Sections 18.6 and 18.7
Analysis of Artificial Neural Networks

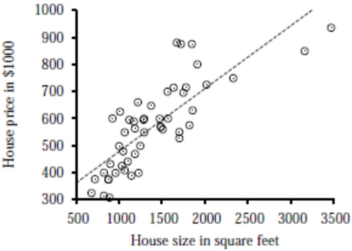
CS4811 - Artificial Intelligence

Nilufer Onder
Department of Computer Science
Michigan Technological University

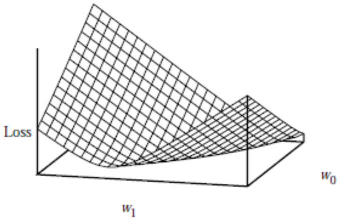
Outline

Univariate linear regression problem

- A *univariate linear function* is a straight line with input x and output y.
- The problem is to "learn" a univariate linear function given a set of data points.
- Given that the formula of the line is $y = w_1 x + w_0$, what needs to be learned are the weights w_0, w_1.
- Each possible line is called a *hypothesis*:
  $h_{\vec{w}} = w_1 x + w_0$



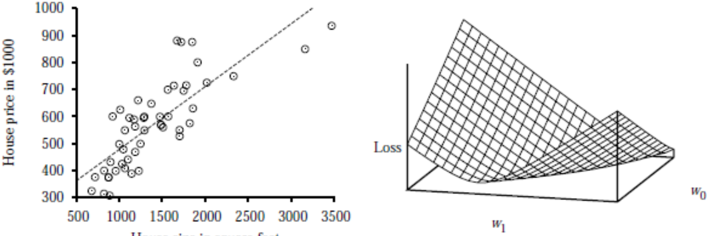
(a)



(b)

Univariate linear regression problem (cont'd)

- There are an infinite number of lines that "fit" the data.
- The task of finding the line that best fits these data is called *linear regression*.
- "Best" is defined as minimizing "loss" or "error."
- A commonly used loss function is the L_2 norm where $Loss(h_{\vec{w}}) = \sum_{j=1}^{N} L_2(y_j, h_{\vec{w}}(x_j)) = \sum_{j=1}^{N}(y_j - h_{\vec{w}}(x_j))^2 = \sum_{j=1}^{N}(y_j - (w_1 x_j + w_0))^2$.



(a) (b)

Minimizing loss

- Try to find $\vec{w}^* = \text{argmin}_{\vec{w}} Loss(h_{\vec{w}})$.
- To mimimize $\sum_{j=1}^{N}(y_j - (w_1 x_j + w_0))^2$, find the partial derivatives with respect to w_0 and w_1 and equate to zero.
- $\frac{\partial}{\partial w_0} \sum_{j=1}^{N}(y_j - (w_1 x_j + w_0))^2 = 0$
- $\frac{\partial}{\partial w_1} \sum_{j=1}^{N}(y_j - (w_1 x_j + w_0))^2 = 0$
- These equations have a unique solution:
  $w_1 = \frac{N(\sum x_j y_j) - (\sum x_j)(\sum y_j)}{N(\sum x_j^2) - (\sum x_j)^2)}$
  $w_0 = (\sum y_j - w_1(\sum x_j))/N$.
- Univariate linear regression is a “solved” problem.

Beyond linear models

- The equations for minimum loss no longer have a closed-form solution.
- Use a *hill-climbing* algorithm, *gradient descent*.
- The idea is to always move to a neighbor that is "better."
- The algorithm is:
  $\vec{w} \leftarrow$ any point in the parameter space
  loop until convergence **do**
  for each w_i **in** $\vec{w}$ **do**
  $w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} Loss(\vec{w})$
- α is called the *step size* or the *learning rate*.

Solving for the linear case

$$\frac{\partial}{\partial w_i} Loss(\vec{w}) = \frac{\partial}{\partial w_i}(y - h_{\vec{w}}(x))^2$$
$$= 2(y - h_{\vec{w}}(x)) \times \frac{\partial}{\partial w_i}(y - h_{\vec{w}}(x))$$
$$= 2(y - h_{\vec{w}}(x)) \times \frac{\partial}{\partial w_i}(y - (w_1 x + w_0))$$

For w_0 and w_1 we get: $\frac{\partial}{\partial w_0} Loss(\vec{w}) = -2(y - h_{\vec{w}}(x))$
$\frac{\partial}{\partial w_1} Loss(\vec{w}) = -2(y - h_{\vec{w}}(x)) \times x$

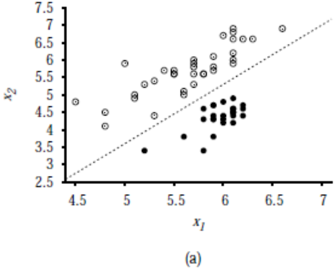
The learning rule becomes:
$w_0 \leftarrow w_0 + \alpha \sum_j (y - h_{\vec{w}}(x))$ and
$w_1 \leftarrow w_1 + \alpha \sum_j (y - h_{\vec{w}}(x)) \times x$

Batch gradient descent

For N training examples, minimize the sum of the individual losses for each example:
$w_0 \leftarrow w_0 + \alpha \sum_j (y_j - h_{\vec{w}}(x_j))$ and
$w_1 \leftarrow w_1 + \alpha \sum_j (y_j - h_{\vec{w}}(x_j)) \times x_j$

- Convergence to the unique global minimum is guaranteed as long as a small enough α is picked.
- The summations require going through all the training data at every step, and there may be many steps
- Using *stochastic gradient descent* only a single training point is considered at a time, but convergence is not guaranteed for a fixed learning rate α.

Linear classifiers with a hard threshold

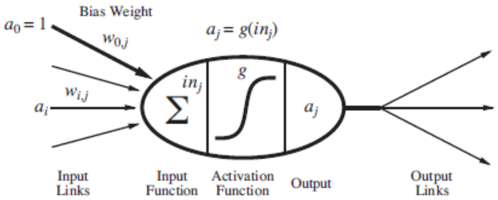


- The plots show two seismic data parameters, body wave magnitude x_1 and surface wave magnitute x_2.
- Nuclear explosions are shown as black circles. Earthquakes (not nuclear explosions) are shown as white circles.
- In graph (a), the line separates the positive and negative examples.
- The equation of the line is:
  $x_2 = 1.7x_1 - 4.9$ or $-4.9 + 1.7x_1 - x_2 = 0$

Classification hypothesis

- The classification hypothesis is:
  $h_{\vec{w}} = 1$ if $\vec{w}.\vec{x} \geq 0$ and 0 otherwise
- It can be thought of passing the linear function $\vec{w}.\vec{x}$ through a *threshold function*.
- Mimimizing *Loss* depends on taking the gradient of the threshold function
- The gradient for the step function is zero almost everywhere and undefined elsewhere!

Perceptron learning



A simple weight update rule that is guaranteed to converge for linearly separable data:

$$w_i \leftarrow w_i + \alpha(y - h_{\vec{w}}(\vec{x})) \times x_i$$

where

y is the true value, and $h_{\vec{w}}(\vec{x})$ is the hypothesis output.

Perceptron learning rule

$w_i \leftarrow w_i + \alpha(y - h_{\vec{w}}(\vec{x})) \times x_i$

- If the output is correct, i.e., $y = h_{\vec{w}}(\vec{x})$, then the weights are not changed.
- If the output is lower than it should be, i.e, y is 1 but $h_{\vec{w}}(\vec{x})$ is 0, then w_i is increased when the corresponding input x_i is positive and decreased when the corresponding input x_i is negative.
- If the output is higher than it should be, i.e, y is 0 but $h_{\vec{w}}(\vec{x})$ is 1, then w_i is decreased when the corresponding input x_i is positive and increased when the corresponding input x_i is negative.

Perceptron learning procedure

- Start with a random assignment to the weights
- Feed the input, let the perceptron compute the answer
- If the answer is correct, do nothing
- If the answer is not correct, update the weights by adding or subtracting the input vector (scaled down by α)
- Iterate over all the input vectors, repeating as necessary, until the perceptron learns

Perceptron learning example

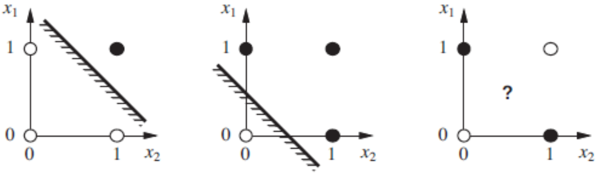
This example teaches the logical or function.
a_0 is the "bias", a_1 and a_2 are the inputs, y is the output.

	a_0	a_1	a_2	y
Example 1	1	0	0	0
Example 2	1	0	1	1
Example 3	1	1	0	1
Example 4	1	1	1	1

With $\alpha = 0.5$ and $(0.1, 0.2, 0.3)$ as the initial weights, the weight vector converges to $(-0.4, 0.7, 0.8)$ after 4 iterations on the examples. The number of iterations changes depending on the initial weights and α (see the spreadsheet).

Expressiveness of perceptrons

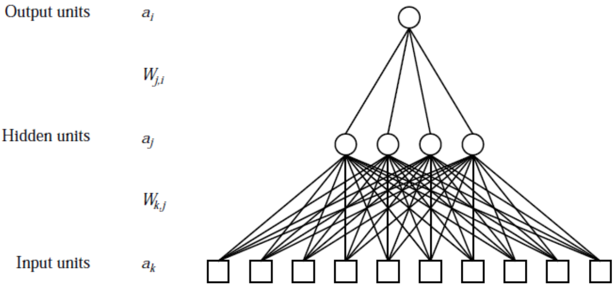
- Consider a perceptron where g is the step function (Rosenblatt, 1957, 1960)
- It can represent AND, OR, NOT, but not XOR (Minsky & Papert, 1969)
- A perceptron represents a *linear separator* in input space: $\sum_j W_j x_j > 0$ or $\mathbf{W} \cdot \mathbf{x} > 0$



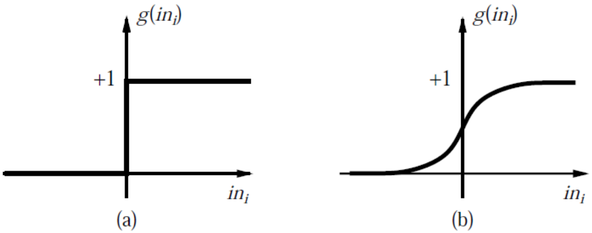
(a) x_1 **and** x_2 (b) x_1 **or** x_2 (c) x_1 **xor** x_2

Multilayer perceptrons (MLPs)

- Remember that a single perceptron will not converge if the inputs are not *linearly separable*.
- In that case, use a multilayer perceptron.
- The numbers of *hidden units* are typically chosen by hand.

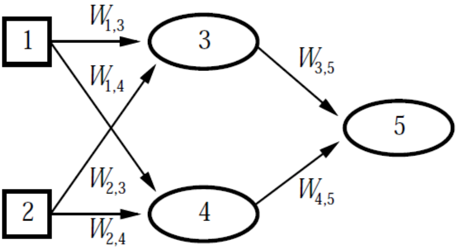


Activation functions



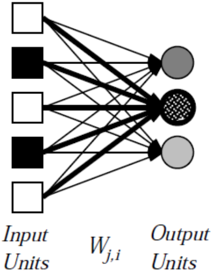
- (a) is a step function or threshold function
- (b) is a sigmoid function $1/(1 + e^{-x})$

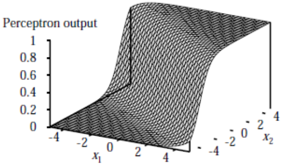
Feed-forward example



- Feed-forward network: parameterized family of nonlinear functions
- Output of unit 5 is $a_5 = g(W_{3,5} \cdot a_3 + W_{4,5} \cdot a_4)$
  $= g(W_{3,5} \cdot g(W_{1,3} \cdot a_1 + W_{2,3} \cdot a_2) + W_{4,5} \cdot g(W_{1,4} \cdot a_1 + W_{2,4} \cdot a_2))$
- Adjusting the weights changes the function:
  do learning this way!

Single-layer perceptrons

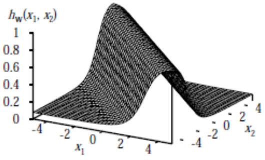




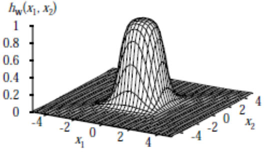
- Output units all operate separately – no shared weights
- Adjusting the weights moves the location, orientation, and steepness of cliff

Expressiveness of MLPs

- All continuous functions with 2 layers,
  all functions with 3 layers
- Ridge: Combine two opposite-facing threshold functions
- Bump: Combine two perpendicular ridges
- Add bumps of various sizes and locations to fit any surface
- Proof requires exponentially many hidden units



(a)



(b)

Back-propagation learning

Output layer: similar to a single-layer perceptron

$$w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta_j$$
$$\text{where } \Delta_j = Err_j \times g'(in_j)$$

Hidden layer: back-propagate the error from the output layer:

$$\Delta_i = g'(in_i) \sum_j w_{i,j} \Delta_j$$

The update rule for weights in hidden layer is the same:

$$w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta_j$$

(Most neuroscientists deny that back-propagation occurs in the brain)

Handwritten digit recognition

- 3-nearest-neighbor classifier (stored images) $= 2.4\%$ error
- Shape matching based on computer vision $= 0.63\%$ error
- 400-300-10 unit MLP $= 1.6\%$ error
- LeNet 768-192-30-10 unit MLP $= 0.9\%$ error
- Boosted neural network $= 0.7\%$ error
- Support vector machine $= 1.1\%$ error
- Current best: virtual support vector machine $= 0.56\%$ error
- Humans $\approx 0.2\%$ error

MLP learners

- MLPs are quite good for complex pattern recognition tasks
- The resulting hypotheses cannot be understood easily
- Typical problems: slow convergence, local minima

Understanding the brain

"The brain is a tissue. It is a complicated, intricately woven tissue, like nothing else we know of in the universe, but it is composed of cells, as any tissue is. They are, to be sure, highly specialized cells, but they function according to the laws that govern any other cells. Their electrical and chemical signals can be detected, recorded and interpreted and their chemicals can be identified, the connections that constitute the brains woven feltwork can be mapped. In short, the brain can be studied, just as the kidney can."

– David H. Hubel (1981 Nobel Prize Winner)

Understanding the brain (cont'd)

"Because we do not understand the brain very well we are constantly tempted to use the latest technology as a model for trying to understand it. In my childhood we were always assured that the brain was a telephone switchboard. (What else could it be?) I was amused to see that Sherrington, the great British neuroscientist, thought that the brain worked like a telegraph system. Freud often compared the brain to hydraulic and electro-magnetic systems. Leibniz compared it to a mill, and I am told that some of the ancient Greeks thought the brain functions like a catapult. At present, obviously, the metaphor is the digital computer."

– John R. Searle (Prof. of Philosophy at UC, Berkeley)

Summary

- Brains have lots of neurons; each neuron $\approx$ perceptron (?)
- None of the neural network models distinguish humans from dogs from dolphins from flatworms. Whatever distinguishes higher cognitive capacities (language, reasoning) may not be apparent at this level of analysis.
- Actually, real neurons fire all the time; what changes is the rate of firing, from a few to a few hundred impulses a second.
- "Neurally inspired computing" rather than "brain science".
- Perceptrons (one-layer networks) are used for linearly separable data.
- Multi-layer networks are sufficiently expressive; can be trained by gradient descent, i.e., error back-propagation.
- Many applications: speech, driving, handwriting, fraud detection, etc.
- Engineering, cognitive modelling, and neural system modelling subfields have largely diverged

Sources for the slides

- AIMA textbook (3^{rd} edition)
- AIMA slides:
  http://aima.cs.berkeley.edu/
- Neuron cell:
  http://www.enchantedlearning.com/subjects/anatomy/brain/Neuron.shtml
  (Accessed December 10, 2011)
- Robert Wilensky's CS188 slides
  http://www.cs.berkeley.edu/ wilensky/cs188
  (Accessed prior to 2009)