Chapter 3

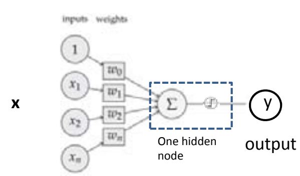
Deep Neural Networks

Qiang Ji

Introduction

- Neural Networks
  - Multilayer Perceptron
- Deep Neural Networks
- Convolutional Neural Networks

The Perceptron Algorithm

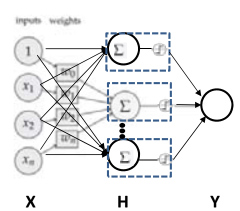


$$y(x) = \phi(\mathbf{w}^t\mathbf{x} + w_0)$$

where nonlinear activation function ϕ() is given by a step function:

$$\phi(\mathbf{w}^t\mathbf{x})\begin{cases} +1 \text{ if } \mathbf{w}^t\mathbf{x} + w_0 > 0 \\ -1 \ \text{ else} \end{cases}$$

Neural Networks



where X represents input, H hidden layer, and Y output layer

Each hidden node is a perceptron and it performs the same operation.

NNs are layers of perceptrons

Multilayer Neural Networks (NNs)

- Input layer
- Output layer
- Hidden layers
- Feed forward – from input though hidden layers to reach output

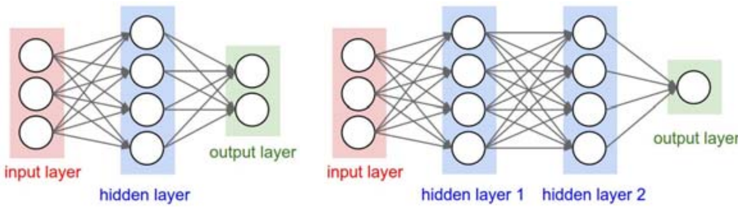
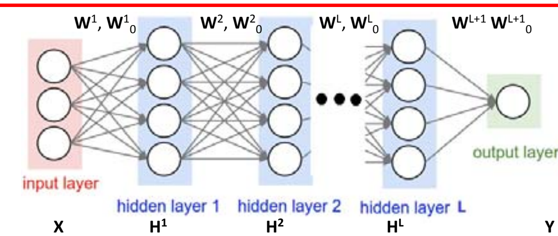


Image Credit: Andrej Karpathy, CS231n

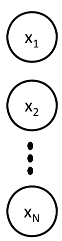
Notations for NNs



- $\mathbf{X} = (x_1, x_2, \ldots, x_N)^t$ represents input layer with N nodes the
- $\mathbf{Y} = (y_1, y_2, \ldots, y_K)^t$ represents output layer with K nodes
- $\mathbf{H}^l = (h^l_1, h^l_2, .. h^l_{N_l})$ represents *l* th hidden layer with N_l nodes, l=0,1,2,..,L+1. with $\mathbf{H}^0 = \mathbf{X}$ **and** $\mathbf{H}^{L+1} = \mathbf{Y}$
- $\mathbf{W}^l$ is the weight matrix (N_{l-1} x N_l) for the lth hidden layer. $\mathbf{W}^l_0$ the bias vector (N_lx1). $\mathbf{W}^{L+1}$ and $\mathbf{W}^{L+1}_0$ are the weight matrix and bias for the output layer.

Input Layer

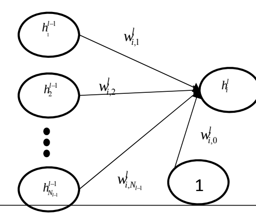
- The number of nodes in input layer is equal to the number of input variables **X**



Input layer

Hidden Layer

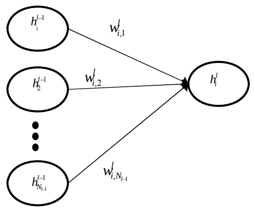
- Each node in the hidden layer is a single perceptron. It is connected to all nodes in the previous layer
- Its value is the linear combination of its inputs through an activation function ϕ()



$$h^l_i = \phi\left(\sum_{j=1}^{N_{l-1}} w^l_{i,j} h^{l-1}_j + w^l_{i,0}\right)$$

$w^l_{i,j}$ is the weight between ith node in layer l and jth node in layer *l*-1 and $w^l_{i,0}$ is the bias for the ith node in layer *l*

Hidden Layer



$$h_i^l = \phi\left(\sum_{j=1}^{N_{l-1}} w_{i,j}^l \, h_j^{l-1} + w_{i,0}^l\right)$$
$$= \phi\left((W_i^l)^t H^{l-1} + w_{i,0}^l\right)$$

- where W_i^l is the ith column of $\mathbf{W}^l$ and H^{l-1} are

$$\mathbf{W}_i = \begin{pmatrix} w_{i,1}^l \\ w_{i,2}^l \\ \dots \\ w_{i,N_{l-1}}^l \end{pmatrix} \qquad H^{l-1} = \begin{pmatrix} h_1^{l-1} \\ h_2^{l-1} \\ \dots \\ h_{N_{l-1}}^{l-1} \end{pmatrix}$$

- $\phi()$ adds non-linearity to the mapping

Hidden Layer

For all nodes in the lth hidden layer H^l, we have

$$\mathbf{H}^l = \begin{pmatrix} h_1^l \\ h_2^l \\ \dots \\ h_{N_l}^l \end{pmatrix} = \begin{pmatrix} \phi((W_1^l)^t \mathbf{H}^{l-1} + w_{1,0}^l) \\ \phi((W_2^l)^t \mathbf{H}^{l-1} + w_{2,0}^l) \\ \dots \\ \phi((W_{N_l}^l)^t \mathbf{H}^{l-1} + w_{N_l,0}^l) \end{pmatrix} = \phi((\mathbf{W}^l)^t \mathbf{H}^{l-1} + \mathbf{W}_0^l)$$

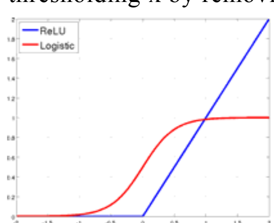
where

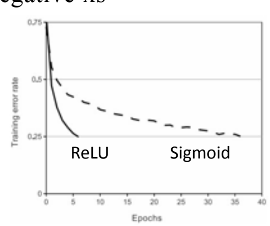
$$\mathbf{W}^l = [\mathbf{W}_1 \; \mathbf{W}_2 \; \dots \; \mathbf{W}_{N_l}] \qquad \mathbf{W}_0^l = \begin{bmatrix} w_{1,0}^l \\ w_{2,0}^l \\ \dots \\ w_{N_l,0}^l \end{bmatrix}$$

where $\phi(\mathbf{x})$ applies to each element of vector $\mathbf{x}$

Activation Functions

- Sigmoid: $\phi(x) = 1 / (1 + e^{-x})$
- Rectified Linear Unit (ReLU): $\phi(x) = \max(0, x)$, basically thresholding x by removing negative xs





- ReLU v.s. Sigmoid: stable gradient, sparse activation, easy computation, etc.

Output Layer

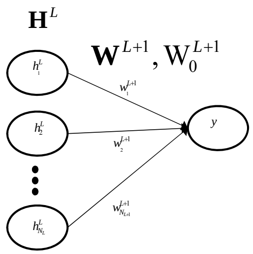
$$\mathbf{y} = g((\mathbf{W}^{L+1})^t \mathbf{H}^L + \mathbf{W}_0^{L+1})$$

g() is the output function and it varies, depending on the type of y

- Scalar regression
- Binary classification
- Multi-classification

Output Layer : Scalar Regression

- The output node value $y \in R$ is computed as a linear combination of its inputs

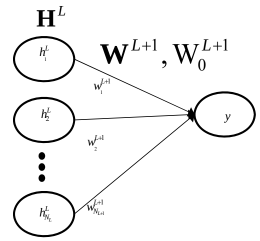


$$y = \sum_{j=1}^{N_L} w_j^{L+1} h_j^L + W_0^{L+1}$$
$$= (\mathbf{W}^{L+1})^t \mathbf{H}^L + W_0^{L+1}$$

Note $\mathbf{W}^{L+1}$ is a vector and W_0^{L+1} is a scalar.

Output Layer : Binary classification

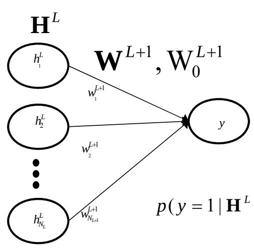
- The output node value $y \in \{+1,-1\}$ is computed via a sigmoid function $\sigma()$



$$p(y \mid \mathbf{H}^L) = \sigma(y((\mathbf{W}^{L+1})^t \mathbf{H}^L + W_0^{L+1}))$$
$$= \frac{1}{1 + e^{-y((\mathbf{W}^{L+1})^t \mathbf{H}^L + W_0^{L+1})}}$$

Output Layer : Binary classification

- The output node value $y \in \{1,0\}$ is computed via a sigmoid function $\sigma()$

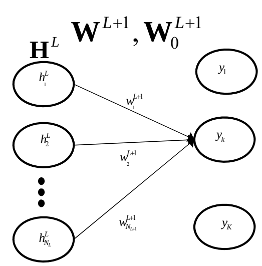


$$p(y=1 \mid \mathbf{H}^L) = \sigma(((\mathbf{W}^{L+1})^t \mathbf{H}^L + W_0^{L+1})) = \frac{1}{1 + e^{-((\mathbf{W}^{L+1})^t \mathbf{H}^L + W_0^{L+1})}}$$

$$p(y \mid \mathbf{H}^L) = \frac{e^{y((\mathbf{W}^{L+1})^t \mathbf{H}^L + W_0^{L+1})}}{1 + e^{((\mathbf{W}^{L+1})^t \mathbf{H}^L + W_0^{L+1})}}$$

Output Layer : Multi-class Classification

- The output node value $y \in \{1,2,\ldots,K\}$. It is represented by an output vector $\mathbf{y}=(y_1, y_2, \ldots, y_K)^t$, and each of its element is computed through a multi-class sigmoid function $\sigma_M()$

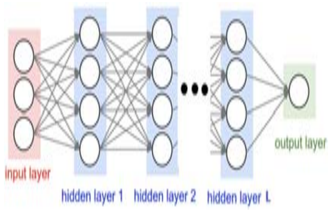


$$y_k = \sigma_M((\mathbf{W}_k^{L+1})^t \mathbf{H}^l + \mathbf{W}_{k,0}^{L+1})$$
$$= \frac{\exp((\mathbf{W}_k^{L+1})^t \mathbf{H}^l + \mathbf{W}_{k,0}^{L+1}))}{\sum_{k=1}^{K} \exp((\mathbf{W}_k^{L+1})^t \mathbf{H}^l + \mathbf{W}_{k,0}^{L+1}))}$$

$$\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \ldots \\ y_K \end{pmatrix}$$

$\mathbf{W}^{L+1}$ is a matrix and $\mathbf{W}^{L+1}_k$ is its kth column and $\mathbf{W}_0^{L+1}$ is a vector

Forward Propagation

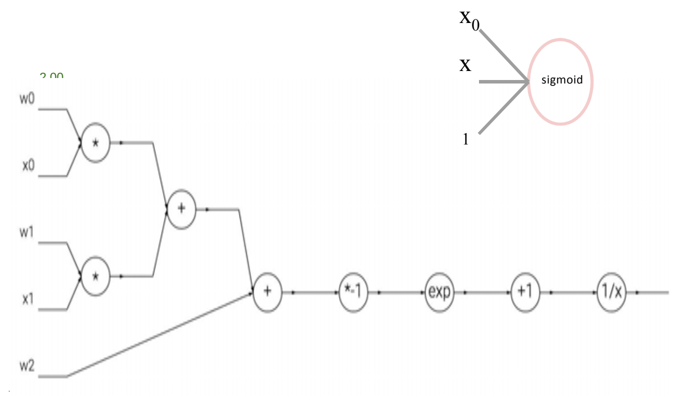


$$H^{1} = \phi((\mathbf{W}^{1})^{t}\mathbf{X} + \mathbf{W}_{0}^{1})$$
$$H^{2} = \phi((\mathbf{W}^{2})^{t}\mathbf{H}^{1} + \mathbf{W}_{0}^{2})$$
....
$$H^{L} = \phi((\mathbf{W}^{L})^{t}\mathbf{H}^{L-1} + \mathbf{W}_{0}^{L})$$
$$y = g((\mathbf{W}^{L+1})^{t}\mathbf{H}^{L} + \mathbf{W}_{0}^{L+1})$$

Given an input x, the output is computed through a series of recursive composition from input layer through the hidden layer until the output layer, i.e.,

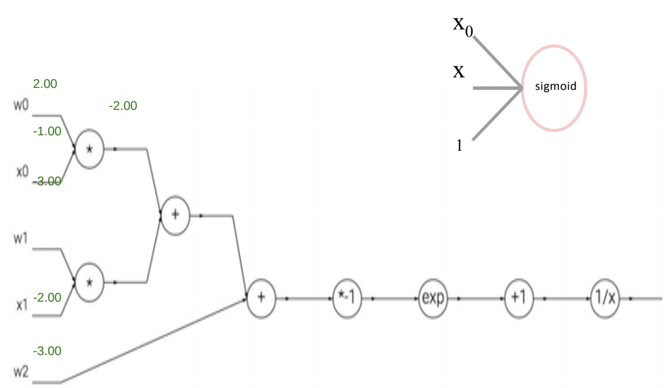
ϕ() is the activation function
g() is the output function

Forward Computation



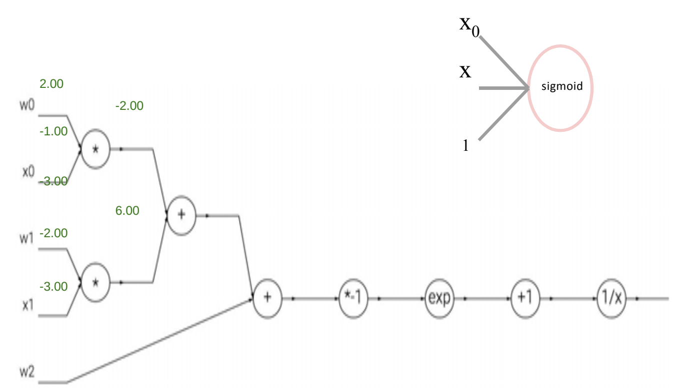
http://cs231n.github.io/optimization-2/

Forward Computation



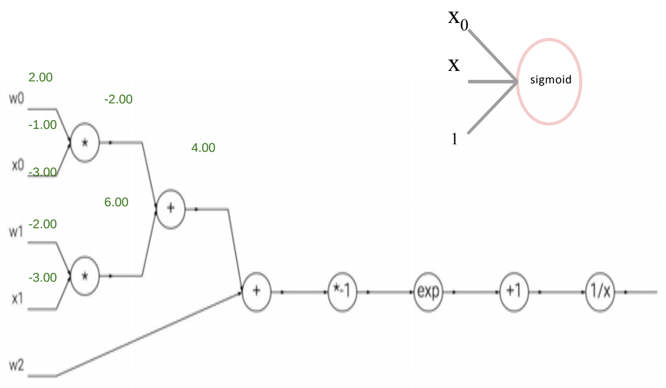
http://cs231n.github.io/optimization-2/

Forward Computation



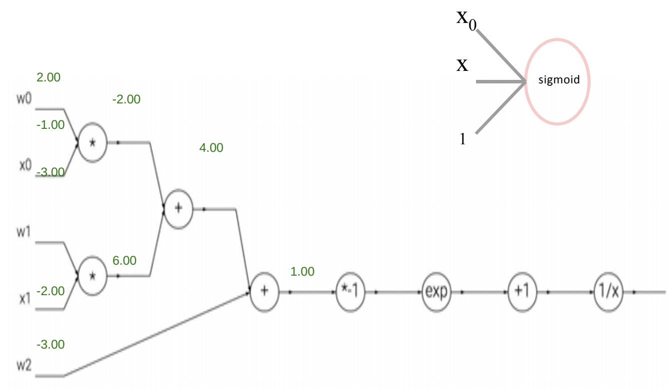
http://cs231n.github.io/optimization-2/

Forward Computation



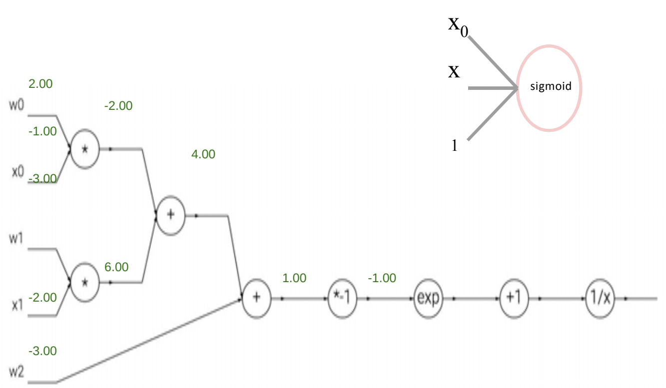
http://cs231n.github.io/optimization-2/

Forward Computation



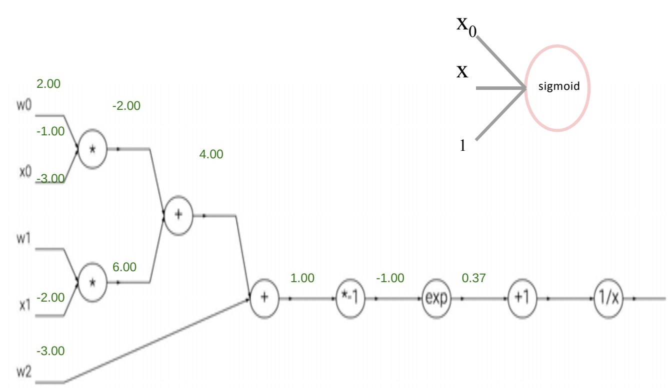
http://cs231n.github.io/optimization-2/

Forward Computation



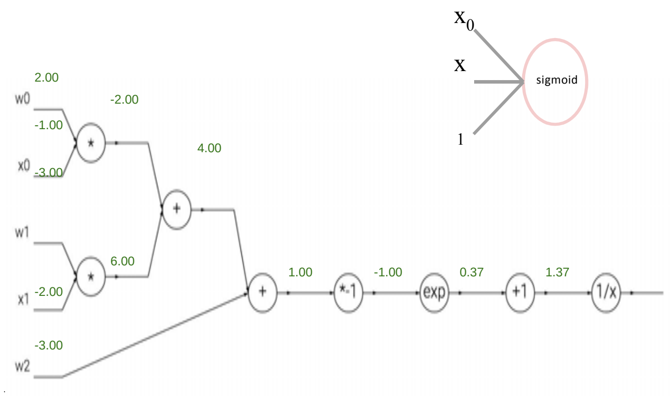
http://cs231n.github.io/optimization-2/

Forward Computation



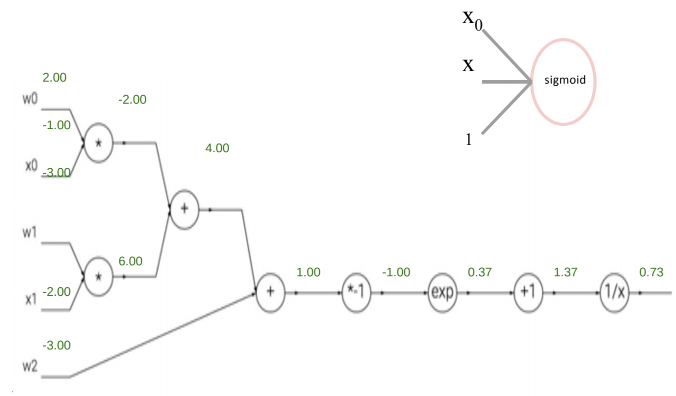
http://cs231n.github.io/optimization-2/

Forward Computation



http://cs231n.github.io/optimization-2/

Forward Computation



http://cs231n.github.io/optimization-2/