ISTA 421 + INFO 521
Introduction to Machine Learning

Lecture 24:
Intro to Neural Networks
The Perceptron

Clay Morrison
claytonm@email.arizona.edu
Harvill 437A
Phone 621-6609

20 November 2017

SISTA 1

Rest of the Semester

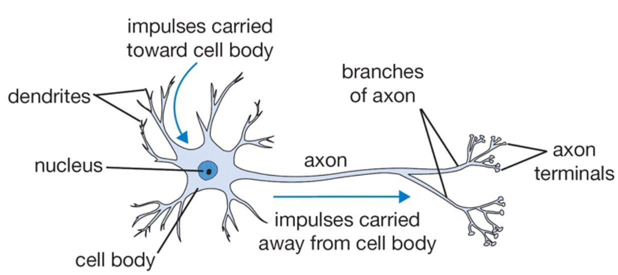
- Mon, Nov 20 – Intro to NN
- Wed, Nov 22 – NN 2: Backpropagation
- Mon, Nov 27 – NN 3: Sparse Autoencoders
- Wed, Nov 29 – Clustering: K-Means
- Mon, Dec 4 – Clustering: Gaussian Mixture Model
- Wed, Dec 6 – Principle Components Analysis

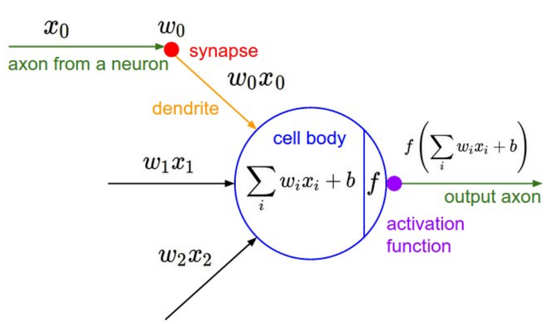
- Homework 5 (NN) due Thurs, Dec 7
- Final Project due Mon, Dec 11

- **Reminder: Please fill out course evaluation (TCE)**

SISTA 2

Neural Networks and Deep Learning

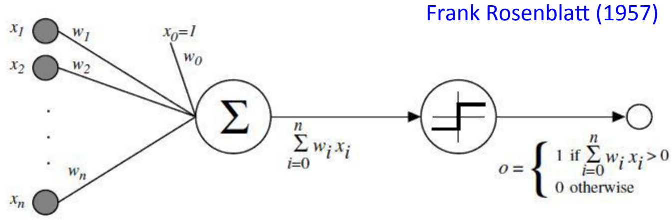




http://cs231n.github.io/neural-networks-1/

The Perceptron

Frank Rosenblatt (1957)



The Perceptron Algorithm (Hebbian training rule)
Initialize weights
Repeat:
  Select next training instance and compute the Perceptron's output
  If output of Perceptron is wrong
    If output should have been 0 but was 1, decrease weights that had input 1
    If output should have been 1 but was 0, increase the weights that had input 1
… until Perceptron achieves desirable performance

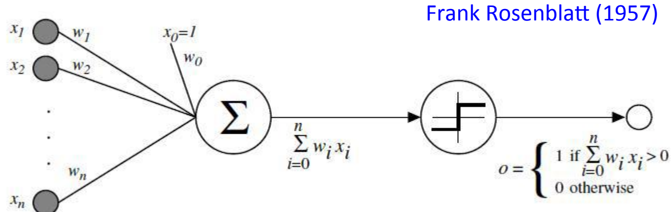
$$w_j := w_j + \Delta w_j$$
$$\Delta w_j = \alpha(\text{target}^{(i)} - \text{output}^{(i)})x_j$$

Recall Widrow-Hoff (aka Adaline) gradient descent for linear regression:

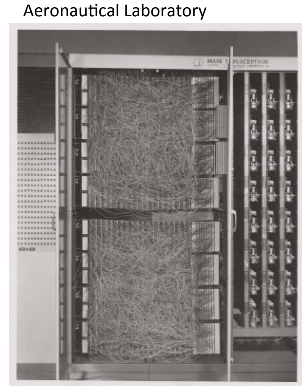
$$\begin{aligned} \mathbf{w} &:= \mathbf{w} - \alpha \frac{\partial \mathcal{L}}{\partial \mathbf{w}} \\ &:= \mathbf{w} - \alpha \left(\mathbf{X}^\top \mathbf{X} \mathbf{w} - \mathbf{X}^\top \mathbf{t} \right) \\ &:= \mathbf{w} - \alpha \sum_{n=1}^{N} \left(t_n - \mathbf{w}^\top \mathbf{x}_n \right) \mathbf{x}_n \end{aligned}$$

The Perceptron

Frank Rosenblatt (1957)



Mark I Perceptron at the Cornell Aeronautical Laboratory



The Perceptron Algorithm (Hebbian training rule)
Initialize weights
Repeat:
  Select next training instance and compute the Perceptron's output
  If output of Perceptron is wrong
    If output should have been 0 but was 1, decrease weights that had input 1
    If output should have been 1 but was 0, increase the weights that had input 1
… until Perceptron achieves desirable performance

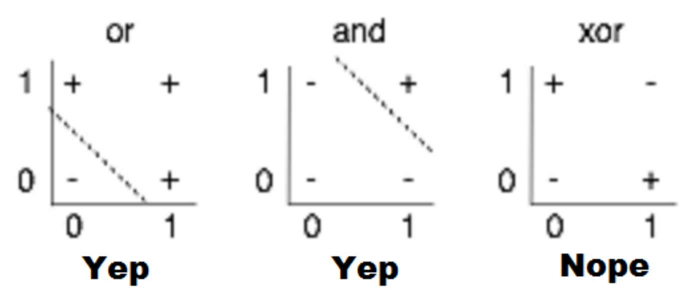
Perceptron vs. Logistic Regression

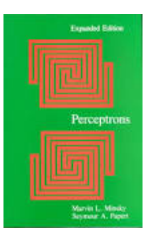
- Perceptron with a sigmoid activation function (and without a hidden layer) and logistic regression are the same.

$$f(z) = \frac{1}{1 + \exp(-z)}$$

- Perceptron (with Hebbian update rule or gradient update) is used for online training.
- Logistic Regression often used for batch.

Limits of the Perceptron

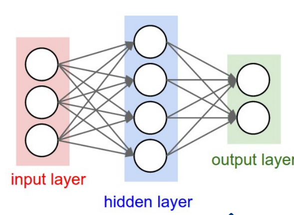




Marvin Minsky and Seymour Papert (1969)

Need multi-layer Perceptron

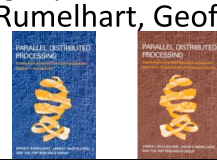
But, at the time, no way to train (Perceptron rule doesn't work)



AI Winter 1

Error Backpropagation

- The big problem: “credit” or “blame” assignment
- A solution:
  - Provide a non-linear activation function that can be differentiated (is smooth)
  - The chain rule can be used to compute the derivative for all the neurons in the prior layer – provides rule for adjusting their weights
  - “backpropagate” the error to the previous layer to determine the previous layer’s target output
- Version developed in early 60’s; implemented on computer by Seppo Linnainmaa in 1970.
- Paul Werbos 1974 PhD thesis: first application to neural networks; not published until 1982!
- Popularized in 1986: “Learning representations by back-propagating errors” by David Rumelhart, Geoffrey Hinton, and Ronald Williams



Homework 5 and
a Note about Notation

- **Tutorial:** Andrew Ng’s *Unsupervised Feature Learning and Deep Learning* (UFLDL) tutorial on building a **sparse autoencoder** for classifying handwriting
  - Main focus: **http://ufldl.stanford.edu/wiki/index.php/UFLDL_Tutorial**
    - The first section on the Sparse Autoencoder is what we’ll focus on.
  - There is also the newer UFLDL tutorial: http://ufldl.stanford.edu/tutorial/
- **Notation:**
  - The UFLDL tutorial (and in many cases in the neural network literature), matrices representing weights going from one layer to the next do so by having column indices for the *from* (source) and row indices for the *to* (destination) weight links.
  - Also, superscripts in parentheses are used to represent individuals in training data or layer index in the network; subscripts used to index into the elements of a vector/matrix.

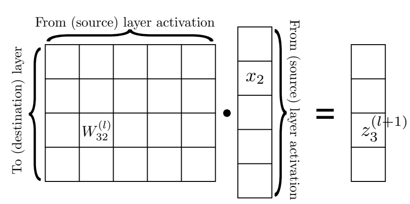
The i^{th} input output value

$(x^{(i)}, y^{(i)})$

The 3rd value of the i^{th} input vector

$x_3^{(i)}$

scalar! (note the subscript)

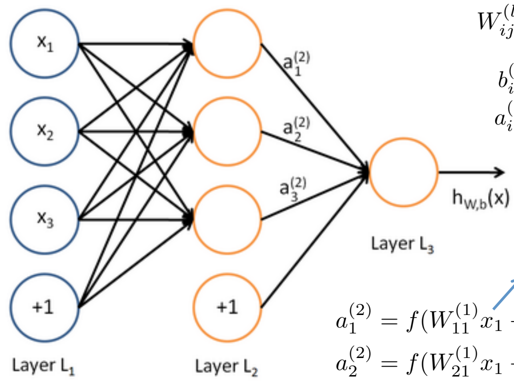


vectors!

$z^{(l+1)} = W^{(l)} x$

matrix!

Feed Forward MLP Calculation (Forward Propagation)



n_l = number of layers = 3

$W_{ij}^{(l)}$ = weight between unit j in layer l and unit i in layer $l+1$

$b_i^{(l)}$ = bias for unit i in layer $l+1$

$a_i^{(l)}$ = activation (output value) of unit i in layer l

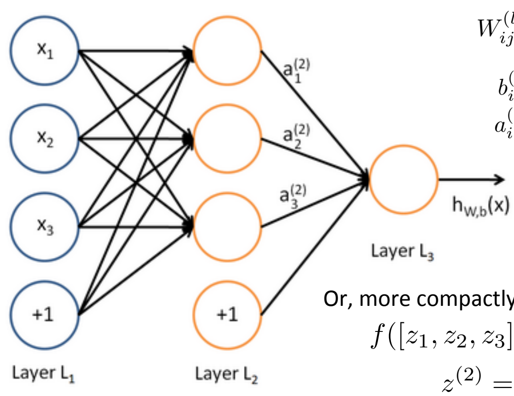
$a_i^{(1)} = x_i$

$$a_1^{(2)} = f(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_1^{(1)})$$
$$a_2^{(2)} = f(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + b_2^{(1)})$$
$$a_3^{(2)} = f(W_{31}^{(1)}x_1 + W_{32}^{(1)}x_2 + W_{33}^{(1)}x_3 + b_3^{(1)})$$
$$h_{W,b}(x) = a_1^{(3)} = f(W_{11}^{(2)}a_1^{(2)} + W_{12}^{(2)}a_2^{(2)} + W_{13}^{(2)}a_3^{(2)} + b_1^{(2)})$$

$z_i^{(l)} = \sum_{j=1}^{n} W_{ij}^{(l-1)}x_j + b_i^{(l-1)}$ and $a_i^{(l)} = f(z_i^{(l)})$

SISTA 11

Feed Forward MLP Calculation (Forward Propagation)



n_l = number of layers = 3

$W_{ij}^{(l)}$ = weight between unit j in layer l and unit i in layer $l+1$

$b_i^{(l)}$ = bias for unit i in layer $l+1$

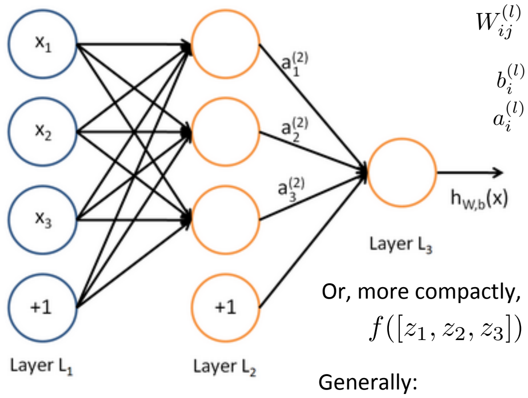
$a_i^{(l)}$ = activation (output value) of unit i in layer l

Or, more compactly, if we make *f*(•) a vector function:

$$f([z_1, z_2, z_3]) = [f(z_1), f(z_2), f(z_3)]$$
$$z^{(2)} = W^{(1)}x + b^{(1)}$$
$$a^{(2)} = f(z^{(2)})$$
$$z^{(3)} = W^{(2)}a^{(2)} + b^{(2)}$$
$$h_{W,b}(x) = a^{(3)} = f(z^{(3)})$$

SISTA 12

Feed Forward MLP Calculation (Forward Propagation)



$$n_l = \text{number of layers} = 3$$
$$W_{ij}^{(l)} = \text{weight between unit } j \text{ in layer } l \text{ and unit } i \text{ in layer } l+1$$
$$b_i^{(l)} = \text{bias for unit } i \text{ in layer } l+1$$
$$a_i^{(l)} = \text{activation (output value) of unit } i \text{ in layer } l$$

Or, more compactly, if we make *f*(•) a vector function:

$$f([z_1, z_2, z_3]) = [f(z_1), f(z_2), f(z_3)]$$

Generally:

$$z^{(l+1)} = W^{(l)}a^{(l)} + b^{(l)} \quad \text{where } a^{(1)} = x$$
$$a^{(l+1)} = f(z^{(l+1)})$$

Gradient Descent

- Given a loss function

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}\left(\underbrace{\|h_\theta(x^{(i)}) - y^{(i)}\|}\right)^2$$

L_2 norm (i.e., Euclidean distance) when h_θ and y are vectors; when scalars, just subtract

- *m* is the number of examples and *h* some function of parameters θ
- Gradient descent updates the parameters in steps:

$$\theta_j = \theta_j - \alpha\frac{\partial J(\theta)}{\partial \theta_j}$$

Backpropagation Algorithm
(for fully connected feedforward network)

training pairs: $\{(x^{(1)}, y^{(1)}), (x^{(m)}, y^{(m)})\}$

Define the loss

Neural networks can be prone to overfitting, so regularize

$$J(W,b) = \left[\frac{1}{m}\sum_{i=1}^{m}\left(\underbrace{\frac{1}{2}\|h_{W,b}(x^{(i)}) - y^{(i)}\|^2}_{\text{Implementation note: use numpy.linalg.norm()}}\right)\right] + \underbrace{\frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_l+1}\left(W_{ji}^{(l)}\right)^2}_{\text{weight decay term}}$$

Goal is to minimize $J(W,b)$

weight decay term

(note that reg term typically not over bias terms)

One iteration of gradient descent updates parameters W, b as follows:

$$W_{ij}^{(l)} = W_{ij}^{(l)} - \alpha\frac{\partial J(W,b)}{\partial W_{ij}^{(l)}} \qquad \frac{\partial J(W,b)}{\partial W_{ij}^{(l)}} = \left[\frac{1}{m}\sum_{k=1}^{m}\frac{\partial J(W,b;x^{(k)},y^{(k)})}{\partial W_{ij}^{(l)}}\right] + \lambda W_{ij}^{(l)}$$

$$b_i^{(l)} = b_i^{(l)} - \alpha\frac{\partial J(W,b)}{\partial b_i^{(l)}} \qquad \frac{\partial J(W,b)}{\partial b_i^{(l)}} = \frac{1}{m}\sum_{k=1}^{m}\frac{\partial J(W,b;x^{(k)},y^{(k)})}{\partial b_i^{(l)}}$$

SISTA 15

Backpropagation Algorithm
(for fully connected feedforward network)

Partial derivatives of cost function for single example (x,y) $\quad \frac{\partial J(W,b;x,y)}{\partial W_{ij}^{(l)}} \quad \frac{\partial J(W,b;x,y)}{\partial b_i^{(l)}}$

Backprop Core (in words):

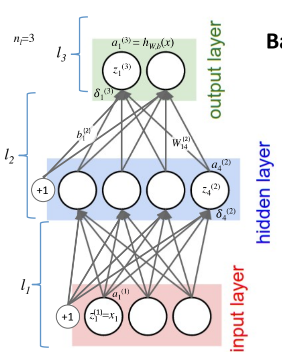
Given training example (x,y) …

Run "forward pass" to compute all activations throughout network

For each node i in layer l, compute "error term" $\delta_i^{(l)}$ Measures how much node was "responsible" for any errors in output

For an output node, can directly measure the difference between network's activation and the true target value, to define $\delta_i^{(n_l)}$

For hidden units, compute $\delta_i^{(l)}$ based on weighted average of the error terms of the nodes that use $a_i^{(l)}$ as an input.



Backprop Core (in more detail):

(1) Perform feedforward pass, computing the activations for layers $L_2, L_3,$ and so on up to the output layer L_{n_l}

(2) For each output unit i in layer n_l (the output layer), set

$$\delta_i^{(n_l)} = \frac{\partial}{\partial z_i^{(n_l)}}\frac{1}{2}\|y - h_{W,b}(x)\|^2 = -(y_i - a_i^{(n_l)}) \cdot f'(z_i^{(n_l)})$$

Indices are correct!
i is layer l, j is layer l+1

(3) For $l = n_l - 1, n_l - 2, n_l - 3, \ldots, 2$

For each node I in layer l, set $\delta_i^{(l)} = \left(\sum_{j=1}^{s_{l+1}} W_{ji}^{(l)}\delta_j^{(l+1)}\right) f'(z_i^{(l)})$

(4) Compute the desired partial derivatives, which are given as:

$$\frac{\partial J(W,b;x,y)}{\partial W_{ij}^{(l)}} = a_j^{(l)}\delta_i^{(l+1)} \qquad \frac{\partial J(W,b;x,y)}{\partial b_i^{(l)}} = \delta_i^{(l+1)}$$

SISTA 16