Neural Networks

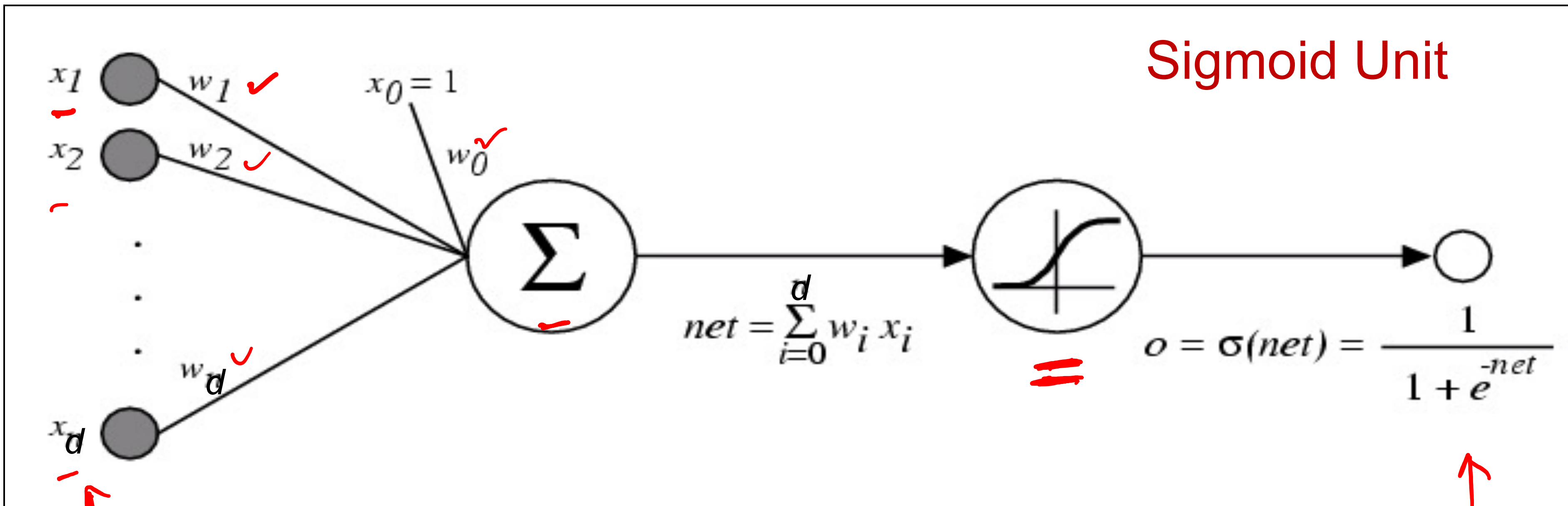
Aarti Singh

Machine Learning 10-315
Feb 14, 2022

Logistic function as a Graph

features of input X

$$\text{Output, } o(\mathbf{x}) = \sigma(w_0 + \sum_i w_i X_i) = \frac{1}{1 + \exp(-(w_0 + \sum_i w_i X_i))}$$

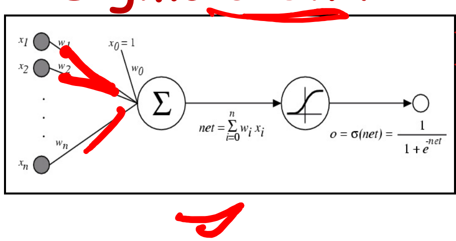


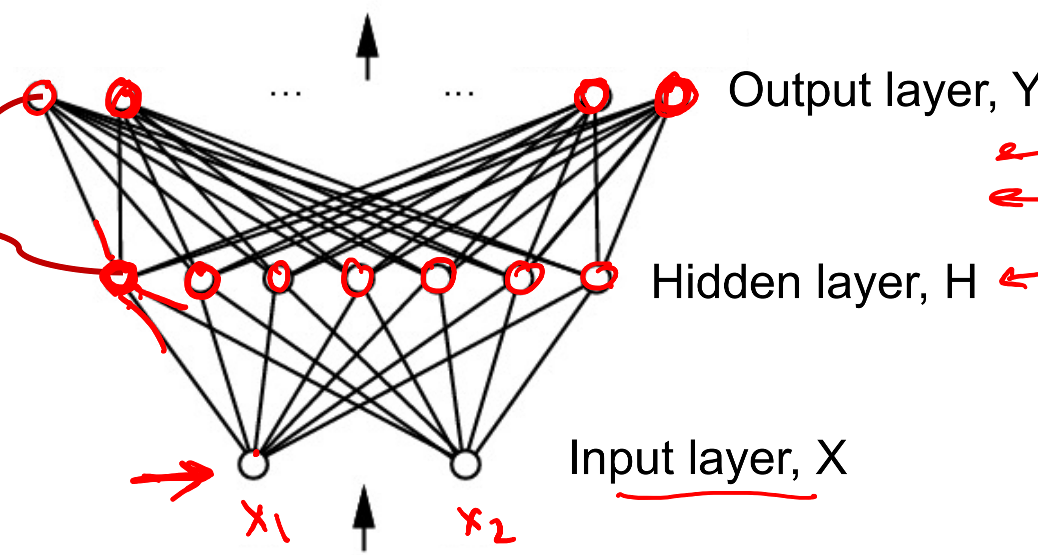
Neural Networks to learn f: X → Y

- 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 sigmoid (more recently ReLU – next lecture) units :

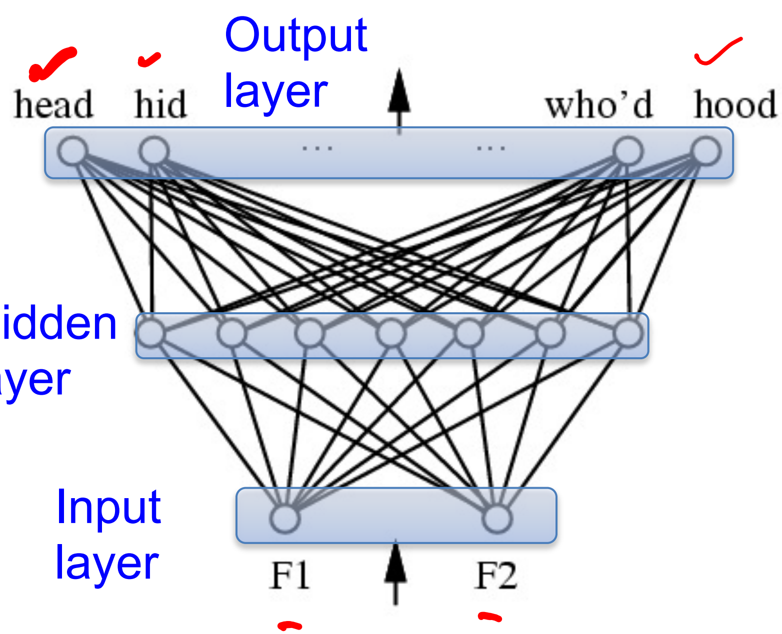
Sigmoid Unit

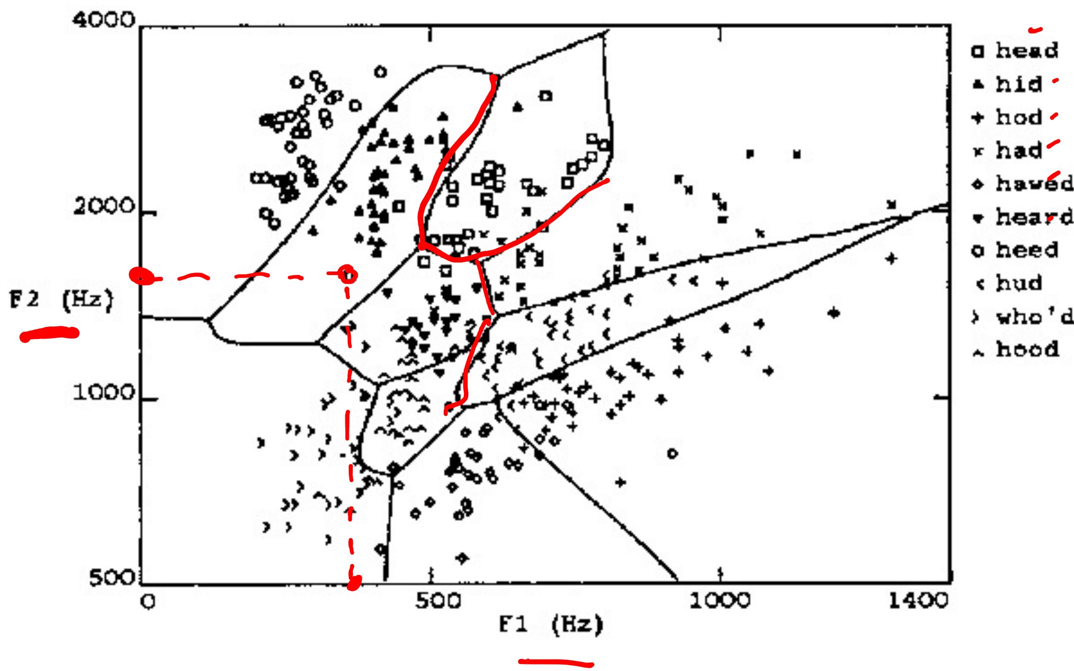




Multilayer Networks of Sigmoid Units

Neural Network trained to distinguish vowel sounds using 2 formants (features)

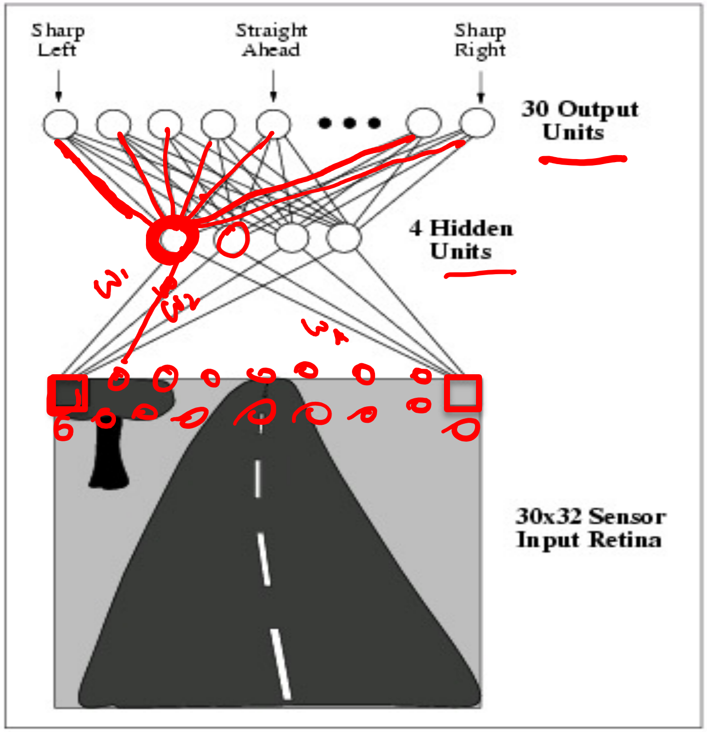




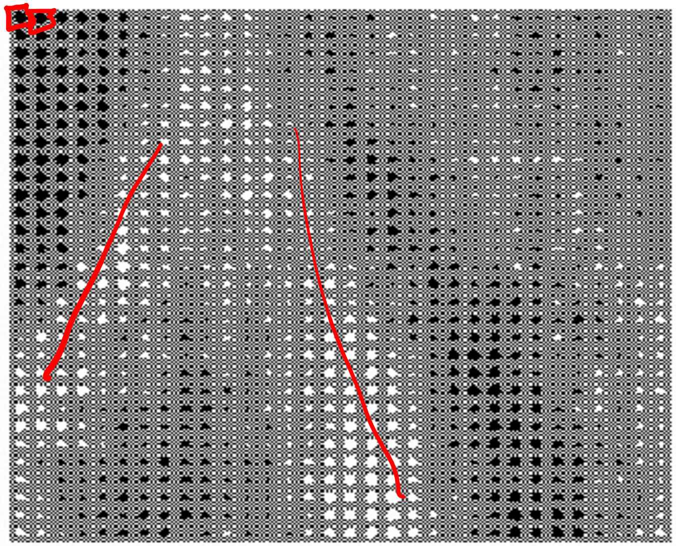
Two layers of logistic units

Highly non-linear decision surface

Neural Network trained to drive a car!



Weights to output units from one hidden unit



Weights of each pixel for one hidden unit

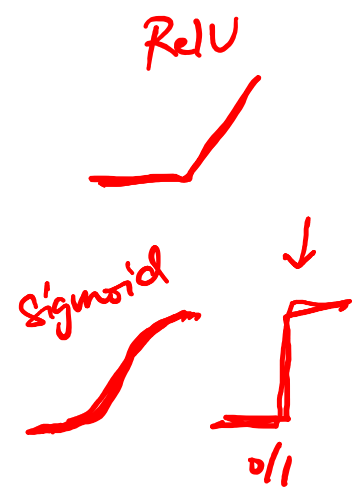
Connectionist Models

brains

Consider humans:

- Neuron switching time ~ .001 second
- Number of neurons ~ 10^{10}
- Connections per neuron ~ 10^{4-5}
- Scene recognition time ~ .1 second
- 100 inference steps doesn't seem like enough

→ much parallel computation



Properties of artificial neural nets (ANN's):

- Many neuron-like threshold switching units
- Many weighted interconnections among units
- Highly parallel, distributed process

Expressive Capabilities of ANNs

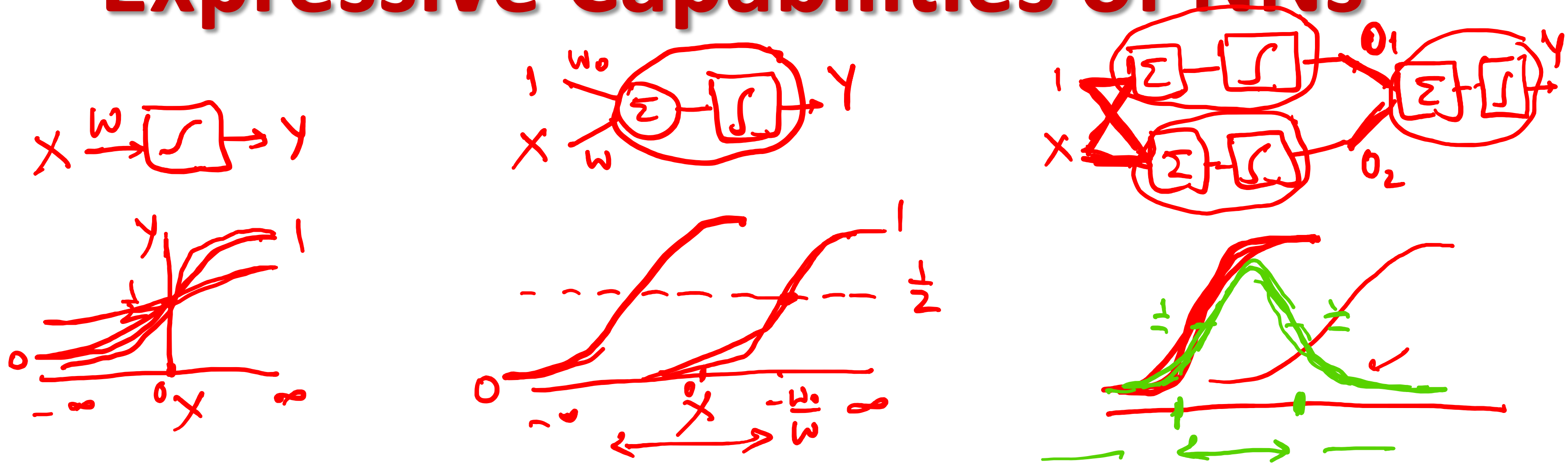
Boolean functions:

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

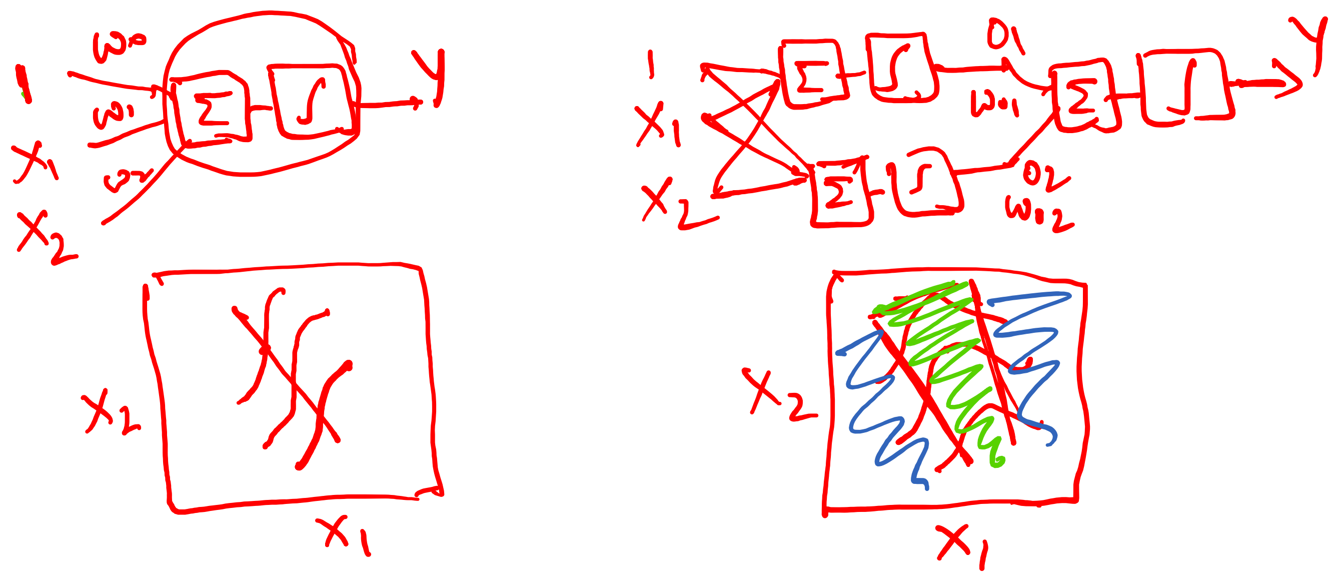
Continuous functions:

- 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].

Expressive Capabilities of NNs



$wx + w_0 = 0$



1 hidden layer NN demo

https://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html

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

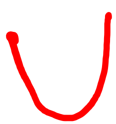
1-Hidden layer,
1 output NN:
$$o(\mathbf{x}) = \sigma\left(\underbrace{w_0 + \sum_h w_h \underbrace{\sigma(w_0^h + \sum_i w_i^h x_i)}_{o_h}}\right)$$

Training Neural Networks – l2 loss

W = { w } first layer ∪ second layer ∪ ... ∪ final layer

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

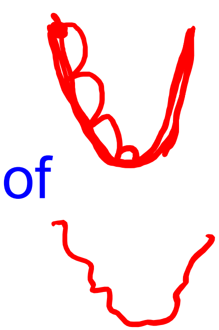
↳ Error



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

Learned neural network

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



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

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.,

$$w_i^{(t+1)} = w_i^{(t)} + \Delta w_i^{(t)}$$

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

Incremental (Stochastic) Gradient Descent

all data

Batch mode Gradient Descent:
Do until satisfied

1. Compute the gradient $\nabla E_D[\vec{w}]$
2. $\vec{w} \leftarrow \vec{w} - \eta \nabla E_D[\vec{w}]$

Using all training data D

$$E_D[\vec{w}] \equiv \frac{1}{2} \sum_{l \in D} (y^l - o^l)^2$$

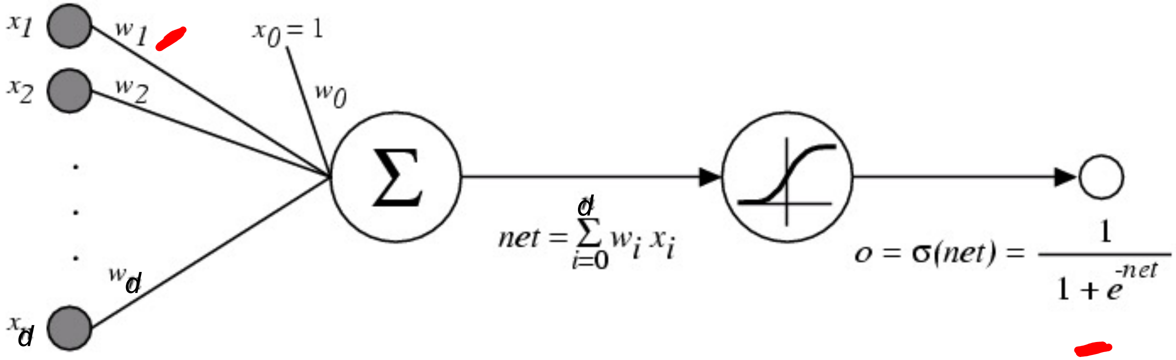
Incremental mode Gradient Descent:
Do until satisfied

- For each training example l in D
  1. Compute the gradient $\nabla E_l[\vec{w}]$
  2. $\vec{w} \leftarrow \vec{w} - \eta \nabla E_l[\vec{w}]$

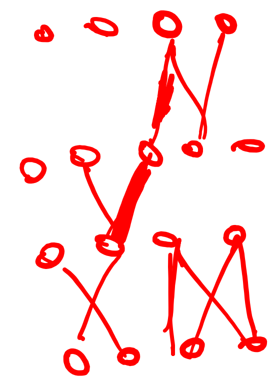
$$E_l[\vec{w}] \equiv \frac{1}{2} (y^l - o^l)^2$$

Incremental Gradient Descent can approximate *Batch Gradient Descent* arbitrarily closely if η made small enough

Training Neural Networks



Err[w]



$\sigma(x)$ is the sigmoid function

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

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

Differentiable

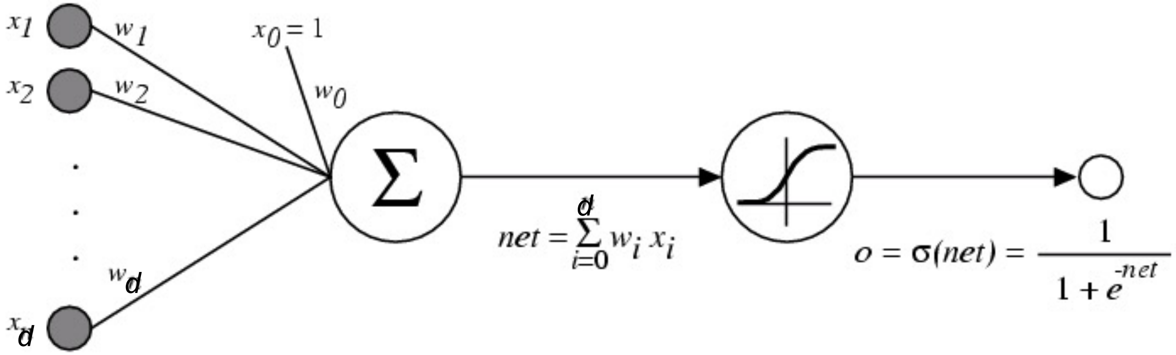
A. σ(x)(1- σ(x)) ✓

B. σ(x) σ(-x)

C. - σ(x)

D. $\sigma(x)^2$

Training Neural Networks



$\sigma(x)$ is the sigmoid function

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

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

We can derive gradient decent rules to train

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

Gradient descent for training NNs

Gradient descent via Chain rule for computing gradients

= **Back-propagation** algorithm for training NNs

Gradient of loss with respect to one weight w_i

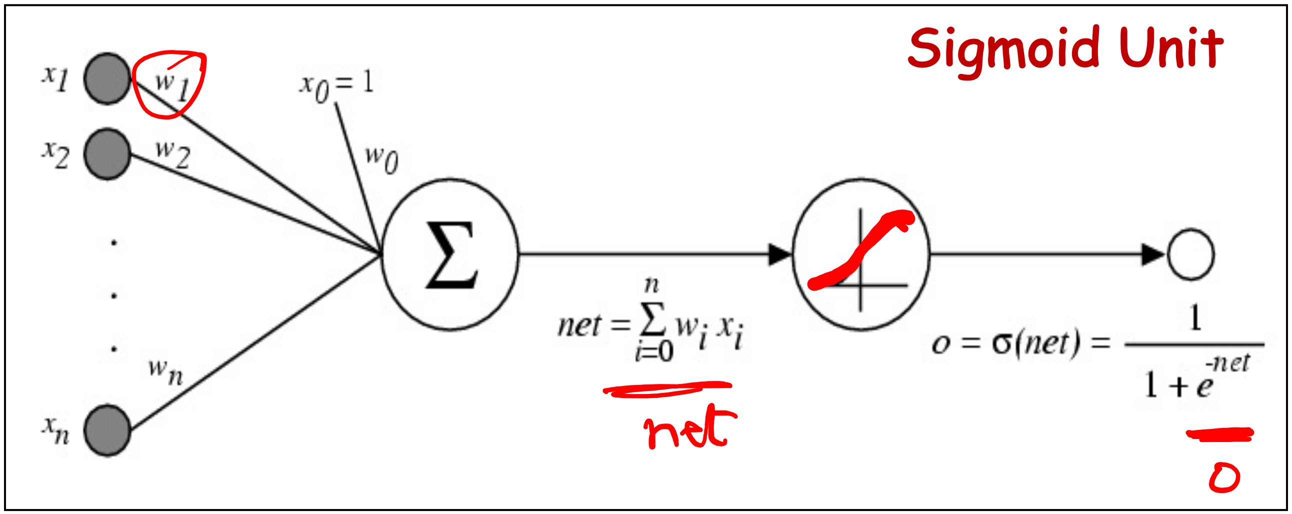
$$= \frac{\partial E}{\partial o^l} \cdot \frac{\partial o^l}{\partial w_i}$$

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

$$w_i^{(t+1)} = w_i^{(t)} + 2\eta \sum_l (y^l - o^l)\left(+\frac{\partial o^l}{\partial w_i}\right)$$

Stochastic gradient: $w_i^{(t+1)} = w_i^{(t)} + 2\eta\,(y^l - o^l)\left(\frac{\partial o^l}{\partial w_i}\right)$

Gradient Descent for 1 node



$\frac{\partial E}{\partial w_i}$

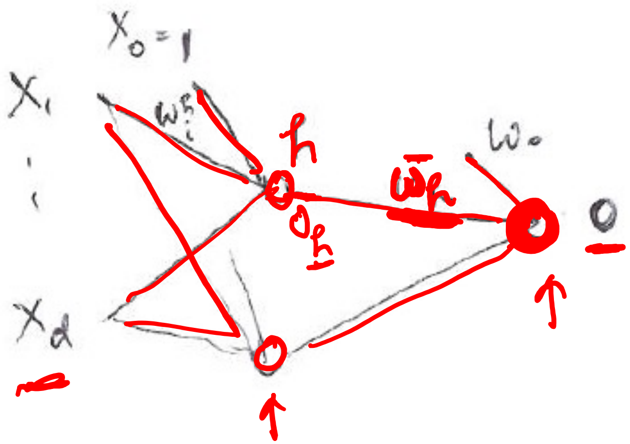
$\sigma(x)(1-\sigma(x))$

$net = \sum_i w_i x_i$

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

Gradient descent step: $w_i^{(t+1)} = w_i^{(t)} + 2\eta(y^l - o^l)\, o^l(1-o^l)\, x_i^l$

Gradient Descent for 1 hidden layer 1 output NN



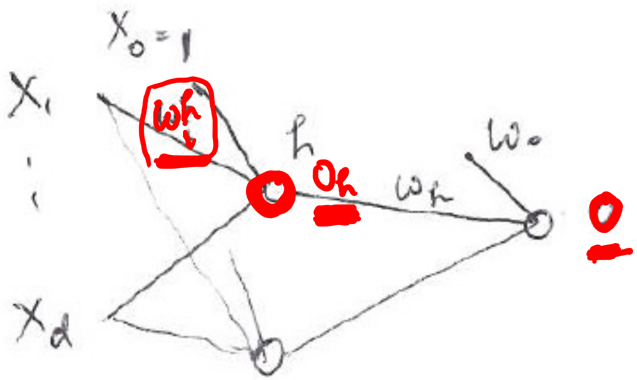
$$o = \sigma\left(w_0 + \sum_h w_h o_h\right) = \sigma\left(\sum_h w_h o_h\right)$$

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

Gradient of the output with respect to one **final** layer weight w_h

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

Gradient Descent for 1 hidden layer 1 output NN

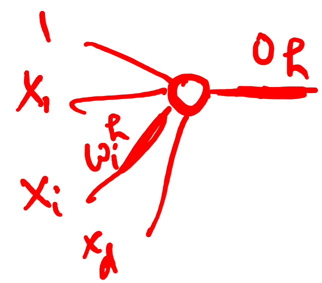


$$o = \sigma\left(w_0 + \sum_h w_h o_h\right) \equiv \sigma\left(\sum_h w_h o_h\right)$$

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

Gradient of the output with respect to one **hidden** layer weight w_i^h

$$\frac{\partial o}{\partial w_i^h} = \frac{\partial o}{\partial o_h} \cdot \frac{\partial o_h}{\partial w_i^h}$$



$$\frac{\partial o_h}{\partial w_i^h} =$$

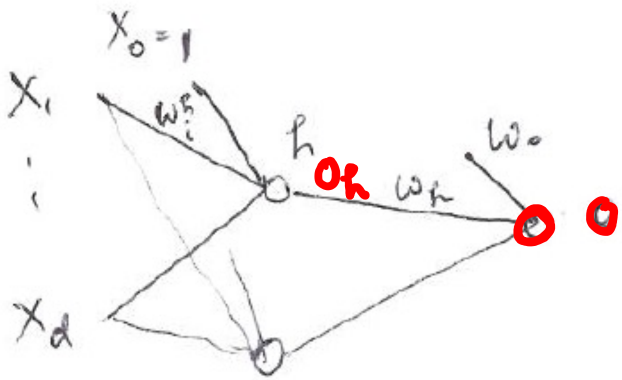
A. $o_h(1-o_h)$

B. $o_h(1-o_h)x_i$

C. $o(1-o)\, x_i$

D. $o(1-o)$

Gradient Descent for 1 hidden layer 1 output NN

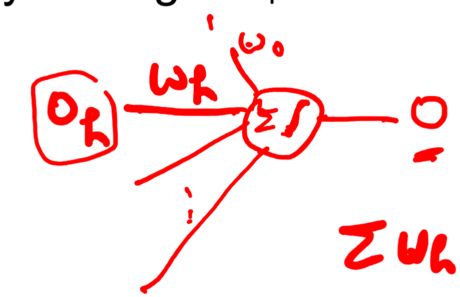


$$o = \sigma\left(w_0 + \sum_h w_h o_h\right) = \sigma\left(\sum_h w_h o_h\right)$$

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

Gradient of the output with respect to one **hidden** layer weight w_i^h

$$\frac{\partial o}{\partial w_i^h} = \frac{\partial o}{\partial o_h} \cdot \frac{\partial o_h}{\partial w_i^h}$$



$\sum w_h o_h$

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

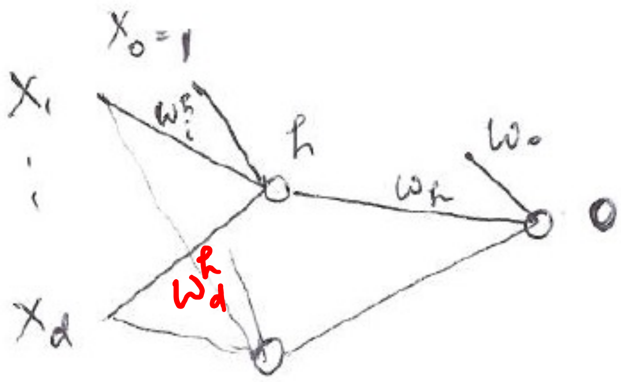
A. $o(1-o)x_i$

B. $o(1-o)o_h$

C. $o(1-o)\, w_h$

D. $o(1-o)$

Gradient Descent for 1 hidden layer 1 output NN



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

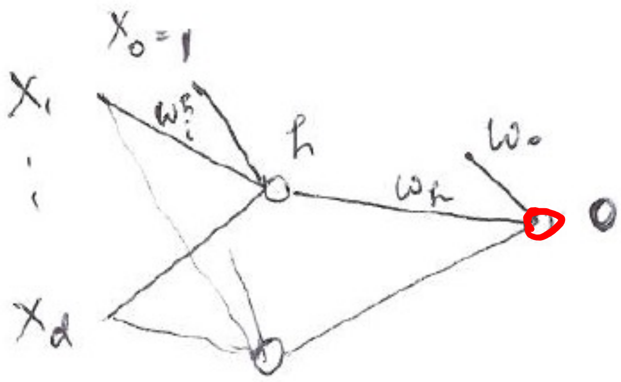
$$o_h = \sigma\left(w_0^h + \sum_i w_i^h x_i\right) = \sigma\left(\sum_i w_i^h x_i\right)$$

Gradient of the output with respect to one **hidden** layer weight w_i^h

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

$$\frac{\partial o}{\partial w_{i'}^h} = \frac{\partial o}{\partial o_h} \cdot \frac{\partial o_h}{\partial w_{i'}^h} = o(1-o)w_h \cdot o_h(1-o_h)x_{i'}$$

Gradient Descent for 1 hidden layer 1 output NN



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

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

Gradient of the output with respect to one **final** layer weight w_h

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

Gradient of the output with respect to one **hidden** layer weight w_i^h

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