What’s on the menu today?

- Unsupervised Learning
  - Sparse Coding and ICA

- Supervised Learning
  - Why supervised learning?
    - Classification
    - Function Approximation
  - Perceptrons & Learning Rule
  - Linear Separability: Minsky-Papert deliver the bad news
  - Multilayer networks to the rescue
  - Function Approximation
  - Backpropagating (errors)
Unsupervised Learning: Sparse Coding and ICA

✦ Suppose input \( u \) is represented by linear superposition of causes \( v_1, v_2, \ldots, v_k \) and “features” \( g_i \):

\[
    u = \sum_i g_i v_i = G v
\]

✦ Problem: For a set of inputs \( u \), estimate causes \( v_i \) for each \( u \) and learn feature vectors \( g_i \) (also called basis vectors/filters)

✦ Idea: Find \( v \) and \( G \) that minimize reconstruction errors:

\[
    E = \frac{1}{2} \left\| u - \sum_i g_i v_i \right\|^2 = \frac{1}{2} (u - Gv)^T (u - Gv)
\]

Probabilistic Interpretation

✦ \( E \) is the same as the negative log likelihood of data

\( \Rightarrow \) Likelihood = Gaussian with mean \( Gv \) and covariance \( I \)

\[
    p[u \mid v; G] = N(u; Gv, I)
\]

\[
    E = -\ln p[u \mid v; G] = \frac{1}{2} (u - Gv)^T (u - Gv) + C
\]

✦ Find \( v \) and \( G \) that maximize:

\[
    F(v, G) = \langle \ln p[v, u; G] \rangle \quad \text{Joint probability of } v \text{ and } u
\]

\[
    = \langle \ln p[u \mid v; G] \rangle + \ln p[v; G] \quad \text{Prior for causes (what should this be?)}
\]
What do we know about the causes \( \mathbf{v} \)?

- We would like the causes to be \textit{independent}.
- If cause A and cause B always occur together, then perhaps they should be treated as a single cause AB?
- Examples:
  - \textbf{Image}: Composed of several independent edges
  - \textbf{Sound}: Composed of independent spectral components
  - \textbf{Objects}: Composed of several independent parts
- Idea 1: We would like: 
  \[
p[\mathbf{v}; G] = \prod_a p[\nu_a; G]
  \]
- Idea 2: If causes are independent, only a few of them will be active for any input \( \Rightarrow \nu_a \) will be 0 most of the time but high for certain inputs \( \Rightarrow \) sparse distribution for \( p[\nu_a; G] \)

Prior Distributions for Causes

Possible prior distributions

\[
p[\nu] = \exp(g(\nu))
\]

Log prior

\[
g(\nu) = -|\nu|
\]

\( g(\nu) = -\ln(1+\nu^2) \)

\[
p[\mathbf{v}; G] \propto \prod_a \exp(g(\nu_a))
\]
Finding the optimal $v$ and $G$

- Want to maximize:
  
  $$F(v, G) = \langle \ln p[u \mid v; G] + \ln p[v; G] \rangle$$
  
  $$= \left\langle -\frac{1}{2} (u - Gv)^T (u - Gv) + \sum_a g(v_a) \right\rangle + K$$

- Alternate between:
  1. Maximize $F$ with respect to $v$ keeping $G$ fixed
     - Set $dv/dt \propto dF/dv$ ("gradient ascent/hill-climbing")
  2. Maximize $F$ with respect to $G$, given the $v$ above
     - Set $dG/dt \propto dF/dG$ ("gradient ascent/hill-climbing")

Network for Estimating $v$ and Learning $G$

\[ \tau \frac{dv}{dt} = \frac{dF}{dv} = G^T (u - Gv) + g'(v) \]

Firing rate dynamics

\[ \tau \frac{dG}{dt} = \frac{dF}{dG} = (u - Gv)v^T \]

Learning rule

\[ \tau_G \frac{dG}{dt} = \frac{dF}{dG} = (u - Gv)v^T \]

Hebbian! (similar to Oja’s rule)
Results of Learning $G$ for Natural Images

Each square is a column $g_i$ of $G$ (obtained by collapsing rows of the square into a vector)

Almost all the $g_i$ represent local edge features

Any image $u$ can be expressed as:

$$u = \sum_i g_i v_i = Gv$$

What if there is a “teacher” telling you the desired output for each input?

Can you learn to generalize to novel inputs?
Supervised Learning

Two Primary Tasks

1. **Classification**
   - Inputs $u_1, u_2, \ldots$ and discrete classes $C_1, C_2, \ldots, C_k$
   - Training examples: $(u_1, C_2), (u_2, C_7), \text{etc.}$
   - Learn the mapping from an arbitrary input to its class
   - Example: Inputs = images, output classes = face, not a face

2. **Function Approximation (regression)**
   - Inputs $u_1, u_2, \ldots$ and continuous outputs $v_1, v_2, \ldots$
   - 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

Classification using “Perceptrons”

- Fancy name for a type of layered feedforward networks
- Uses artificial neurons (“units”) with binary inputs and outputs

![Single-layer Perceptron](image1.png)  

![Multilayer Perceptron](image2.png)
Perceptrons use “Threshold Units”

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

\[
v_j = \Theta(\sum_j w_{ij} u_j - \mu_i)
\]

\( \Theta(x) = 1 \) 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

\[
\sum_j w_{ij} u_j - \mu_i = 0
\]

- 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 is linearly separable
  - $a \text{ AND } b = 1$ if and only if $a = 1$ and $b = 1$

Linear hyperplane $v = 1.5$

Perceptron for AND

Perceptron Learning Rule

- Given inputs $u$ and desired output $v^d$, adjust $w$ as follows:
  
  1. Compute error signal $e = (v^d - v)$ where $v$ is the current output
  
  2. Change weights according to the error $e$
     - For positive inputs, increase weights if error is positive and decrease if error is negative (opposite for negative inputs)
     
     $$w \rightarrow w + \varepsilon(v^d - v)u$$
     
     $A \rightarrow B$ means replace $A$ with $B$
Linear Inseparability

- Single-layer perceptron with threshold units fails if classification task is not linearly separable
  - Example: XOR
  - $a \text{ XOR } b = 1 \text{ iff } (a = -1, b = 1) \text{ or } (a = 1, b = -1)$
  - No single line can separate the “yes” (+1) outputs from the “no” (-1) outputs!

Solution in 1980s: 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 $x + y - 2\Theta(x + y - 1.5) - 0.5 > 0$

(Here, inputs $x, y$ are assumed to be 0 or 1)
What if you want to approximate a continuous function?

Can a network learn to drive?

Example Network

Get steering angle from a human driver

Desired Output: \( \mathbf{d} = (d_1, d_2, \ldots, d_{30}) \)

Get current camera image

Input \( \mathbf{u} = (u_1, u_2, \ldots, u_{960}) \) = image pixels
Function Approximation

- We want networks that can learn a function
- Network maps real-valued inputs to real-valued outputs
- Want to generalize to predict outputs for new inputs
- Idea: Given input data, minimize errors between network’s output and desired output by adapting weights

Sigmoidal Networks

The most common activation function:

Sigmoid function:

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

(Non-linear “squashing” function)
Gradient-Descent Learning (“Hill-Climbing”)

✦ Given training examples \((u^m, d^m)\) \((m = 1, \ldots, N)\), define an error function (cost function or “energy” function)

\[
E(w) = \frac{1}{2} \sum_m (d^m - v^m)^2 \quad v^m = g(w^T u^m)
\]

✦ Would like to change \(w\) so that \(E(w)\) is minimized

\(\Rightarrow\) Gradient Descent: Change \(w\) in proportion to \(-dE/dw\) (why?)

\[
w \rightarrow w - \epsilon \frac{dE}{dw}
\]

\[
\frac{dE}{dw} = -\sum_m (d^m - v^m) \frac{dv^m}{dw} = -\sum_m (d^m - v^m) g'(w^T u^m) u^m
\]

“Stochastic” (or On-line) Gradient Descent

✦ What if the inputs only arrive one-by-one?

✦ Stochastic gradient descent approximates sum over all inputs with an “on-line” running sum:

\[
w \rightarrow w - \epsilon \frac{dE_i}{dw}
\]

\[
\frac{dE_i}{dw} = -(d^m - v^m) g'(w^T u^m) u^m
\]

Also known as the “delta rule” or “LMS rule”

delta = error
But wait….

- Delta rule tells us how to modify the connections from input to output (one layer network)
  - One layer networks are not that interesting (remember XOR?)
- What if we have multiple layers?

\[
\begin{align*}
\text{Input } u &= (u_1 \ u_2 \ldots u_K)^T \\
\text{Output } v &= (v_1 \ v_2 \ldots v_J)^T; \quad \text{Desired } d
\end{align*}
\]

How do we adapt these “hidden” connections?

Let’s Backpropagate (Errors)

- Backpropagation = gradient-descent learning for multilayer feedforward networks
- **Idea**: Propagate credit/blame for errors back to internal nodes
  - Use chain rule (from calculus) to change weights for internal “hidden” nodes

\[
\text{error } = (d - v)
\]

Backpropagate this to correct all weights

\[
\begin{align*}
\text{Output } v &= (v_1 \ v_2 \ldots v_J)^T \\
\text{Input } u &= (u_1 \ u_2 \ldots u_K)^T
\end{align*}
\]

Delta rule
Backprop rule
Notation for Backprop

\[ v_i^m = g\left(\sum_j W_{ij} g\left(\sum_k w_{jk} u_k^m\right)\right) \]

Find \( W \) and \( w \) that minimize total squared output error:

\[ E(W, w) = \frac{1}{2} \sum_m \left\| d^m - v^m \right\|^2 \]

\[ = \frac{1}{2} \sum_{m,i} (d_i^m - v_i^m)^2 \]

Backpropagation (for Math lovers’ eyes only!)

✦ Learning rule for hidden-output connection weights:

\[ W_{ij} \rightarrow W_{ij} - \epsilon \frac{\partial E}{\partial W_{ij}} \]

\[ \frac{dE}{dW_{ij}} = -\sum_w (d^w_i - v^w_i)g'(\sum_j W_{ij} x_j^w)x_j^w \quad \text{Delta rule} \]

✦ Backpropagation rule for input-hidden connection weights:

\[ w_{jk} \rightarrow w_{jk} - \epsilon \frac{\partial E}{\partial w_{jk}} \quad \text{But: } \frac{\partial E}{\partial w_{jk}} = \frac{\partial E}{\partial x_j^m} \cdot \frac{\partial x_j^m}{\partial w_{jk}} \quad \{\text{chain rule}\} \]

\[ \frac{dE}{dw_{jk}} = -\sum_{m,i} (d_i^m - v_i^m)g'(\sum_j W_{ij} x_j^m)W_{ij} \cdot g'(\sum_k w_{jk} u_k^m)u_k^m \]
Learning to Drive using Backprop

One of the learned “road features” $w_i$

ALVINN (Autonomous Land Vehicle in a Neural Network)

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)
Next Class: Reinforcement Learning

✦ Things to do:
  ➔ Read Chapter 9
  ➔ Finish Last Homework (due Thu, May 24)
  ➔ Work on mini-project

I’ll be bäck (for reinf. learning)