Neural Networks (recap)

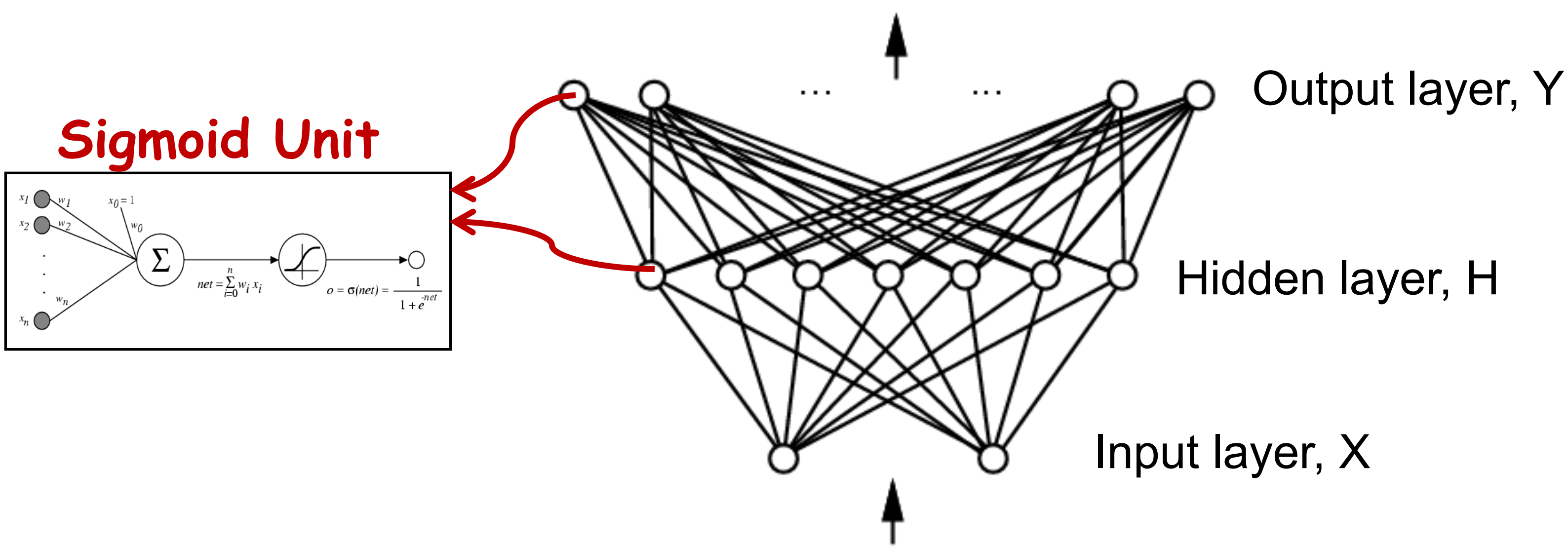
Aarti Singh

Machine Learning 10-315
Sept 18, 2019

Neural Networks to learn f: X → Y

- f can be a **non-linear** function
- X (vector of) continuous and/or discrete variables
- Y (**vector** of) continuous and/or discrete variables

- Neural networks - Represent f by ***network*** of logistic/sigmoid units:



Prediction using Neural Networks

Prediction – Given neural network (hidden units and weights), use it to predict the label of a test point

Forward Propagation –
Start from input layer
For each subsequent layer, compute output of sigmoid unit

Sigmoid unit:

$$o(\mathbf{x}) \;=\; \sigma(w_0 + \sum_i w_i x_i)$$

1-Hidden layer,
1 output NN:

$$o(\mathbf{x}) \;=\; \sigma\left(w_0 + \sum_h w_h \underbrace{\sigma(w_0^h + \sum_i w_i^h x_i)}_{o_h}\right)$$

Training Neural Networks – l2 loss

Train weights of all units to minimize sum of squared errors of predicted network outputs

$$W \leftarrow \arg\min_{W} E[W]$$

$$W \leftarrow \arg\min_{W} \sum_{l} (y^l - \widehat{f}(x^l))^2$$

Learned neural network

Where $\widehat{f}(x^l) = o(x^l)$, output of neural network for training point x^l

Minimize using Gradient Descent

For Neural Networks,
***E*[*w*] no longer convex in w**

Gradient

$$\nabla E[\vec{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots \frac{\partial E}{\partial w_n}\right]$$

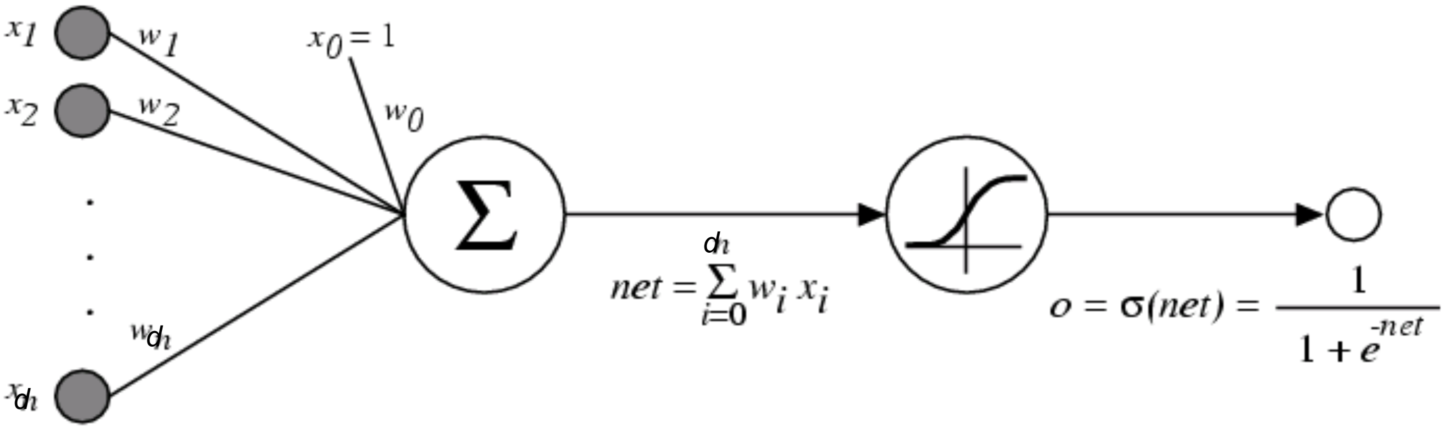
Training rule:

$$\Delta \vec{w} = -\eta \nabla E[\vec{w}]$$

i.e.,

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

Gradient Descent for 1 sigmoid unit



$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{l \in D} (y^l - o^l)^2 = \sum_l (y^l - o^l) \left(-\frac{\partial o^l}{\partial w_i} \right)$$

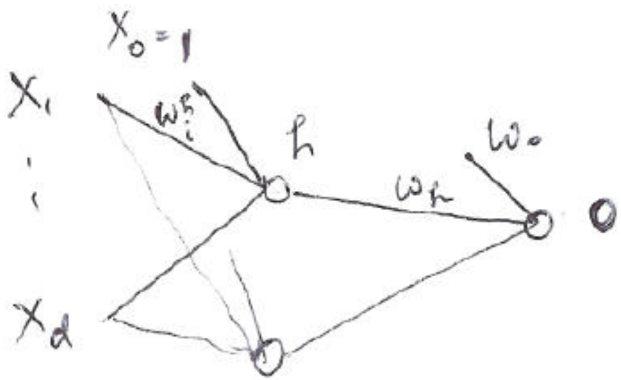
Gradient of the sigmoid function output wrt its input

$$\frac{\partial \sigma(net)}{\partial net} = \sigma(net)(1 - \sigma(net)) = o(1 - o)$$

Gradient of the sigmoid unit output wrt input weights

$$\frac{\partial o}{\partial w_i} = \frac{\partial o}{\partial net} . \frac{\partial net}{\partial w_i} = o(1 - o) x_i$$

Gradient Descent for 1 hidden layer 1 output NN



$$o = \sigma\left(w_0 + \sum_h w_h o_h\right) \equiv \sigma\left(\sum_h w_h o_h\right)$$

$$o_h = \sigma\left(w_0^h + \sum_i w_i^h x_i\right) \equiv \sigma\left(\sum_i w_i^h x_i\right)$$

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{l \in D} (y^l - o^l)^2 = \sum_l (y^l - o^l) \left(-\frac{\partial o^l}{\partial w_i}\right)$$

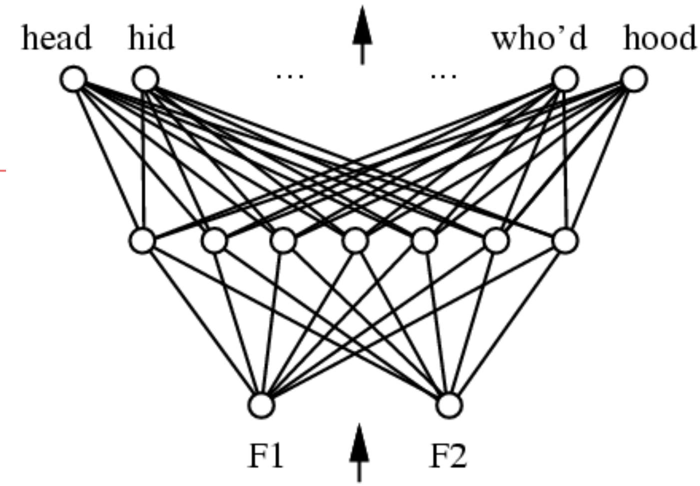
Gradient of the output with respect to w_h

$$\frac{\partial o}{\partial w_i} = o(1 - o)o_h$$

Gradient of the output with respect to input weights w^h_i

$$\frac{\partial o}{\partial w_i^h} = o(1 - o)o_h(1 - o_h)w_h x_i$$

Backpropagation Algorithm (MLE) using Stochastic gradient descent



Initialize all weights to small random numbers.
Until satisfied, Do

- For each training example, Do
  1. Input the training example to the network and compute the network outputs → Using Forward propagation
  2. For each output unit k
  $$\delta_k^l \leftarrow o_k^l(1 - o_k^l)(y_k^l - o_k^l)$$
  3. For each hidden unit h
  $$\delta_h^l \leftarrow o_h^l(1 - o_h^l) \sum_{k \in outputs} w_{h,k}\delta_k^l$$
  4. Update each network weight $w_{i,j}$
  $$w_{i,j} \leftarrow w_{i,j} + \Delta w_{i,j}^l$$
  where
  $$\Delta w_{i,j}^l = \eta \delta_j^l o_i^l$$

l = training example

y_k = target output (label) of output unit k

$o_{k(h)}$ = unit output (obtained by forward propagation)

w_{ij} = wt from i to j

Note: if i is input variable, $o_i = x_i$

More on Backpropagation

- Gradient descent over entire *network* weight vector
- Easily generalized to arbitrary directed graphs
- Will find a local, not necessarily global error minimum
  - In practice, often works well (can run multiple times)
- Often include weight *momentum* α

$$\Delta w_{i,j}(n) = \eta \delta_j x_{i,j} + \alpha \Delta w_{i,j}(n-1)$$

- Minimizes error over *training* examples
  - Will it generalize well to subsequent examples?
- Training can take thousands of iterations $\rightarrow$ slow!
- Using network after training is very fast

Objective/Error no longer convex in weights

Expressive Capabilities of ANNs

Boolean functions:

- Every boolean function can be represented by network with single hidden layer
- but might require exponential (in number of inputs) hidden units

Continuous functions:

- Every bounded continuous function can be approximated with arbitrarily small error, by network with one hidden layer [Cybenko 1989; Hornik et al. 1989]
- Any function can be approximated to arbitrary accuracy by a network with two hidden layers [Cybenko 1988].