Backpropagation Learning

15-496/782: Artificial Neural Networks
David S. Touretzky
Spring 2004

Reading: HK&P sections 5.4 and 6.1-6.3

LMS / Widrow-Hoff Rule

\[ \Delta w_i = -\eta (y-d)x_i \]

Works fine for a single layer of trainable weights. What about multi-layer networks?

With Linear Units, Multiple Layers Don’t Add Anything

\[ y = U \times (Vx) = (U \times V)x \]

1 layer of trainable weights

What Can be Done with Linear Threshold Units?

Linear operators are closed under composition. Equivalent to a single layer of weights \( W = U \times V \)

But with non-linear units, extra layers add computational power.
What Can be Done with Linear Threshold Units?

2 layers of trainable weights

convex polygon region

How Do We Train A Multi-Layer Network?

Can't use perceptron training algorithm because we don't know the 'correct' outputs for hidden units.

Define sum-squared error:

\[ E = \frac{1}{2} \sum_{p} (d^p - y^p)^2 \]

Use gradient descent error minimization:

\[ \Delta w_j^k = -\eta \frac{\partial E}{\partial w_j^k} \]

Works if the nonlinear transfer function is differentiable.
Gradient Descent Learning

\[ y = \sum w_i x_i \]

\[ E = \frac{1}{2} \sum (d^p - y^p)^2 \]

\[ \frac{\partial E}{\partial y} = y - d \]

\[ \frac{\partial E}{\partial w_i} = \frac{\partial E}{\partial y} \frac{\partial y}{\partial w_i} = (y - d) x_i \]

\[ \Delta w_i = -\eta \frac{\partial E}{\partial w_i} = -\eta (y - d) x_i \]

How do we extend this to two layers?

Now We Can Use The Chain Rule

\[ \frac{\partial E}{\partial y_k} = (y_k - d_k) \]

\[ \delta_k = \frac{\partial E}{\partial \text{net}_k} = (y_k - d_k) f'(\text{net}_k) \]

\[ \frac{\partial E}{\partial w_{kj}} = \frac{\partial E}{\partial \text{net}_k} \frac{\partial \text{net}_k}{\partial w_{kj}} = \frac{\partial E}{\partial \text{net}_k} y_j \]

\[ \frac{\partial E}{\partial w_{kj}} = \frac{\partial E}{\partial \text{net}_j} \frac{\partial \text{net}_j}{\partial w_{kj}} = \delta_j y_i \]

\[ \delta_j = \frac{\partial E}{\partial \text{net}_j} = \frac{\partial E}{\partial y_j} f'(\text{net}_j) \]

\[ \frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial \text{net}_i} \frac{\partial \text{net}_i}{\partial w_{ij}} = \delta_i \]

\[ \Delta w_{kj} = -\eta \frac{\partial E}{\partial w_{kj}} \]

\[ \Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} \]

Use Smooth Nonlinear Units

\[ \text{net}_j = \sum w_{ij} y_i \]

\[ y_j = f(\text{net}_j) \quad \text{f must be differentiable} \]

Common choices for f:

\[ f(x) = \frac{1}{1+e^{-x}} \]

\[ f'(x) = f(x)(1-f(x)) \]

\[ f(x) = \tanh(x) \]

\[ f'(x) = 1/cosh^2(x) \]

Weight Updates
Function Approximation

Bumps from which we compose \( f(x) \)
\[
\tanh(w_0 + w_1 x)
\]

3n+1 free parameters for \( n \) hidden units

Encoder Problem

Input patterns: 1 bit on out of \( N \).
Output pattern: same as input.
Only 2 hidden units: bottleneck!

5-2-5 Encoder Problem

Training patterns:
- **A:** 0 0 0 0 1
- **B:** 0 0 0 1 0
- **C:** 0 0 1 0 0
- **D:** 0 1 0 0 0
- **E:** 1 0 0 0 0

Hidden code:
- **A:** 2, 0
- **B:** 0, 2
- **C:** 1, -1
- **D:** -1, 1
- **E:** -1, 0

Solving XOR

\[
\begin{array}{ccc}
    x_1 & x_2 & y \\
    0 & 0 & 0 \\
    0 & 1 & 1 \\
    1 & 0 & 0 \\
    1 & 1 & 0 \\
\end{array}
\]

Two solutions:
\[
(x_1 \lor x_2) \land \neg x_1 \land x_2
\]

Try the bpxor demo. Which solution does it use?
Local Minima

One problem with backprop is that the error surface is no longer bowl-shaped.

Gradient descent can get trapped in local minima.

In practice, this does not usually prevent learning.

“Noise” can get us out of local minima. Large learning rates can be a source of noise due to overshooting.

Flat Spots

If weights become large, net becomes large, derivative of f() goes to zero.

Fahlman's trick: add a small constant to f'(x) to keep the derivative from going to zero. Typical value is 0.1.

Reachable Targets

Targets of 0 and 1 are unreachable by the logistic or tanh functions.

Weights get large as the algorithm tries to force each output unit to reach its asymptotic value.

Trying to get a “correct” output from 0.95 up to 1.0 wastes time and resources that should be concentrated elsewhere.

Solution: use “reachable targets” of 0.1 and 0.9 instead of 0/1. And don't penalize the network for overshooting these targets.

Error Signal Attenuation

The error signal δ gets attenuated as it moves backward through multiple layers.

So different layers learn at different rates.

Input-to-hidden weights learn more slowly than hidden-to-output weights.

Solution: have different learning rates η for different layers.
Fan-In Affects Learning Rate

Solution: scale learning rate by fan-in.

Momentum

Learning is slow if the learning rate is set too low.
Gradient may be steep in some directions but shallow in others.
Solution: add a momentum term $\alpha$.

$$\Delta w_{ij}(t) = -\eta \frac{\partial E}{\partial w_{ij}(t)} + \alpha \Delta w_{ij}(t-1)$$

Typical value for $\alpha$ is 0.9.
If the direction of the gradient remains constant, the algorithm will take increasingly large steps.

Weights Can Oscillate If Learning Rate Set Too High

Solution: calculate the cosine of the angle between successive weight vectors.

$$\cos \theta = \frac{\hat{w}(t) \cdot \hat{w}(t-1)}{||\hat{w}(t)|| ||\hat{w}(t-1)||}$$

If cosine close to 1, things are going well.
If cosine < 0.95, reduce the learning rate.
If cosine < 0, we’re oscillating: cancel the momentum.

$$\Delta w(t) = -\eta \frac{\partial E}{\partial w} + \alpha \Delta w(t-1)$$

The “Herd Effect” (Fahlman)

Hidden units all move in the same direction at once, instead of spreading out to divide and conquer.

Solution: use initial random weights, not too large (to avoid flat spots), to encourage units to diversify.

Use a small initial learning rate to give units time to sort out their “specialization” before taking large steps in weight space.

Add hidden units one at a time. (Cascor algorithm.)
How Many Layers Do We Need?

Two layers of weights suffice to compute any “reasonable” function.
But it may require a lot of hidden units!

Why does it work out this way?
Lapedes & Farmer: any reasonable function can be approximated by a linear combination of localized “bumps” that are each nonzero over a small region.
These bumps can be constructed by a network with two layers of weights.

Early Application of Backprop:
From DECtalk to NETtalk

DECtalk was a text-to-speech program that drove a Votrax speech synthesizer board.

Contained 700 rules for English pronunciation, plus a large dictionary of exceptions.

Developed over several years by a team of linguists and programmers.

NETtalk Learns to Read

In 1987, Sejnowski & Rosenberg made national news when they used backprop to “teach” a neural network to “read aloud”.

Output: 23 phonetic feature units
plus 3 for stress, syll. boundaries.
Hidden layer: 0-120 units.
Input: 7 letter window containing 7x29 = 206 units.

Training the network with 10,000 weights took 24 hours on a VAX-780 computer. (Today it would take a few minutes.)

Why Was NETtalk Interesting?

No explicit rules. No exception dictionary. Trained in less than a day. Programmers now obsolete!

NETtalk went through “developmental stages” as it learned to read. Analogous to child development?
CV alternation: “babbling” word boundaries recognized: “pseudo-words” many words intelligible understandable text

Graceful response to “damage” (some weights deleted, or noise added.) Rapid recovery with retraining. Analogous to human stroke patients?
Learning Curves for 0-120 Hidden Units

Training set was a 1000 word dictionary corpus; many irregular words.
No hidden units: 82% best guess.
120 hidden units: 98% best guess.
Errors in the no “hidden units” case were often inappropriate.
Hidden units allow for more contextual influence by recognizing higher order features in the input.

Test of Generalization Performance

Initial training: 1000 words, with 120 hidden units.
Testing set was a 20,012 word dictionary.

No additional training:
77% best guess
28% perfect match

After 5 training passes:
90% best guess
48% perfect match

Effects of Damage

Std. dev. of the original, undamaged weights was 1.2

Random weight perturbations in [-.5,.5] had little effect.

So each weight must convey only a few bits of information.

Relearning After Damage

Relearning was about 10 times faster to achieve similar performance.

Analogy to rapid recovery of language in stroke patients?
Was NETtalk Really Competitive?

Couldn't handle words with context-dependent pronunciations ("lead") or stresses ("survey").

Couldn't handle grammatical structure, e.g., questions vs. declarative sentences.

Lacked clever contextual tricks, such as:
  "he dove" vs. "the dove"
  "Dr. Smith" vs. "51 Rodeo Dr."

But not bad for a seven letter window!