Lecture 10. Adaptive Networks II. Gradient Descent and Backpropagation

Reading Assignments:

HBTNN:
   Backpropagation (Werbos)
   Sensorimotor Learning (Massone)

TMB2
   8.2 Adaptive Networks
Training Hidden Units

In the simple Perceptron (Rosenblatt 1962), only the weights to the output units can change. This architecture can only support linearly separable maps. The problem for many years was to extend the perceptron concept to multilayered networks.

The credit assignment problem: "How does a neuron deeply embedded within a network 'know' what aspect of the outcome of an overall action was 'its fault'?

I.e.: given an "error" which is a global measure of overall system performance, what local changes can serve to reduce global error?
Back-Propagation

**Backpropagation**: a method for training a loop-free network which has three types of unit:
- input units;
- hidden units carrying an internal representation;
- output units.

![Diagram of a neural network with input, hidden, and output layers.](image)
Neurons

Each unit has both input and output taking continuous values in some range \([a,b]\). The response is a sigmoidal function of the weighted sum.

Thus if a unit has inputs \(x_k\) with corresponding weights \(w_{ik}\), the output \(x_i\) is given by

\[
x_i = f_i(\sum_k w_{ik} x_k)
\]

where \(f_i\) is the sigmoid function

\[
f_i(x) = \frac{1}{1 + \exp (- (x - \theta_i))}
\]

with \(\theta_i\) a bias (threshold) for the unit.

Unlike the Heaviside step function, this \(f\) is differentiable.
Choose a training set $T$ of pairs $(p,t)$ each comprising an input pattern $p$ and the corresponding desired output vector $t$.

At each trial, we choose an input pattern $p$ from $T$ and consider the corresponding restricted error

$$E = \Sigma_k (t_k - o_k)^2$$

where $k$ ranges over designated "output units" with $(t_1, ..., t_n)$ the target output vector, and $(o_1, ..., o_n)$ the observed output vector.

The net has many units interconnected by weights $w_{ij}$. The learning rule is to change $w_{ij}$ so as to reduce $E$ by gradient descent.

To descend the hill, reverse the derivative.

$$\Delta w_{ij} = - \frac{\partial E}{\partial w_{ij}} = 2 \Sigma_k (t_k - o_k) \frac{\partial o_k}{\partial w_{ij}}$$
Backpropagation

In a layered loop-free net, changing the weights $w_{ij}$ according to the gradient descent rule may be accomplished equivalently by back propagation, working back from the output units:

**Proposition:** Consider a layered loop-free net with energy $E = \sum_k (t_k - o_k)^2$, where $k$ ranges over designated "output units," and let the weights $w_{ij}$ be changed according to the gradient descent rule

$$\Delta w_{ij} = - \frac{\partial E}{\partial w_{ij}} = 2 \sum_k (t_k - o_k) \frac{\partial o_k}{\partial w_{ij}}.$$ 

Then the weights may be changed inductively, working back from the output units …..
Output Units

We saw:

$$
\Delta w_{ij} = - \frac{\partial E}{\partial w_{ij}} = 2 \sum_k (t_k - o_k) \frac{\partial o_k}{\partial w_{ij}}.
$$

with $w_{ij}$ connecting from $j$ to $i$. So, for output unit $i$,

$$
\Delta w_{ij} = 2 (t_i - o_i) \frac{\partial o_i}{\partial w_{ij}}
$$

because $\frac{\partial o_k}{\partial w_{ij}} = 0$ for $k \neq i$

since unit $k$ receives inputs only through weights of the form $w_{kj}$ from other units $j$. 

![Diagram showing connections between units i, j, and k]
Laurent Itti: CS564 - Brain Theory and Artificial Intelligence. Adaptive Networks II

**Backpropagation Rule**

\[ \Delta w_{ij} = 2 \left( t_i - o_i \right) \frac{\partial o_i}{\partial w_{ij}} \]

with

\[ o_i = f_i \left( \sum_k w_{ik} o_k \right) = f_i(u_i) \]

where \( o_k \) is the activity of unit \( k \) in the previous layer (so, the sum over \( k \) does not include \( i! \)) connecting to unit \( i \).

\[ \Delta w_{ij} = 2 \left( t_i - o_i \right) \frac{\partial o_i}{\partial u_i} \frac{\partial u_i}{\partial w_{ij}} \]

\[ = 2 \left( t_i - o_i \right) \frac{\partial f_i(u_i)}{\partial u_i} \frac{\partial (\Sigma_k w_{ik} o_k)}{\partial w_{ij}} \]

\[ = 2 \left( t_i - o_i \right) f_i'(u_i) o_j \]

\( \Delta w_{ij} \) is proportional to \( \delta_i o_j \), where:

**Basis Step:** \[ \delta_i = (t_i - o_i)f_i' \] for an output unit.

[cf. Perceptron - but with added \( f_i' \) term. Widrow-Hoff 1960]
Hidden Units

Induction Step: If i is a hidden unit, and if $\delta_k$ is known for all units which receive unit i's output, then:

$$\delta_i = (\Sigma_k \delta_k w_{ki}) f'_i$$

where k runs over all units which receive unit i's output, and $\Delta w_{ij}$ is proportional to $\delta_i o_j$

[see TMB2 8.2 for proof]

The "error signal" $\delta_i$ propagates back layer by layer.

$\Sigma_k \delta_k w_{ki}$: unit i receives error propagated back from a unit k to the extent to which i affects k.
A Non-Biological Heuristic

The above theorem tells us how to compute $\Delta w_{ij}$ for gradient descent.

It does not guarantee that the above step-size is appropriate to reach the minimum;
It does not guarantee that the minimum, if reached, is global.

The back-propagation rule defined by this proposition is thus a heuristic rule, not one guaranteed to find a global minimum.
Since it is heuristic, it may also be applied to neural nets which are loop-free, even if not strictly layered.
Sensorimotor Learning

An act cannot be evaluated in terms of the motor pattern produced by the net, but rather by the sensory pattern produced by the motion.

Example: Learning to pronounce words:

Let \( X \) be the set of neural motor commands;
\( Y \) be the set of sensory patterns they produce

Physics (Muscles + Vocal Tract + Sound Waves + Ear + Auditory System) defines a function \( f: X \rightarrow Y \). \( f(x) \) is the neural code for what is heard when neural command \( x \) is sent to the articulators.
Critique

What such learning methods achieve:

In "many cases" (the bounds are not yet well defined)

- if we train a net $N$ with repeated presentations of the various $(x_k, y_k)$ from some training set

- then it will converge to a set of connections which enable $N$ to compute a function $f: X \rightarrow Y$ with the property that as $k$ runs from 1 to $n$, the $f(x_k)$ "correlate fairly well" with the $y_k$. 
An end to programming? NO!!

Consider three issues:

a) complexity: Is the network complex enough to encode a solution method?

b) practicality: Can the net achieve such a solution within a feasible period of time?

c) efficacy: How do we guarantee that the generalization achieved by the machine matches our conception of a useful solution?
Given a complex problem, programmers will still need to

- Decompose it into subproblems
- Specify an initial structure for a separate network for each subproblem
- Place suitable constraints on (the learning process for) each network; and, finally,
- Apply debugging techniques to the resultant system.

We may expect that the initial design and constraint processes may in some cases suffice to program a complete solution to the problem without any use of learning at all.
which we will probably not have time to cover in class, but which can help your reading of TMB2…
First learn MM: Mental Model, then MP: Motor Program

MM: a network which models the "physics" \( f \) predicting the sensory effects of every command.

Once MM is built, its connections are fixed — or slowly varying to provide a stable reference for MP.

Then MP is trained by adjusting the total network \([\text{MP} \to \text{MM}]\) (but with MM fixed) by back-propagation.

MP is to implement a function \( g: Y \to X \) with the property \( f(g(y)) \approx y \) for each desired sensory situation \( y \):

generating the motor program for a desired response.
Learning to repeat words that have been heard

MP, the Motor Program, receives as input the "intention," a vector representing a word as heard and is to yield an output which drives the articulators to reproduce that word.

The teacher cannot specify the right motor activity to produce the word. The net, MM for Mental Model, is to model the physics of the articulators and the ears.
**Stage 1: The "babble phase"**

(or, in motor control: a "flailing phase"), using more or less random sounds and movements to build up MM.

**Expectation** = the current output of MM for a given input
i.e., motor pattern generated by MP

(Sensory Feedback - Expectation)$^2$
is the error to train MM by backpropagation.

MM is a kind of efference copy device, making a copy of the desired effect available within the network. The system could get locked in if MM is not doing a good enough job; it thus requires a sufficiently varied set of inputs from MP to "span" the motor space $X$. Later MM can be fine-tuned while training MP on its repertoire.
Stage 2

Provide the network MP for generating motor programs as template the sound of a word we wish it to produce, and have MP repeatedly try to utter a word and hear if it sounds the same.

MP is adjusted by applying back propagation to the combined network MP+MM, but with the connections in MM being "clamped" as being already correct.

Heuristics suggest that the system will still converge to the desired "identity function" for MP+MM.
Computing Shape from Shaded Images

Given a single image, we can often infer shape from shading.

Characterize the surface locally by 2 curvatures \( (k = 1/r) \) -- the maximum and minimum curvatures, \( k_1 \) and \( k_2 \):

- a. \( k_1 = k_2 = 0 \) for a plane
- b. \( k_1 > 0, \ k_2 = 0 \) for a cylinder
- c. \( k_1 > 0, \ k_2 < 0 \) at a saddle
- d. \( k_1 < 0, \ k_2 < 0 \) on a concave surface
- e. \( k_1 > 0, \ k_2 > 0 \) on a convex surface.
Neural Network Inferring Curvature

Sejnowski and Lehky 1987 constructed a three-layer neural network model to infer curvature at a point of a surface from a local sample of its shaded image.

Input is from an array of on-center and off-center receptive fields on a hexagonal array.

These are connected via hidden units to output cells with responses tuned for, e.g., the orientation (direction of axis of maximal curvature in horizontal section of surface) and curvature of the surface at a specified point of the surface.
Neural Network Inferring Curvature

They use a rectangular array of 24 output units. There are 6 different orientation angles labeling the columns of the array:
- the top 2 rows code ± for the maximum curvature
- the bottom 2 rows code ± for the minimum curvature.

The net is trained by backpropagation to compute a set of connections which minimizes error on output layer between desired patterns and the firing rates that are actually produced — but they do not claim that this rule governs the brain's maturation.

They measure the performance of the network by measuring the correlation between actual and desired outputs.
- With no hidden units, the correlation asymptotes at 0.7.
- With 12 or 27 or more hidden units, it asymptotes at 0.9.
They examined many hidden units to see what their response properties were, and found cells with oriented receptive fields similar to those of Hubel-Wiesel simple cells in the visual cortex of cats and monkeys that respond optimally to oriented bars and edges.

The images used here **had no edges** as such, only shading!!

The **proposed circuit is akin to a column**
Many such must be integrated for looking at a complex figure.