, a similar effect can be achieved by replicating each data point a number of times and transforming each copy independently



Regularization in Neural Networks

Approach 2 leaves the data set unchanged but modifies the error function through the addition of a regularizer.





Regularization in Neural Networks

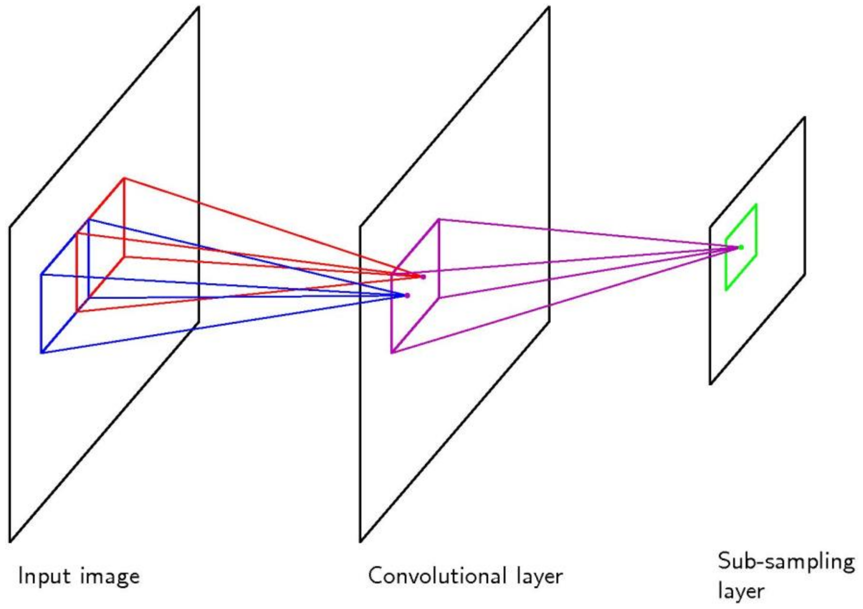
An advantage of approach 3 is that it can correctly extrapolate well beyond the range of transformations included in the training set.

However, it can be difficult to find hand-crafted features with the required invariances that do not also discard information that can be useful for discrimination.

Regularization in Neural Networks

Another approach to creating models that are invariant to certain transformation of the inputs is to build the invariance properties into the structure of a neural network.

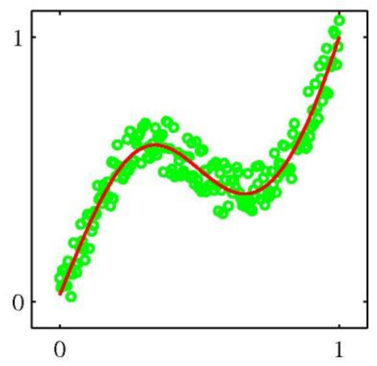
This is the basis for the convolutional neural network, which has been widely applied to image data.

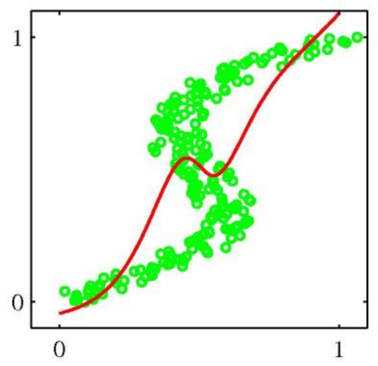


Mixture Density Networks

The goal of supervised learning is to model a conditional distribution p(t|x), which for many simple regression problems is chosen to be Gaussian.

However, practical machine learning problems can often have significantly non-Gaussian distributions.





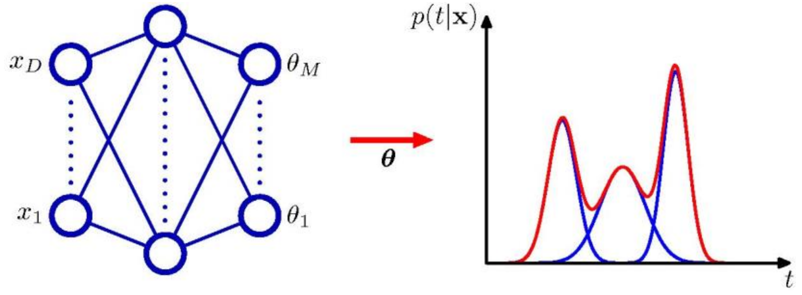
We therefore seek a general framework for modelling conditional probability distributions

Mixture Density Networks

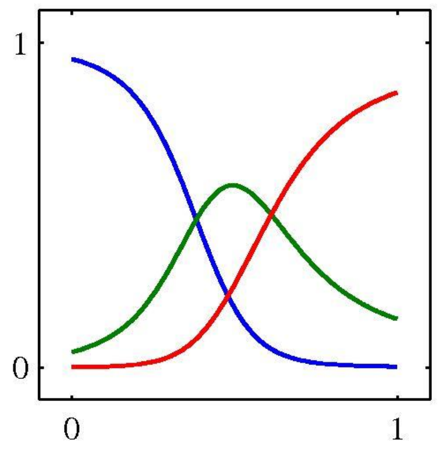
By using a mixture model for p(t|x) in which both the mixing coefficients as well as the component densities are flexible functions of the input vector x, we are giving rise to the mixture density network.

$$p(\mathbf{t}|\mathbf{x}) = \sum_{k=1}^{K} \pi_k(\mathbf{x}) \mathcal{N}\left(\mathbf{t}|\boldsymbol{\mu}_k(\mathbf{x}), \sigma_k^2(\mathbf{x})\right)$$

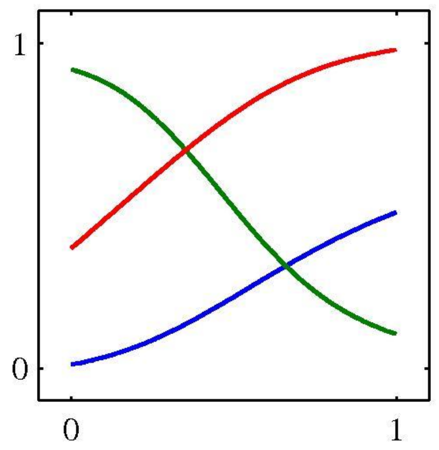
$$\sum_{k=1}^{K} \pi_k(\mathbf{x}) = 1, \qquad 0 \leqslant \pi_k(\mathbf{x}) \leqslant 1$$



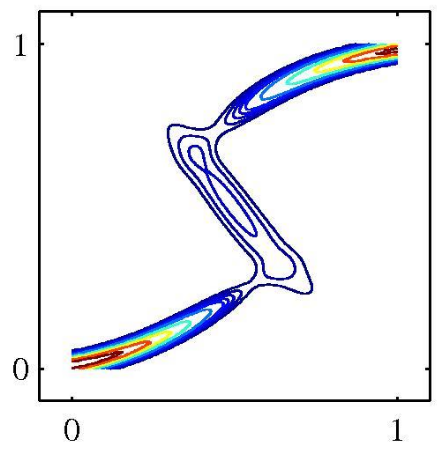
Mixture Density Networks



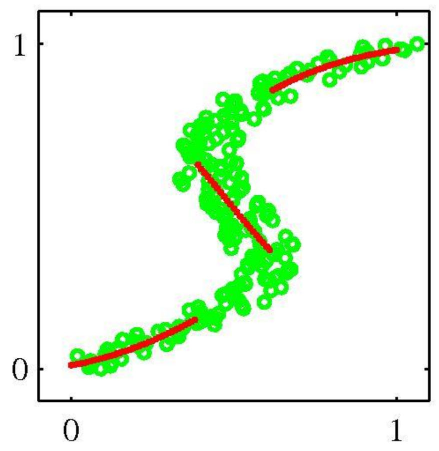
(a)



(b)



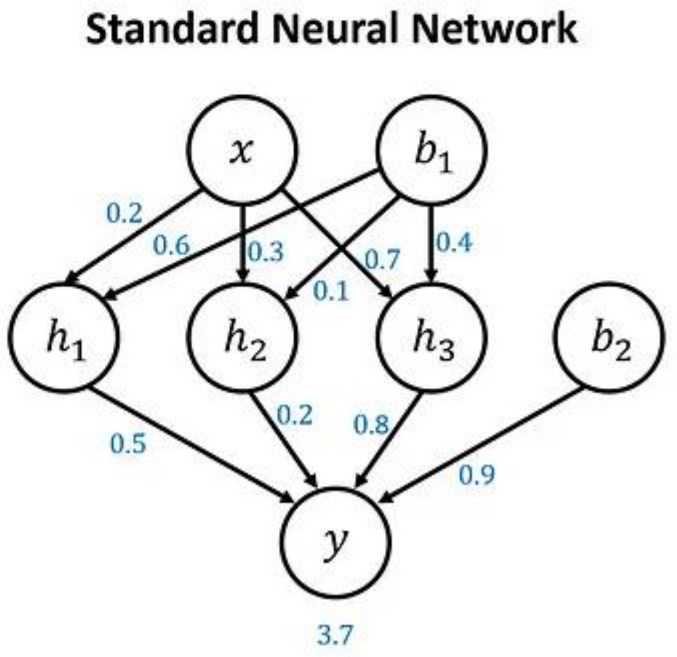
(c)

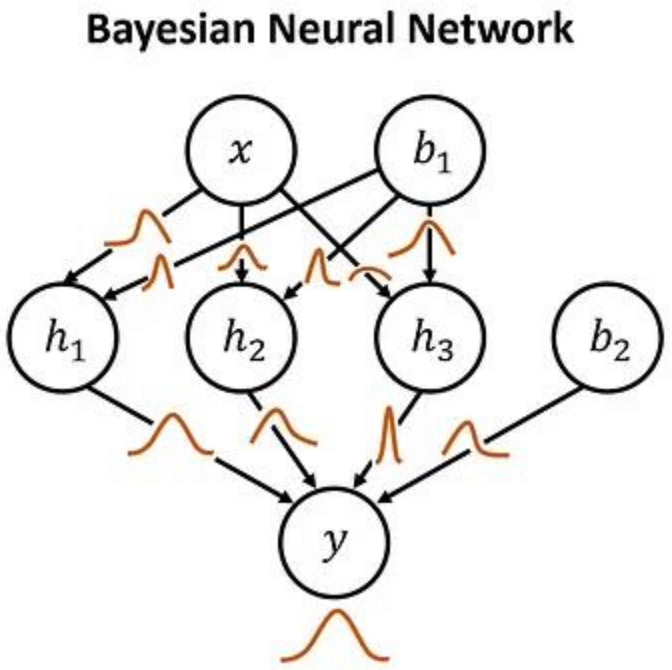


(d)

Bayesian Neural Networks

Bayesian neural networks combine neural network with Bayesian inferences





https://blog.cyda.hk/