MultiLayer Neural Networks

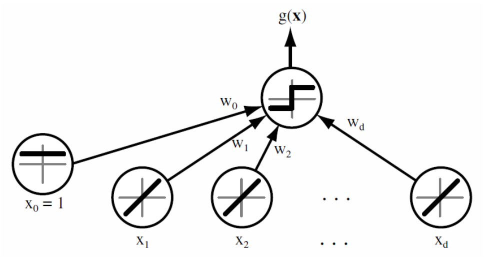
Xiaogang Wang

xgwang@ee.cuhk.edu.hk

March 27, 2014

Outline

Two-layer neural networks model linear classifiers

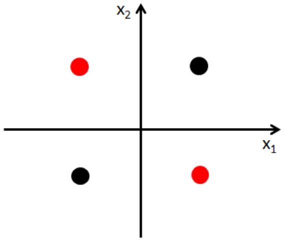


$$g(\mathbf{x}) = f(\sum_{i=1}^{d} x_i w_i + w_0) = f(\mathbf{w}^t \mathbf{x})$$

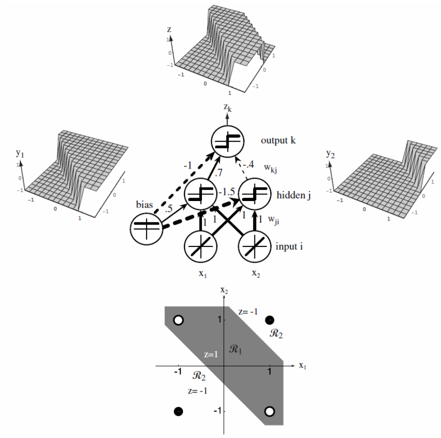
$$f(s) = \begin{cases} 1, & \text{if } s \geq 0 \\ -1, & \text{if } s < 0 \end{cases} .$$

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

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

- Output unit emits

$$z_k = f(net_k)$$

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

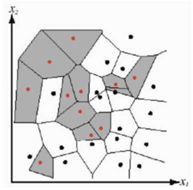
$$g_k(\mathbf{x}) = z_k = f\left(\sum_{j=1}^{n_H} w_{kj} f\left(\sum_{i=1}^{d} w_{ji} x_i + w_{j0}\right) + w_{k0}\right)$$

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

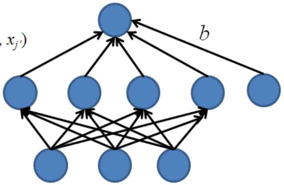


output $g(x) = label_j$

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

hidden $d(x, x_j)$

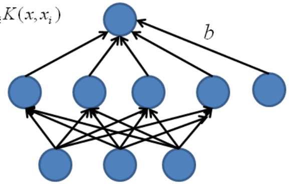
input x



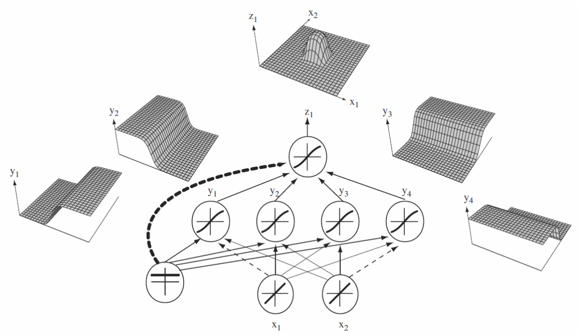
output $g(x) = b + \sum_i \alpha_i K(x, x_i)$

SVM hidden $K(x, x_j)$

input x



Expressive power of a three-layer neural network

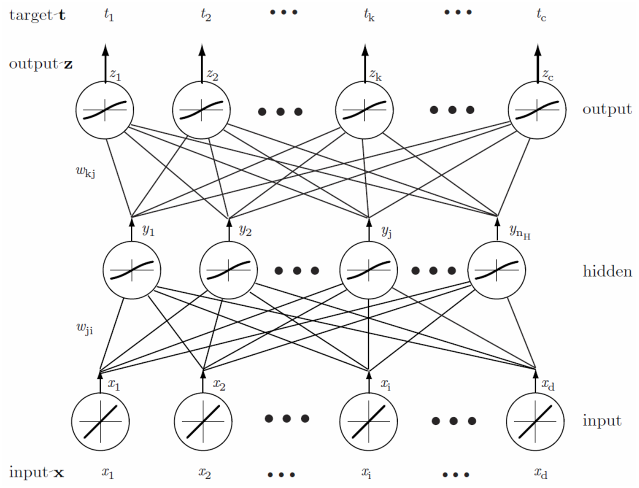


With a tanh activation function $f(s) = (e^s - e^{-s})/(e^s + e^{-s})$, the hidden unit outputs are paired in opposition thereby producing a "bump" at the output unit. With four hidden units, a local mode (template) can be modeled. Given a sufficiently large number of hidden units, any continuous function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

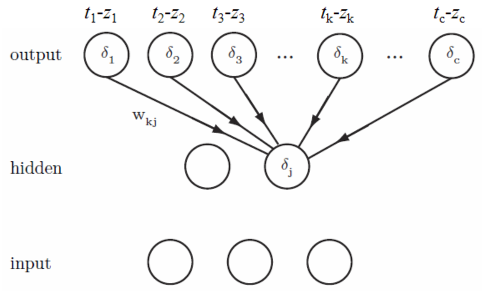
- Sensitivity for a hidden unit j

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

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

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

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

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

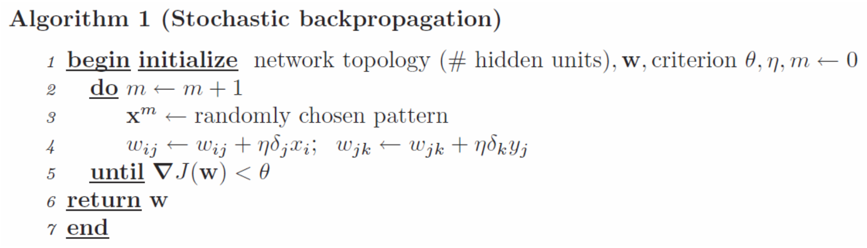
- Batch gradient descent

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

- 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):

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

Stochastic backpropagation



Algorithm 1 (Stochastic backpropagation)

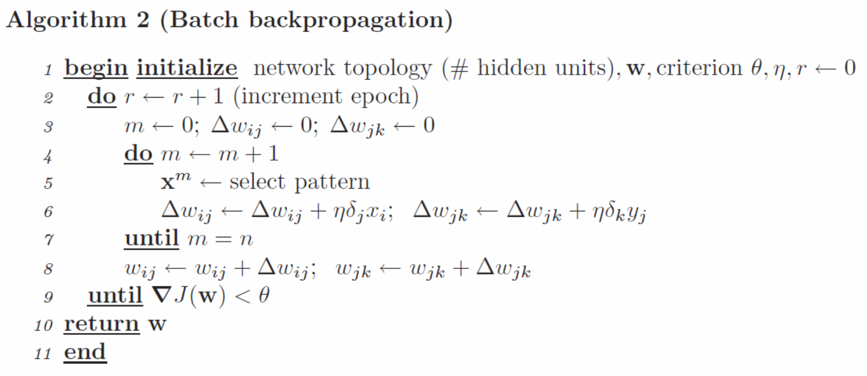
```
1 begin initialize network topology (# hidden units), $\mathbf{w}$, criterion $\theta, \eta, m \leftarrow 0$
2   do $m \leftarrow m + 1$
3       $\mathbf{x}^m \leftarrow$ randomly chosen pattern
4       $w_{ij} \leftarrow w_{ij} + \eta\delta_j x_i; \quad w_{jk} \leftarrow w_{jk} + \eta\delta_k y_j$
5   until $\nabla J(\mathbf{w}) < \theta$
6 return $\mathbf{w}$
7 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



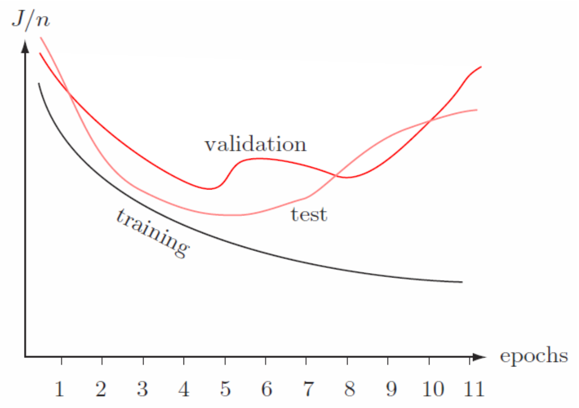
Algorithm 2 (Batch backpropagation)

```
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

Plot learning curves on the training and validation sets



Plot the average error per pattern (i.e. $1/n \sum_p J_p$) versus the number of epochs.

Learning curve on the training set

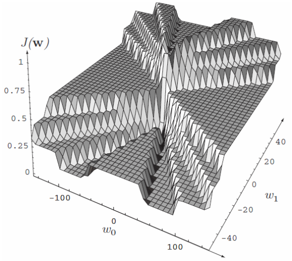
- The average training error typically decreases with the number of epochs and reaches an asymptotic value
- This asymptotic value could be high if **underfitting** happens. The reasons could be
  - The classification problem is difficult (Bayes error is high) and there are a large number of training samples
  - The expressive power of the network is not enough (the numbers of weights, layers and nodes in each layer)
  - Bad initialization and get stuck at local minimum (pre-training for better initialization)
- If the learning rate is low, the training error tends to decrease monotonically, but converges slowly. If the learning rate is high, the training error may oscillate.

Learning curve on the test and validation set

- The average error on the validation or test set is virtually always higher than on the training set. It could increase or oscillate when **overfitting** happen. The reasons could be
  - Training samples are not enough
  - The expressive power of the network is too high
  - Bad initialization and get stuck at local minimum (pre-training for better initialization)
- Stop training at a minimum of the error on the validation set

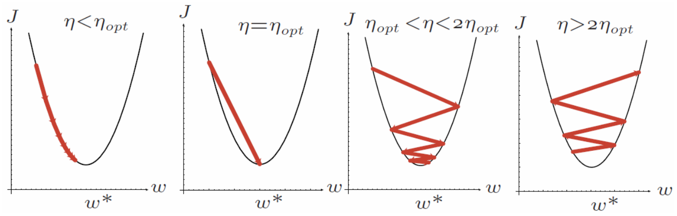
Error surfaces

- Backpropagation is based on gradient descent and tries to find the minimum point of the error surface $\mathbf{J}(\mathbf{w})$
- Generally speaking, it is unlikely to find the global minimum since the error surface is usually very complex
- Backpropagation stops at local minimum and plateaus (regions where error varies only slightly as a function of weights)
- Therefore, it is important to find a good initialization for backpropagation (through pre-training)



Learning rate

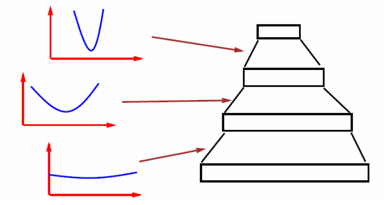
- Decrease the learning rate when the weight vector “oscillates” and increase it when the weight vector follows a steady direction
- One can choose a different learning rate for each weights, so that all the weights in the network converge roughly at the same speed



Gradient descent in a 1D quadratic criterion with different learning rates. The optimal learning rate is found by $\eta_{opt} = \left(\frac{\partial^2 J}{\partial^2 w^2}\right)^{-1}$.

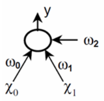
Learning rate

- Learning rates in the lower layers should generally be larger than in the higher layers, since the second derivative is often smaller in the lower layers

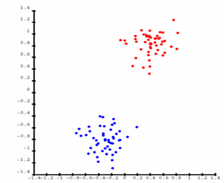


Learning rate

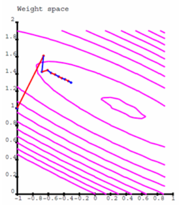
- Example of linear network trained in a batch mode.



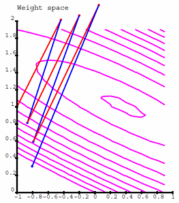
(a)



(b)



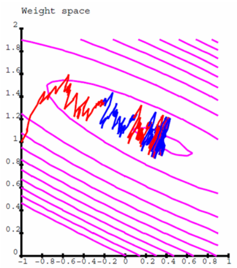
(c) η= 1.5

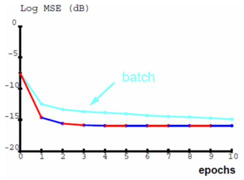


(d) η= 2.5

Learning rate

- Stochastic learning with $\eta = 0.2$





Incorpotation of momentum into stochastic gradient descent

- Error surfaces often have plateaus where there are "to many" weights (especially when the number of layers is large) and thus the error depends only weakly upon any one of them.
- Include some fraction α of the previous weight update in stochastic backpropagation

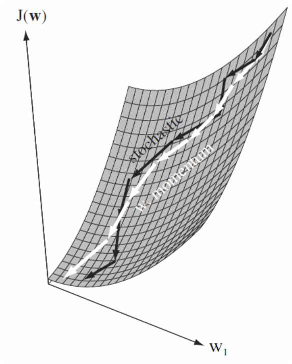
$$\mathbf{w}(m+1) = \mathbf{w}(m) + (1-\alpha)\Delta\mathbf{w}_{bp}(m) + \alpha\Delta\mathbf{w}(m-1)$$

where $\Delta\mathbf{w}_{bp}(m)$ is the change in $\mathbf{w}(m)$ that would be called for by the backpropagation algorithm

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

- Allow the network to learn more quickly when plateaus in the error surface exists

Incorpotation of momentum into stochastic gradient descent



Data augmentation

- If the training set is small, one call synthesize some training samples by adding Gaussian noise to real training samples
- Domain knowledge can be used to synthesize training samples. For example, in image classification, more training images can be synthesized by translation, scaling, and rotation.

Normalizing input

- If the dynamic range of one input feature is much larger than others, during training, the network will mainly adjust weights on this feature while ignore others
- We do not want to prefer one feature over others just because they differ solely measured units
- To avoid such difficulty, the input patterns should be shifted so that the average over the training set of each feature is zero, and then be scaled to have the same variance as 1 in each feature
- Input variables should be uncorrelated if possible
    - If inputs are uncorrelated then it is possible to solve for the value of one weight without any concern for other weights
    - With correlated inputs, one must solve for multiple weights simultaneously, which is a much harder problem
    - PCA can be used to remove linear correlations in inputs

Shuffling the training samples

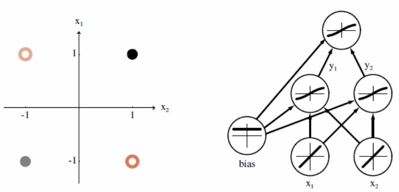
- Networks learn the fatest from the most unexpected sample
- Shuffle the training set so that successive training examples never (rarely) belong to the same class
- Present input examples that produce a large error more frequently than examples that produce a small error
    - This technique applied to data containing outliers can be disastrous because outliers can produce large errors yet should not be presented frequently

Dropout

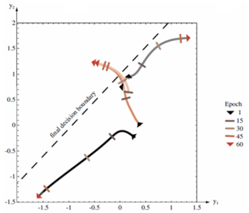
- Radomly set some input features and the ouputs of hidden units as zero during the training process
- Feature co-adaptation: a feature is only helpful when other specific features are present
    - Because of the existence of noise and data corruption, some features or the responses of hidden nodes can be misdetected
- Dropout prevents feature co-adaptation and can significantly improve the generalization of the trained network
- Can be considered as another approach to regularization
- It can be viewed as averaging over many neural networks
- Slower convergence

Nonlinear feature mapping

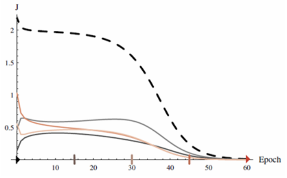
(a)



(b)



(c)

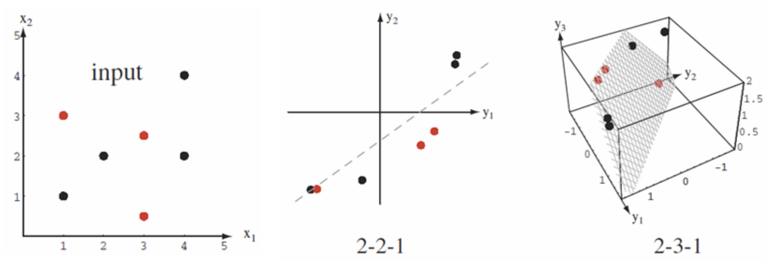


(a) 2-2-1 backpropagation network and the four patterns of the XOR problem;
(b) Outputs of the hidden units for each of the four patterns; these outputs move across the y_1y_2-space as the network learns;
(c) Learning curves of individual patterns and total error as a function of epoch

Nonlinear feature mapping

- The multilayer neural networks provide nonlinear mapping of the input to the feature representation at the hidden units
- With small initial weights, the net activation of each hidden unit is small, and thus the linear portion of their activation function is used. Such a linear transformation from $\mathbf{x}$ to $\mathbf{y}$ leaves the patterns linearly inseparable in the XOR problem.
- As learning progresses and the input-to-hidden weights increase in magnitude, the nonlinearities of the hidden units warp and distort the mapping from input to the hidden unit space
- The linear decision boundary at the end of learning found by the hidden-to-output weights is shown by the straight dashed line; the nonlinearly separable problem at the inputs is transformed into a linearly separable at the hidden units

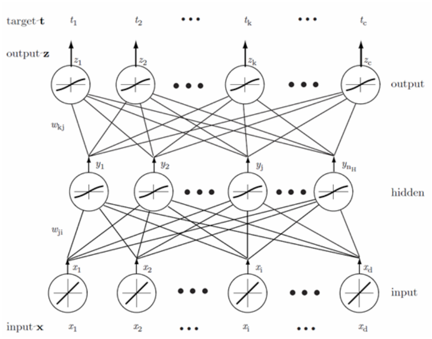
Nonlinear feature mapping

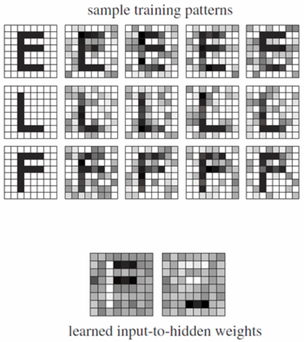


- The expressive power of the 2-2-1 network is not high enough to separate all the seven patterns, even after the global minimum error is reached by training.
- These patterns can be separated by increasing one more hidden unit to enhance the expressive power.

Filter learning

- The input-to-hidden weights at a single hidden unit describe the input patterns that leads to maximum activation of that hidden unit, analogous to a “matched filter”
- Hidden units find feature groupings useful for the linear classifier implemented by the hidden-to-output layer weights





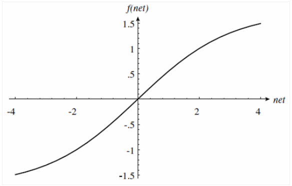
Filter learning

- The top images represent patterns from a large training set used to train a 64-2-3 neural network for classifying three characters
- The bottom figures show the input-to-hidden weights, represented as patterns, at the two hidden units after training
- One hidden unit is tuned to a pair of horizontal bars while the other is tuned to a single lower bar
- Both of these feature groups are useful building blocks for the pattern presented

A recommended sigmoid function for activation function

$$f(x) = 1.79159 \frac{e^{\frac{2}{3}x} - e^{-\frac{2}{3}x}}{e^{\frac{2}{3}x} + e^{-\frac{2}{3}x}}$$

$f(\pm 1) = \pm 1$, liner in the range of $-1 < x < 1$, $f''(x)$ has extrema near $x = \pm 1$.



Y. LeCun, Generalization and network design strategies. *Proc. Int'l Cibf, Cinectuibusn ub Oersoectuve*, 1988.

Desirable properties of activation functions

- Must be nonlinear: otherwise it is equivalent to a linear classifier
- Its output has maximum and minimum value: keep the weights and activations bounded and keep training time limited
  - Desirable property when the output is meant to represent a probability
  - Desirable property for models of biological neural networks, where the output represents a neural firing rate
  - May not be desirable in networks for regression, where a wide dynamic range may be required
- Continuous and differentiable **almost everywhere**
- Monotonicity: otherwise it introduces additional local extrema in the error surface
- Linearity for a small value of *net*, which will enable the system to implement a linear model if adequate for yielding low error

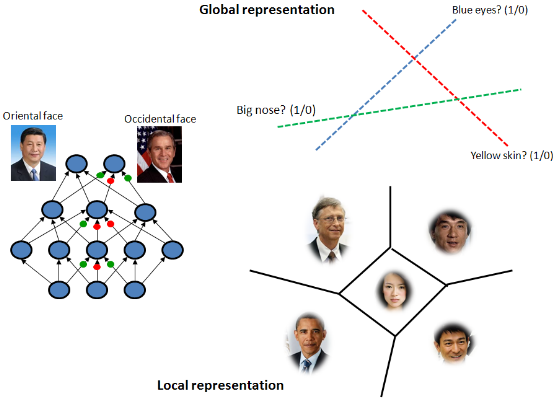
Desirable properties of activation functions

- The average of the outputs at a node is close to zero because these outputs are the inputs to the next layer
- The variance of the outputs at a node is also 1.

Global versus local representations

- Tanh function (shifting the center of *sigmoid* to 0) has all the properties above
  - It has large response for input in a large range. Any particular input **x** is likely to yield activity through several hidden units. This affords a **distributed** or **global** representation of the input.
  - If hidden units have activation functions that have significant response only for input within a small range, then an input **x** generally leads to fewer hidden units being active - a **local** representation.
  - It is often in practice that when there are few training points, distributed representations are superior because more of the data influences the posteriors at any given input region.
  - The global representation can be better achieved with more layers

Global versus local representations



Choosing target values

- Avoid setting the target values as the sigmoid's asymptotes
  - Since the target values can only be achieved asymptotically, it drives the output and therefore the weights to be very large, which the sigmoid derivative is close to zero. So the weights may become stuck.
  - When the outputs saturate, the network gives no indication of confidence level. Large weights fore all outputs to the tails of the sigmoid instead of being close to decision boundary.
- Insure that the node is not restricted to only the linear part of the sigmoid
- Choose target values at the point of the maximum second derivative on the sigmoid so as to avoid saturating the output units

Initializing weights

- Randomly initialize weights in the liner region. But they should be large enough to make learning proceed.
  - The network learns the linear part of the mapping before the more difficult nonlinear parts
  - If weights are too small, gradients are small, which makes learning slow
- To obtain a standard deviation close to 1 at the output of the first hidden layer, we just need to use the recommended sigmoid and require that the input to the sigmoid also have a standard deviation $\sigma_y = 1$. Assuming the inputs to a unit are uncorrelated with variance 1, the standard deviation of σ_{y_i} is

$$\sigma_{y_i} = (\sum_{j=1}^{m} w_{ij}^2)^{1/2}$$

Initializing weights

- To ensure $\sigma_{y_i} = 1$, weights should be randomly drawn from a distribution (e.g. uniform) with mean zero and standard deviation

$$\sigma_w = m^{-1/2}$$

Recommended readings

- R. O. Duda, P. E. Hart, and D. G. Stork, “Pattern Classification,” Chapter 6, 2000.
- Y. LeCun, L. Bottou, G. B. Orr, and K. Muller, “Efficient BackProp,” Technical Report, 1998.