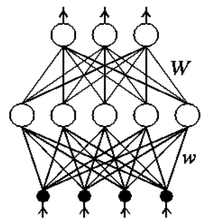
CSE 473

Chapters 20 & 21

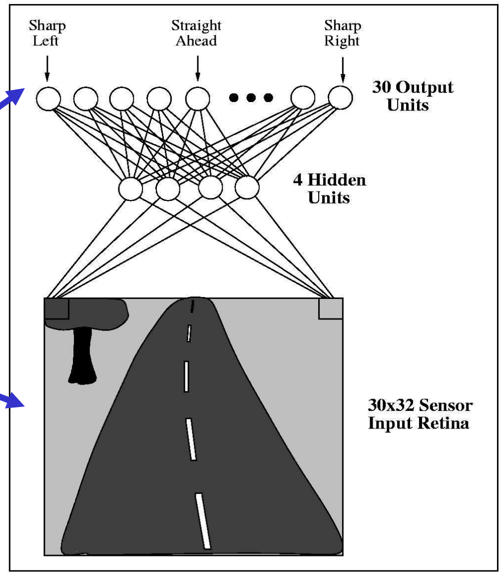
Machine Learning: Backpropagation and Reinforcement Learning



Learning to Drive

How would you use a neural network to drive?

Example Network

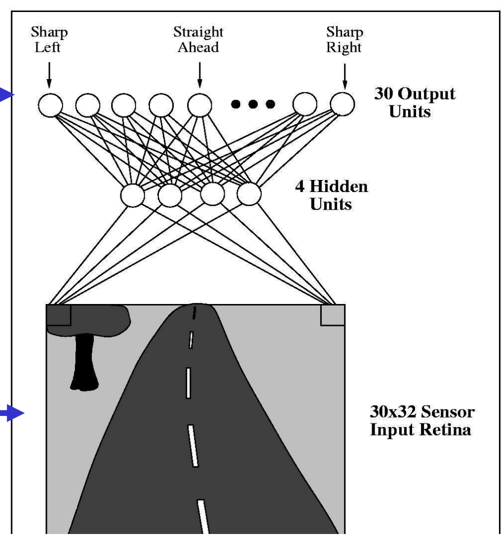


Where will the training inputs come from?

Example Network

Get steering angle from a human driver

Training Output:
$\mathbf{d} = (d_1\ d_2\ \dots\ d_{30})$

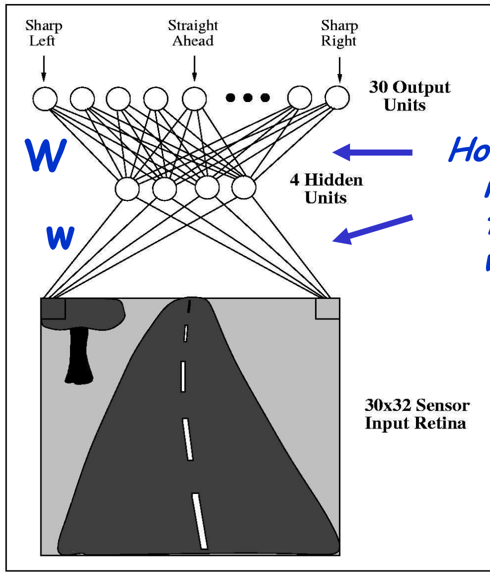


Get current camera image

Training Input $\mathbf{u} = (u_1\ u_2\ \dots\ u_{960})$ = image pixels

Example Network

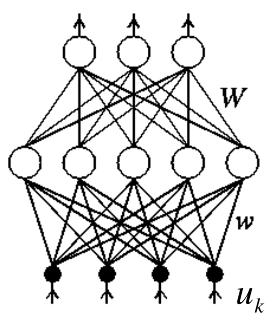
You are given training input-output pairs (**u**,**d**)



How do we modify these weights?

Idea

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



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

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

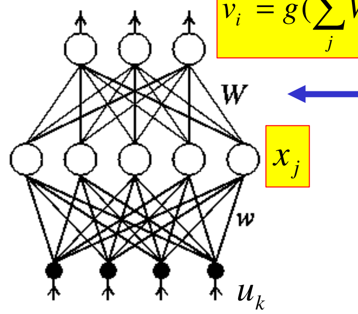
Find **W** and **w** that minimize total squared output error over all output units (labeled i):

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

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 hidden-output weights **W**:

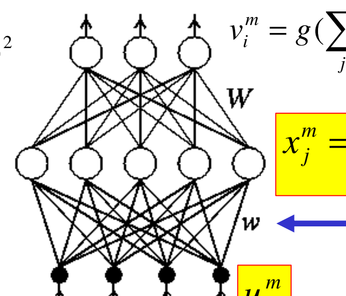
$$W_{ji} \rightarrow W_{ji} - \varepsilon \frac{dE}{dW_{ji}} \qquad \{\textit{gradient descent}\}$$

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

Backpropagation: Hidden Weights

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

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



$$\boxed{x_j^m = g(\sum_k w_{kj} u_k^m)}$$

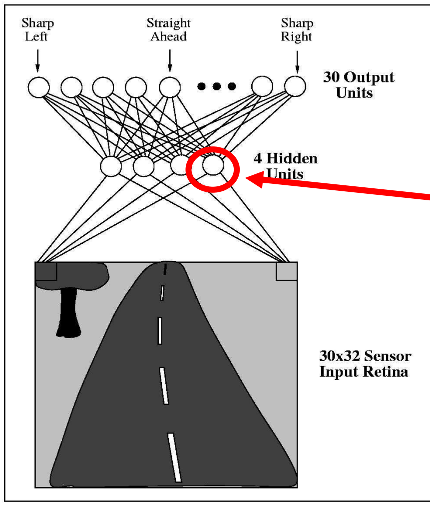
$$\boxed{u_k^m}$$

Learning rule for input-hidden weights **w**:

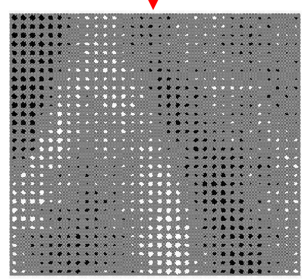
$$w_{kj} \rightarrow w_{kj} - \varepsilon \frac{dE}{dw_{kj}} \qquad \text{But}: \frac{dE}{dw_{kj}} = \frac{dE}{dx_j} \cdot \frac{dx_j}{dw_{kj}} \qquad \{\textit{chain rule}\}$$

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

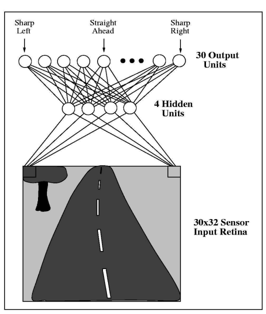
Learning to Drive using Backprop



One of the learned "road features" w_i



ALVINN (Autonomous Land Vehicle in a Neural Network)



CMU Navlab

Trained using human driver + camera images

After learning:

- Drove up to 70 mph on highway
- Up to 22 miles without intervention
- Drove cross-country largely autonomously

(Pomerleau, 1992)

Other Demos

- Function Approximation:
  http://neuron.eng.wayne.edu/bpFunctionApprox/bpFunctionApprox.html
- Pattern Recognition
  http://www-cse.uta.edu/%7Ecook/ai1/lectures/applets/hnn/JRec.html
- Image Compression
  http://neuron.eng.wayne.edu/bpImageCompression9PLUS/bp9PLUS.html
- Backpropagation for Control: Ball Balancing
  http://neuron.eng.wayne.edu/bpBallBalancing/ball5.html

Demos: Pole Balancing and Backing up a Truck

(by Keith Grochow, CSE 599, 2001)

Neural network learns to balance a pole on a cart

System:

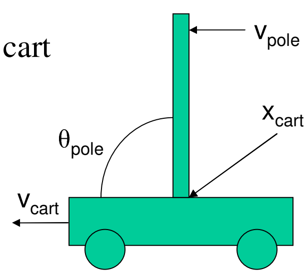
4 state variables: x_{cart}, v_{cart}, θ_{pole}, v_{pole}

1 input: Force on cart

Backprop Network:

Input: State variables

Output: New force on cart



NN learns to back a truck into a loading dock

System (Nyugen and Widrow, 1989):

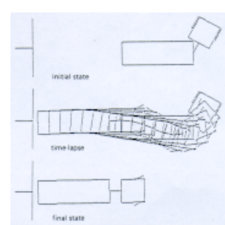
State variables: x_{cab}, y_{cab}, θ_{cab}

1 input: new $\theta_{steering}$

Backprop Network:

Input: State variables

Output: Steering angle $\theta_{steering}$

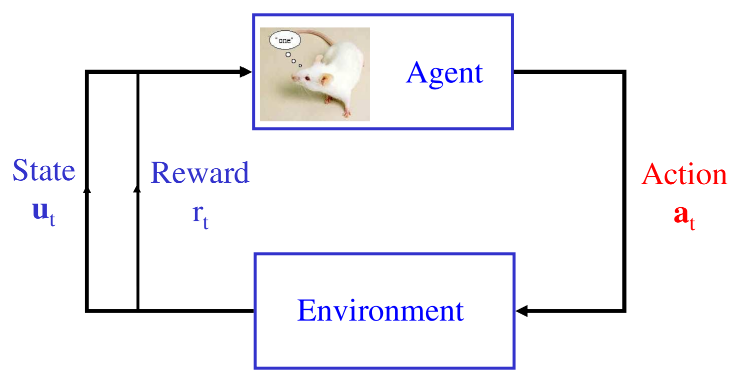


Human drivers don't usually get exact supervisory signals (commands for muscles) for learning to drive!

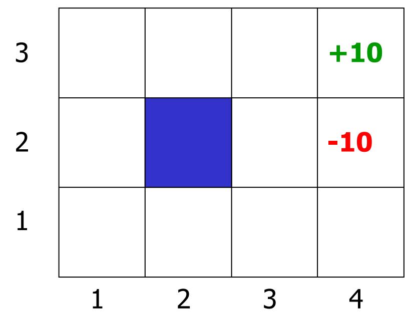
Must learn by trial-and-error
Might get "rewards and punishments" along the way

Enter…Reinforcement Learning

The Reinforcement Learning "Agent"



Example: Rat in a Maze



Food (cheese)

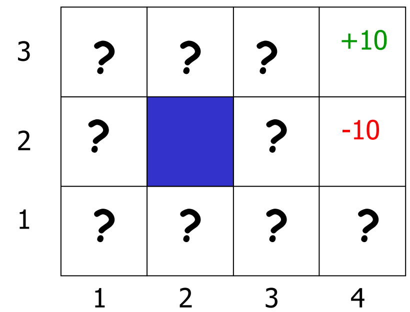
Trap

States = Maze locations (1,1), (1,2),…
Actions = Move forward, left, right, back
Rewards = +10 at (3,4), -10 at (2,4)
-1 at others (cost of moving)

Actions might be noisy

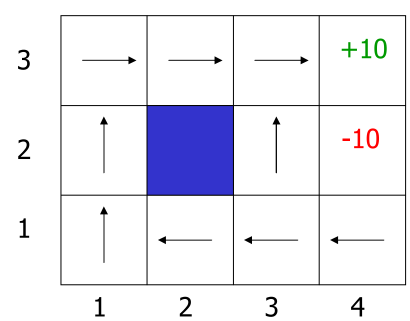
- Executing "Forward" action may not succeed
  E.g. 0.9 probability of moving forward, 0.1 probability of remaining in current location
- Characterized by transition probabilities:
  P(next state | current state, action)

Goal: Learn a "Policy"



Policy = for each state, what is the best action that maximizes my expected reward?

Goal: Learn a "Policy"



The Optimal Policy

Reinforcement Learning

- How can an agent learn behaviors when it doesn't have a teacher to tell it how to perform?
  - The agent has a task to perform
  - It takes some actions in the world
  - At some later point, it gets feedback telling it how well it did on performing the task
  - The agent performs the same task over and over again
- This problem is called **reinforcement learning:**
  - The agent gets *positive reinforcement* for tasks done well
  - The agent gets *negative reinforcement* for tasks done poorly

Reinforcement Learning (cont.)

- The goal is to get the agent to act in the world so as to maximize its rewards
- The agent has to figure out what it did that made it get the reward/punishment
  - This is known as the **credit assignment** problem
- Reinforcement learning approaches can be used to train computers to do many tasks
  - backgammon and chess playing
  - job shop scheduling
  - controlling robot limbs

Next Time

- How to learn based on rewards
  - Predicting delayed rewards (TD learning)
  - Learning policies
  - Q-learning