Multilayer Perceptron

Multilayer Feed-forward Neural Network

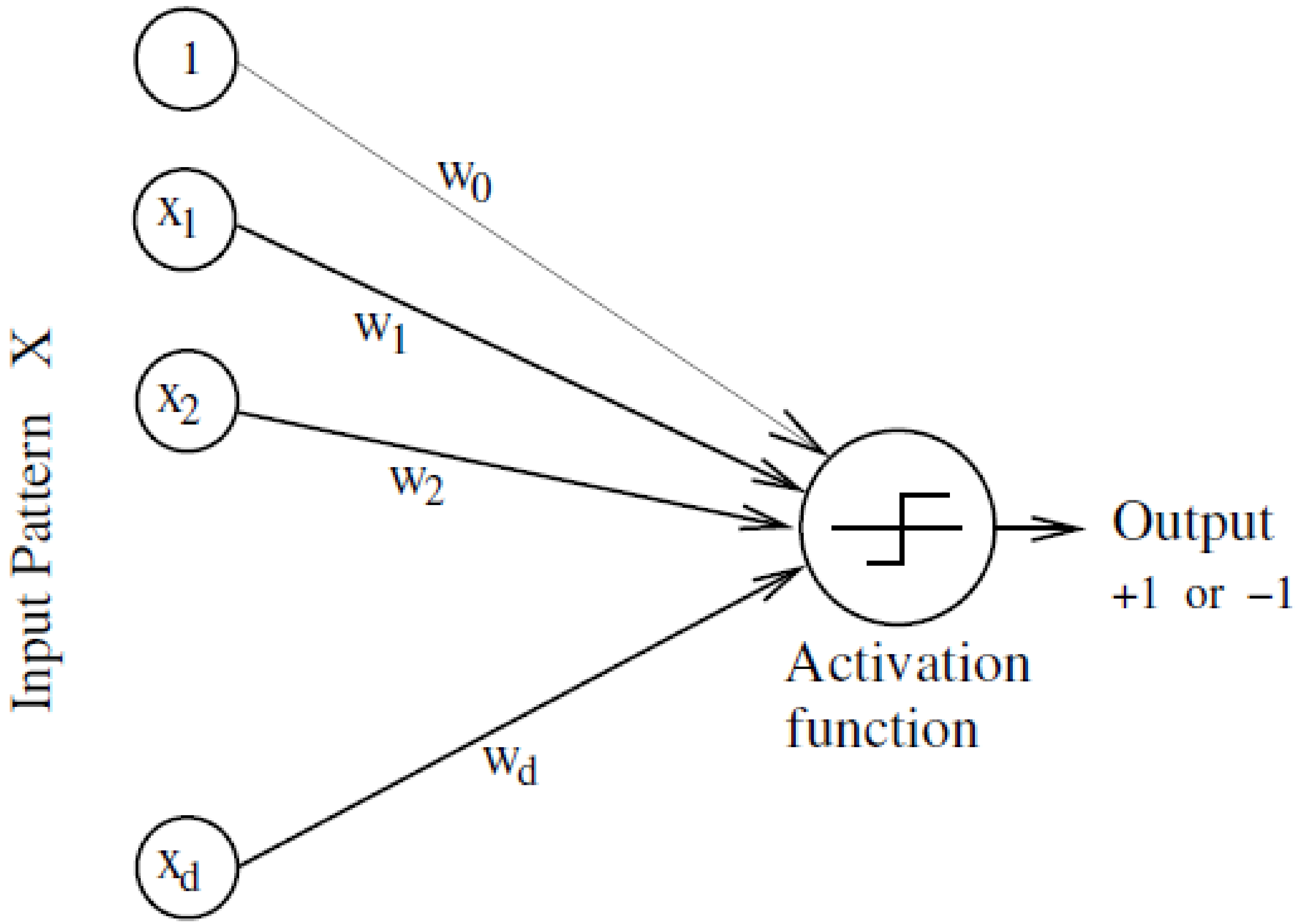
Non-linear discriminants

- Non-linear discriminants are more powerful than linear discriminants.
- The number of parameters to be learned are larger than that with linear discriminants and hence can create some problems.
- These can overcome the drawbacks of the single [a] layer networks (Perceptrons).
- One of the popular methods for training a multilayer network is based on gradient descent procedure called the *backpropagation algorithm*.

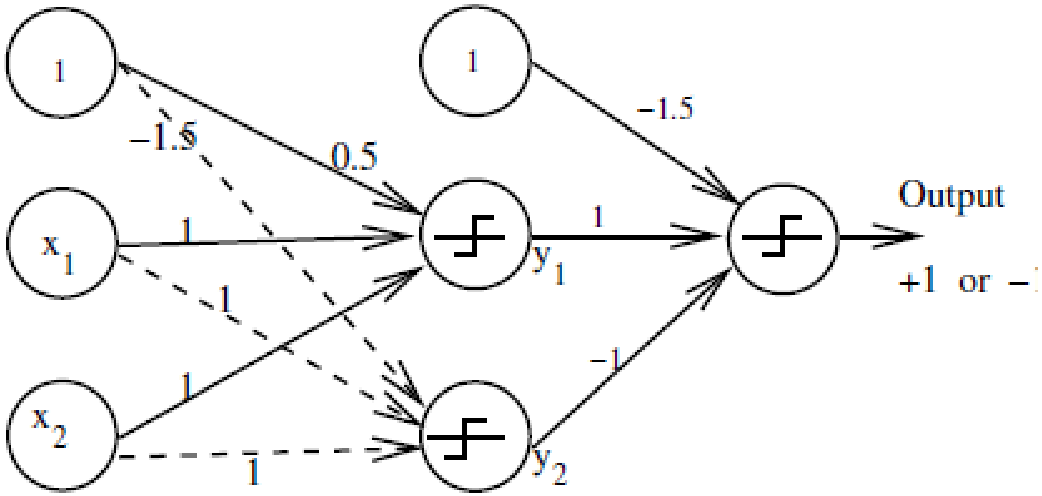
---

[a] Some say Perceptron has single layer others say it has two layers

Two layer network



Three layer network : XOR Problem



x_1	x_2	y_1	y_2	Output
+1	+1	+1	+1	−1
+1	−1	+1	−1	+1
−1	+1	+1	−1	+1
−1	−1	−1	−1	−1

Multilayer Networks

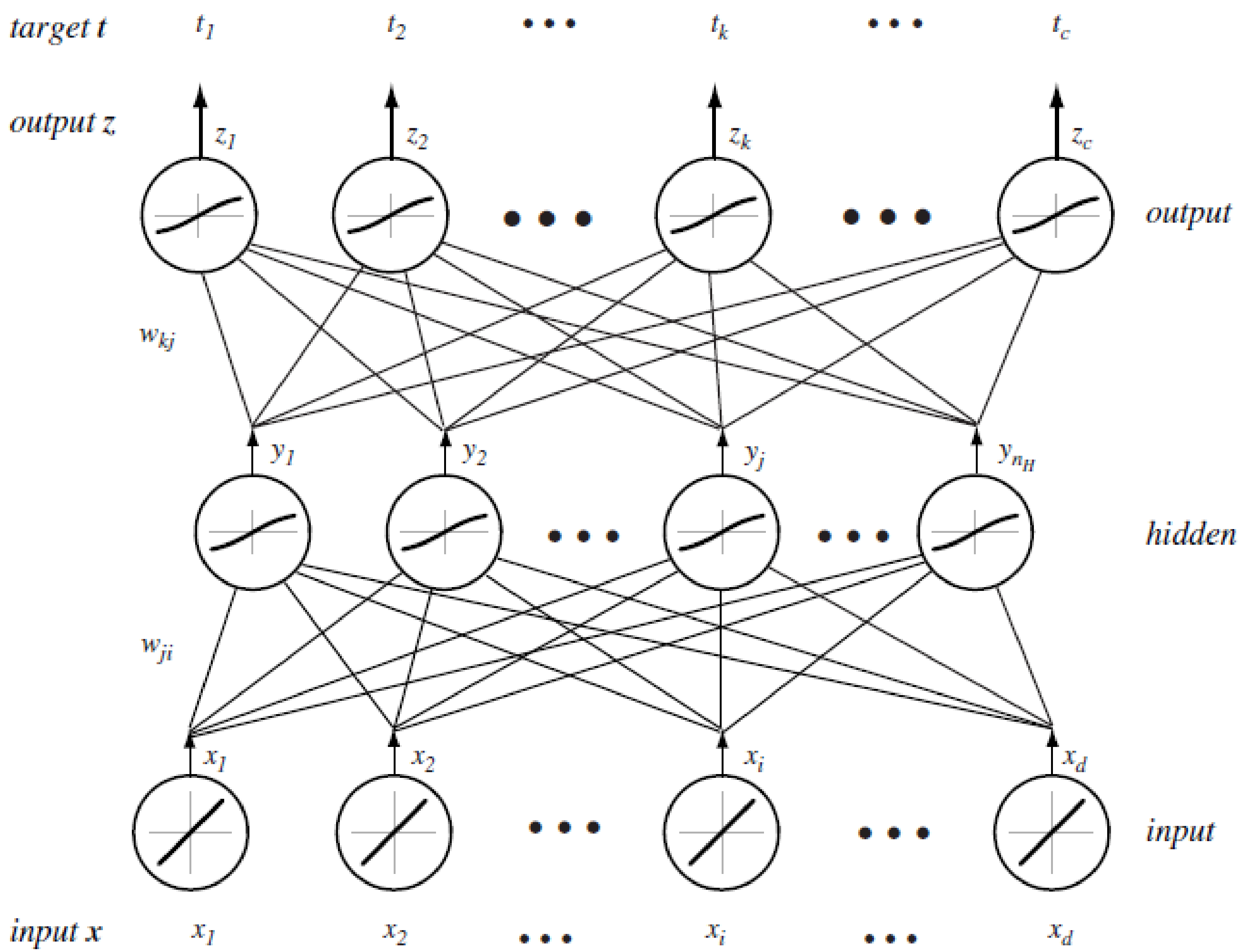
- With sufficient number of hidden units, Multilayer networks can represent any function.

  Indeed, a three layer network with sufficient number of hidden units can implement any function. This is mathematically proved.

- To generalize it to the c class problem, the number of output units are c where each one computes the discriminant function $g_k(X)$.

- The activation function need not be the only *sign (Signum)* function. Indeed we often require the activation function to be continuous and differentiable. Also, activation functions can be different for different units.

Three layer network

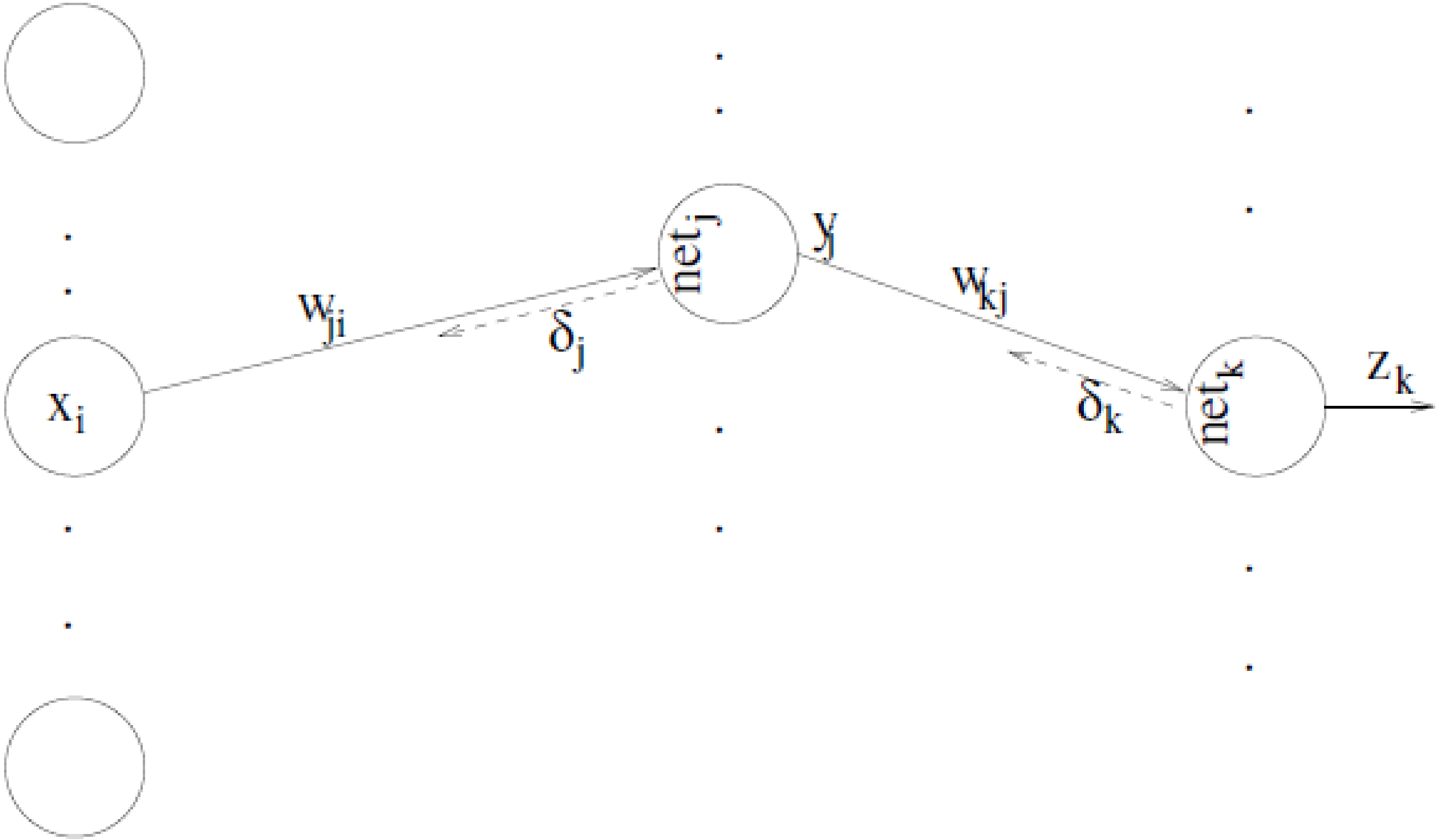


Three layer network

- Let the weight connecting the i th input unit to the j th hidden unit be w_{ji}. Similarly, let the weight connecting the j th hidden unit to the k th output unit be w_{kj}.
- Let the number of hidden units are n_H.
- Let the output from the k th output unit be z_k.
- Then the discriminant $g_k(X)$ is:

$$g_k(X) \equiv z_k = f\left(\sum_{j=1}^{n_H} w_{kj} f\left(\sum_{i=1}^{d} w_{ji}x_i + w_{j0}\right) + w_{k0}\right)$$

Error Backpropagation



Error Backpropagation

- Let the desired output (target) of the output unit k be t_k.
- The actual output of the unit be z_k.
- The objective,

$$J(W) = \frac{1}{2} \sum_{r=1}^{c} (t_r - z_r)^2$$

- We apply gradient descent to get a W for which $J(W)$ is minimum.
- If w is the weight of an edge then the update rule after the m^{th} iteration should be

$$w_{m+1} \quad = \quad w_m - \eta \frac{\partial J}{\partial w} \quad = \quad w_m + \triangle w$$

Error Backpropagation

Update rule for weights between hidden and output units:

- Let w_{kj} be the weight between j^{th} hidden node and k^{th} output node.
- We need to find $\partial J / \partial w_{kj}$
- $J(W) = \frac{1}{2}\sum_{r=1}^{c}(t_r - z_r)^2$, So $\frac{\partial J}{\partial z_k} = -(t_k - z_k)$
- $$\frac{\partial J}{\partial w_{kj}} = \frac{\partial J}{\partial \text{net}_k}\underbrace{\frac{\partial \text{net}_k}{\partial w_{kj}}}_{y_j} = \frac{\partial J}{\partial z_k}\frac{\partial z_k}{\partial \text{net}_k} y_j = \underbrace{-(t_k - z_k) f'(\text{net}_k)}_{\frac{\partial J}{\partial net_k} = -\delta_k} y_j$$
  $$= -\delta_k y_j$$
- $\triangle w_{kj} = \eta y_j \delta_k$

Error Backpropagation

Update rule for weights between input and hidden units:

- Let w_{ji} be the weight between i^{th} input node and j^{th} hidden node.

$$\frac{\partial J}{\partial w_{ji}} = \underbrace{\frac{\partial J}{\partial net_j}}_{-\delta_j} \underbrace{\frac{\partial net_j}{\partial w_{ji}}}_{x_i} = \left(\sum_{r=1}^{c} \underbrace{\frac{\partial J}{\partial \text{net}_r}}_{-\delta_r} \underbrace{\frac{\partial \text{net}_r}{\partial y_j}}_{w_{rj}} \underbrace{\frac{\partial y_j}{\partial \text{net}_j}}_{f'(net_j)} \right) x_i$$

$$= \underbrace{\sum_{r=1}^{c} -\delta_r w_{rj} f'(net_j)\, x_i}_{\frac{\partial J}{\partial net_j} = -\delta_j}$$

- $\triangle w_{ji} = \eta x_i \delta_j$

Learning – training the network

- We will call the process of running 1 example through the network (and training the network on that 1 example) a **weight update iteration**.
- Training the network once on each example of your training set is called an **epoch**. Typically, you have to train the network for many epochs before it converges.

Training Protocols

- The exact behavior of the backpropagation depends on the starting point.
- We cannot start with $W = 0$, i.e., all weights being zeros. Because the updates will be zero and the training cannot proceed.
- So one should start with a random weight vector.

- Two common approaches

1. Offline or Batch Protocol
2. Online Protocol

Batch Backpropagation

- In the batch training protocol, all the training patterns are presented first and their corresponding weight updates summed; only then are the actual weights in the network are updated.

- Stopping criteria:
  - When weight updation between successive epochs is small enough.
  - Or, overall error accumulated for all training examples is small enough.

Online

- For each example –
  - Present the example to the network
  - update the weights.
  - Goto the next example.
- When all examples are presented, we say one epoch is completed.

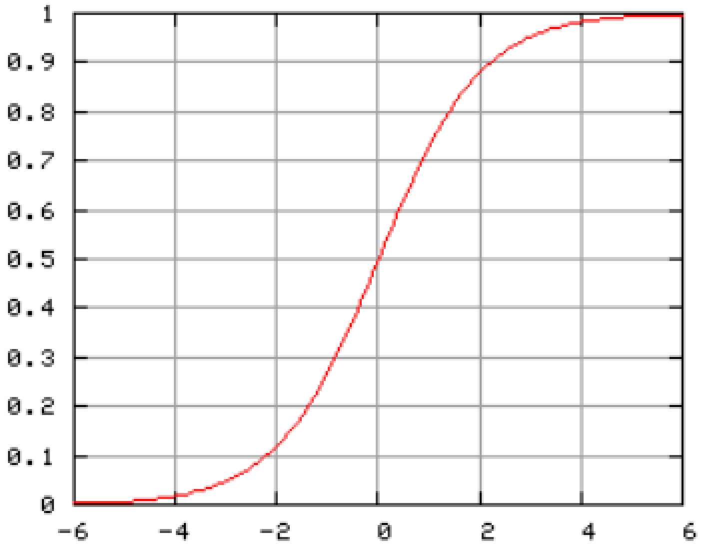
Activation Function

- Backpropagation will work with any activation function $f(\cdot)$, provided that it is continuous and differentiable.
- But the activation function needs to be non-linear, otherwise it will not have any computational power over Perceptrons.
- A second desirable property is that $f(\cdot)$ has some bounded range. It should have some limited values for maximum and minimum. This will keep the weights to be bounded, and thus keeping the training time limited.

Activation function

Sigmoid function

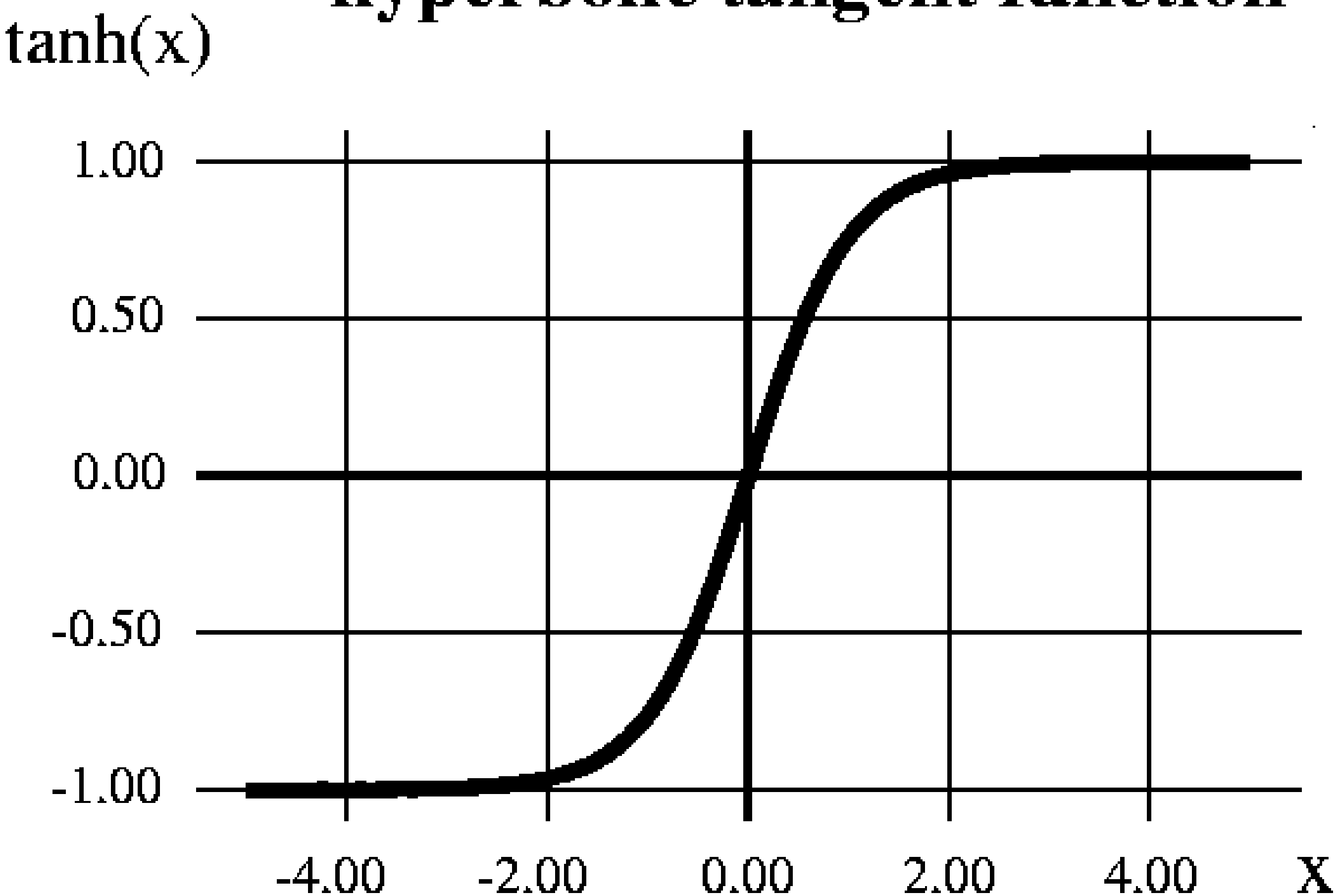
$$f'(\boldsymbol{t}) = \boldsymbol{f}(\boldsymbol{t})(\mathbf{1} - \boldsymbol{f}(\boldsymbol{t}))$$



$$f(t) = \frac{1}{1+e^{-t}}$$

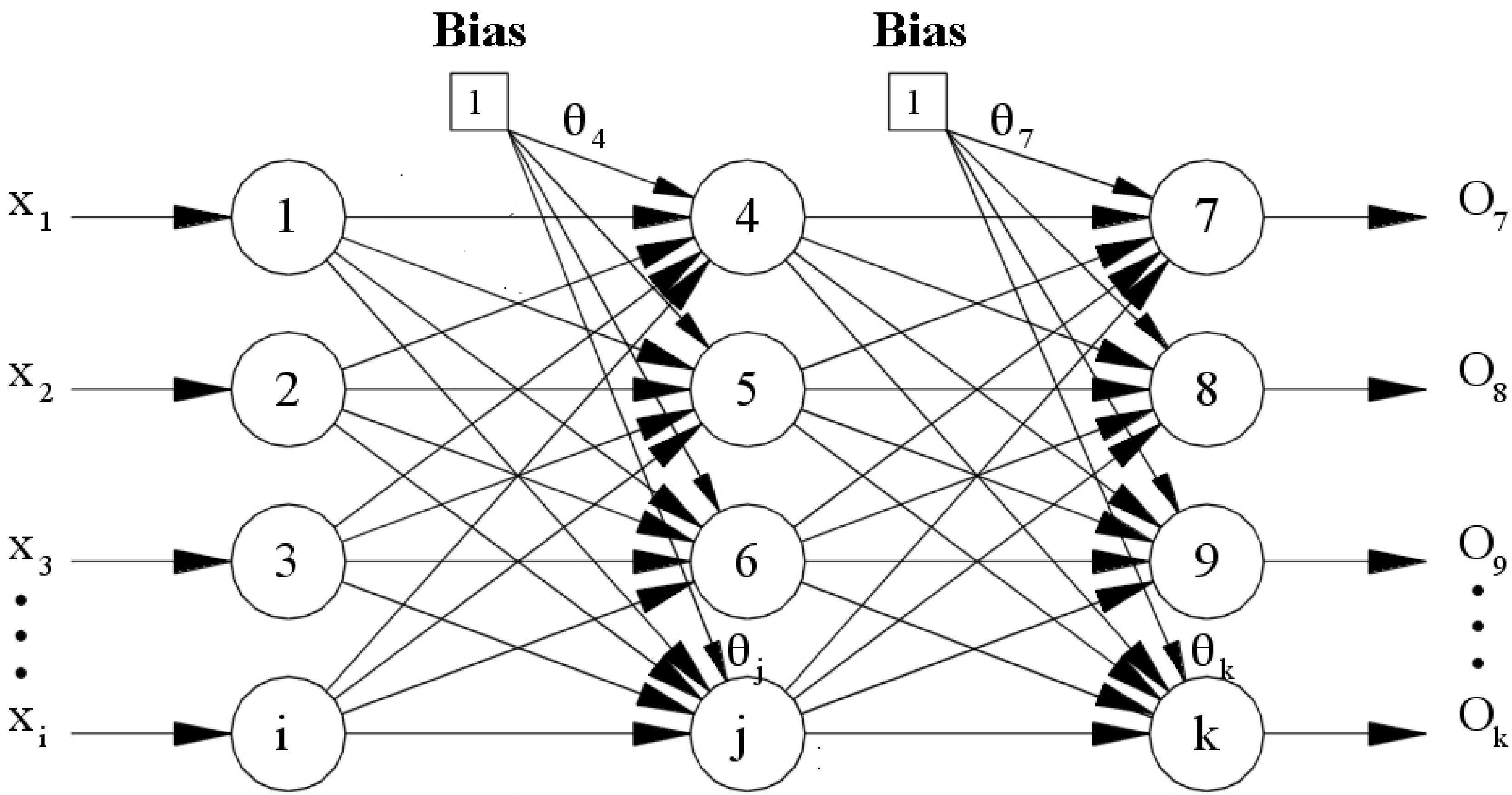
Various parametrized sigmoid functions are there, but, we will not talk about them.

hyperbolic tangent function



- $f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

Universal approximation requires bias term



Scaling Input

- A feature with whose values could be of larger magnitude (say it is in thousands), can swamp the features whose values are smaller (say in fractions).
- It is better to normalize the training set to have *zero-mean* and *unit-variance* for each feature.

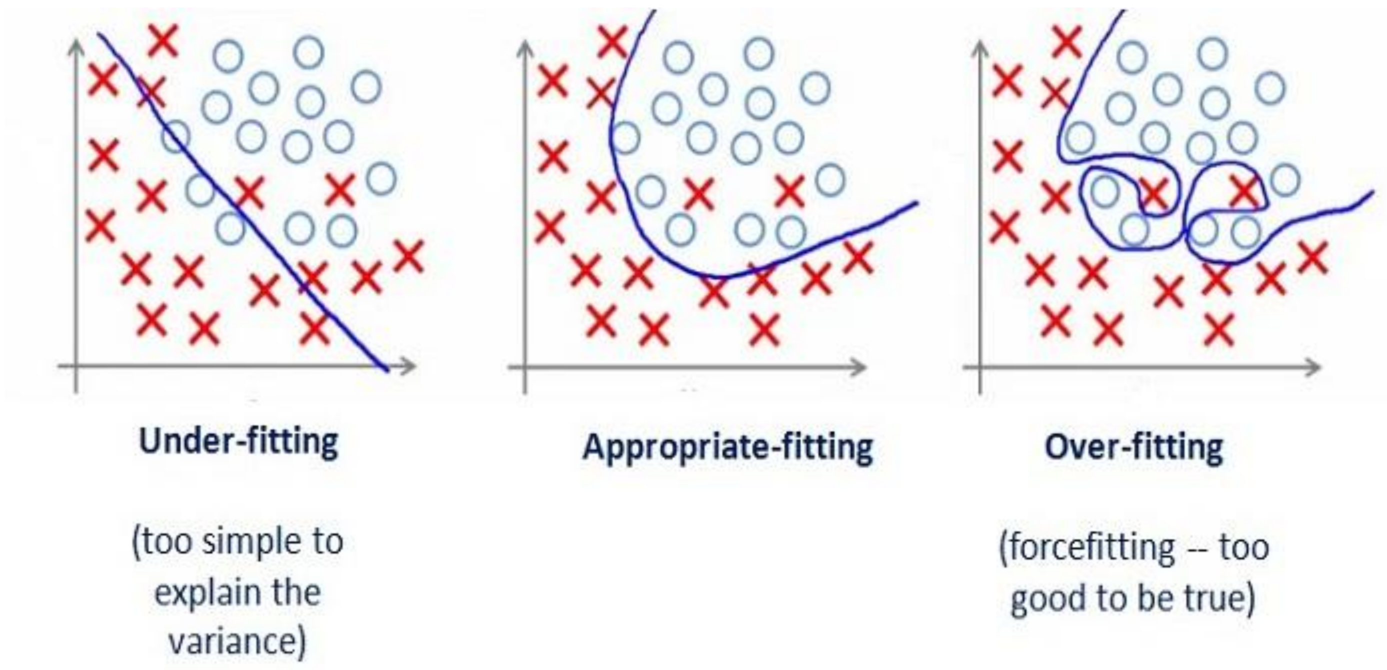
Improvements

- It can be shown that with infinite number of training patterns, multilayer networks converges to the Bayes classifier.
- So, larger the training set, better the classifier.
- Some times it is possible for us to create training patterns based on domain knowledge, etc.
  - In OCR data, the images of the characters can be rotated slightly to create new patterns.
  - Probabilistic dependency between the features can be taken into account to create new patterns.
- Drawbacks with larger training sets is that of increased space and training time requirements.

Number of Hidden Units

- The number of input and output units are dictated by the problem, but the number of hidden units (n_H) is not.
- Large n_H is required to learn complicated functions. So large n_H means more expressive power for the net.
  - Drawback: Over-fitting. Good performance over the training set doesnot necessarily mean good performance over the independent test set. One should not respect noisy patterns.
- Small n_H means, the net doesnot have enough free parameters to fit the training data well, and again the test error is high.

Over-fitting versus **under-fitting**



- Regularization is a principled way to overcome the over-fitting problem.

Number of Hidden Units

- n_H determines the total number of weights in the net – which can be considered as number of degrees of freedom – and thus it can be seen that we should not have more weights than the total number of training patterns.
- A convenient rule of thumb is to choose the total number of weights is roughly $N/10$ where N is the number of training patterns.

Learning Rates

- In principle, small learning rates will lead to convergence, but slowly.
- The Hessian matrix (second order information) can guide to optimal learning rate, as we saw for Perceptrons.
- Large learning rates can have negative effect.
- In practice, however thumb-rules are used. For example, $\eta \simeq 0.1$ might be alright, which can be lowered or increased based the changes in the J values.

- Adding Momentum can speed-up the process.
  - A fraction of Previous update can be added …
- $\Delta \mathrm{w_{ij}}(t) \quad = -\eta \frac{\partial J}{\partial w_{ij}} + \alpha\, \Delta \mathrm{w_{ij}}(t-1)$

When to stop the training?

- Excessive training can give us a classifier which is doing very well with the training data.
- This, anyhow, doesnot guarantee better performance with the test set.
- Cross validation technique can be used to decide when to stop the training.
- Some times early stopping is advocated.

Stopping Criterion

- $$W_{new} = W_{old} + \Delta W$$

  Stop when ΔW is sufficiently small.

  That is, if ($||\Delta W|| < \epsilon$), STOP.

  ϵ could be something like 0.01

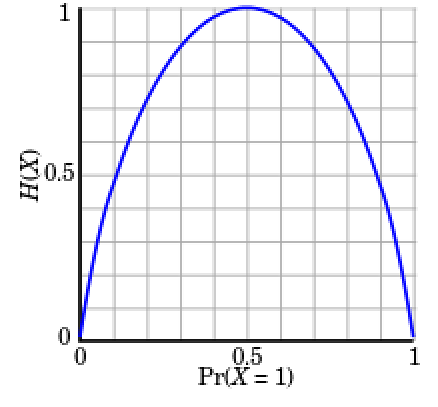
- Or, after the fixed number of epochs, the training can be stopped.

Other Loss Functions

- **Cross-entropy** is also a widely used loss function.
- **Entropy:** For a distribution, it is a measure of uncertainty.

$$H(X) = -\sum_{X=x_i} p(x_i) \log p(x_i)$$

- For a binary random variable



Cross (relative) entropy

- Distribution q is away from distribution p, by

$$H(p,q) = -\sum_{x_i} p(x_i) \log q(x_i)$$

- Distribution p is the target, q is the network's output.
- For binary classification, a single output neuron is enough.

  Let the two classes are labeled 0 and 1.

  The cross entropy is: $-t \log z - (1-t) \log(1-z)$.

$$\boxed{J(W) = -\sum_{\mathrm{r}=1}^{\mathrm{c}} t_r \log z_r + (1-t_r) \log(1-z_r)}$$

Sigmoid (logistic activation)

- Remember, we used the same activation, i.e., the Sigmoid one.
- Now, one can apply back-propagation with this error being defined.
- For details, it is quite similar to what we did with sum of squared deviations. Reference is given below.
  - But, notation used in the following reference is different.

Reference: https://www.ics.uci.edu/~pjsadows/notes.pdf