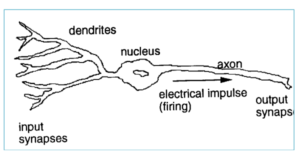
Machine Learning: Neural Networks

CSE 415: Introduction to Artificial Intelligence
University of Washington
Winter, 2019

Outline

- History of neural networks research
- The Perceptron
- Examples
- Training algorithm
- Fundamental training theorem
- Two-level perceptrons
- 2-Level feedforward neural nets with sigmoid activation functions
- Backpropagation and the Delta rule.

The Biological Neuron



The human brain contains approximately 10^{11} neurons.
Activation process:
Inputs are transmitted electrochemically across the input synapses
Input potentials are summed.
If the sum reaches a threshold, a pulse moves down the axon. (The neuron has "fired".)
The pulse is distributed at the axonal arborization to the input synapses of other neurons.
After firing, there is a refractory period of inactivity.

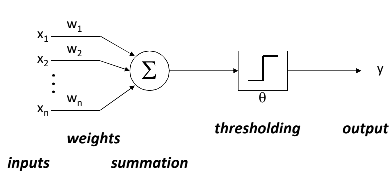
History of Neural Networks Research

1943 McCulloch & Pitts model of neuron.
$n_i(t+1) = \Theta(\Sigma_j w_{ij} n_j(t) - \mu_i)$, $\Theta(x) =$ 1 if $x \geq 0$; 0, otherwise.
1962 Frank Rosenblatt's book gives a training algorithm for finding the weights w_{ij} from examples.
Principles of neurodynamics: Perceptrons and the theory of brain mechanisms

1969 Marvin Minsky and Seymour Papert publish *Perceptrons*, and prove that 1-layer perceptrons are incapable of computing image connectedness.
1974-89, 1982: Associated content-addressable memory.

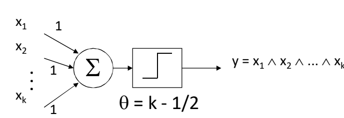
Backpropagation: Werbos 1974, Parker 1985, Rumelhart, Hinton, & Williams 1986.

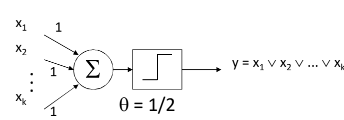
The Perceptron



$$y = \begin{cases} 1 & \text{if } \Sigma w_i x_i \geq \theta; \\ 0, & \text{otherwise.} \end{cases}$$

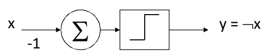
Perceptron Examples: Boolean AND and OR.





$x_i \in \{0, 1\}$

Perceptron Examples: Boolean NOT



$\theta = -1/2$

$x_i \in \{0, 1\}$

Perceptron Example: Template Matching

-1	-1	1	-1	-1
-1	1	-1	1	-1
1	1	1	1	1
1	-1	-1	-1	1
1	-1	-1	-1	1

weights w_1 through w_{25}

$x_i \in \{-1, 1\}$

$\theta = 25 - \varepsilon$

Recognizes the letter A provided the exact pattern is present.

Perceptron Training Sets

Let $X = X^+ \cup X^-$ be the set of training examples.

$S_X = \langle X_1, X_2, ..., X_k, ... \rangle$ is a *training sequence* on X, provided:

(1) Each X_k is a member of X, and

(2) Each element of X occurs infinitely often in S_X.

An element *e* occurs *infinitely often* in a sequence

$z = \langle z_1, z_2, ... \rangle$

provided that for any nonzero integer *i*, there exists a nonnegative integer *j* such that there is an occurrence of *e* in

$z_i, z_{i+1}, \ldots, z_j$.

Perceptron Training Algorithm

Let $X = X^+ \cup X^-$ be the set of training examples. and let $S_X = \langle X_1, X_2, ..., X_k, ... \rangle$ be a training sequence on X.

Let w_k be the weight vector at step *k*.

Choose w_0 arbitrarily. For example. $w_0 = (0, 0, ..., 0)$.

Each each step *k*, *k* = 0, 1, 2, . . .

Classify X_k using w_k.

If X_k is correctly classified, take $w_{k+1} = w_k$.

If X_k is in X^- but misclassified, take $w_{k+1} = w_k - c_k X_k$.

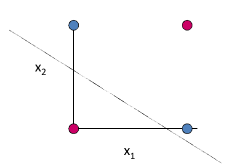
If X_k is in X^+ but misclassified, take $w_{k+1} = w_k + c_k X_k$.

The sequence c_k should be chosen according to the data. Overly large constant values can lead to oscillation during training. Values that are too small will increase training time. However, $c_k = c_0/k$ will work for any positive c_0.

Perceptron Limitations

Perceptron training always converges if the training data X^+ and X^- are linearly separable sets.

The boolean function XOR (exclusive or) is not linearly separable. (Its positive and negative instances cannot be separated by a line or hyperplane.) It cannot be computed by a single-layer perceptron. It cannot be learned by a single-layer perceptron.

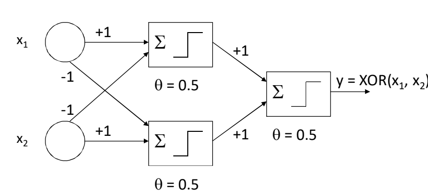


$X^+ = \{ (0, 1), (1, 0) \}$

$X^- = \{ (0, 0), (1, 1) \}$

$X = X^+ \cup X^-$

Two-Layer Perceptrons



Two-Layer Perceptrons (cont.)

Two-Layer perceptrons are computationally powerful.

However: they are not trainable with a method such as the perceptron training algorithm, because the threshold units in the middle level “block” updating information; there is no way to know what the correct updates to first-level weights should be.

Two-Layer Feedforward Networks with Sigmoid Activation Functions

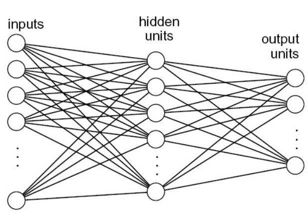
We get: the power of 2-level perceptrons,

plus

the trainability of 1-level perceptrons (well, sort of).

These are sometimes called (a) “backpropagation networks,” (because the training method is called backpropagation) and (b) “two-layer feedforward neural networks.”

Structure of a Backprop. Network



Hidden Node Input Activation

As with perceptrons, a weighted sum is computed of values from the previous level:

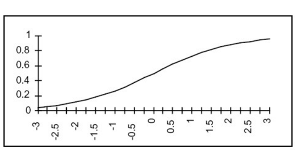
$$h_j = \Sigma_i \, w_{ij} \, x_i$$

However the hidden node does not apply a threshold, but a sigmoid function ...

Sigmoid Activation Functions

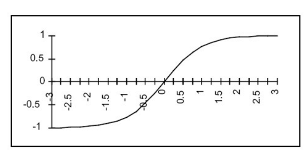
Instead of using threshold functions, which are neither continuous nor differentiable, we use a *sigmoid* function, which is a sort of smoothed threshold function.

$$g_1(h) = 1/(1 + e^{-h})$$



An Alternative Sigmoid Func.

$$g_2(h) = \tanh(h) = (e^h - e^{-h})/(e^h + e^{-h})$$



Sigmoid Function Properties

Both g_1 and g_2 are continuous and differentiable.

$$g_1(h) = 1/(1 + e^{-h})$$

$$g_1'(h) = g_1(h)\,(1 - g_1(h))$$

$$g_2(h) = \tanh(h) = (e^h - e^{-h})/(e^h + e^{-h})$$

$$g_2'(h) = 1 - g_2(h)^2$$

Training Algorithm

Each *training example* has the form $\langle X_i, T_i\rangle$, were X_i is the vector of inputs, and T_i is the desired corresponding output vector.

An *epoch* is one pass through the training set, with an adjustment to the networks weights for each training example. (Use the "delta rule" for each example.)

Perform as many epochs of training as needed to reduce the classification error to the required level.

If there are not enough hidden nodes, then training might not converge.

Delta Rule

For each training example $\langle X_i, T_i\rangle$, Compute $F(X_i)$, the outputs based on the current weights.

To update a weight w_{ij}, add ∇w_{ij} to it, where

$$\nabla w_{ij} = \eta\, \delta_j\, F_j \qquad (\eta \text{ is the training rate.})$$

If w_{ij} leads to an output node, then use

$$\delta_j = (t_j - F_j)\, g'_j(h_j)$$

If w_{ij} leads to a hidden node, then use "backpropagation":

$$\delta_j = g'_j(h_j)\, \Sigma_k\, \delta_k\, w_{kj}$$

The δ_k in this last formula comes from the output level, as computed above.

Performance of Backpropagation

Backpropagation is slow compared with 1-layer perceptron training.

The training rate η can be set large near the beginning and made smaller in later epochs.

In principle, backpropagation can be applied to networks with more than one layer of hidden nodes, but this slows the algorithm much more.

By "ganging" weights together into groups, the number of free parameters to learn can be greatly reduced, making multilayer ("deep") neural networks learnable in practice. In computer vision, the weights in the first 1 to 3 levels can be considered to be convolutional filtering kernels. Such systems are called *Deep Convolutional Neural Nets*.

Setting the Number of Hidden Nodes

The number of nodes in the hidden layer affects generality and convergence.

If too few hidden nodes: convergence may fail.

Few but not too few nodes: possibly slow convergence but good generalization

Too many hidden nodes: Rapid convergence, but "overfitting" happens.

Overfitting: the learned network handles the training set, but fails to generalize effectively to similar examples not in the training set.

Applications of 2-Layer Feedforward Neural Networks

These networks are very popular as trainable classifiers for a wide variety of pattern data.

Examples:

•Speech recognition and synthesis

•Visual texture classification

•Optical character recognition

•Control systems for robot actuators