MultiLayer Neural Networks

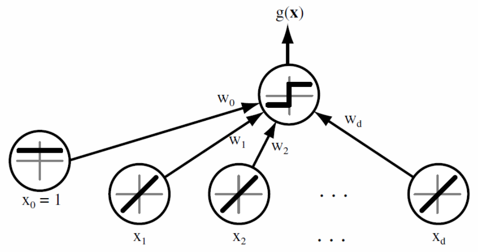
Xiaogang Wang

xgwang@ee.cuhk.edu.hk

March 23, 2014

Outline

Two-layer neural networks model linear classifiers

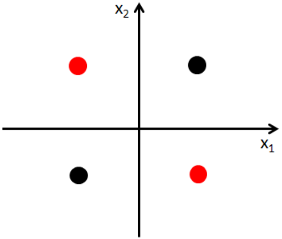


$$g(\mathbf{x}) = f(\sum_{i=1}^{d} x_i w_i + w_0) = f(\mathbf{w}^t \mathbf{x})$$

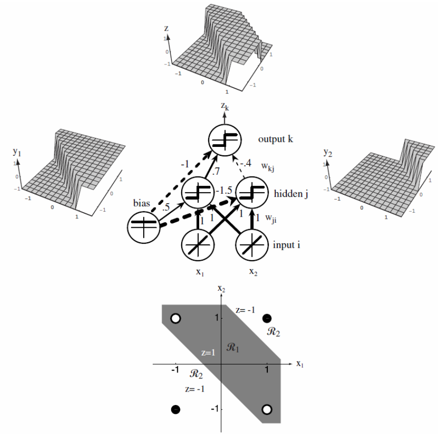
$$f(s) = \begin{cases} 1, & \text{if } s \geq 0 \\ -1, & \text{if } s < 0 \end{cases}.$$

Two-layer neural networks model linear classifiers

- A linear classifier cannot solve the simple exclusive-OR prolem



Non-linear classifiers can be modeled by adding a hidden layer



Three-layer neural network

- Net activation: each hidden unit j computes the weighted sum of its inputs

$$net_j = \sum_{i=1}^{d} x_i w_{ji} + w_{j0} = \sum_{i=0}^{d} x_i w_{ji} = \mathbf{w}_j^t \mathbf{x}$$

- Activation function: each hidden unit emits an output that is a **nonlinear** function of its activation

$$y_j = f(net_j)$$

$$f(net) = Sgn(net) = \begin{cases} 1, & \text{if } net \geq 0 \\ -1, & \text{if } net < 0 \end{cases}.$$

There are multiple choices of the activation function as long as they are continuous and differentiable **almost everywhere**. Activation functions could be different for different nodes.

Three-layer neural network

- Net activation of an output unit k

$$net_k = \sum_{i=1}^{n_H} y_j w_{kj} + w_{k0} = \sum_{j=0}^{n_H} y_j w_{kj} = \mathbf{w}_k^t \mathbf{y}$$

- Output unit emits

$$z_k = f(net_k)$$

- The output of the neural network is equivalent to a set of discriminant functions

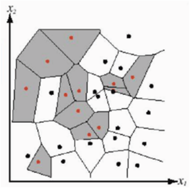
$$g_k(\mathbf{x}) = z_k = f\left(\sum_{j=1}^{n_H} w_{kj} f\left(\sum_{i=1}^{d} w_{ji} x_i + w_{j0}\right) + w_{k0}\right)$$

Expressive power of a three-layer neural network

- It can represent **any** discriminant function
- However, the number of hidden units required can be very large...
- Most widely pattern recognition models (such as SVM, boosting, and KNN) can be approximated as neural networks with one or two hidden layers. They are called models with shallow architectures.
- Shallow models divide the feature space into regions and match templates in local regions. $O(N)$ parameters are needed to represent N regions.
- Deep architecture: the number of hidden nodes can be reduced exponentially with more layers for certain problems.

Expressive power of a three-layer neural network

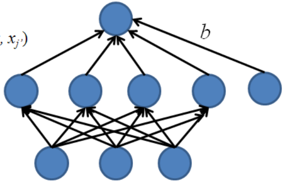
KNN



output $g(x) = label_j$
$j = \text{argmin}_{j'}\, d(x, x_{j'})$

hidden $d(x, x_j)$

input x

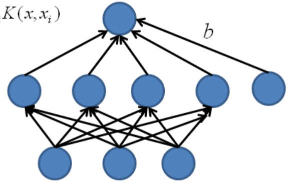


SVM

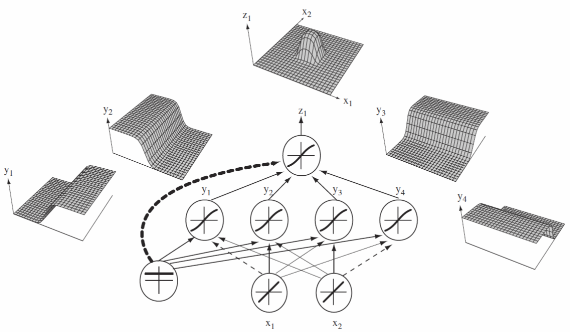
output $g(x) = b + \sum_i \alpha_i K(x, x_i)$

hidden $K(x, x_j)$

input x



Expressive power of a three-layer neural network

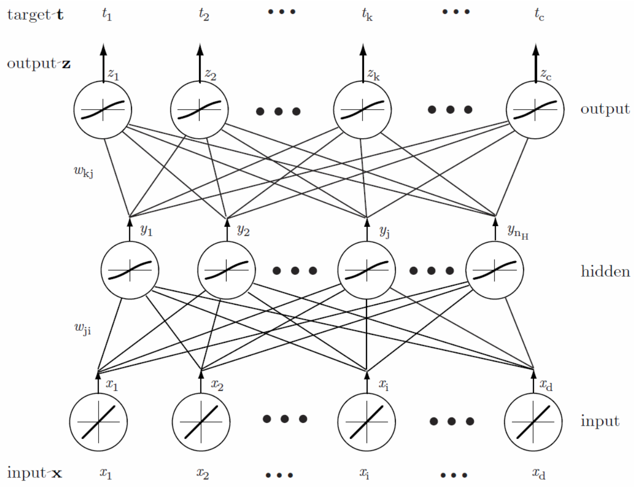


With a sigmoidal activation function $f(s) = (e^s - e^{-s})/(e^s + e^{-s})$, the hidden unit outputs are paired in opposition thereby producing a "bump" at the output unit. With four hidden units, a local mode (template) can be modeled. Given a sufficiently large number of hidden units, any continuous function from input to output can be approximated arbitrarily well by such a network.

Backpropagation

- The most general method for supervised training of multilayer neural network
- Present an input pattern and change the network parameters to bring the actual outputs closer to the target values
- Learn the input-to-hidden and hidden-to-output weights
- However, there is no explicit teacher to state what the hidden unit's output should be. Backpropagation calculates an effective error for each hidden unit, and thus derive a learning rule for the input-to-hidden weights.

A three-layer network for illustration



$$z_k = f(net_k)$$

$$net_k = \sum_{i=1}^{n_H} y_j w_{kj} + w_{k0}$$

$$y_j = f(net_j)$$

$$net_j = \sum_{i=1}^{d} x_i w_{ji} + w_{j0}$$

Training error

$$J(\mathbf{w}) = \frac{1}{2}\sum_{k=1}^{c}(t_k - z_k)^2 = \frac{1}{2}||\mathbf{t} - \mathbf{z}||^2$$

- Differentiable
- There are other choices, such as cross entropy

$$J(\mathbf{w}) = -\sum_{k=1}^{c} t_k \log(z_k)$$

Both $\{z_k\}$ and $\{t_k\}$ are probability distributions.

Gradient descent

- Weights are initialized with random values, and then are changed in a direction reducing the error

$$\Delta \mathbf{w} = -\eta \frac{\partial J}{\partial \mathbf{w}},$$

or in component form

$$\Delta w_{pq} = -\eta \frac{\partial J}{\partial w_{pq}}$$

where η is the learning rate.

- Iterative update

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

Hidden-to-output weights w_{kj}

$$\frac{\partial J}{\partial w_{kj}} = \frac{\partial J}{\partial net_k} \frac{\partial net_k}{\partial w_{kj}} = -\delta_k \frac{\partial net_k}{\partial w_{kj}}$$

- Sensitivity of unit k

$$\delta_k = -\frac{\partial J}{\partial net_k} = -\frac{\partial J}{\partial z_k} \frac{\partial z_k}{\partial net_k} = (t_k - z_k) f'(net_k)$$

  Describe how the overall error changes with the unit's net activation.

- Weight update rule. Since $\partial net_k / \partial w_{kj} = y_j$,

$$\Delta w_{kj} = \eta \delta_k y_j = \eta (t_k - z_k) f'(net_k) y_j.$$

Activation function

- Sign function is not a good choice for $f(\cdot)$. Why?
- Popular choice of $f(\cdot)$
  - Sigmoid function

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

  - Tanh function (shift the center of Sigmoid to the origin)

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

  - Rectified linear unit (ReLU)

$$f(s) = \max(0, x)$$

Input-to-hidden weights

$$\frac{\partial J}{\partial w_{ij}} = \frac{\partial J}{\partial y_j} \frac{\partial y_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ij}}$$

- How the hidden unit output y_j affects the error at each output unit

$$\begin{aligned}
\frac{\partial J}{\partial y_j} &= \frac{\partial}{\partial y_j}\left[\frac{1}{2}\sum_{k=1}^{c}(t_k - z_k)^2\right] \\
&= -\sum_{k=1}^{c}(t_k - z_k)\frac{\partial z_k}{\partial y_j} \\
&= -\sum_{k=1}^{c}(t_k - z_k)\frac{\partial z_k}{\partial net_k}\frac{\partial net_k}{\partial y_j} \\
&= -\sum_{k=1}^{c}(t_k - z_k)f'(net_k)w_{kj} = \sum_{k=1}^{c}\delta_k w_{kj}
\end{aligned}$$

Input-to-hidden weights

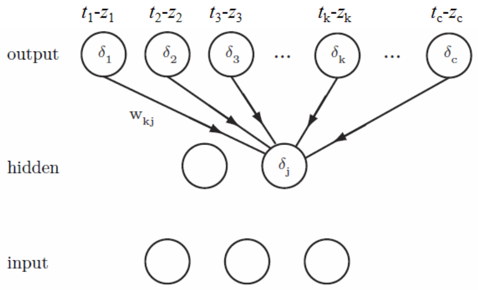
- Sensitivity for a hidden unit j

$$\delta_j = -\frac{\partial J}{\partial net_j} = -\frac{\partial J}{\partial y_j}\frac{\partial y_j}{\partial net_j} = f'(net_j)\sum_{k=1}^{c} w_{kj}\delta_k$$

- $\sum_{k=1}^{c} w_{kj}\delta_k$ is the effective error for hidden unit j
- Weight update rule. Since $\partial net_j / \partial w_{ji} = x_i$,

$$\Delta w_{ji} = \eta x_i \delta_j = \eta \left[\sum_{k=1}^{c} w_{kj}\delta_k\right] f'(net_j)x_i$$

Error backpropagation



The sensitivity at a hidden unit is proportional to the weighted sum of the sensitivities at the output units: $\delta_j = f'(net_j) \sum_{k=1}^{c} w_{kj}\delta_k$. The output unit sensitivities are thus propagated "back" to the hidden units.

Stochastic gradient descent

- Given n training samples, our target function can be expressed as

$$J(\mathbf{w}) = \sum_{p=1}^{n} J_n(\mathbf{w})$$

- Batch gradient descent

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \sum_{p=1}^{n} \nabla J_p(\mathbf{w})$$

- In some cases, evaluating the sum-gradient may be computationally expensive. Stochastic gradient descent samples a subset of summand functions at every step. This is very effective in the case of large-scale machine learning problems. In stochastic gradient descent, the true gradient of $J(\mathbf{w})$ is approximated by a gradient at a single example (or a mini-batch of samples):

$$\mathbf{w} \leftarrow \mathbf{w} - \eta J_p(\mathbf{w})$$

Stochastic backpropagation

Algorithm 1 (Stochastic backpropagation)

begin initialize network topology (# hidden units), $\mathbf{w}$, criterion $\theta, \eta, m \leftarrow 0$
    do $m \leftarrow m + 1$
        $\mathbf{x}^m \leftarrow$ randomly chosen pattern
        $w_{ij} \leftarrow w_{ij} + \eta\delta_j x_i; \quad w_{jk} \leftarrow w_{jk} + \eta\delta_k y_j$
    until $\nabla J(\mathbf{w}) < \theta$
return $\mathbf{w}$
end

- In stochastic training, a weight update may reduce the error on the single pattern being presented, yet increase the error on the full training set.

Mini-batch based stochastic gradient descent

- Divide the training set into mini-batches.
- In each epoch, randomly permute mini-batches and take a mini-batch sequentially to approximate the gradient
  - One epoch corresponds to a single presentations of all patterns in the training set
- The estimated gradient at each iteration is more reliable
- Start with a small batch size and increase the size as training proceeds

Batch backpropagation

Algorithm 2 (Batch backpropagation)

begin initialize network topology (# hidden units), $\mathbf{w}$, criterion $\theta, \eta, r \leftarrow 0$
    do $r \leftarrow r + 1$ (increment epoch)
        $m \leftarrow 0;\ \Delta w_{ij} \leftarrow 0;\ \Delta w_{jk} \leftarrow 0$
        do $m \leftarrow m + 1$
            $\mathbf{x}^m \leftarrow$ select pattern
            $\Delta w_{ij} \leftarrow \Delta w_{ij} + \eta \delta_j x_i;\quad \Delta w_{jk} \leftarrow \Delta w_{jk} + \eta \delta_k y_j$
        until $m = n$
        $w_{ij} \leftarrow w_{ij} + \Delta w_{ij};\quad w_{jk} \leftarrow w_{jk} + \Delta w_{jk}$
    until $\nabla J(\mathbf{w}) < \theta$
return $\mathbf{w}$
end

Summary

- Stochastic learning
  - Estimate of the gradient is noisy, and the weights may not move precisely down the gradient at each iteration
  - Faster than batch learning, especially when training data has redundance
  - Noise often results in better solutions
  - The weights fluctuate and it may not fully converge to a local minimum
- Batch learning
  - Conditions of convergence are well understood
  - Some acceleration techniques only operate in batch learning
  - Theoretical analysis of the weight dynamics and convergence rates are simpler