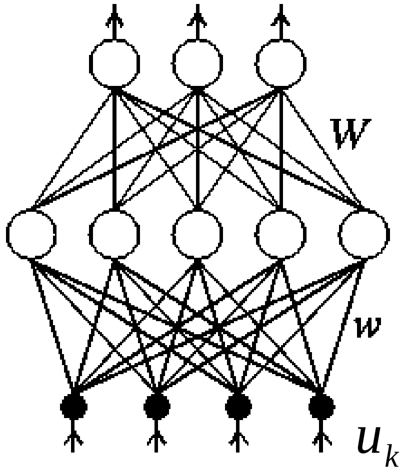
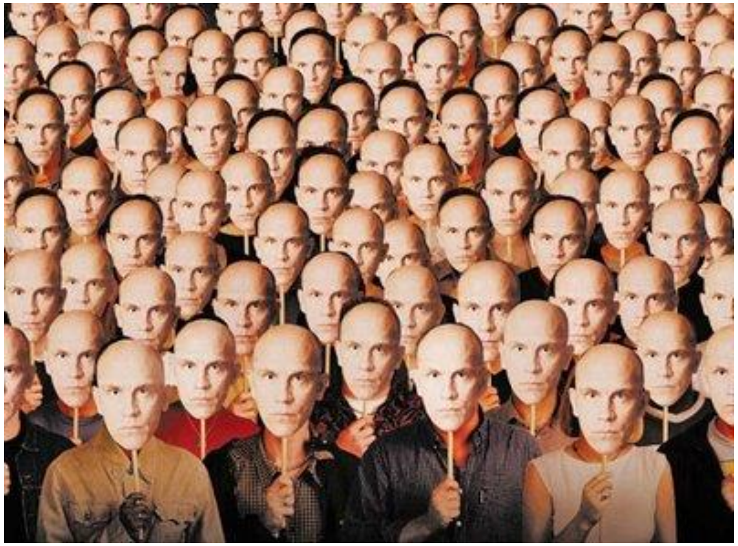
CSE 473

Lecture 28
(Chapter 18)

Neural Networks and Ensemble Learning





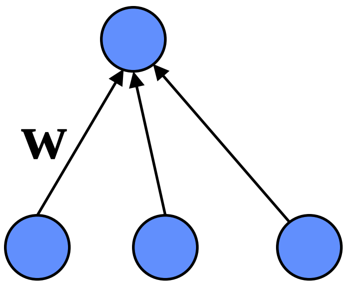
What if you want your neural network to predict *continuous* outputs rather than +1/-1 (i.e., perform regression)?

E.g., Teaching a network to drive

Image Source: Wikimedia Commons

Continuous Outputs with Sigmoid Networks

$$Output\ \ v = g(\mathbf{w}^T\mathbf{u}) = g(\sum_i w_i u_i)$$

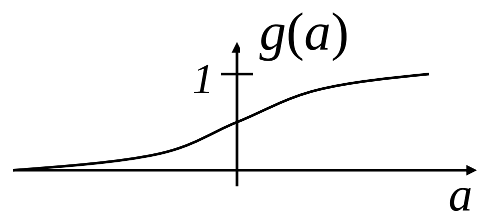


Input nodes

$\mathbf{u} = (u_1 \quad u_2 \quad u_3)^T$

Sigmoid output function:

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



Parameter β controls the slope

Learning the weights

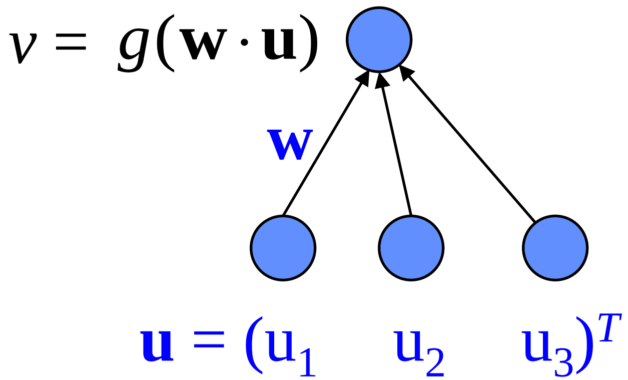
Given: Training data (input **u**, desired output **d**)

Problem: How do we learn the weights **w**?

Idea: *Minimize squared error* between network's output and desired output:

$$E(\mathbf{w}) = (d - v)^2$$

where $v = g(\mathbf{w} \cdot \mathbf{u})$



Starting from random values for **w**, want to change **w** so that *E*(**w**) is minimized – How?

Learning by Gradient-Descent
(opposite of "Hill-Climbing")

Change w so that *E*(w) is minimized

- Use Gradient Descent: Change w in proportion to -d*E*/dw (why?)

$$E(\mathbf{w}) = (d - v)^2 \qquad v = g(\mathbf{w} \cdot \mathbf{u})$$

$$\mathbf{w} \to \mathbf{w} - \varepsilon \frac{dE}{d\mathbf{w}}$$

Derivative of sigmoid

$$\frac{dE}{d\mathbf{w}} = -2(d - v)\frac{dv}{d\mathbf{w}} = -2\underbrace{(d - v)}_{\text{delta = error}}g'(\mathbf{w} \cdot \mathbf{u})\mathbf{u}$$

Also known as the "delta rule" or "LMS (least mean square) rule"

But wait!

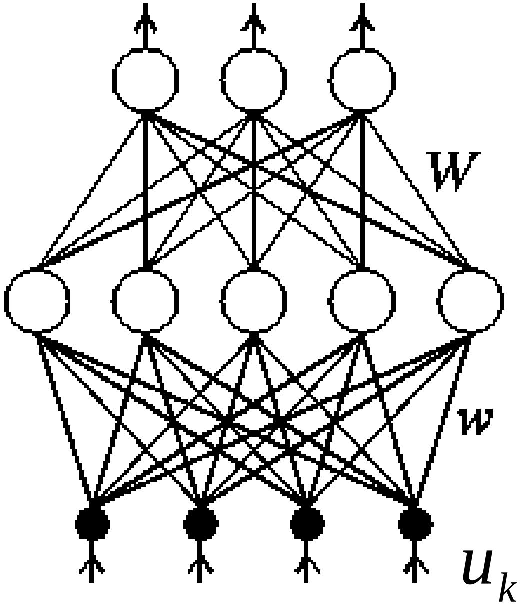
This rule is for a one layer network

- One layer networks are not that interesting!! (remember XOR?)

What if we have multiple layers?

Learning Multilayer Networks

$$v_i = g\left(\sum_j W_{ji}\, g\left(\sum_k w_{kj} u_k\right)\right)$$



Start with random weights **W**, **w**

Given input vector **u**, network produces output vector **v**

Use gradient descent to find **W** and **w** that minimize total error over all output units (labeled *i*):

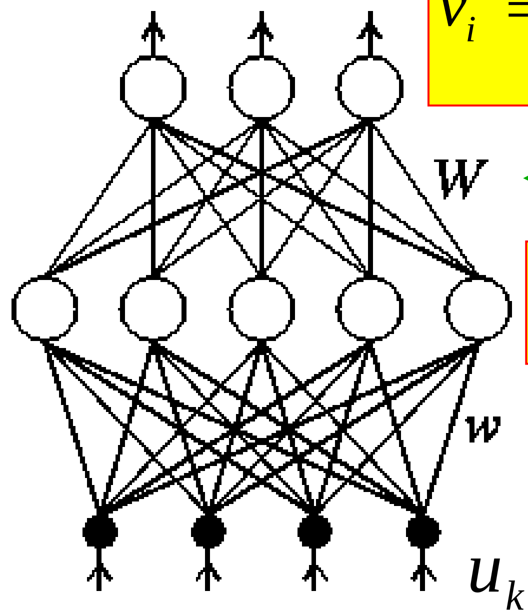
$$E(\mathbf{W},\mathbf{w}) = \frac{1}{2}\sum_i (d_i - v_i)^2$$

This leads to the famous "Backpropagation" learning rule

Backpropagation: Output Weights

$$E(\mathbf{W},\mathbf{w}) = \frac{1}{2}\sum_i (d_i - v_i)^2$$

$$\boxed{v_i = g(\sum_j W_{ji} x_j)}$$



$$\boxed{x_j}$$

Learning rule for <u>hidden-output weights W</u>:

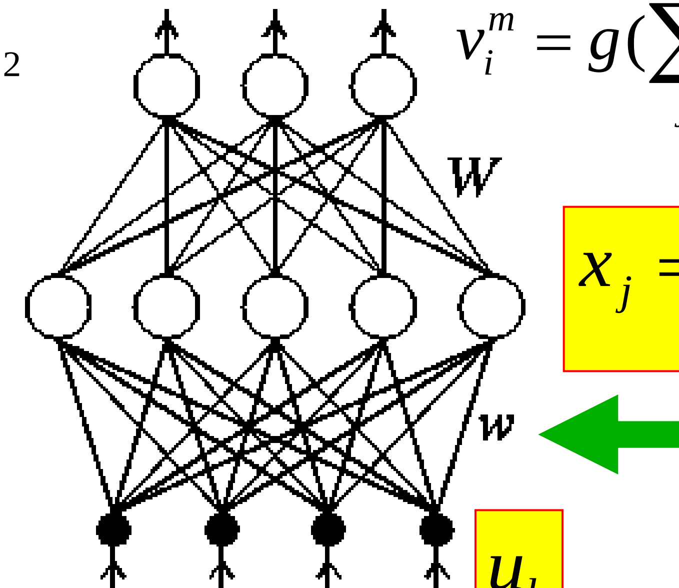
$$W_{ji} \rightarrow W_{ji} - \varepsilon \frac{dE}{dW_{ji}}$$ {gradient descent}

$$\frac{dE}{dW_{ji}} = -(d_i - v_i) g'(\sum_j W_{ji} x_j) x_j$$ {delta rule}

Backpropagation: Hidden Weights

$$E(\mathbf{W},\mathbf{w}) = \frac{1}{2}\sum_i (d_i - v_i)^2$$

$$v_i^m = g\left(\sum_j W_{ji} x_j\right)$$



$$\boxed{x_j = g\left(\sum_k w_{kj} u_k\right)}$$

$$\boxed{u_k}$$

Learning rule for <u>input-hidden weights w</u>:

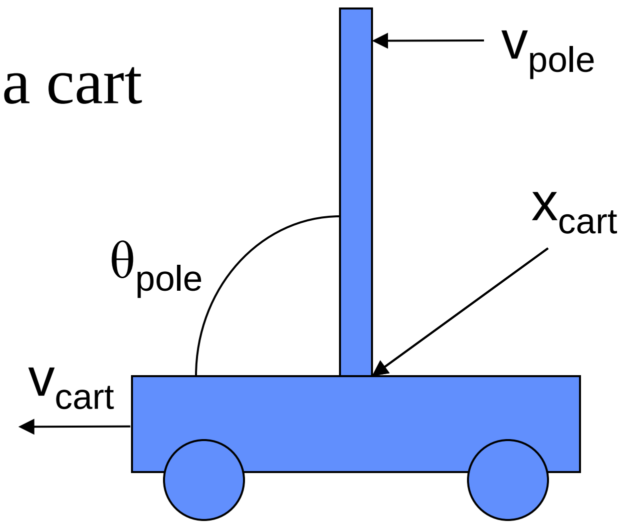
$$w_{kj} \rightarrow w_{kj} - \varepsilon \frac{dE}{dw_{kj}} \qquad \text{But}: \frac{dE}{dw_{kj}} = \frac{dE}{dx_j} \cdot \frac{dx_j}{dw_{kj}} \quad \{\text{chain rule}\}$$

$$\frac{dE}{dw_{kj}} = \left[-\sum_i (d_i - v_i) g'\left(\sum_j W_{ji} x_j\right) W_{ji} \right] \cdot \left[g'\left(\sum_k w_{kj} u_k\right) u_k \right]$$

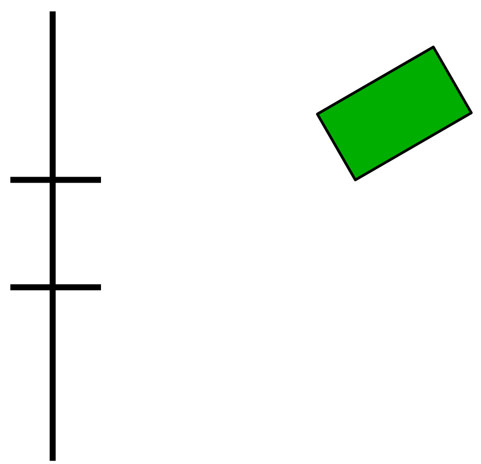
Examples: Pole Balancing and Backing up a Truck

(courtesy of Keith Grochow)

- Neural network learns to balance a pole on a cart
  - Input: x_{cart}, v_{cart}, θ_{pole}, v_{pole}
  - Output: New force on cart



- Network learns to back a truck into a loading dock
  - Input: x, y, θ of truck
  - Output: Steering angle



Ensemble Learning

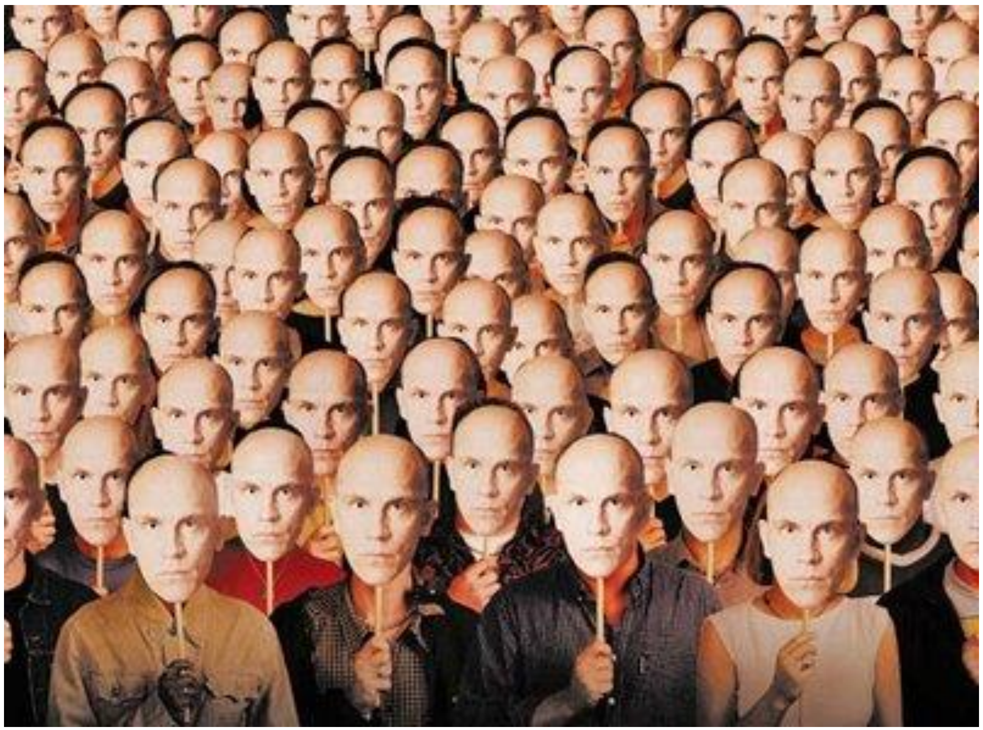
Sometimes each learning technique yields a different "hypothesis" (function)

But no perfect hypothesis…

Could we combine several imperfect hypotheses to get a better hypothesis?

Why Ensemble Learning?

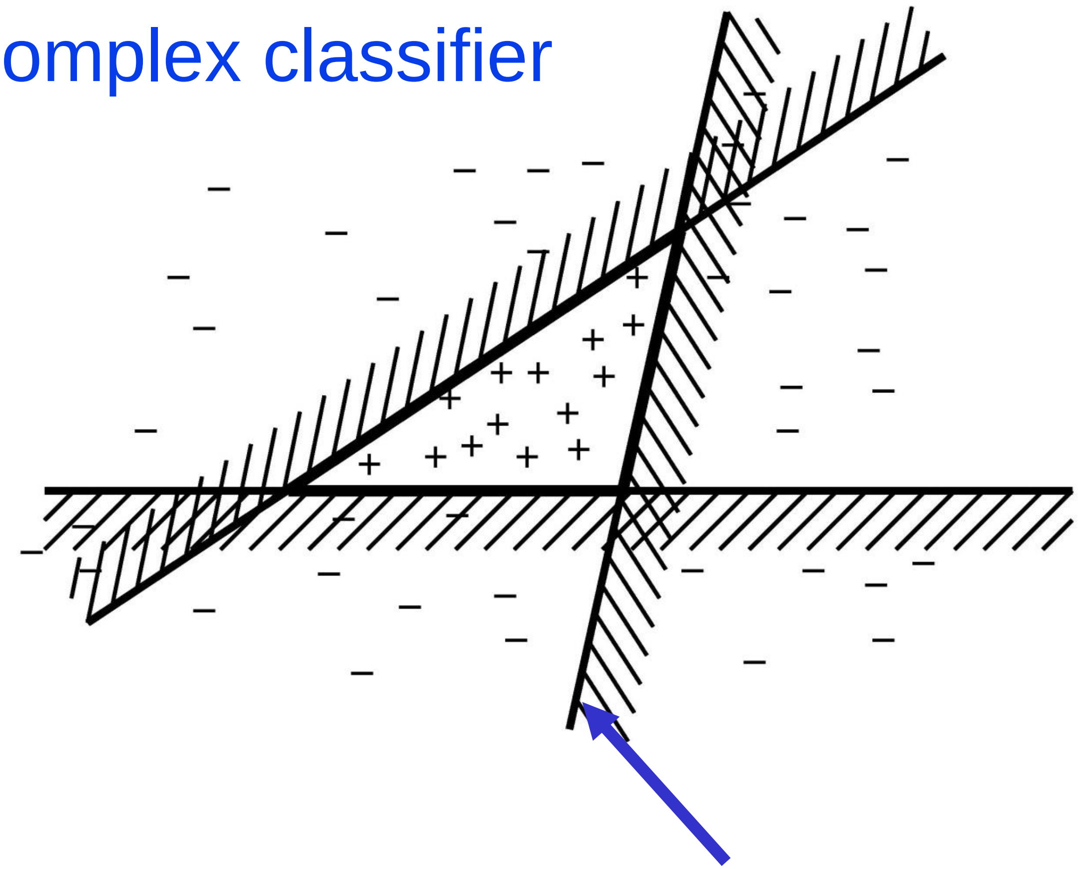
Wisdom of the Crowds…



Example

Combine 3 linear classifiers

$\Rightarrow$ More complex classifier



This line is one simple classifier saying that everything to the left is + and everything to the right is -

Ensemble Learning: Motivation

Analogies:

- Elections combine voters' choices to pick a good candidate (hopefully)
- Committees combine experts' opinions to make better decisions
- Students working together on a capstone project

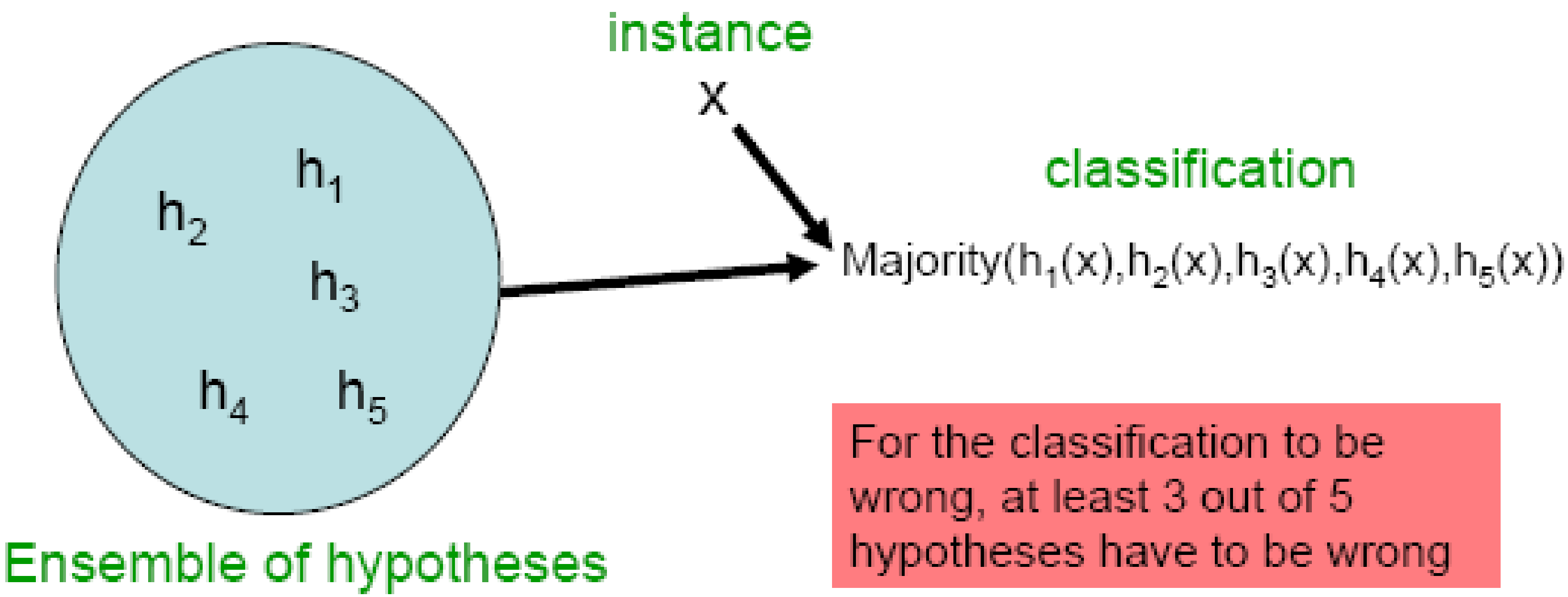
Intuitions:

Individuals make mistakes but the "majority" may be less likely to

Individuals often have partial knowledge; a committee can pool expertise to make better decisions

Ensemble Technique 1: Bagging

Combine hypotheses (classifiers) via majority voting



For the classification to be wrong, at least 3 out of 5 hypotheses have to be wrong

Bagging: Details

1. Generate *m* new training datasets by sampling with replacement from the given dataset

2. Train *m* classifiers $h_1, \ldots, h_m$ (e.g., decision trees), one from each newly generated dataset

3. Classify a new input by running it through the *m* classifiers and choosing the class that receives the most "votes"

Example: *Random forest* = Bagging with *m* decision tree classifiers, each tree constructed from random subset of attributes

Bagging: Analysis

- Assumptions:
  - Each h_i makes error with probability p
  - The hypotheses are independent

- Majority voting of n hypotheses:
  - k hypotheses make an error: $\binom{n}{k} p^k(1-p)^{n-k}$
  - Majority makes an error: $\Sigma_{k>n/2} \binom{n}{k} p^k(1-p)^{n-k}$
  - With n=5, p=0.1 → err(majority) < 0.01

Error probability went down from 0.1 to 0.01!

Weighted Majority Voting

In practice, hypotheses rarely independent

Some hypotheses have less errors than others $\Rightarrow$ all votes are not equal!

Idea: Let's take a weighted majority

How do we compute the weights?

Next Time

- Weighted Majority Ensemble Classification
    - Boosting
- Survey of AI Applications
- To Do:
    - Project 4 due tonight!
    - Finish Chapter 18