Neural Networks

in 1 lecture!

Machine Learning – 10701/15781

Carlos Guestrin

Carnegie Mellon University

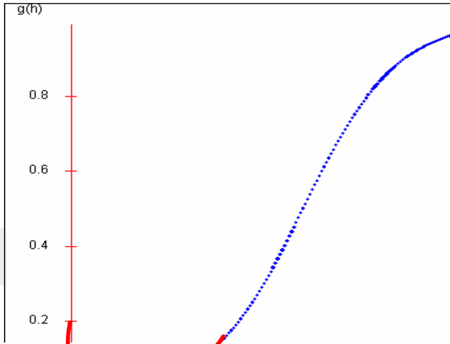
January 31st, 2005

Announcements

- First homework due Wednesday
  - **Beginning** of class
  - If using late days (24 hours per late day), timestamp and hand to Sharon Cavlovich, Wean Hall 5315

w_0 is the weight for constant: $x_0^j = 1 \ \forall j$

Logistic regression



- P(Y|X) represented by:

$$P(Y = 1 \mid x, W) = \frac{1}{1 + e^{-(w_0 + \sum_i w_i x_i)}}$$
$$= g(w_0 + \sum_i w_i x_i)$$

Sigmoid

- Learning rule – MLE:

$$\frac{\partial \ell(W)}{\partial w_i} = \sum_j x_i^j [y^j - P(Y^j = 1 \mid x^j, W)]$$

data

$$= \sum_j x_i^j [y^j - g(w_0 + \sum_i w_i x_i^j)]$$

$$\boxed{w_i \leftarrow w_i + \eta \sum_j x_i^j \delta^j}$$

how will j is classified

$$\delta^j = y^j - g(w_0 + \sum_i w_i x_i^j)$$

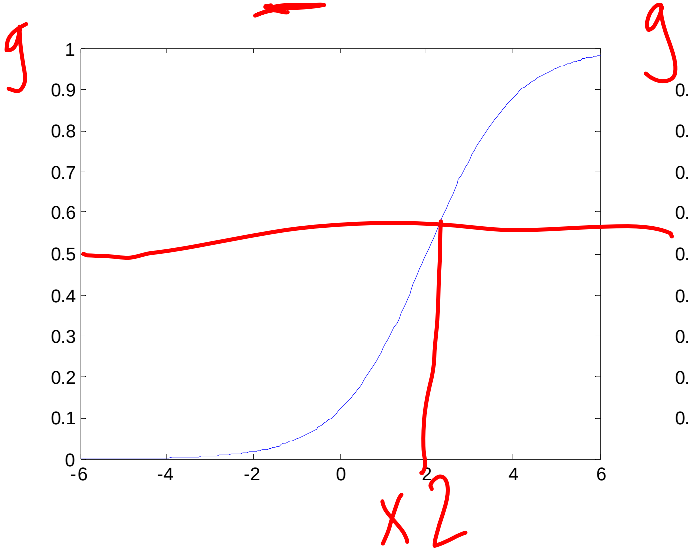
learning rate

Sigmoid

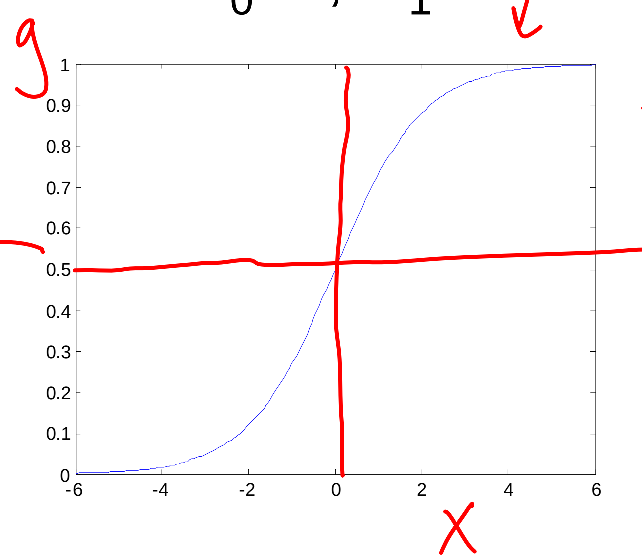
$$g(w_0 + \sum_i w_i x_i) = \frac{1}{1 + e^{-(w_0 + \sum_i w_i x_i)}}$$

change weight constant

w_0=2, w_1=1

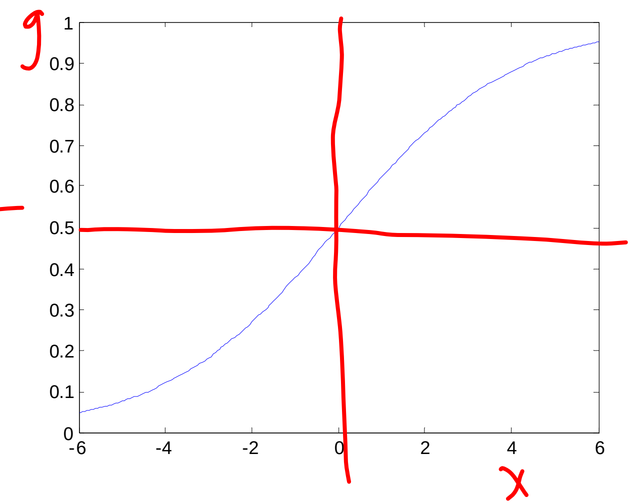


w_0=0, w_1=1

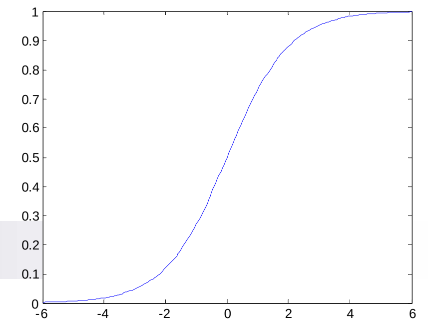


change w_1

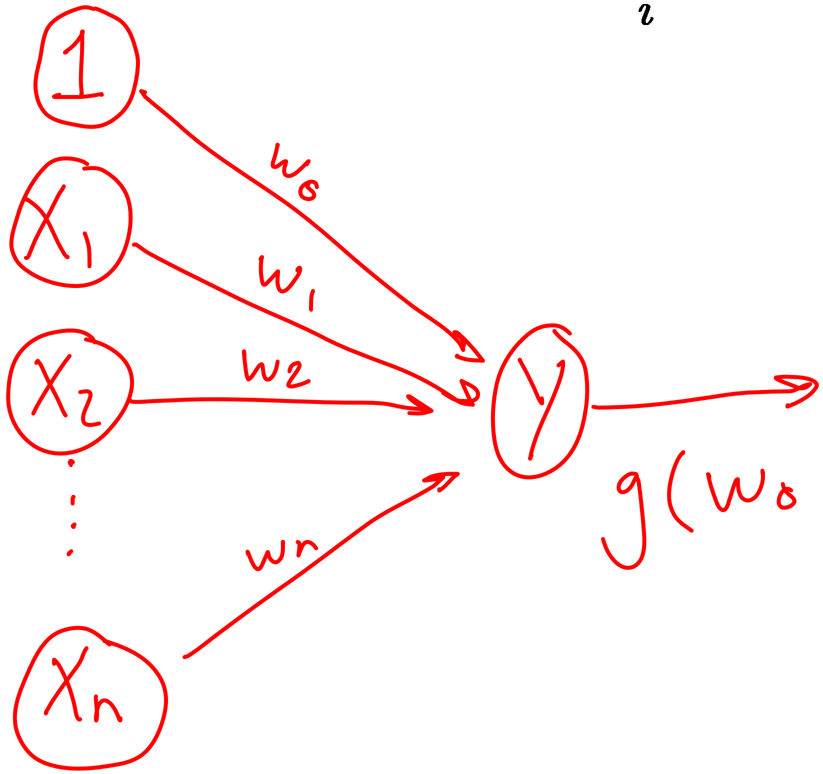
w_0=0, w_1=0.5



Perceptron as a graph



$$g(w_0 + \sum_i w_i x_i) = \frac{1}{1 + e^{-(w_0 + \sum_i w_i x_i)}}$$

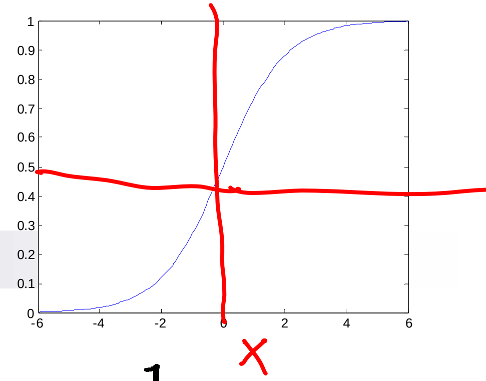


$$g(w_0 + \sum_i w_i x_i)$$

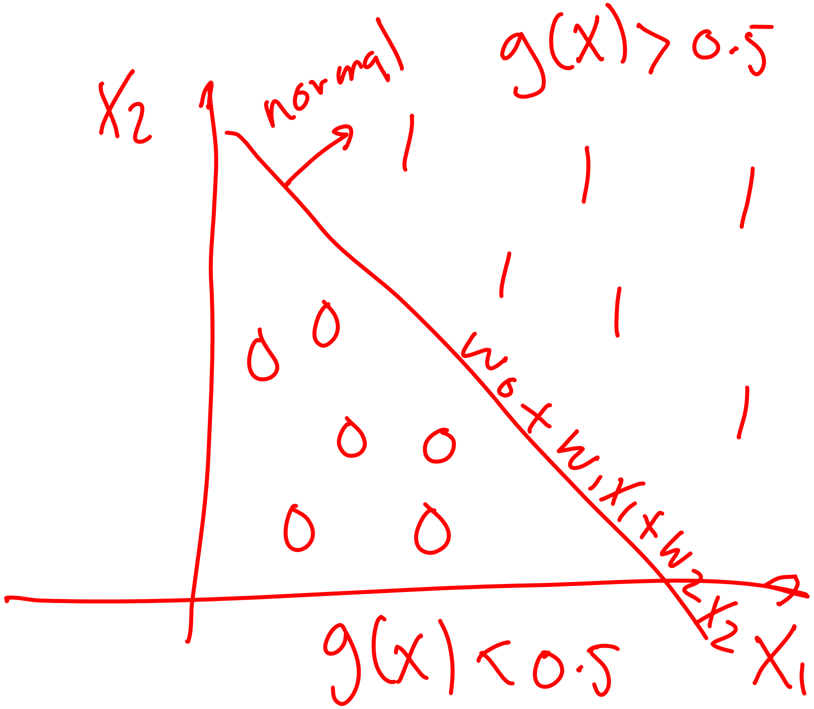
Linear perceptron classification region

$g(x) > 0.5$ when $x > 0$

$g(x) < 0.5$ when $x < 0$



$$g(w_0 + \sum_i w_i x_i) = \frac{1}{1 + e^{-(w_0 + \sum_i w_i x_i)}}$$



$g(w_0 + \sum_i w_i x_i)$

$g(\text{line})$

Optimizing the perceptron

- Trained to minimize sum-squared error

$$\ell(W) = \frac{1}{2}\sum_j \underbrace{[y^j - g(w_0 + \sum_i w_i x_i^j)]^2}$$

$$\frac{\partial \ell(W)}{\partial w_i} = \sum_j -[y^j - g(w_0 + \sum_i w_i x_i^j)] *$$

$$\frac{\partial}{\partial w_i} g(w_0 + \sum_i w_i x_i^j)$$

derivative of Sigmoids

Derivative of sigmoid

$$\frac{\partial \ell(W)}{\partial w_i} = -\sum_j [y^j - g(w_0 + \sum_i w_i x_i^j)] \; x_i^j \; g'(w_0 + \sum_i w_i x_i^j)$$

$$g(x) = \frac{1}{1+e^{-x}}$$

$$\frac{\partial g(x)}{\partial x} = \frac{+e^{-x}}{(1+e^{-x})^2}$$

$$= \frac{1 - 1 + e^{-x}}{(1+e^{-x})^2}$$

$$= \overbrace{\boxed{\frac{1}{(1+e^{-x})}}}^{g(x)} \left(\frac{-1}{1+e^{-x}} + \cancelto{1}{\frac{1+e^{-x}}{1+e^{-x}}} \right)$$

$$\frac{\partial g(x)}{\partial x} = g(x)\left(1 - g(x)\right)$$

The perceptron learning rule

learing rate

training example feature

$$\boxed{w_i \leftarrow w_i + \eta \sum_j x_i^j \delta^j}$$

difference

$$\delta^j = \underbrace{[y^j - g(w_0 + \sum_i w_i x_i^j)]}\, g^j (1 - g^j)$$

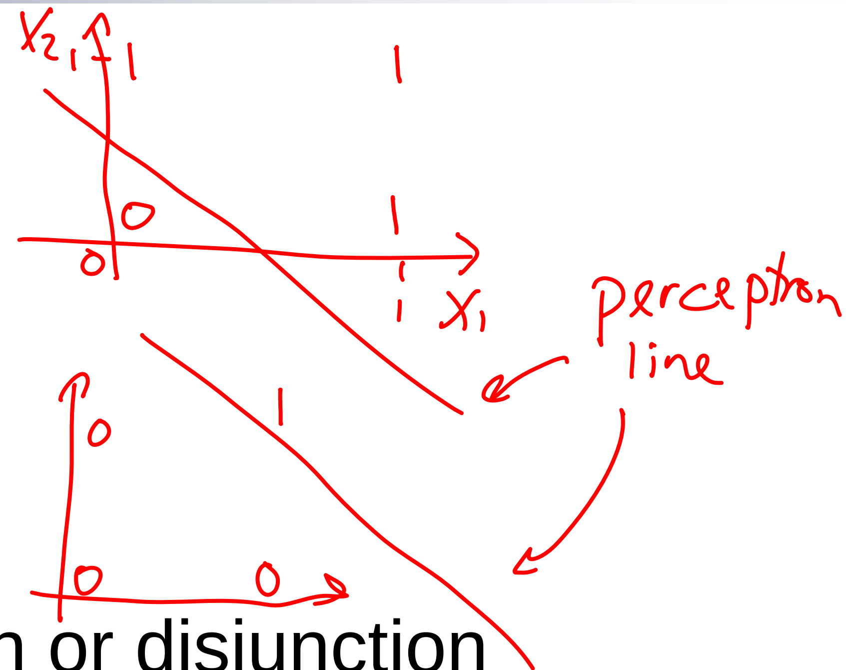
$$g^j = g(w_0 + \sum_i w_i x_i^j)$$

- Compare to MLE:

$$\boxed{w_i \leftarrow w_i + \eta \sum_j x_i^j \delta^j} \qquad \delta^j = \overbrace{[y^j - g(w_0 + \sum_i w_i x_i^j)]}$$

Percepton, linear classification, Boolean functions

- Can learn $x_1 \vee x_2$
- Can learn $x_1 \wedge x_2$
- Can learn any conjunction or disjunction



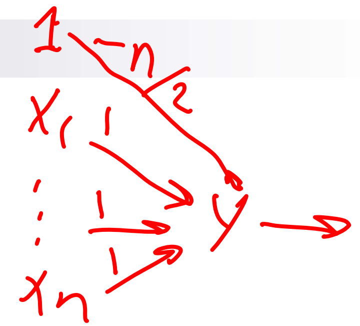
$x_1 \wedge \sim x_2 \wedge x_3 \wedge x_5 \wedge \sim x_6$

$x_1 \vee \sim x_2 \vee x_3 \vee x_5 \vee \sim x_6$

Percepton, linear classification, Boolean functions

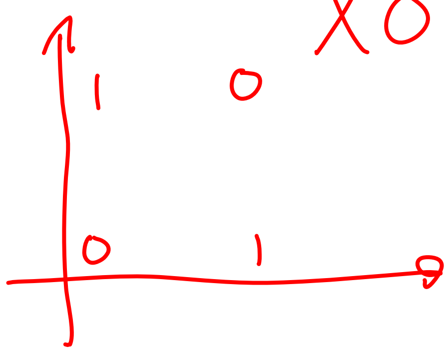
- Can learn majority

$$\sum_{i=1}^{n} x_i > n/2$$



- Can perceptrons do everything?

XOR!!!



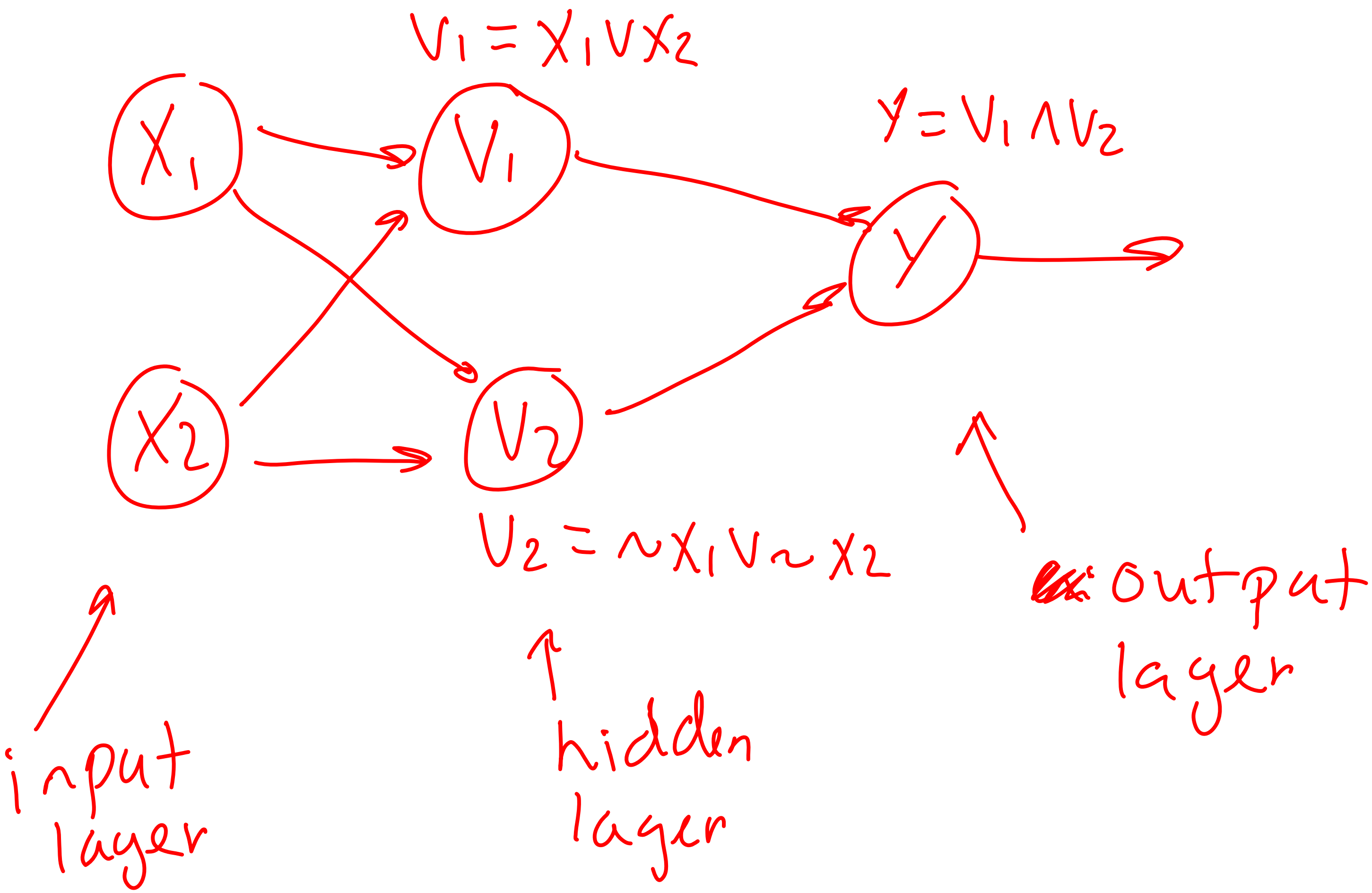
$$(x_1 \vee x_2) \wedge (\sim x_1 \vee \sim x_2)$$

no line can separate

Going beyond linear classification

- Solving the XOR problem

$(x_1 \vee x_2) \wedge (\sim x_1 \vee \sim x_2)$



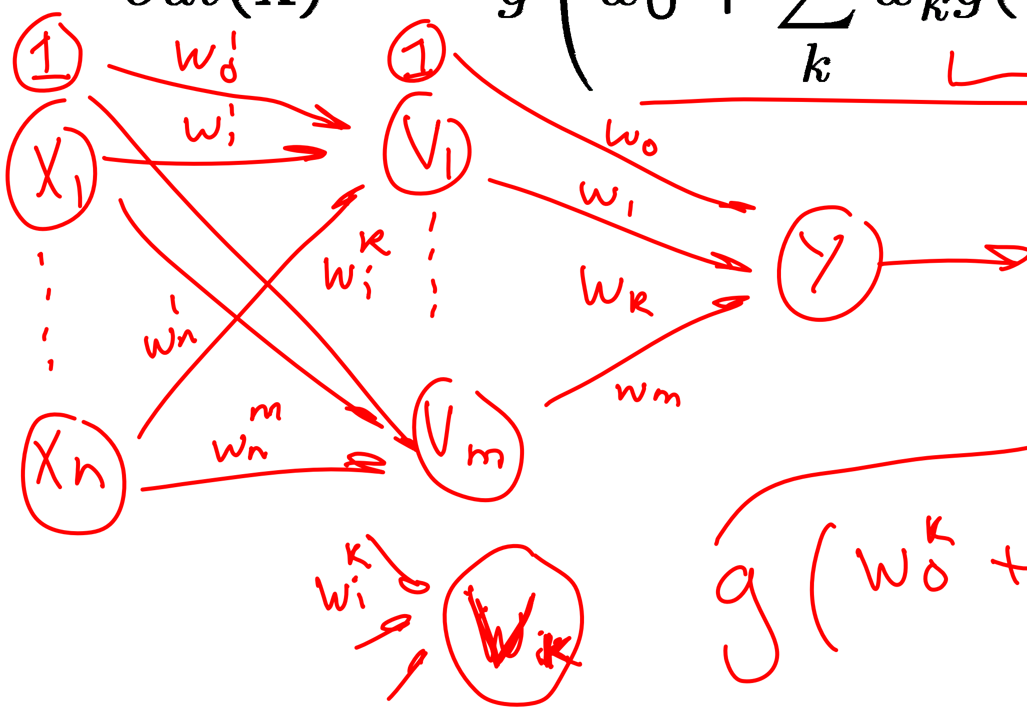
Hidden layer

- Perceptron: $$out(\mathbf{x}) = g(w_0 + \sum_i w_i x_i)$$

- 1-hidden layer:

superscripts are indexes, not exp!

$$out(\mathbf{x}) = g\left(w_0 + \sum_k w_k \underbrace{g(w_0^k + \sum_i w_i^k x_i)}\right)$$



$$g(w_0^k + \sum_i w_i^k x_i)$$

Dropped w_0 for simpler notation

Gradient descent for 1-hidden layer

perceptron → $\boxed{w_i \leftarrow w_i + \eta\, x_i\, \delta}$

$$\delta = \underbrace{[y - g(w_0 + \sum_i w_i x_i)]}_{}\underline{g(1-g)}$$

"x_k" $= g(\sum_i w_i^k x_i)$

"x_i" $= x_i$

"δ" $= [y - g(\ )]$, $\times$ (*)

$$out(\mathbf{x}) = g\left(\sum_k w_k g(\sum_i w_i^k x_i)\right)$$

$$\frac{\partial \ell(W)}{\partial w} = -[y - out(\mathbf{x})]\frac{\partial out(\mathbf{x})}{\partial w}$$

output layer:

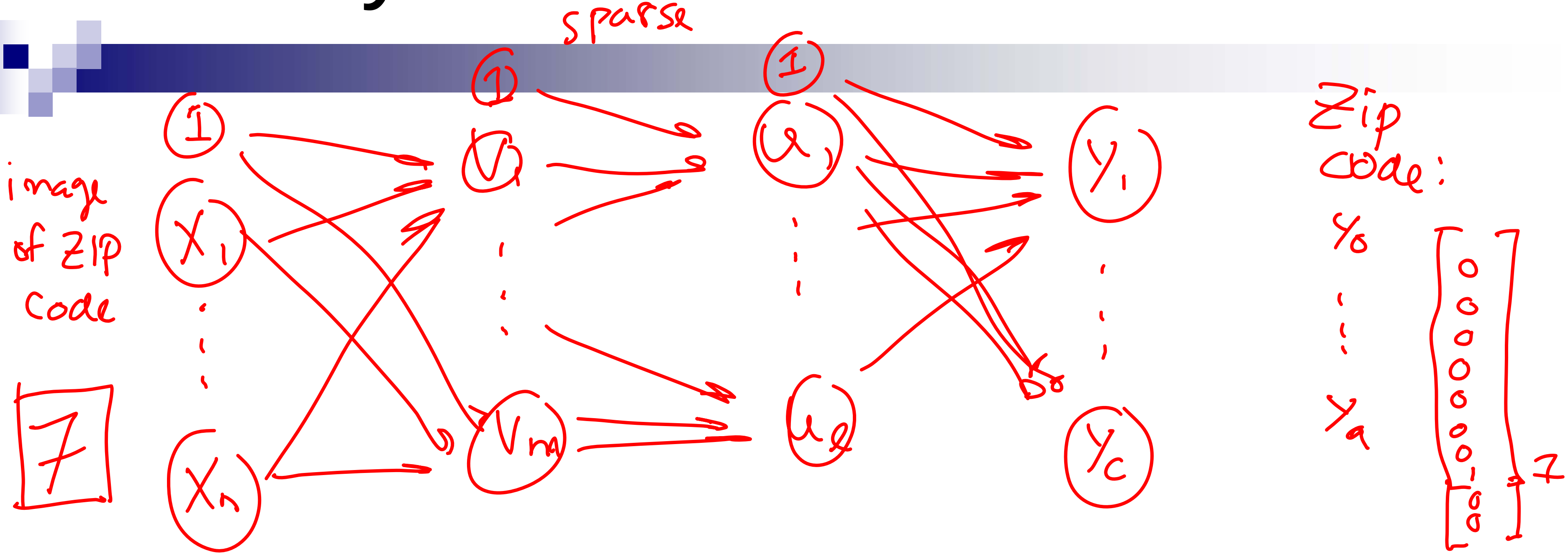
$$\frac{\partial out(x)}{\partial w_k} = g(\sum_i w_i^k x_i) \cdot \overbrace{g'(\sum_k w_k g(\sum_i w_i^k x_i))}^{(*)}$$

$$= \underbrace{g(\sum_i w_i^k x_i)}_{x_i \text{ in the perceptron}}\ g\left[\sum_k w_k g(\sum_i w_i^k x_i)\right]\left[1 - g(\sum_k w_k g(\sum_i w_i^k x_i))\right]$$

hidden Layer:

$$\frac{\partial out(x)}{\partial w_i^k} = \underbrace{g'(\sum_k w_k g(\sum_i w_i^k x_i))}_{(*)} \cdot \underbrace{\frac{\partial}{\partial w_i^k}\left[\sum_k w_k g(\sum_i w_i^k x_i)\right]}_{w_k\, g'(\sum_i w_i^k x_i)\, x_i}$$

Multilayer neural networks

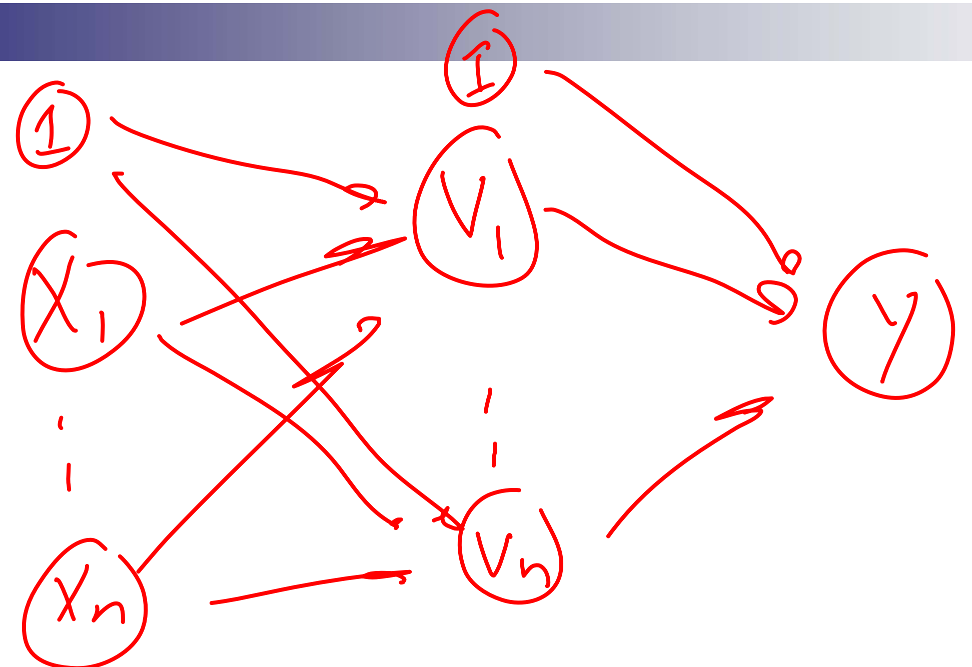


training data

input $(x_1, \ldots, x_n) \to$ output $(y_1, \ldots, y_c)$

Forward propagation – prediction

given new example x



Forward propagation

plug x → compute each hidden unit → compute output based on hidden

For multi layer: Recursion!

Back-propagation – learning

Basically gradient descent

For each example x, y

Forward propagation

Backwards updating weights

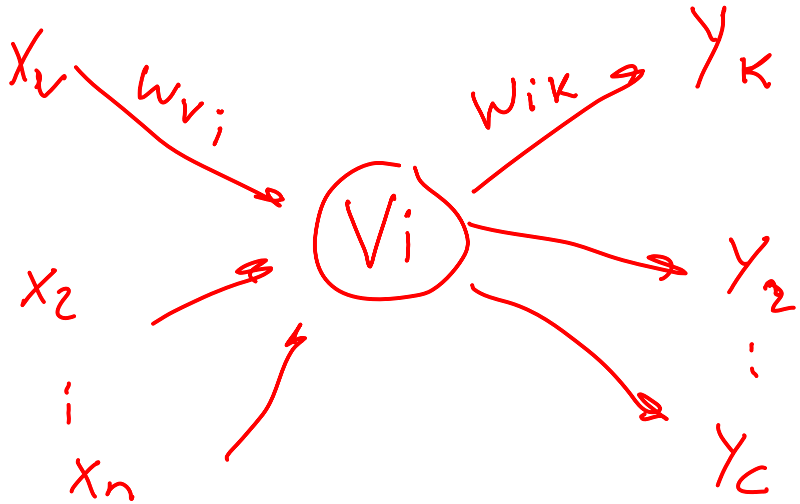
Back-propagation – learning

$$\boxed{w_i \leftarrow w_i + \eta \; x_i \; \delta}$$

$$\delta = [y - g(w_0 + \sum_i w_i x_i)]g(1-g)$$

Hidden unit:

x_v computed by forward propagation



$w_{vi} \leftarrow w_{vi} + \eta \, x_v \cdot \delta_i$

$\delta_i = g(\sum_v w_{vi} x_v) * [1 - g(\sum_v w_{vi} x_v)] * \sum_k w_{ik} \delta_k$

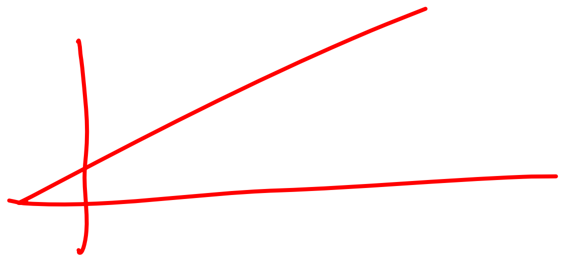
Start of recursion for δ:

output layer: $\delta = [y - g(\text{output})]$

Many possible response functions

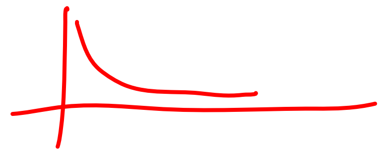
- Sigmoid

- Linear

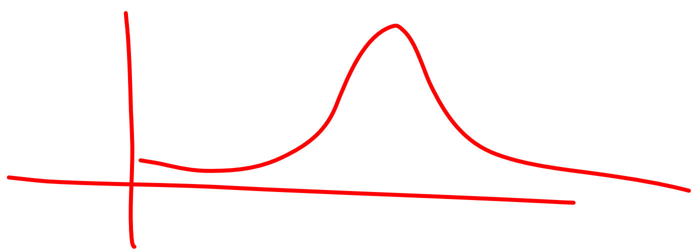


$g = \sum_i w_i x_i$

- Exponential



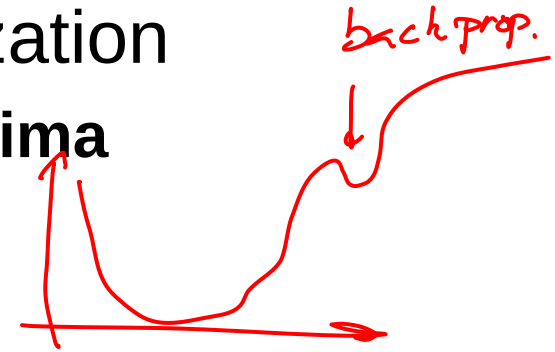
- Gaussian



- …

Convergence of backprop

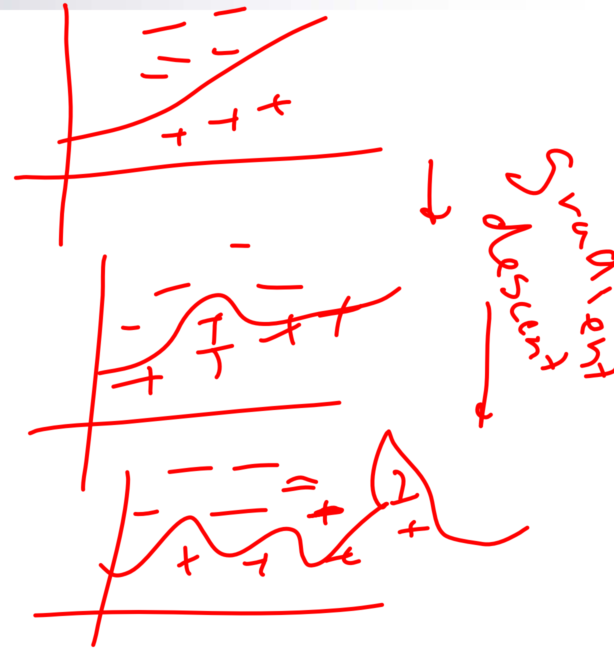
- Perceptron leads to convex optimization
  - Gradient descent reaches **global minima**



- Multilayer neural nets **not convex**
  - Gradient descent gets stuck in local minima
  - Hard to set learning rate
  - Selecting number of hidden units and layers = fuzzy process
  - NNs falling in disfavor in last few years
  - As we'll see in second half of the semester, *kernel trick* is a good alternative
  - Nonetheless, neural nets are one of the most used ML approaches

Training set error

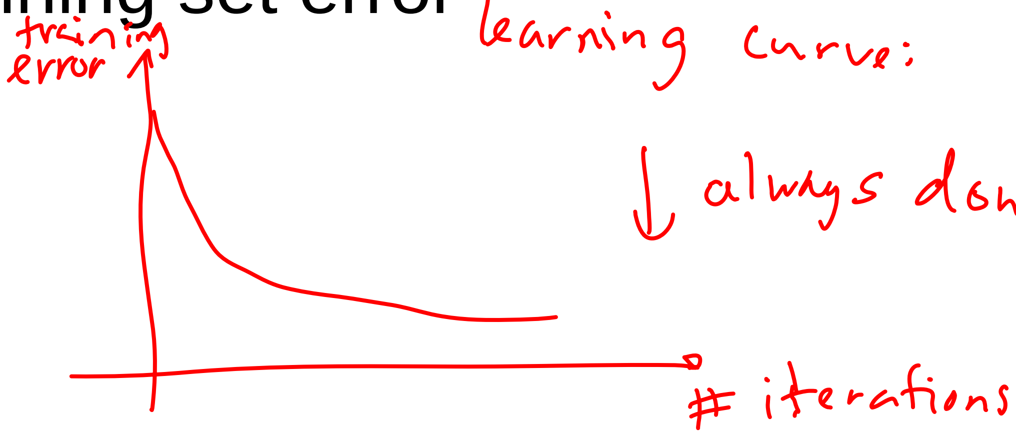
- Neural nets represent complex functions
  - Output becomes more complex with gradient steps



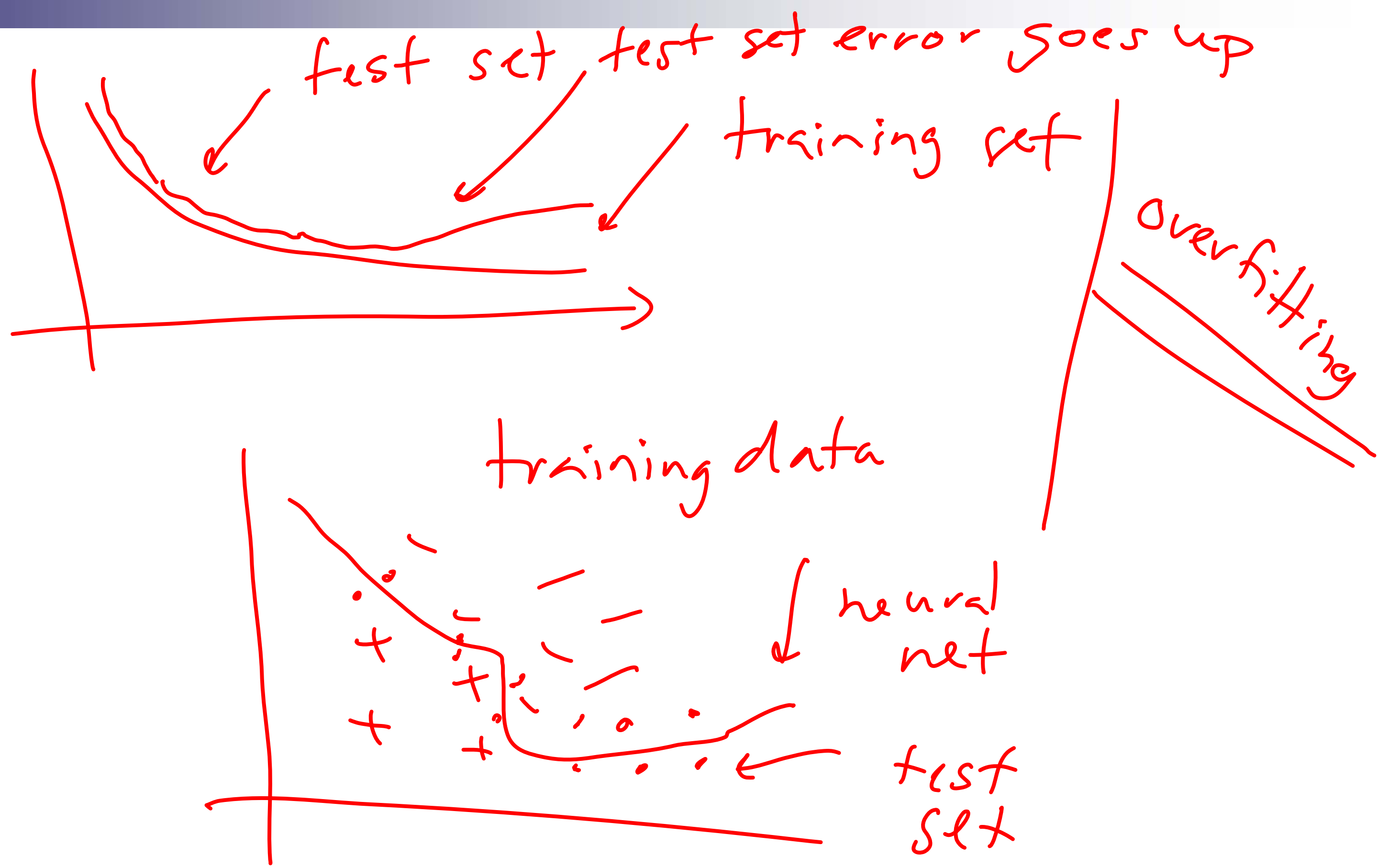
- Training set error

learning curve:

always down

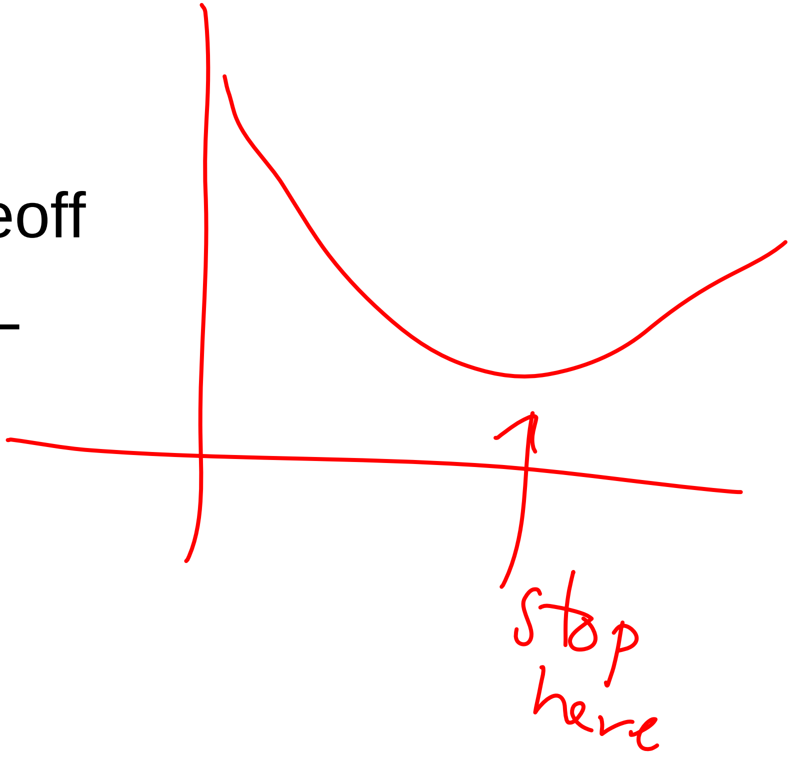


What about test set error?



Overfitting

- Output fits training data "too well"
  - Poor test set accuracy
- Overfitting the training data
  - Related to bias-variance tradeoff
  - One of central problems of ML
- Avoiding overfitting?
  - More training data
  - Regularization
  - Early stopping
  - Next few lectures



What you need to know

- Perceptron:
  - Representation
  - Perceptron learning rule
  - Derivation
- Multilayer neural nets
  - Representation
  - Derivation of backprop
  - Learning rule
- Overfitting
  - Definition
  - Training set versus test set
  - Learning curve
  - Next few classes: strategies for dealing with overfitting