Math Modeling, Week 8

Multilayer neural networks

Like a chain of Rescorla-Wagner networks

Layers 0 through M

N_m nodes at layer m

Layer 0: input; layer M: output; intermediate: "hidden layers"

Weights w_{ij}^m from layer m-1 (node j) to layer m (node i)

Bias b_i^m for each node, like a weight from constant node

Input to each node

$$v_i^m = \sum_{j=1}^{N_{m-1}} w_{ij}^m a_j^{m-1} + b_i^m$$

$$\mathbf{v}^m = \mathrm{W}^m \mathbf{a}^{m-1} + \mathbf{b}^m$$

Activation (output) of each node

$a_i^m = f_{\mathrm{act}}(v_i^m)$

Often sigmoid activation function: $a = \tanh(v) \in [-1,1]$ or $a = \mathrm{logistic}(v) = \frac{1}{1+e^{-v}} = \frac{1}{2}\left(1 + \tanh\frac{v}{2}\right) \in [0,1]$

Needed to introduce nonlinearity

Otherwise layers would be redundant: $\mathbf{a}^M = \left(\prod_{m=1}^{M} \mathrm{W}^m\right)\mathbf{a}^0 + \tilde{\mathbf{b}}^M$

Universal approximation

Can match any continuous function $\mathbf{a}^0 \mapsto \mathbf{v}^M$ to arbitrary precision, given enough hidden nodes/layers

Back-propagation

Learning algorithm for multilayer networks

Gradient descent on sum-squared error

$E = \frac{1}{2}\sum_{i=1}^{N_M}\left(v_i^M - R_i\right)^2$

R_i is feedback or correct value on output node i

Update by moving down the gradient: $\Delta w_{ij}^m = -\epsilon \frac{\mathrm{d}E}{\mathrm{d}w_{ij}^m}$

One-layer network

$\frac{\mathrm{dE}}{\mathrm{d}w_j} = \frac{\mathrm{d}v}{\mathrm{d}w_j} \cdot \frac{\mathrm{d}}{\mathrm{d}v}\left[\frac{1}{2}(v-R)^2\right] = a_j \cdot (v - R)$

Rescorla-Wagner rule: $\Delta w_j = \epsilon(R - v)a_j$ prediction error times cue value (a_j)

Multilayer network

$\frac{\mathrm{dE}}{\mathrm{d}w_{ij}^m}$ hard to compute directly for early layers m

Output depends on weight by exponentially many routes through intermediate layers

Recursive solution

Derivative of error wrt all nodes' inputs and outputs, using chain rule

Final layer: $\frac{\mathrm{d}E}{\mathrm{d}v_i^M} = v_i^M - R_i$

Node output: $\frac{\mathrm{d}E}{\mathrm{d}a_j^{m-1}} = \sum_{i=1}^{N_m} \frac{\mathrm{d}v_i^m}{\mathrm{d}a_j^{m-1}} \cdot \frac{\mathrm{d}E}{\mathrm{d}v_i^m} = \sum_{i=1}^{N_m} w_{ij}^m \cdot \frac{\mathrm{d}E}{\mathrm{d}v_i^m}$

Node input (m < M): $\frac{\mathrm{d}E}{\mathrm{d}v_i^m} = \frac{\mathrm{d}a_i^m}{\mathrm{d}v_i^m} \cdot \frac{\mathrm{d}E}{\mathrm{d}a_i^m} = f'_{\mathrm{act}}(v_i^m) \cdot \frac{\mathrm{d}E}{\mathrm{d}a_i^m}$

tanh: $f'_{\mathrm{act}}(v) = \mathrm{sech}^2\, v = 1 - a^2$

logistic: $f'_{\mathrm{act}}(v) = \frac{e^{-v}}{(1+e^{-v})^2} = a(1-a)$

Weight: $\frac{\mathrm{d}E}{\mathrm{d}w_{ij}^m} = \frac{\mathrm{d}v_i^m}{\mathrm{d}w_{ij}^m} \cdot \frac{\mathrm{d}E}{\mathrm{d}v_i^m} = a_j^{m-1} \cdot \frac{\mathrm{d}E}{\mathrm{d}v_i^m}$

Bias: $\frac{\mathrm{d}E}{\mathrm{d}b_i^m} = \frac{\mathrm{d}v_i^m}{\mathrm{d}b_i^m} \cdot \frac{\mathrm{d}E}{\mathrm{d}v_i^m} = \frac{\mathrm{d}E}{\mathrm{d}v_i^m}$

Algorithm

Define d_i^m for all nodes, as $d_i^m = \frac{\mathrm{d}E}{\mathrm{d}v_i^m}$

Set $d_i^M = v_i^M - R_i$ (negative prediction error on output layer)

Inductively set $d_j^{m-1} = f'_{\mathrm{act}}\left(v_j^{m-1}\right) \sum_{i=1}^{N_m} w_{ij}^m d_i^m$ for $m = M, \ldots, 2$

Update each weight by $\Delta w_{ij}^m = -\epsilon a_j^{m-1} d_i^m$

Update each bias by $\Delta b_i^m = -\epsilon d_i^m$