Introduction to Machine Learning

CSE474/574: Lecture 8

Varun Chandola <chandola@buffalo.edu>

11 Feb 2015

Outline

Contents

1 Recap - Perceptrons

Training Rule for Gradient Descent

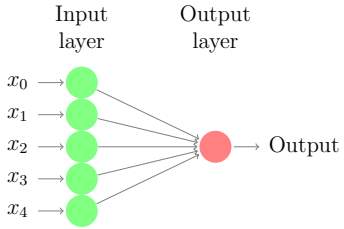
$$\vec{\mathbf{w}} = \vec{\mathbf{w}} - \eta \nabla E(\vec{\mathbf{w}})$$

For each weight component:

$$w_i = w_i - \eta \frac{\partial E}{\partial w_i}$$

The key operation in the above update step is the calculation of each partial derivative. This can be computed for perceptron error function as follows:

$$\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i} \frac{1}{2} \sum_j (y_j - \vec{\mathbf{w}}^\top \vec{\mathbf{x}}_j)^2 \\
&= \frac{1}{2} \sum_j \frac{\partial}{\partial w_i} (y_j - \vec{\mathbf{w}}^\top \vec{\mathbf{x}}_j)^2 \\
&= \frac{1}{2} \sum_j 2(y_j - \vec{\mathbf{w}}^\top \vec{\mathbf{x}}_j) \frac{\partial}{\partial w_i} (y_j - \vec{\mathbf{w}}^\top \vec{\mathbf{x}}_j) \\
&= \sum_j (y_j - \vec{\mathbf{w}}^\top \vec{\mathbf{x}}_j)(-x_{ij})
\end{aligned}$$



where x_{ij} denotes the i^{th} attribute value for the j^{th} training example. The final weight update rule becomes:

$$w_i = w_i + \eta \sum_j (y_j - \vec{\mathbf{w}}^\top \vec{\mathbf{x}}_j) x_{ij}$$

1.1 Issues with Gradient Descent

- Slow convergence
- Stuck in local minima

One should note that the second issue will not arise in the case of Perceptron training as the error surface has only one global minima. But for general setting, including multi-layer perceptrons, this is a typical issue.

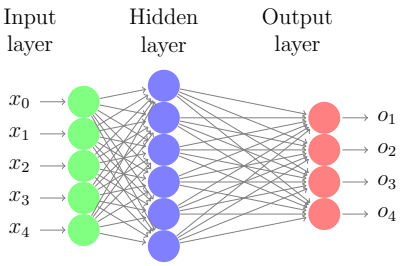
More efficient algorithms exist for batch optimization, including *Conjugate Gradient Descent* and other *quasi*-Newton methods. Another approach is to consider training examples in an online or incremental fashion, resulting in an online algorithm called **Stochastic Gradient Descent** [**?**], which will be discussed next.

1.2 Stochastic Gradient Descent

- **Update weights after every training example.**
- For sufficiently small η, closely approximates Gradient Descent.

Gradient Descent	**Stochastic Gradient Descent**
Weights updated after summing error over all examples	Weights updated after examining each example
More computations per weight update step	Significantly lesser computations
Risk of local minima	Avoids local minima

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



The reason we do not use the thresholded perceptron is because the error function is not differentiable. To understand this, recall that to compute the gradient for perceptron learning we compute the partial derivative of the error with respect to every component of the weight vector.

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i}\frac{1}{2}\sum_j (y_j - \vec{\mathbf{w}}^\top \vec{\mathbf{x}}_j)^2$$

Now if we use the thresholded perceptron, we need to replace $\vec{\mathbf{w}}^\top \vec{\mathbf{x}}_j$ with o in the above equation, where o is -1 if $\vec{\mathbf{w}}^\top \vec{\mathbf{x}}_j < 0$ and 1, otherwise. Obviously, given that o is not smooth, the error function is not differentiable. Hence we work with the unthresholded perceptron unit.

2 Multi Layered Perceptrons

2.1 Generalizing to Multiple Labels

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

Multi-class classification is more applicable than binary classification. Applications include, handwritten digit recognition, robotics, etc.

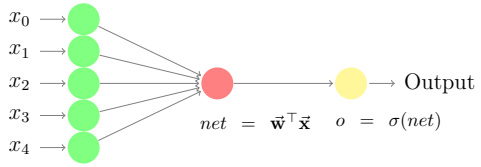
2.2 An Example of a Multilayer Neural Network

One significant application of neural networks in the area of autonomous driving is ALVINN or *Autonomous Land Vehicle In a Neural Network* developed at Carnegie Mellon University in the early 1990s. More information available here: `http://www.ri.cmu.edu/research_project_detail.html?project_id=160&menu_id=261`. ALVINN is a perception system which learns to control the NAVLAB vehicles by watching a person drive. ALVINN's architecture consists of a single hidden layer back-propagation network. The input layer of the network is a 30x32 unit two dimensional "retina" which receives input from the vehicles video camera. Each input unit is fully connected to a layer of five hidden units which are in turn fully connected to a layer of 30 output units. The output layer is a linear representation of the direction the vehicle should travel in order to keep the vehicle on the road.

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

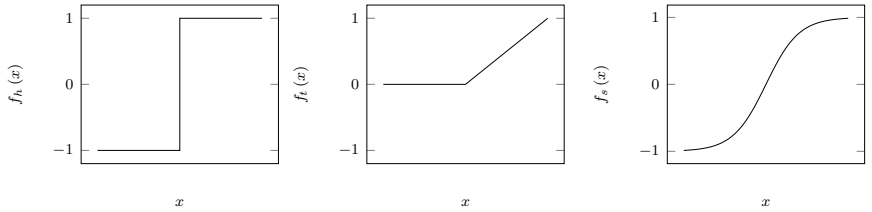
$$\sigma(net) = \frac{1}{1+e^{-net}}$$

  - Non-linear output



As mentioned earlier, the perceptron unit cannot be used as it is not differentiable. The linear unit is differentiable but only learns linear discriminating surfaces. So to learn non-linear surfaces, we need to use a non-linear unit such as the sigmoid.

2.3 Properties of Sigmoid Function

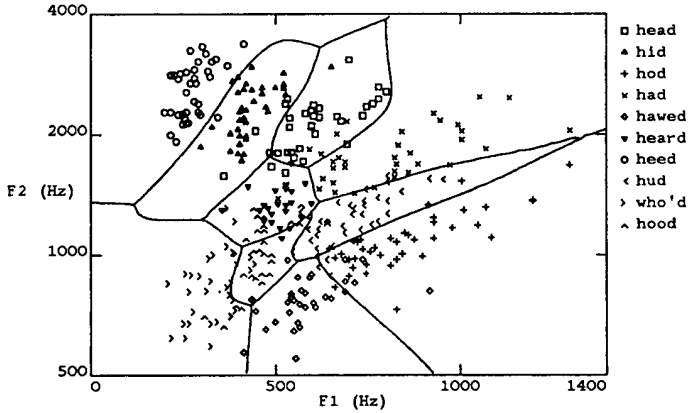


The threshold output in the case of the sigmoid unit is continuous and smooth, as opposed to a perceptron unit or a linear unit. A useful property of sigmoid is that its derivative can be easily expressed as:

$$\frac{d\sigma(y)}{dy} = \sigma(y)(1-\sigma(y))$$

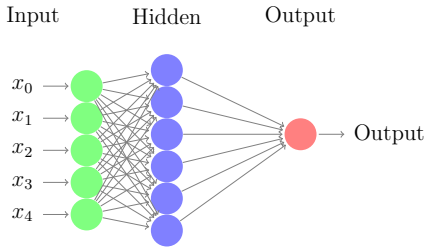
One can also use e^{-ky} instead of e^{-y}, where k controls the "steepness" of the threshold curve.

2.4 Motivation for Using Non-linear Surfaces



The learning problem is to recognize 10 different vowel sounds from the audio input. The raw sound signal is compressed into two features using spectral analysis.

3 Feed Forward Neural Networks



- d input nodes
- m hidden nodes
- k output nodes
- At hidden nodes: $\vec{\mathbf{w}_\mathbf{j}}, 1 \leq j \leq m$, $\vec{\mathbf{w}_\mathbf{j}} \in \Re^d$
- At output nodes: $\vec{\mathbf{w}_\mathbf{j}}, 1 \leq j \leq k$, $\vec{\mathbf{w}_\mathbf{j}} \in \Re^m$

The multi-layer neural network shown above is used in a feed forward mode, i.e., information only flows in one direction (forward). Each hidden node "collects" the inputs from all input nodes and computes a weighted sum of the inputs and then applies the sigmoid function to the weighted sum. The output of each hidden node is forwarded to every output node. The output node "collects" the inputs (from hidden layer nodes) and computes a weighted sum of its inputs and then applies the sigmoid function to obtain the final output. The class corresponding to the output node with the largest output value is assigned as the predicted class for the input.

For implementation, one can even represent the weights as two matrices, W_h $(m \times d)$ and W_o $(k \times m)$.

References