Introduction to Machine Learning

Neural Networks

Varun Chandola

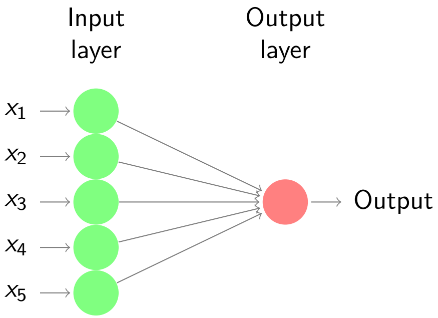
Computer Science & Engineering
State University of New York at Buffalo
Buffalo, NY, USA
chandola@buffalo.edu

University at Buffalo
Department of Computer Science and Engineering
School of Engeering and Applied Sciences

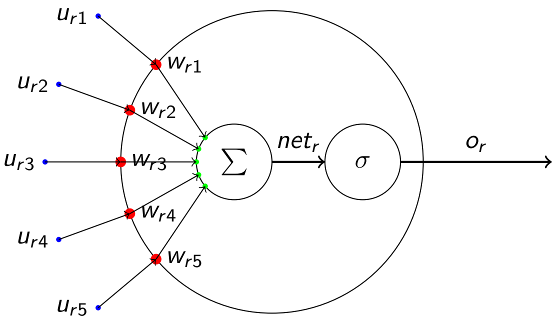
Outline

Extending Perceptrons

- Questions?
  - Why not work with thresholded perceptron?
    - Not differentiable
  - How to learn non-linear surfaces?
  - How to generalize to multiple outputs, numeric output?



Anatomy of a Sigmoid Unit (r)

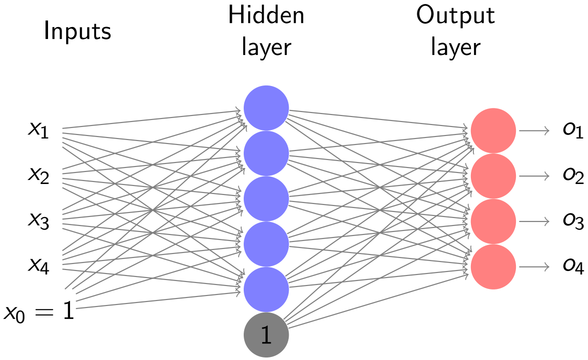


Generalizing to Multiple Labels

- Distinguishing between multiple categories
- *Solution:* Add another layer - **Multi Layer Neural Networks**

Generalizing to Multiple Labels

- Distinguishing between multiple categories
- *Solution:* Add another layer - **Multi Layer Neural Networks**



What Threshold Unit to Use?

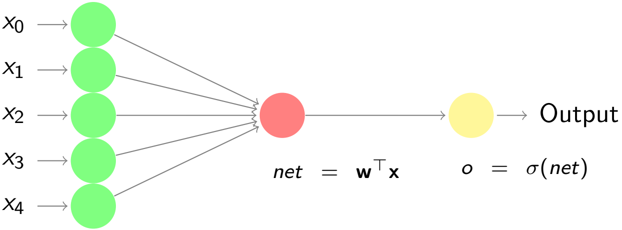
- ~~Linear Unit~~
- ~~Perceptron Unit~~

What Threshold Unit to Use?

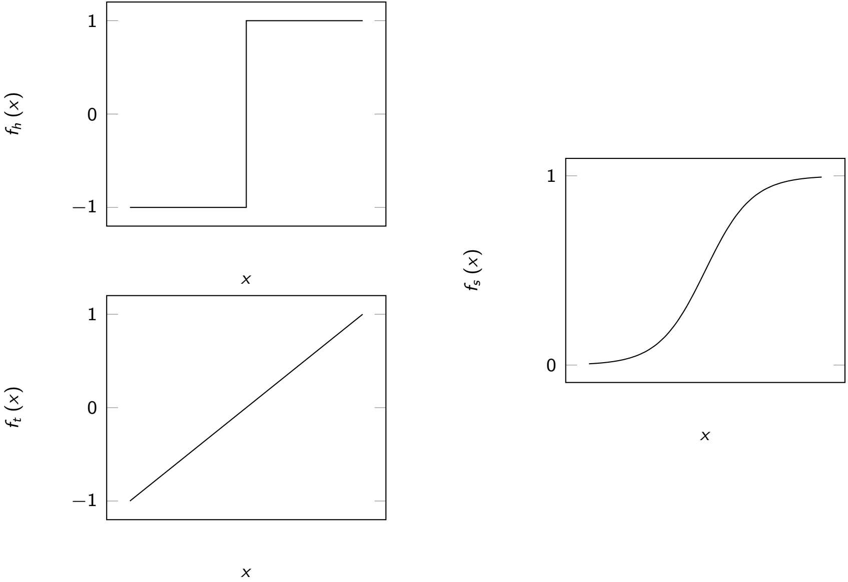
- ~~Linear Unit~~
- ~~Perceptron Unit~~
- Sigmoid Unit
  - Smooth, differentiable threshold function

$$\sigma(net) = \frac{1}{1 + e^{-net}}$$

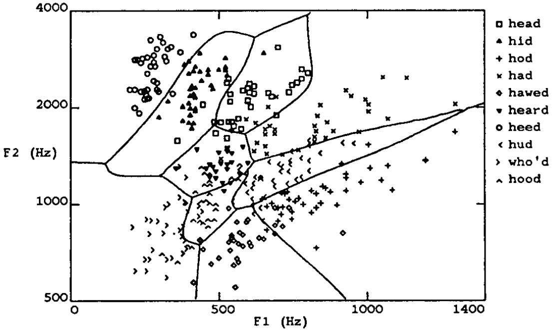
  - Non-linear output



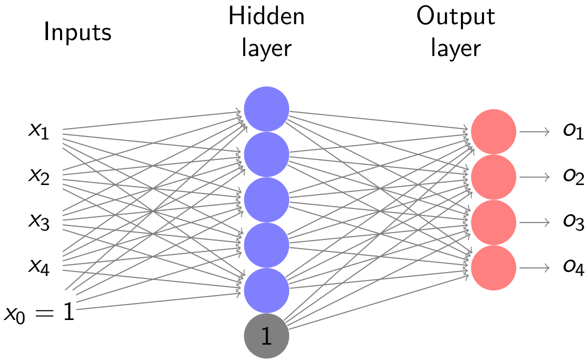
Properties of Sigmoid Function



Motivation for Using Non-linear Surfaces



Feed Forward Neural Networks - Architecture



Feed Forward Neural Networks

- D input nodes (excluding bias)
- M hidden nodes (excluding bias)
- K output nodes

- At hidden nodes: $\mathbf{w_j}, 1 \leq j \leq M$, $\mathbf{w_j} \in \mathbb{R}^{D+1}$
- At output nodes: $\mathbf{w_l}, 1 \leq l \leq K$, $\mathbf{w_l} \in \mathbb{R}^{M+1}$

Learning Weights of the Multi-layer Network

- Assume that the network structure is predetermined (number of hidden nodes and interconnections)
- Objective function for N training examples:

$$J = \sum_{i=1}^{N} J_i = \frac{1}{2}\sum_{i=1}^{N}\sum_{l=1}^{K}(y_{il} - o_{il})^2$$

- y_{il} - Target value associated with l^{th} class for input ($\mathbf{x}_i$)
- $y_{il} = 1$ when k is true class for $\mathbf{x}_i$, and 0 otherwise
- o_{il} - Predicted output value at l^{th} output node for $\mathbf{x}_i$

What are we learning?

Weight vectors for all output and hidden nodes that minimize J

The Backpropagation Algorithm

1. Initialize all weights to *small values*
2. For each training example, $\langle \mathbf{x}, \mathbf{y} \rangle$:
    2.1 **Propagate input forward** through the network
    2.2 **Propagate errors backward** through the network

Backpropagation Algorithm - Continued

Gradient Descent

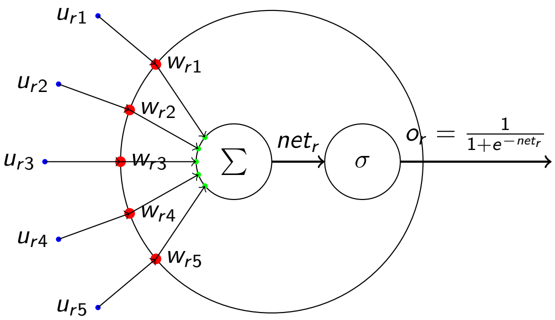
- Move in the opposite direction of the **gradient** of the objective function
- $-\eta \nabla J$

$$\nabla J = \sum_{i=1}^{N} \nabla J_i$$

- What is the gradient computed with respect to?
  - Weights - m at hidden nodes and k at output nodes
  - $\mathbf{w}_j$ $(j = 1 \ldots m)$
  - $\mathbf{w}_l$ $(l = 1 \ldots k)$
- $\mathbf{w}_j \leftarrow \mathbf{w}_j - \eta \frac{\partial J}{\partial \mathbf{w}_j} = \mathbf{w}_j - \eta \sum_{i=1}^{N} \frac{\partial J_i}{\partial \mathbf{w}_j}$
- $\mathbf{w}_l \leftarrow \mathbf{w}_l - \eta \frac{\partial J}{\partial \mathbf{w}_l} = \mathbf{w}_l - \eta \sum_{i=1}^{N} \frac{\partial J}{\partial \mathbf{w}_l}$

$$\nabla J_i = \begin{bmatrix} \frac{\partial J_i}{\partial \mathbf{w}_1} \\ \frac{\partial J_i}{\partial \mathbf{w}_2} \\ \vdots \\ \frac{\partial J_i}{\partial \mathbf{w}_{m+k}} \end{bmatrix}$$

Anatomy of a Sigmoid Unit (r)



$$\frac{\partial J_i}{\partial \mathbf{w}_r} = \begin{bmatrix} \frac{\partial J_i}{\partial w_{r1}} \\ \frac{\partial J_i}{\partial w_{r2}} \\ \vdots \end{bmatrix}$$

- Need to compute $\frac{\partial J_i}{\partial w_{rq}}$
- Update rule for the q^{th} entry in the r^{th} weight vector:

$$w_{rq} \leftarrow w_{rq} - \eta \frac{\partial J}{\partial w_{rq}} = w_{rq} - \eta \sum_{i=1}^{N} \frac{\partial J_i}{\partial w_{rq}}$$

Derivation of the Backpropagation Rules

Assume that we only one training example, i.e., $i = 1$, $J = J_i$. Dropping the subscript i from here onwards.

- Consider any weight w_{rq}
- Let u_{rq} be the q^{th} element of the input vector coming in to the r^{th} unit.

Observation 1

Weight w_{rq} is connected to J through $net_r = \sum_i w_{rq} u_{rq}$.

$$\frac{\partial J}{\partial w_{rq}} = \frac{\partial J}{\partial net_r}\frac{\partial net_r}{\partial w_{rq}} = \frac{\partial J}{\partial net_r} u_{rq}$$

Analyzing Output Nodes

Observation 2

net_l for an **output node** is connected to J only through the output value of the node (or o_l)

$$\frac{\partial J}{\partial net_l} = \frac{\partial J}{\partial o_l}\frac{\partial o_l}{\partial net_l}$$

Analyzing Output Nodes

Observation 2

net_l for an **output node** is connected to J only through the output value of the node (or o_l)

$$\frac{\partial J}{\partial net_l} = \frac{\partial J}{\partial o_l}\frac{\partial o_l}{\partial net_l}$$

Update Rule for Output Units

$$w_{lj} \leftarrow w_{lj} + \eta\delta_l u_{lj}$$

where $\delta_l = (y_l - o_l)o_l(1 - o_l)$.

- *Question:* What is u_{lj} for the l^{th} output node?

Analyzing Hidden Nodes

Observation 3

net_j for a **hidden node** is connected to J through all output nodes

$$\frac{\partial J}{\partial net_j} = \sum_{l=1}^{K} \frac{\partial J}{\partial net_l} \frac{\partial net_l}{\partial net_j}$$

Analyzing Hidden Nodes

Observation 3

net_j for a **hidden node** is connected to J through all output nodes

$$\frac{\partial J}{\partial net_j} = \sum_{l=1}^{K} \frac{\partial J}{\partial net_l} \frac{\partial net_l}{\partial net_j}$$

Update Rule for Hidden Units

$$w_{jp} \leftarrow w_{jp} + \eta \delta_j u_{jp}$$

Analyzing Hidden Nodes

Observation 3

net_j for a **hidden node** is connected to J through all output nodes

$$\frac{\partial J}{\partial net_j} = \sum_{l=1}^{K} \frac{\partial J}{\partial net_l} \frac{\partial net_l}{\partial net_j}$$

Update Rule for Hidden Units

$$w_{jp} \leftarrow w_{jp} + \eta \delta_j u_{jp}$$

$$\delta_j = o_j(1 - o_j) \sum_{l=1}^{K} \delta_l w_{lj}$$

$$\delta_l = (y_l - o_l) o_l (1 - o_l)$$

- *Question:* What is u_{jp} for the j^{th} hidden node?

Final Algorithm

- While not converged:
  - *Move forward* to compute outputs at hidden and output nodes
  - *Move backward* to propagate errors back
    - Compute δ errors at output nodes (δ_l)
    - Compute δ errors at hidden nodes (δ_j)
  - Update all weights according to weight update equations

Conclusions about Neural Networks

- Error function contains many local minima
- No guarantee of convergence
  - Not a "big" issue in practical deployments
- Improving backpropagation

Conclusions about Neural Networks

- Error function contains many local minima
- No guarantee of convergence
  - Not a "big" issue in practical deployments
- Improving backpropagation
  - Adding momentum
  - Using stochastic gradient descent
  - Train multiple times using different initializations

Bias Variance Tradeoff

- Neural networks are *universal function approximators*
  - By making the model more complex (increasing number of hidden layers or m) one can lower the error
- Is the model with least training error the best model?

Bias Variance Tradeoff

- Neural networks are *universal function approximators*
  - By making the model more complex (increasing number of hidden layers or m) one can lower the error
- Is the model with least training error the best model?
  - The simple answer is **no**!
  - Risk of overfitting (chasing the data)
  - Overfitting $\Leftarrow$ **High generalization error**

High Variance - Low Bias

- "Chases the data"
- Very low training error
- Poor performance on unseen data

Low Variance - High Bias

- Less sensitive to training data
- Higher training error
- Better performance on unseen data

Getting the Right Balance

- General rule of thumb – If two models are giving similar training error, choose the **simpler** model
- What is simple for a neural network?
- Low weights in the weight matrices?
  - Why?

Introducing Bias in Neural Network Training

- Penalize solutions in which the weights are high
- Can be done by introducing a penalty term in the objective function
  - **Regularization**

Regularization for Backpropagation

$$\widetilde{J} = J + \frac{\lambda}{2n}\left(\sum_{j=1}^{M}\sum_{i=1}^{D+1}(w_{ji}^{(1)})^2 + \sum_{l=1}^{K}\sum_{j=1}^{M+1}(w_{lj}^{(2)})^2\right)$$

Other Extensions?

- Use a different loss function (why)?
  - Quadratic (Squared), Cross-entropy, Exponential, KL Divergence, etc.
- Use a different activation function (why)?
  - Sigmoid

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

  - Tanh

$$f(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

  - Rectified Linear Unit (ReLU)

$$f(z) = max(0, z)$$

References