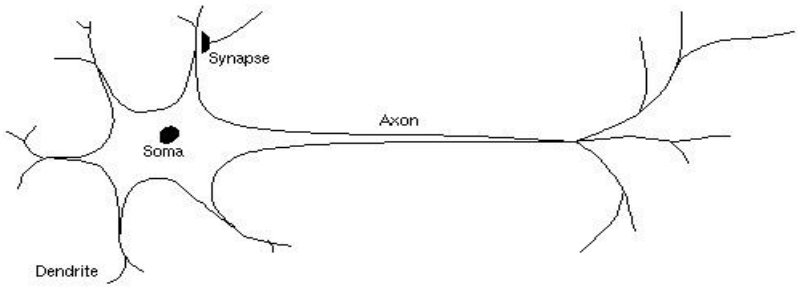
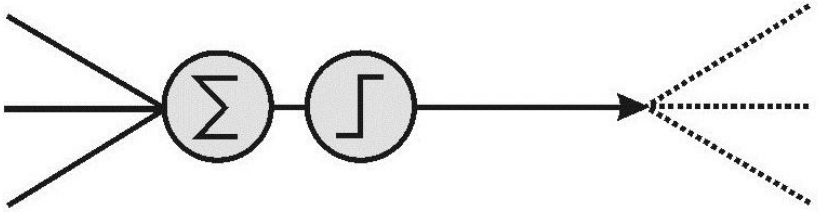
Artificial Neural Networks

By: Mikhail Davydov

Concept Overview

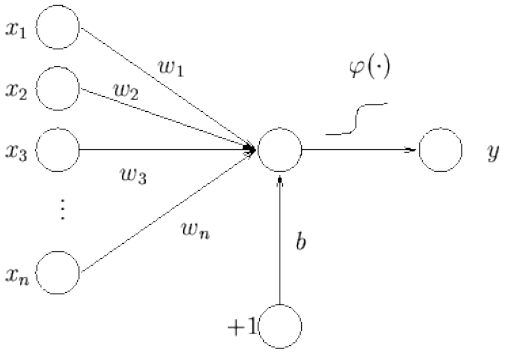
- Map input values to values/classes in the output space
- Functional form relating input and desired output is not known, nor are any assumptions about it are made in the design of the net
  - Nonparametric Regression [typically needs larger sample size, data runs the whole show, no model hypothesis]
  - Focus on the network perfor
    statistical characteristics of

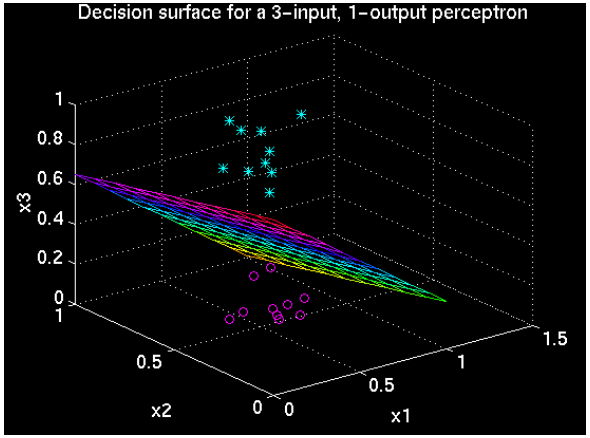


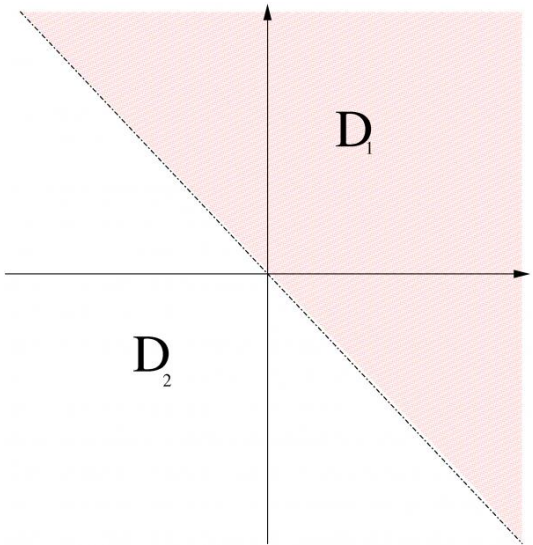


Single Perceptron Overview

- The inputs are classified into two regions
  - n-dimensional input space is divided by a hyperplane into two decision regions







- 2D single perceptron decision boundary

$$Plane: \sum_{i=1}^{n} w_i x_i - \theta = 0$$

$$w_1 x_1 + w_2 x_2 - \theta = 0$$

$$x_2 = -\frac{w_1}{w_2} x_1 + \theta$$

Single Perceptron Learning

- Weights are adjusted to reduce the difference between the actual and desired outputs

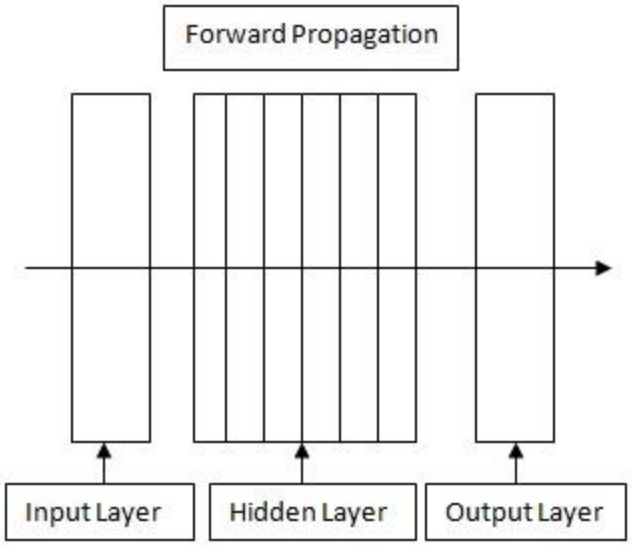
  - Error on output: $e(p) = Y_d(p) - Y(p)$
  - Learning rule: $w_i(p+1) = w_i(p) + \alpha\, x_i(p) \cdot e(p)$

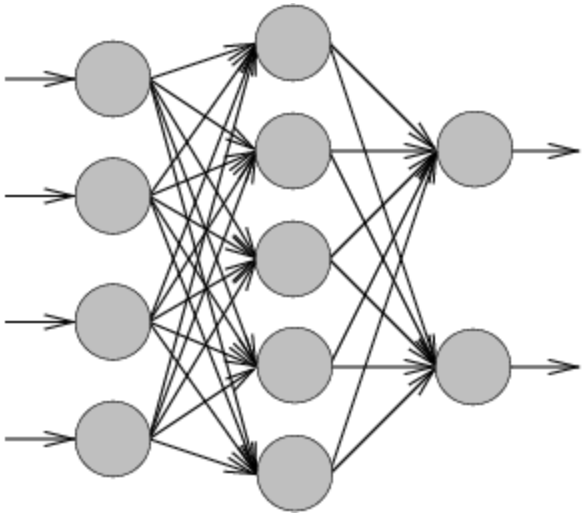
    Can be used to derive the learning algorithm
  - **Initialization:** Weights are initialized randomly (usually in [-0.5, 0.5])
  - **Activation:** $Y(p) = step/logsig[\sum_{i=1}^{n} w_i x_i - \theta]$

    Step/exposure p in training
  - **Delta rule:** $\Delta w_i(p) = \alpha x_i(p) \cdot e(p) \;\rightarrow\; w_i(p+1) = w_i(p) + \Delta w_i(p)$
  - **Iteration:** p = p + 1 → Activate the neuron with new input vector $\vec{x}$ and update the weights as needed. Repeat until e(p) is zero or has reached the desired characteristics.

Multilayer Perceptron

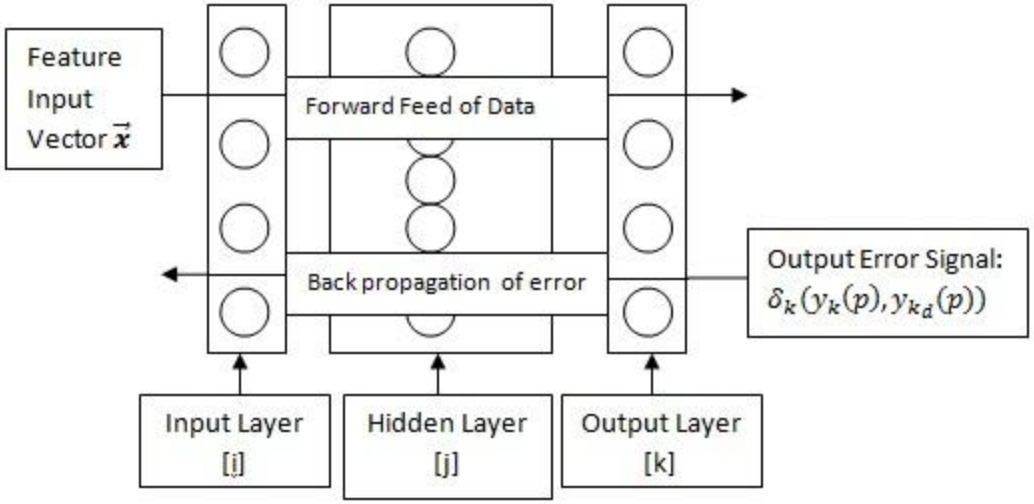
- The layers between the input layer and output layer constitute the hidden layer
  - So called since we don't really have nor need an idea about what the outputs of those layers should look like
  - They are there to add dimensionality to the solution to deal with the complexity of the problem as needed





Backward Propagation Algorithm

- Two phase learning:
  - Feeding the network forward
  - Calculate the error starting at the output layer and modify the weights on the way to the input layer as the output error signal is backward propagated
- The weights are randomly assigned as before (the range implemented can be optimized)



Activation and Training

Activation

a) Input data activates the hidden layer:

$$.y_j(p) = sigmoid[\sum_{i=1}^{n} x_i(p) \cdot w_{ij}(p) - \theta_j]$$

b) The output neurons process the hidden layer data:

$$.y_k(p) = sigmoid[\sum_{j=1}^{m} x_j(p) \cdot w_{jk}(p) - \theta_k]$$

Training

- Update of weights connecting the output to the hidden layer
  - a) Error gradient: $\delta_k(p) = y_k(p)[1 - y_k(p)]e_k(p)$
  - b) Weight update: $\Delta w_{jk}(p) = \alpha y_j(p) \cdot \delta_k(p)$

    $$.w_{jk}(p+1) = w_{jk}(p) + \Delta w_{jk}(p)$$
- Hidden Layer input weight training
  - c) Error gradient: $\delta_j(p) = y_j(p)[1 - y_j(p)] \cdot \sum_{k=1}^{l} \delta_k(p) \cdot w_{jk}(p)$
  - d) Weight update: $\Delta w_{ij}(p) = \alpha x_i(p) \cdot \delta_j(p)$

    $$.w_{ij}(p+1) = w_{ij}(p) + \Delta w_{ij}(p)$$
- Iteration
  - e) Set p = p+1 and populate the network with $w_{jk}(p+1)$ and $w_{ij}(p+1)$
  - f) Repeat until satisfied with error

The Credit Assignment Problem

- Which weight should be modified and by how much?
  - Error must be defined and propagated
  - Final feature saliency must be determined
- Solution [in attempt to generalize]:
  - Use of a monotonic continuous transfer function allows for a neat solution of the above tasks
  - By assuming the final output error [at iteration p] of the form:

$$E_p = \frac{1}{2}\sum_{j=1}^{N^L}(t_{pj} - y_{pj})^2, where\ y_{pj} = o_{pj}^L\left(\sum_{i=1}^{N^{L-1}} w_{ji}^L \cdot o_{pi}^{L-1} + \theta_{pj}^L\right)$$

Net Stimulus

we can construct a weight updating mechanism for the whole network through gradient descent minimization.

  - This is plausible because the output of each of the final layer neurons is the result of weighted processing of all of the neurons in the layers prior to its own

Error Gradient [output layer]

After all (P) training iterations have been completed, the total error is:

$$E = \sum_{p=1}^{P} E_p$$

We are to minimize this error with respect to the weights in the layers (l):

$$\frac{\partial E}{\partial w_{ji}^{l}} = \sum_{p=1}^{P} \frac{\partial E_p}{\partial w_{ji}^{l}} \quad \rightarrow \quad \frac{\partial E_p}{\partial w_{ji}^{l}} = \frac{\partial E_p}{\partial net_{pj}^{l}} \frac{\partial net_{pj}^{l}}{\partial w_{ji}^{l}}$$

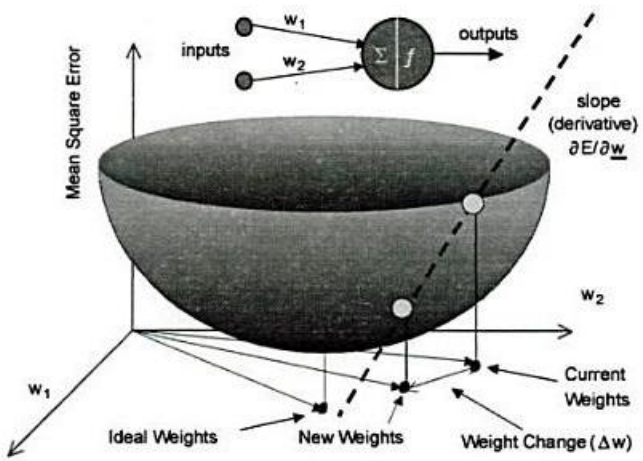


Figure 1.10 Gradient descent for mean-squared-error cost function.

$$\frac{\partial net_{pj}^{l}}{\partial w_{ji}^{l}} = \frac{\partial}{\partial w_{ji}^{l}} \sum_{k=1}^{N^{l-1}} w_{ji}^{l} \cdot o_{pk}^{l-1} + \theta_{pj}^{l} = o_{pi}^{l-1}; \quad \frac{\partial E_p}{\partial net_{pj}^{l}} = -\delta_{pj}^{l} \quad \rightarrow \quad \frac{\partial E_p}{\partial w_{ji}^{l}} = -\delta_{pj}^{l} o_{pi}^{l-1}$$

Setting l = L and working back into the network we have:

$$-\delta_{pj}^{L} = \frac{\partial E_p}{\partial net_{pj}^{L}} = \frac{\partial E_p}{\partial o_{pj}^{L}} \left(\frac{\partial o_{pj}^{L}}{\partial net_{pj}^{L}} = \dot{T}_j^{L}(net_{pj}^{L}) \right);$$

$$\frac{\partial E_p}{\partial o_{pj}^{L}} = \frac{\partial}{\partial o_{pj}^{L}} \left[\frac{1}{2} \sum_{j=1}^{N^{L}} \left(t_{pj} - o_{pj}^{L} \right)^2 \right] = -\left(t_{pj} - o_{pj}^{L} \right) = -e_j(p) \quad \rightarrow \quad \delta_{pj}^{L} = \dot{T}_j^{L}(net_{pj}^{L}) e_j(p)$$

Error Gradient [inner layers]

Hidden layers ($l < L$):

$$\frac{\partial E_p}{\partial o_{pj}^{l}} = \sum_{k=1}^{N^{l+1}} \left(\frac{\partial E_p}{\partial net_{pk}^{l+1}} \frac{\partial net_{pk}^{l+1}}{\partial o_{pj}^{l}} \right)$$

$$\frac{\partial E_p}{\partial net_{pk}^{l+1}} = -\delta_{pk}^{l+1}; \quad \frac{\partial net_{pk}^{l+1}}{\partial o_{pj}^{l}} = \frac{\partial}{\partial o_{pj}^{l}} \left(\sum_{i=1}^{N^{l}} w_{ki}^{l+1} \cdot o_{pi}^{l} + \theta_{pk}^{l+1} \right) = w_{kj}^{l+1} \quad \rightarrow \quad \frac{\partial E_p}{\partial o_{pj}^{l}} = -\sum_{k=1}^{N^{l+1}} \delta_{pk}^{l+1} \cdot w_{kj}^{l+1}$$

This is reflects the fact that a change in a given layer affects the layers after it.

$$\therefore \delta_{pj}^{l} = -\frac{\partial E_p}{\partial net_{pj}^{l}} = -\frac{\partial E_p}{\partial o_{pj}^{l}} \left(\frac{\partial o_{pj}^{l}}{\partial net_{pj}^{l}} = \dot{T}_j^{l}(net_{pj}^{l}) \right) = \dot{T}_j^{l}(net_{pj}^{l}) \cdot \sum_{k=1}^{N^{l+1}} \delta_{pk}^{l+1} \cdot w_{kj}^{l+1} \; for \; l < L$$

The backward propagation learning algorithm can now be seen as the natural choice for MLP networks.

Weight Updating Scheme

- $\frac{\partial E_p}{\partial w_{ji}^l}$ Gives the direction in which the error reduces
- The learning rate scales how fast we get to the minimum:

$$\Delta w_{pji}^l = -\eta \frac{\partial E_p}{\partial w_{ji}^l} = \eta \delta_{pj}^l T_{pi}^{l-1}$$

  - Eta too low – the algorithm takes very long to complete
  - Eta too high – the algorithm will keep overshooting the desired degree of error in attempt to minimize
- In essence, eta sets the learning step granularity
- The weights can either be updated with each iteration (online) or at the end of the last iteration (batch)
- For batch learning:

$$\Delta w_{ji}^l{}_{algo} = \sum_{p=1}^{\mu} \Delta w_{pji}^l$$

Weight Updating Scheme

- Iteration momentum (Heavy Ball Method):
  - Adds to the progression toward the MSE absolute minimum
  - Filters variations in the error surface (local extremes)
  - Affects learning acceleration

$$\Delta w_{ji}^{l}(p) = \eta \delta_{pj}^{l} T_{pi}^{l} + \alpha \Delta w_{ji}^{l}(p-1)$$

- Exponential Smoothing:

$$\Delta w_{ji}^{l}(p) = (1-\sigma)\eta \delta_{pj}^{l} T_{pi}^{l} + \sigma \Delta w_{ji}^{l}(p-1)$$

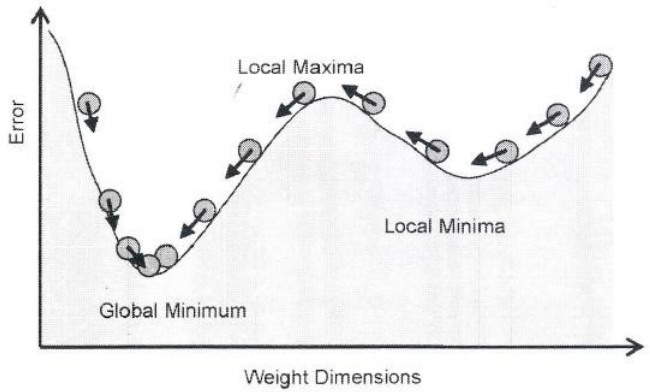


Figure A.2 The use of momentum is illustrated. Think of it as a heavy-ball analogy: A heavy ball rolling downhill will roll up and over a small hill, representing a local maxima.

Feature Saliency

A trained network can be used to determine which features are important

Simplest intuitive metric:

$$\Lambda_i = \sum_{j=1}^{N^l} |w_{ji}^l|$$

Upon further analysis, it can be seen that Λ_i "communicates" the Bayesian nature of neural networks

With enough hidden layer neurons, the training data PDF can be modeled

The output neurons approximate the posterior probabilities of the classes in the problem

$$z_j = P(C_j|\vec{x}) \quad \rightarrow \quad P_{error}(j,\vec{x}) = 1 - \sum_j P(C_j|\vec{x}) = \sum_{k \neq j} z_k$$

The sensitivity of a given output to some feature is given by

$$\frac{\partial P_{error}(j,\vec{x})}{\partial x_i} = \frac{\partial}{\partial x_i} \sum_{k \neq j} z_k = \frac{\partial}{\partial x_i} \sum_{k \neq j} T_k^l \left(\sum_{m=1}^{N^{l-1}} w_{km}^l \cdot o_m^{l-1} + \theta_k^l \right)$$

Feature analysis for a three layer network [with a sigmoid transfer function]:

$$\frac{\partial P_{error}(j,\vec{x})}{\partial x_i} = \frac{\partial}{\partial x_i}\sum_{k\neq j} T_k^3\Big(\sum_{m=1}^{N^2} w_{km}^3 \cdot o_m^2 + \theta_k^3\Big) = \sum_{k\neq j} z_k(1-z_k)\cdot \sum_{m=1}^{N^2} w_{km}^3 \cdot \frac{\partial}{\partial x_i} o_m^2(net_m^2)$$

$$\frac{\partial P_{error}(j,\vec{x})}{\partial x_i} = \sum_{k\neq j} \delta_k^3 \cdot \sum_{m=1}^{N^2} w_{km}^3\, \delta_m^2 \cdot \sum_{n=1}^{N^1} w_{mn}^2\, \delta_n^1 \cdot w_{ni}^1$$

A saliency metric can now be defined from the sensitivity of outputs to input features:

$$\Omega_i = \sum_j \sum_{\vec{x}\epsilon S} \sum_{x_i \epsilon D_i} \left| \frac{\partial}{\partial x_i} \sum_{k\neq j} z_k \right|$$

Using the triangle inequality:

$$\Omega_i \leq \sum_j \sum_{\vec{x}\epsilon S} \sum_{x_i \epsilon D_i} \sum_{k\neq j} \left| \delta_k^3 \cdot \sum_{m=1}^{N^2} w_{km}^3\, \delta_m^2 \cdot \sum_{n=1}^{N^1} w_{mn}^2\, \delta_n^1 \cdot w_{ni}^1 \right|$$

All of the arguments of the norm are vector elements, so the triangle inequality can be used to further simplify:

$$\Omega_i \leq \sum_j \sum_{\vec{x}\epsilon S} \sum_{x_i \epsilon D_i} \sum_{k\neq j} \delta_k^3 \cdot \sum_{m=1}^{N^2} \left| w_{km}^3 \right| \delta_m^2 \cdot \sum_{n=1}^{N^1} \left| w_{mn}^2 \right| \delta_n^1 \cdot \left| w_{ni}^1 \right| \sim \Lambda_i$$

Advantages and Disadvantages of the Backpropagation Algorithm

- Few parameters to tune
- Easy to implement
- Fairly universal in its applications
- Architecture optimization is hard to determine for a problem of given complexity
- No training session memory
- Can be unnecessarily slow

Sources

- Kevin L. Priddy: Artificial Neural Networks, An Introduction
- C. Bishop: Neural Networks for Pattern Recognition