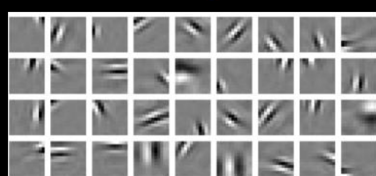
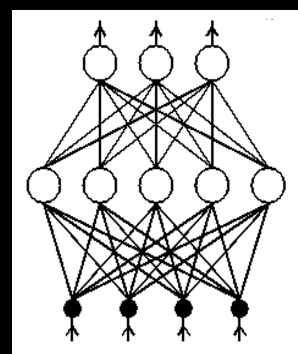
CSE/NB 528

Lecture 13: From Unsupervised to Reinforcement Learning

(Chapters 8-10)





Today's Agenda: All about Learning

- Unsupervised Learning
  - Sparse Coding
  - Predictive Coding
- Supervised learning
  - Perceptrons and Backpropagation
- Reinforcement Learning
  - TD and Actor-Critic learning

Recall from Last Time: Linear Generative Model

Causes **v**

Generative model

Data **u**

Suppose input **u** was generated by a linear superposition of causes v_1, v_2, …, v_k with basis vectors (or "features") $\mathbf{g}_i$

$$\mathbf{u} = \sum_i \mathbf{g}_i v_i + noise = G\mathbf{v} + n$$

(Assume noise is Gaussian white noise with mean zero)

Bayesian approach

- Find **v** and G that maximize posterior:

$$p[\mathbf{v} \mid \mathbf{u}; G] = k \cdot p[\mathbf{u} \mid \mathbf{v}; G] p[\mathbf{v}; G]$$

- Equivalently, find **v** and G that maximize log posterior:

$$F(\mathbf{v}, G) = \log p[\mathbf{u} \mid \mathbf{v}; G] + \log p[\mathbf{v}; G] + \log k$$

$\mathbf{u} = G\mathbf{v} + n$

log of Gaussian

$$\log N(\mathbf{u}; G\mathbf{v}, I)$$
$$= -\frac{1}{2}(\mathbf{u} - G\mathbf{v})^T (\mathbf{u} - G\mathbf{v}) + C$$

If v_a independent

$$p[\mathbf{v}; G] = \prod_a p[v_a; G]$$
$$\log p[\mathbf{v}; G] = \sum_a \log p[v_a; G]$$

Prior for individual causes (what should this be?)

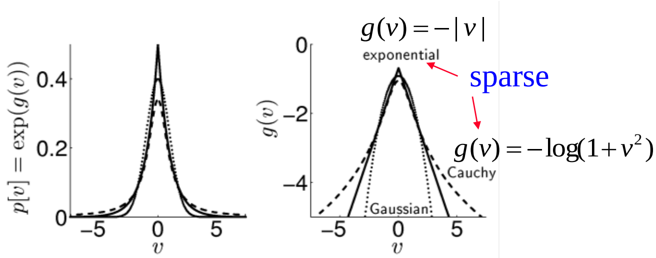
What do we know about the causes **v**?

- Idea: Causes independent: only a few of them will be active for any input
  - v_a will be 0 most of the time but high for a few inputs
  - Suggests a sparse distribution for $p[v_a;G]$: peak at 0 but with heavy tail (also called *super-Gaussian* distribution)

Examples of Prior Distributions for Causes

Possible prior distributions

Log prior



$$p[\mathbf{v};G] = c \cdot \prod_a \exp(g(v_a))$$

$$\log p[\mathbf{v};G] = \sum_a g(v_a) + c$$

Finding the optimal **v** and *G*

- Want to maximize:

$$F(\mathbf{v}, G) = \log p[\mathbf{u} \mid \mathbf{v}; G] + \log p[\mathbf{v}; G] + \log k$$
$$= -\frac{1}{2}(\mathbf{u} - G\mathbf{v})^T(\mathbf{u} - G\mathbf{v}) + \sum_a g(v_a) + K$$

- Alternate between two steps:
  - Maximize *F* with respect to **v** keeping G fixed
    - How?
  - Maximize *F* with respect to G, given the **v** above
    - How?

Estimating the causes **v** for a given input

Gradient ascent

Derivative of g

$$\frac{d\mathbf{v}}{dt} \propto \frac{dF}{d\mathbf{v}} = G^T(\mathbf{u} - G\mathbf{v}) + g'(\mathbf{v})$$

Reconstruction (prediction) of **u**

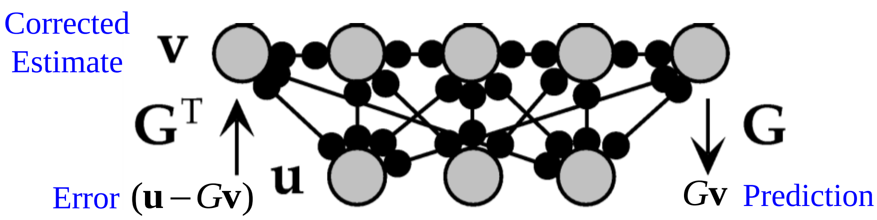
$$\tau \frac{d\mathbf{v}}{dt} = G^T(\mathbf{u} - G\mathbf{v}) + g'(\mathbf{v})$$

Firing rate dynamics (Recurrent network)

Error Sparseness constraint

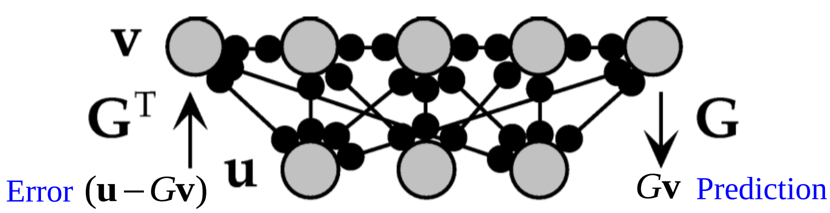
Sparse Coding Network for Estimating $\mathbf{v}$

$$\tau \frac{d\mathbf{v}}{dt} = G^T(\mathbf{u} - G\mathbf{v}) + g'(\mathbf{v})$$



[Suggests a role for feedback pathways in the cortex (Rao & Ballard, 1999)]

Learning the Synaptic Weights G

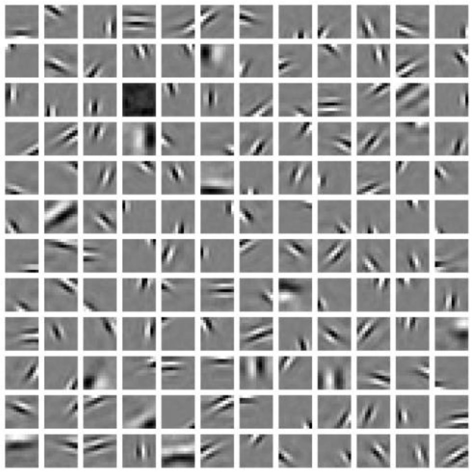


Gradient ascent $\quad \frac{dG}{dt} \propto \frac{dF}{dG} = (\mathbf{u} - G\mathbf{v})\mathbf{v}^T$

Learning rule $\quad \tau_G \frac{dG}{dt} = (\mathbf{u} - G\mathbf{v})\mathbf{v}^T$

Hebbian!
(similar to Oja's rule)

Result: Learning G for Natural Images



(Olshausen & Field, 1996)

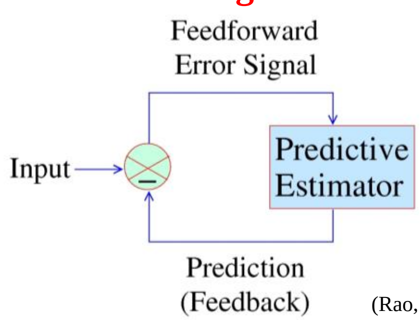
Each square is a column $\mathbf{g}_i$ of G (obtained by collapsing rows of the square into a vector)

Almost all the $\mathbf{g}_i$ represent local edge features

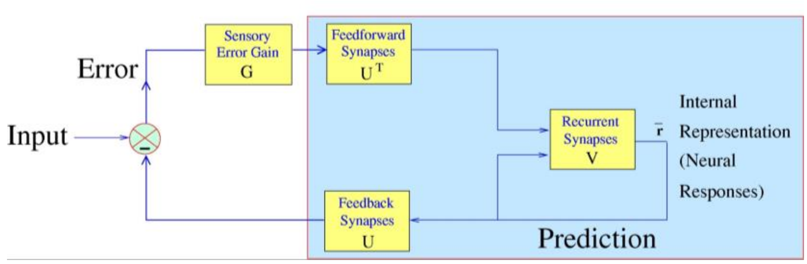
Any image patch $\mathbf{u}$ can be expressed as:

$$\mathbf{u} = \sum_i \mathbf{g}_i v_i = G\mathbf{v}$$

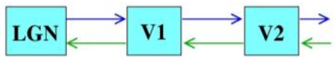
Sparse Coding Network is a special case of Predictive Coding Networks

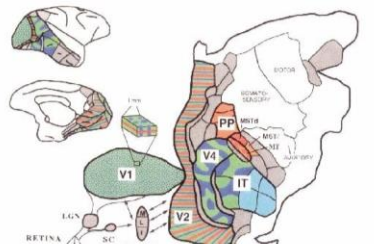


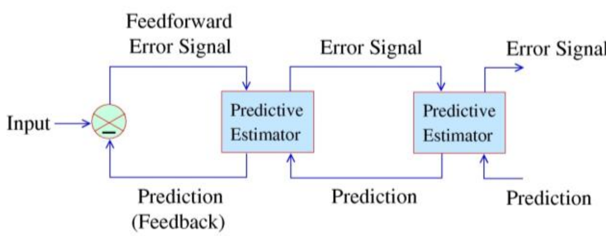
(Rao, *Vision Research*, 1999)



Predictive Coding Model of Visual Cortex





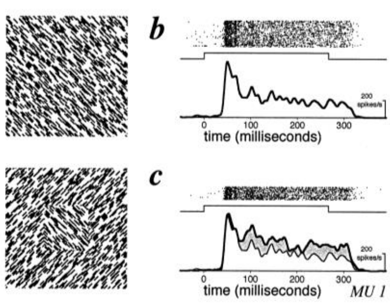


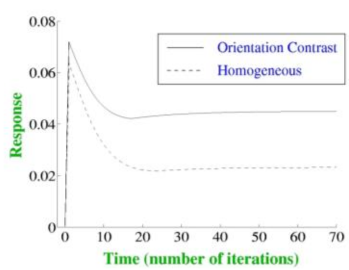
(Rao & Ballard, *Nature Neurosci*., 1999)

Predictive coding model explains contextual effects

Monkey Primary Visual Cortex

Model

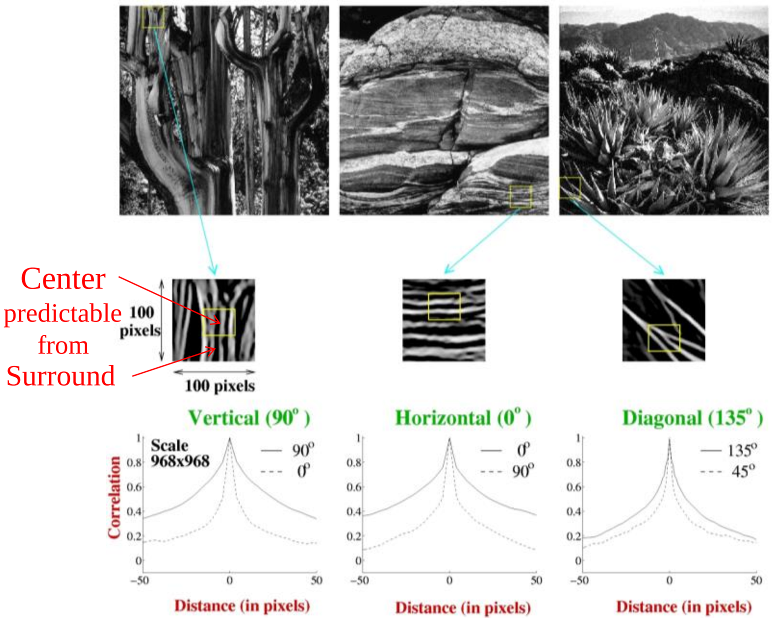




(Zipser et al., *J. Neurosci*., 1996)

Increased activity for non-homogenous input interpreted as prediction error (i.e., anomalous input): center is not predicted by surrounding context.

Natural Images as a Source of Contextual Effects



(Rao & Ballard, *Nature Neurosci.*, 1999)

What if your data comes with not just inputs but also outputs?

Enter…Supervised Learning

Supervised Learning

- Two Primary Tasks

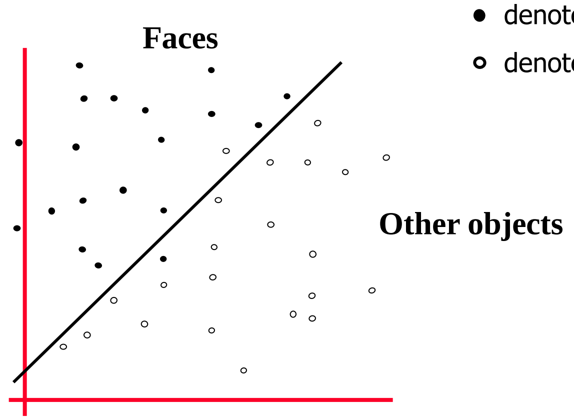
1. Classification

- Inputs u_1, u_2, … and discrete classes C_1, C_2, …, C_k
- Training examples: (u_1, C_2), (u_2, C_7), etc.
- Learn the mapping from an arbitrary input to its class
- Example: Inputs = images, output classes = face, not a face

2. Regression

- Inputs u_1, u_2, … and continuous outputs v_1, v_2, …
- Training examples: (input, desired output) pairs
- Learn to map an arbitrary input to its corresponding output
- Example: Highway driving
  Input = road image, output = steering angle

The Classification Problem



- ● denotes output of +1 (faces)
- ○ denotes output of -1 (other)

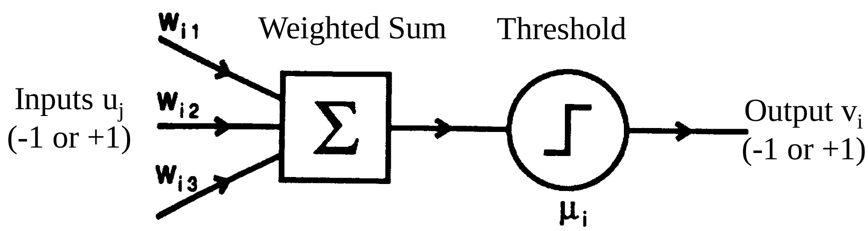
Idea: Find a separating hyperplane (line in this case)

Neurons as Classifiers: The "Perceptron"

- Artificial neuron:
  - m binary inputs (-1 or 1) and 1 output (-1 or 1)
  - Synaptic weights w_{ij}
  - Threshold μ_i

$$v_i = \Theta\left(\sum_j w_{ij} u_j - \mu_i\right)$$

$\Theta(x)$ = +1 if $x \geq 0$ and -1 if $x < 0$



What does a Perceptron compute?

- Consider a single-layer perceptron
  - Weighted sum forms a *linear hyperplane (line, plane, …)*

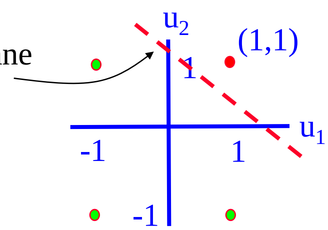
$$\sum_j w_{ij} u_j - \mu_i = 0$$

  - Everything *on one side* of hyperplane is in class 1 (output = +1) and everything *on other side* is class 2 (output = -1)
  - Any function that is linearly separable can be computed by a perceptron

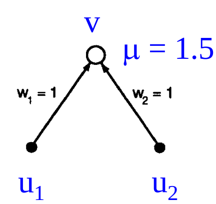
Linear Separability

- Example: AND function is linearly separable
  - a AND b = 1 if and only if a = 1 and b = 1

Linear hyperplane



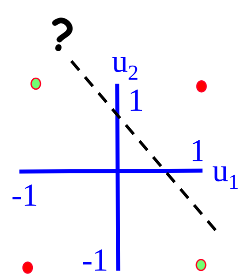
● +1 output
● -1 output



Perceptron for AND

What about the XOR function?

u_1	u_2	XOR
-1	-1	+1
1	-1	-1
-1	1	-1
1	1	+1

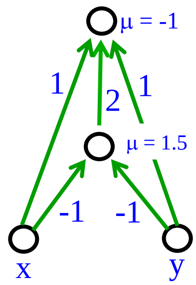


● +1 output
● -1 output

Can a straight line separate the +1 outputs from the -1 outputs?

Multilayer Perceptrons

- Removes limitations of single-layer networks
  - Can solve XOR
- An example of a two-layer perceptron that computes XOR



- Output is +1 if and only if $x + y + 2\Theta(-x - y - 1.5) > -1$

(Inputs x and y can be +1 or -1)

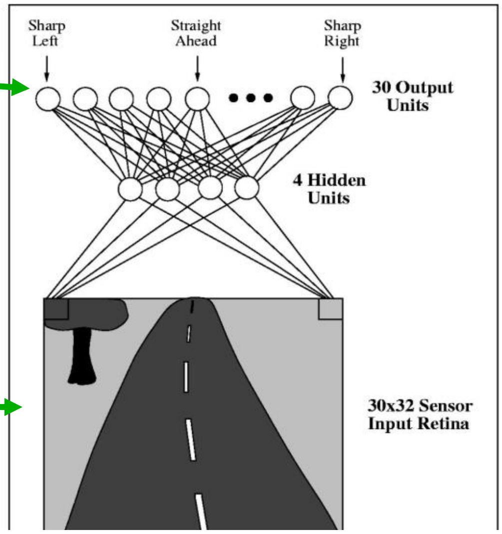
What if you want to approximate a *continuous* function (i.e., regression)?

Can a network learn to drive?

Example Network

Steering angle

Desired Output:
$\mathbf{d} = [d_1\ d_2\ \ldots\ d_{30}]$

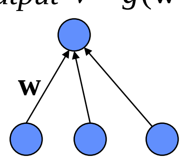


Current image

Input $\mathbf{u} = [u_1\ u_2\ \ldots\ u_{960}]$ = image pixels

Sigmoid Networks

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

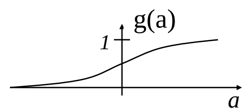


Input nodes

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

Sigmoid output function:

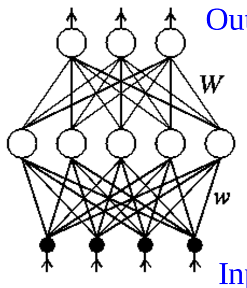
$$g(a) = \frac{1}{1 + e^{-\beta a}}$$



Sigmoid is a non-linear "squashing" function: Squashes input to be between 0 and 1. Parameter β controls the slope.

Multilayer Sigmoid Networks

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



Output $\mathbf{v} = (v_1\ v_2\ \ldots\ v_J)^T$; Desired = $\mathbf{d}$

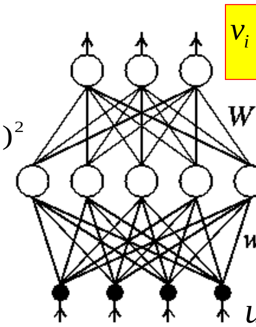
How do we learn these weights?

Input $\mathbf{u} = (u_1\ u_2\ \ldots\ u_K)^T$

Backpropagation Learning: Uppermost layer

Minimize output error:

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



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

$\boxed{x_j}$

u_k

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

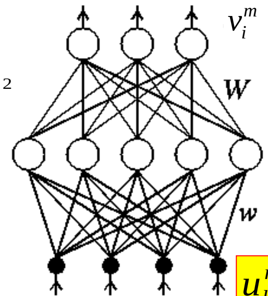
$$W_{ji} \rightarrow W_{ji} - \varepsilon \frac{dE}{dW_{ji}}$$ **{*gradient descent*}**

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

Backpropagation: Inner layer (chain rule)

Minimize output error:

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



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

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

$$\boxed{u_k^m}$$

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

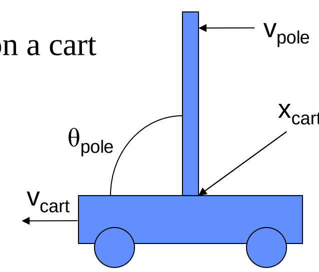
$$w_{kj} \rightarrow w_{kj} - \varepsilon \frac{dE}{dw_{kj}} \qquad \text{But}: \frac{dE}{dw_{kj}} = \frac{dE}{dx_j} \cdot \frac{dx_j}{dw_{kj}} \qquad \{\textit{chain rule}\}$$

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

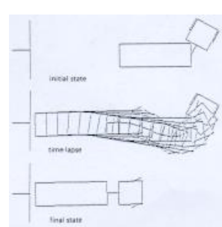
Demos: Pole Balancing and Backing up a Truck

(courtesy of Keith Grochow, CSE 599)

- 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 $\theta_{steering}$



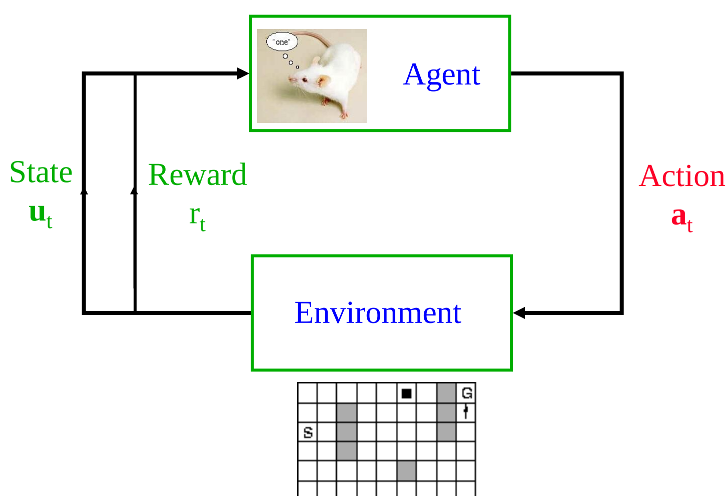
Humans (and animals in general) don't get exact supervisory signals (commands for muscles) for learning to talk, walk, ride a bicycle, play the piano, drive, etc.

We learn by trial-and-error
(with hints from others)

Might get "rewards and punishments" along the way

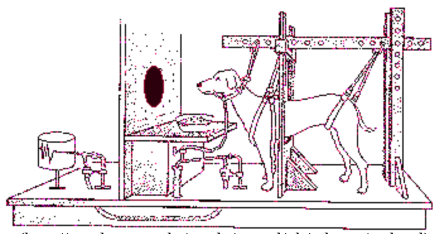
Enter…Reinforcement Learning

The Reinforcement Learning "Agent"



Early Results: Pavlov and his Dog

- Classical (Pavlovian) conditioning experiments
- Training: Bell → Food
- After: Bell → Salivate
- Conditioned stimulus (bell) predicts future reward (food)



(http://employees.csbsju.edu/tcreed/pb/pdoganim.html)

Predicting Delayed Rewards

- Reward is typically delivered at the end (when you know whether you succeeded or not)
- Time: $0 \le t \le T$ with stimulus $u(t)$ and reward $r(t)$ at each time step t (Note: $r(t)$ can be zero at some time points)
- Key Idea: Make the output $v(t)$ predict *total expected future reward* starting from time t

$$v(t) \approx \left\langle \sum_{\tau=0}^{T-t} r(t+\tau) \right\rangle$$

Learning to Predict Delayed Rewards

- Use a set of modifiable weights $w(t)$ and *predict based on all past stimuli* $u(t)$:

$$v(t) = \sum_{\tau=0}^{t} w(\tau)u(t-\tau)$$

- Would like to find the weights (or filter) $w(\tau)$ that minimize:

$$\left(\sum_{\tau=0}^{T-t} r(t+\tau) - v(t)\right)^2$$

(Can we minimize this using gradient descent and delta rule?)

Yes, BUT…not yet available are the future rewards

Temporal Difference (TD) Learning

- **Key Idea**: Rewrite squared error to get rid of future terms:

$$\left(\sum_{\tau=0}^{T-t} r(t+\tau) - v(t)\right)^2 = \left(r(t) + \sum_{\tau=0}^{T-t-1} r(t+1+\tau) - v(t)\right)^2$$

$$\approx \left(r(t) + v(t+1) - v(t)\right)^2$$

Minimize this using gradient descent!

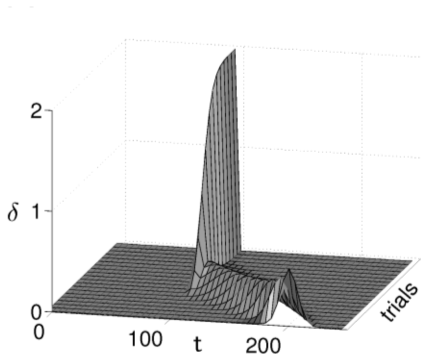
- **Temporal Difference (TD) Learning**:

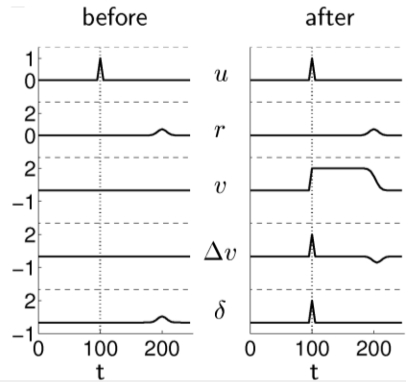
$$w(\tau) \rightarrow w(\tau) + \varepsilon\,[\overbrace{\underbrace{r(t) + v(t+1)}_{\text{Expected future reward}} - v(t)}^{\delta}]\,u(t-\tau)$$

Prediction

Predicting Delayed Reward: TD Learning

Stimulus at t = 100 and reward at t = 200



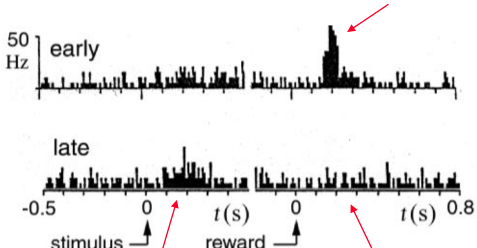


Prediction error δ for each time step
(over many trials)

Possible Reward Prediction Error Signal in the Primate Brain

Dopaminergic cells in Ventral Tegmental Area (VTA)

Reward Prediction error? $[r(t)+v(t+1)-v(t)]$



Before Training

After Training

$[0+v(t+1)-v(t)]$

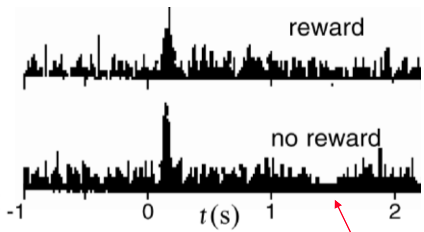
No error

$v(t) \approx r(t)+v(t+1)$

$[r(t)+v(t+1)-v(t)] \approx 0$

More Evidence for Prediction Error Signals

Dopaminergic cells in VTA



Negative error

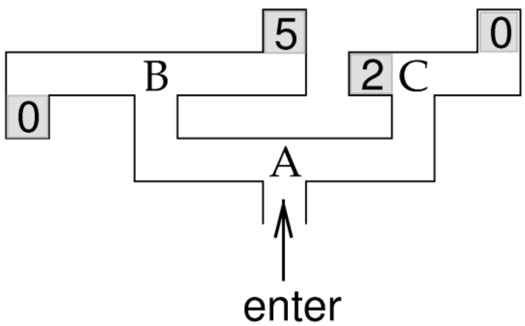
$$r(t) = 0, v(t+1) = 0$$

Reward predicted but not delivered

$$[r(t) + v(t+1) - v(t)] = -v(t)$$

That’s great, but how does all that math help me get food in a maze?

Selecting Actions when Reward is Delayed

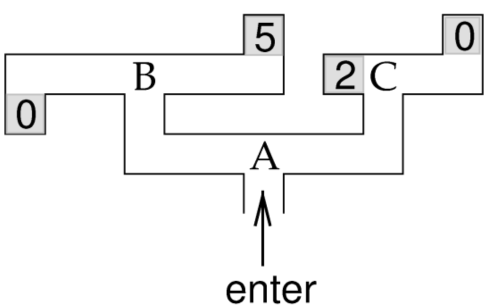


States: A, B, or C

Possible actions at any state: Left (L) or Right (R)

If you randomly choose to go L or R (random "policy"), what is the expected value v of each state?

Policy Evaluation



For random policy:

$$v(B) = \frac{1}{2}\cdot 0 + \frac{1}{2}\cdot 5 = 2.5$$

$$v(C) = \frac{1}{2}\cdot 2 + \frac{1}{2}\cdot 0 = 1$$

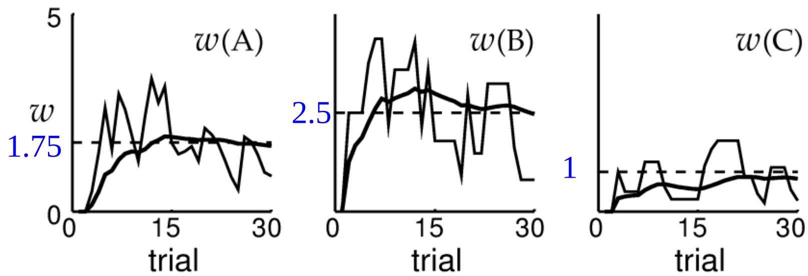
$$v(A) = \frac{1}{2}\cdot v(B) + \frac{1}{2}\cdot v(C) = 1.75$$

(Location, action) ⇨ new location
(u,a) ⇨ u'
Let value of location $v(u)$ = weight $w(u)$

Can learn value of locations using TD learning:

$$w(u) \to w(u) + \varepsilon\,[r_a(u) + v(u') - v(u)]$$

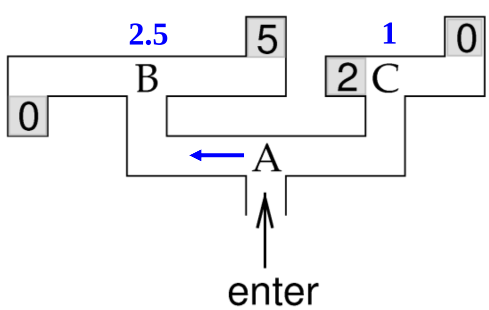
Maze Value Learning for Random Policy



(For all three, $\varepsilon = 0.5$)

Once I know the values, I can pick the action that leads to the higher valued state!

Selecting Actions based on Values



Values act as surrogate immediate rewards → Locally optimal choice leads to globally optimal policy (for "Markov" environments)

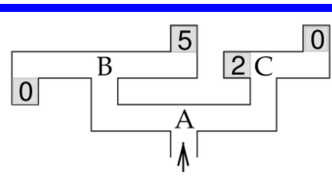
Related to *Dynamic Programming* in CS (see appendix in text)

Actor-Critic Learning

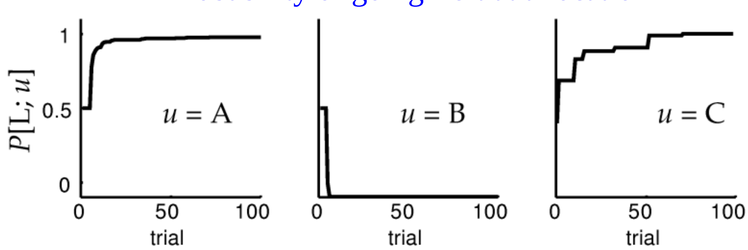
- Two separate components: Actor (maintains policy) and Critic (maintains value of each state)

1. Critic Learning ("Policy Evaluation"):
   Value of state $u = v(u) = w(u)$
   $$w(u) \to w(u) + \varepsilon\,[r_a(u) + v(u') - v(u)]$$ (same as TD rule)
2. Actor Learning ("Policy Improvement"):
   $$P(a;u) = \frac{\exp(\beta Q_a(u))}{\sum_b \exp(\beta Q_b(u))}$$
   Use this to select an action a at state u
   For all a':
   $$Q_{a'}(u) \to Q_{a'}(u) + \varepsilon[r_a(u) + v(u') - v(u)](\delta_{aa'} - P(a';u))$$
3. Interleave 1 and 2

Actor-Critic Learning in the Maze Task

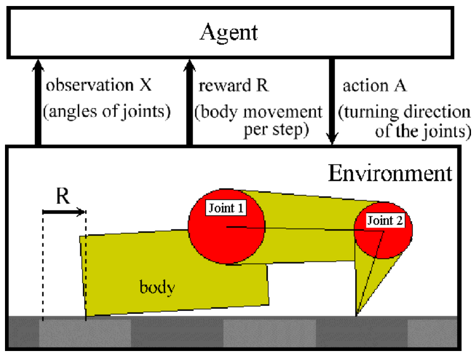


Probability of going Left at a location



Demo of Reinforcement Learning in a Robot

(from http://sysplan.nams.kyushu-u.ac.jp/gen/papers/JavaDemoML97/robodemo.html)



Things to do:

Finish homework 3
Work on group project