Neural Networks

- Feed forward networks of (usually) sigmoid functions (continuous approximation of LTUs)
- Powerful models - can represent any boolean function (with exponentially many hidden nodes)
- Learn by gradient descent
  - many local minima
- Not like an artificial brain: brain much more connected, many more nodes, much more parallel, spike trains, etc.

Example

| 1 | 2 | 3 | 4 |

Node $i$

$z_i = f(a)$

$a = \sum_j w_{ji} z_j + b$

Bias $b$ (fixed to 1)

$z_i$ values coming in

Output $z_i$ to other nodes

$z_i$ is output of node $i$

$w_{i,j}$ is weight from $i$ to $j$

Alpaydin notation different:

$w$, $v$: weights

$x$, $z$, $y$: node vals

Architecture usually fixed (nodes, functions at nodes, and connections)

Weights $w_{ij}$ learned from data

To compute value, put attributes at input nodes, each node computes an activation $a_i = \sum_j w_{ji} z_j$ (weighted sum of outputs of nodes feeding into $i$)

Node output $z_i$ is some $f(a_i)$

- Exception: for input nodes, output $z_i = x_i$

Common $f(a)$ are tanh and logistic sigmoid

Net for XOR

Hidden nodes learn useful subpatterns

Biases important

Inputs $x_i$ are 0/1; $f(a) = \tanh(a)$
Backpropagation used to learn weights

• Given a new example \((x,r)\) compute error \((output - r)^2\)
• Want \(\partial \text{error} / \partial w\) for each weight and bias in network, update each \(w := w - \eta \frac{\partial \text{error}}{\partial w}\)
• Useful quantity: \(\delta_j = \frac{\partial \text{error}}{\partial a_j}\)
• \(\partial \text{error} / \partial w_{ij} = \delta_i z_j\) (already have \(z_j\))
• Otherwise, \(\delta_i = f'(a_i) \sum_k \delta_k w_{ik}\) (sum over nodes using \(v_i\))
  – Need \(\delta_k\) for \(k\) using \(z_i\): backpropagate \(\delta\) values

Backpropagation notes:

• Evaluate forward, backpropagate to get gradients
• Computationally efficient, but many iterations through data
• Can do on-line (one example, stochastic GD) or batch updates
• Can have multiple output nodes, and output nodes can be linear (instead of sigmoid)
• Complicated surface (many local minimums): do multiple runs, pick best

Improving Convergence

• Momentum
  \[\Delta w'_i = -\eta \frac{\partial E}{\partial w'_i} + \alpha \Delta w_{i-1}\]
• Adaptive learning rate
  \[\Delta \eta = \begin{cases} 4\eta & \text{if } E^{\text{new}} < E^t \\ -b\eta & \text{otherwise} \end{cases}\]

Backpropagation Algorithm

1. Forward pass: compute all \(a_i\) and \(z_i\) values
2. Compute errors for output node(s)
3. Compute \(\delta\) values for each node in a backwards pass through net
4. Update \(w\)’s based on gradient (computed from activations and \(\delta\)’s)

Initialization issues

Saturation \((a_i\text{ big})\) is bad because sigmoid flat and gradient small
- Solution: make initial weights small
Symmetry must be broken (so hidden nodes learn different things)
- Solution: use random initialization
Finding good topology
- Solution: It is a black art, but there are “brain surgery” techniques

Overfitting/training (Alpaydin)

Number of weights: \(H(d+1)+(H+1)\times\#\text{outputs}\)
ANN notes:

- MultiClass:
  - Use one output $z_c$ per class $c \in C$
  - Combine with softmax: $P(\text{class}) = \frac{\exp(v_{\text{class}})}{\sum \exp(v_c)}$
- 1 hidden layer is a universal approximator, but multiple layers may give simpler nets
- Weight sharing and wt regularization
- Evolutionary methods?

Dimensionality Reduction

Neural Net (classic) Successes

- Pronunciation - mapping text to phonemes (NETtalk, 1987) used 7 character window
- Handwritten character recognition (LeCun 1989) three hidden layers sparsely connected, compiled into silicon and used for mail sorting
- Driving: ALVINN (Pomerleau, 1993) maps from video to steering direction, actually driven on highways.

Neural Net Summary

- Model: flexible, very flexible over all topologies, but must pick topology
- Data: Numeric
- Interpretable? No (but some pretty pictures)
- Missing values? No
- Noise/outliers? Very good
- Irrelevant features? Bad
- Comp. efficiency? Good (but local minima)