Neural Networks

ANN for Artificial Neural Networks
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

$z_i$ is output of node $i$

$W_{j,i}$ is weight from $i$ to $j$
• Architecture usually fixed (nodes, functions at nodes, and topology)
• Weights $w_{j,i}$ learned from data
• To compute value, put attributes at input nodes, each node $j$ computes an activation $a_j = \sum_i w_{j,i} z_i$ (weighted sum of outputs of nodes feeding into $j$)
• Node output $z_j$ is some $f_j(a_j)$
  – but for input nodes, output $z_i = x_i$
• Common $f(a_j)$ are tanh and logistic sigmoid
Node $i$

$z_i = f(a_i)$

$a_i = \sum_j w_{j,i} z_j + w_{0,i} b$

Bias $b$ (fixed to 1)

$z_j$ values coming in

Output $z_i$ to other nodes

$f = \tanh(a)$

$\sigma(a) = 1/(1+\exp(-a))$
Net for XOR

Hidden nodes learn useful subpatterns
Biases important

Inputs $x_i$ are 0/1; $f(a) = \sigma(a)$
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.
ANN error surface

\[ E(w) \]

\[ \nabla E \]
Backpropagation used to learn weights

- For new example \((x,t)\) compute error \((output - t)^2\)
- Want \(\frac{\partial \text{error}}{\partial w}\) for each weight and bias in network, update each \(w := w - \eta \frac{\partial \text{error}}{\partial w}\)
- Useful quantities: \(\delta_j = \frac{\partial \text{error}}{\partial a_j}\)
- \(\frac{\partial \text{error}}{\partial w_{j,i}} = \delta_j z_i\) (already have \(z_i\))
- If \(j\) is output node, \(\delta_j = 2 f'(a_j)(t-z_j)\)
- Otherwise, \(\delta_j = f'(a_j) \sum_k \delta_k w_{k,j}\) (sum over nodes using \(z_j\))
  - Need \(\delta_k\) for nodes \(k\) using \(z_i\): \textit{backpropagate} \(\delta\) vals
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)
Backpropagation notes:

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

Saturation ($a_i$ 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, “brain surgery” techniques proposed
Improving Convergence

- **Momentum**
  \[
  \Delta w_i^t = -\eta \frac{\partial E^t}{\partial w_i} + \alpha \Delta w_i^{t-1}
  \]

- **Adaptive learning rate**
  \[
  \Delta \eta = \begin{cases} 
  +c & \text{if } E^{t+\tau} < E^t \\
  -b\eta & \text{otherwise}
  \end{cases}
  \]
Overfitting/training (Alpaydin)

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

• **MultiClass:**
  – Use one output $z_c$ per class $c$ in $C$
  – Combine with softmax: $P(\text{class} \mid X) = \frac{\exp(z_{\text{class}})}{\sum_c \exp(z_c)}$

• 1 hidden layer is a universal approximator, but multiple layers may give simpler nets
• Weight sharing and weight regularization
• Evolutionary methods?
Dimensionality Reduction

Encoder

Decoder

Linear

Nonlinear
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? Fair (but local minima)