Gradient Descent

To understand, consider simpler *linear unit*, where

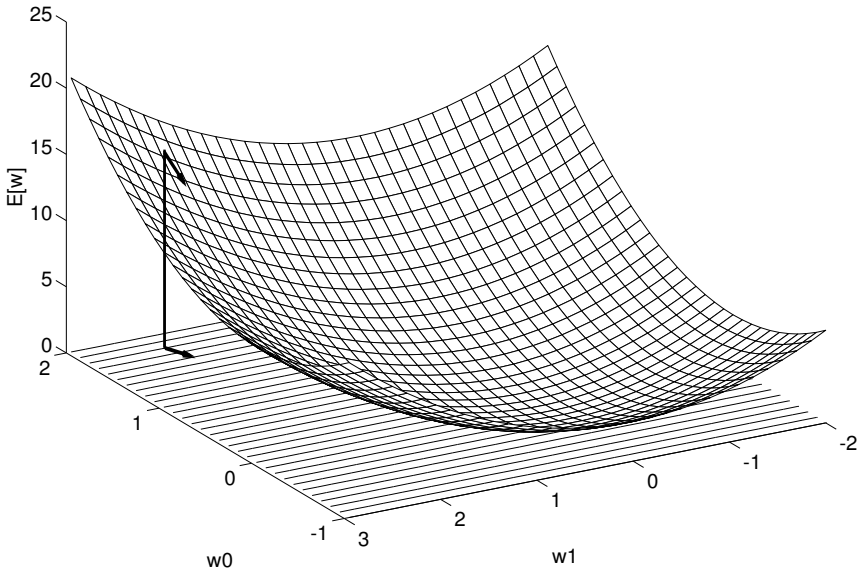
$$o = w_0 + w_1 x_1 + \cdots + w_n x_n$$

Let's learn w_i's that minimize the squared error

$$E[\vec{w}] \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

Where D is set of training examples

Gradient Descent



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

$$
\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i} \frac{1}{2} \sum_d (t_d - o_d)^2 \\
&= \frac{1}{2} \sum_d \frac{\partial}{\partial w_i} (t_d - o_d)^2 \\
&= \frac{1}{2} \sum_d 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d) \\
&= \sum_d (t_d - o_d) \frac{\partial}{\partial w_i} (t_d - \vec{w} \cdot \vec{x_d}) \\
\frac{\partial E}{\partial w_i} &= \sum_d (t_d - o_d)(-x_{i,d})
\end{aligned}
$$

Gradient Descent

GRADIENT-DESCENT($training\_examples, \eta$)

Each training example is a pair of the form $\langle \vec{x}, t \rangle$*, where* $\vec{x}$ *is the vector of input values, and* t *is the target output value.* η *is the learning rate (e.g., .05).*

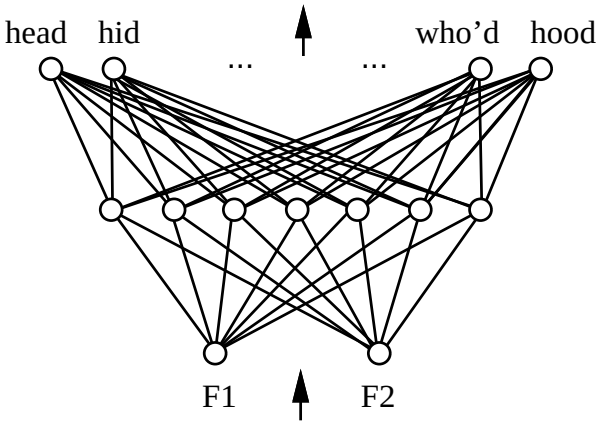
- Initialize each w_i to some small random value
- Until the termination condition is met, Do
  - Initialize each Δw_i to zero.
  - For each $\langle \vec{x}, t \rangle$ in *training_examples*, Do
    - Input the instance $\vec{x}$ to the unit and compute the output o
    - For each linear unit weight w_i, Do

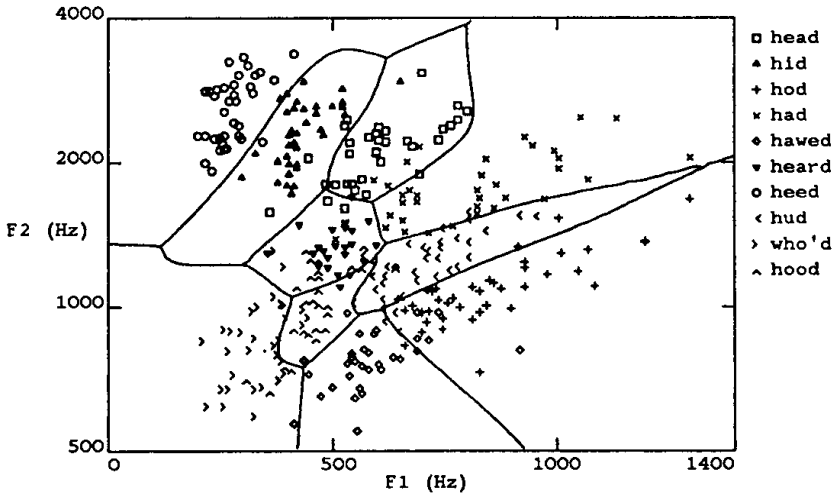
$$\Delta w_i \leftarrow \Delta w_i + \eta(t - o)x_i$$

  - For each linear unit weight w_i, Do

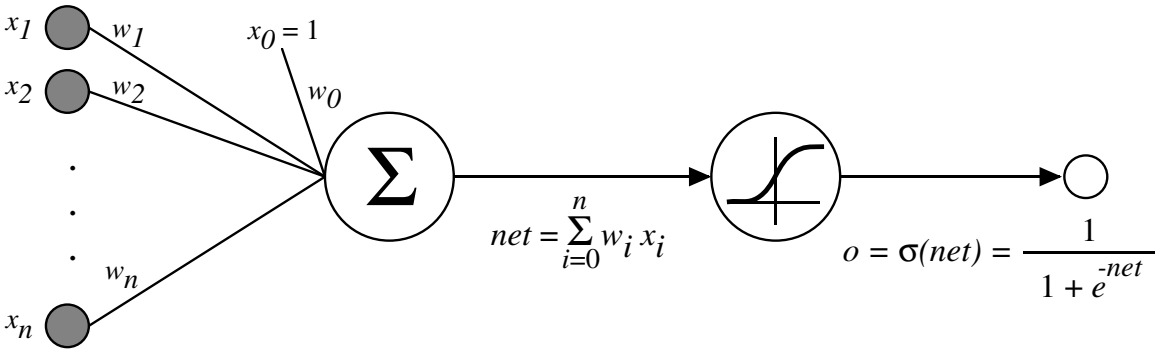
$$w_i \leftarrow w_i + \Delta w_i$$

Multilayer Networks of Sigmoid Units





Sigmoid Unit



$\sigma(x)$ is the sigmoid function

$$\frac{1}{1 + e^{-x}}$$

Nice property: $\frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x))$

We can derive gradient decent rules to train

- One sigmoid unit
- *Multilayer networks* of sigmoid units $\rightarrow$ Backpropagation

Error Gradient for a Sigmoid Unit

$$\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2 \\
&= \frac{1}{2} \sum_d \frac{\partial}{\partial w_i} (t_d - o_d)^2 \\
&= \frac{1}{2} \sum_d 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d) \\
&= \sum_d (t_d - o_d) \left(-\frac{\partial o_d}{\partial w_i} \right) \\
&= -\sum_d (t_d - o_d) \frac{\partial o_d}{\partial net_d} \frac{\partial net_d}{\partial w_i}
\end{aligned}$$

But we know:

$$\frac{\partial o_d}{\partial net_d} = \frac{\partial \sigma(net_d)}{\partial net_d} = o_d(1 - o_d)$$

$$\frac{\partial net_d}{\partial w_i} = \frac{\partial (\vec{w} \cdot \vec{x}_d)}{\partial w_i} = x_{i,d}$$

So:

$$\frac{\partial E}{\partial w_i} = -\sum_{d \in D} (t_d - o_d) o_d (1 - o_d) x_{i,d}$$

Backpropagation Algorithm

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
  2. For each output unit k

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$$

  3. For each hidden unit h

$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{h,k}\delta_k$$

  4. Update each network weight $w_{i,j}$

$$w_{i,j} \leftarrow w_{i,j} + \Delta w_{i,j}$$

     where

$$\Delta w_{i,j} = \eta \delta_j x_{i,j}$$

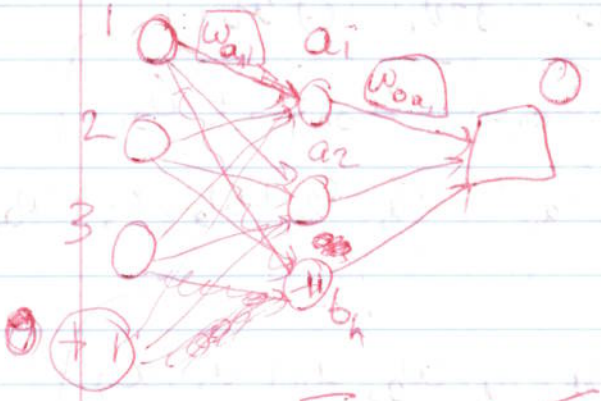
More on Backpropagation

- 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

DERIVATION OF WEIGHT UPDATES



$$E = \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

$$\frac{\partial E}{\partial w_{oa_1}} = \frac{1}{2} \sum_{d \in D} \frac{\partial (t_d - o_d)^2}{\partial w_{oa_1}}$$

$$\Rightarrow \frac{1}{2} \sum_{d \in D} (t_d - o_d) \left(-\frac{\partial o_d}{\partial w_{oa_1}} \right) \Rightarrow$$

$$= \frac{1}{2} \sum_{d \in D} -(t_d - o_d) \frac{\partial o_d}{\partial net_h} \times \frac{\partial net_h}{\partial w_{oa_1}}$$

where $net_h = \sigma(w_{oa_1} a_1 + w_{oa_2} + w_{ob_h})$

$$\therefore \frac{\partial E}{\partial w_{oa_1}} = \frac{1}{2} \sum_{d \in D} -(t_d - o_d) o_d (1 - o_d) a_1$$

δ_k

(from $\frac{\partial \sigma(z)}{\partial z} = \sigma(z)(1 - \sigma(z))$)

Now, for weights going from input to hidden layer

$$\frac{\partial E}{\partial w_{a_1}} = \frac{1}{2} \sum_{d \in D} \frac{\partial (t_d - o_d)^2}{\partial w_{a_1}} = \frac{1}{2} \sum_{d \in D} t_d - o_d \left(\frac{\partial o_d}{\partial w_{oa_1}} \right)$$

$o_d(1-o_d)$ w_{oa_1} $a_1(1-a_1)$ x_1

$$= \frac{1}{2} \sum_{d \in D} -(t_d - o_d) \left(\frac{\partial o_d}{\partial net_h} \right) \left(\frac{\partial net_h}{\partial a_1} \right) \left(\frac{\partial a_1}{\partial net_i} \right) \times \left(\frac{\partial net_i}{\partial w_{a,1}} \right)$$

where, $net_h = \sigma(w_{oa_1} a_1 + w_{oa_2} a_2 + w_{o_{b_h}})$

$net_j = \sigma(w_{a,1} x_1 + w_{a,2} x_2 + w_{a,3} x_3 + w_{b,0})$

Hence,

$$\frac{\partial E}{\partial w_{a,1}} = \frac{1}{2} \sum_{d \in D} -(t_d - o_d) o_d(1-o_d) w_{oa_1} a_1(1-a_1) x_1$$

∂_k

∂_h

Experiments and tricks

Lets talk about how various implementation decisions affect the performance of the neural networks. The following things can be played with:

- Connection of a neuron- Keep your network fully connected
- Number of units in the hidden layer- Do not have a vrey large number of neurons
- Different learning rates- Don’t let your training error oscillate
- Initialization of weights- start with small random weights
- Number of iterations

Tips

- For regression problem scale your output label t to be in [0,1]
- Scale your attribute ranges so that all the attributes are in a similar range
- For eg. year= ((2000-1900)/50)-1 will result in a [-1,1]
- For input attributes only, 'no' should not be 0. -1 should be good.
- For regression problem, you can choose not to use a sigmoid for output(for output only, hidden layers still have sigmoid)
- Your weight update rule will change significantly if you don't have a sigmoid over output(Its easy to derive)
- However, using a sigmoid fuction for regression is acceptable too. Remember to scale your 't' to [0,1]

Slide 8 on backpropagation is not perfect

- The iteration scheme is that of stochastic gradient descent on that slide. We recommend you to follow normal gradient descent scheme.
- the δ_k s should not be multiplied by input attributes. They should be multiplied by their hidden neuron values.
- $x_{i,j}$ should be $x_{i,d}$ where i is the attribute number and d is the training instance.