

Lecture 12.

MLP (IV): Programming & Implementation

Outline

- Matrix formulation
- Matlab Implementation

Summary of Equations (per epoch)

- Feed-forward pass: $z_0^{(\ell-1)}(k) \equiv 1$

For $k = 1$ to K , $\ell = 1$ to L , $i = 1$ to $N(\ell)$,

$$z_i^{(\ell)}(k) = f(u_i^{(\ell)}(k)) = 1/(1 + \exp[-u_i^{(\ell)}(k)])$$

$$u_i^{(\ell)}(k) = \sum_{j=0}^N w_{ij}^{(\ell)}(t) z_j^{(\ell-1)}(k)$$

t : epoch index

k : sample index

- Error-back-propagation pass:

For $k = 1$ to K , $\ell = 1$ to L , $i = 1$ to $N(\ell)$,

$$\delta_i^{(\ell)}(k) = \begin{cases} f'(u_i^{(\ell)}(k)) \cdot \sum_{m=1}^{N(\ell+1)} \delta_m^{(\ell+1)}(k) \cdot w_{im}^{(\ell+1)}(t) & \ell < L, \\ f'(u_i^{(L)}(k)) \cdot [d_i(k) - z_i^{(L)}(k)] & \ell = L. \end{cases}$$

Summary of Equations (cont'd)

- Weight update pass:

For $k = 1$ to K , $\ell = 1$ to L , $i = 1$ to $N(\ell)$,

$$-\frac{\partial E}{\partial w_{ij}^{(\ell)}(t)} = \sum_{k=1}^K \delta_i^{(\ell)}(k) z_j^{(\ell-1)}(k)$$

$$w_{ij}^{(\ell)}(t+1) = w_{ij}^{(\ell)}(t) - \mu \frac{\partial E}{\partial w_{ij}^{(\ell)}(t)} + \mu (w_{ij}^{(\ell)}(t) - w_{ij}^{(\ell)}(t-1))$$

Matrix Formulation

- The feed-forward, back-propagation, and weight-update loops can be formulated as matrix-vector product, or matrix-matrix product operations.
- Many computers can implement matrix operations using special instructions (e.g. MMX) to speed up the computation and hence improve training speed.
- Implementation using Matlab can be made quite efficient when the algorithm is formulated in terms of matrix, vector operations rather than loops (too much overhead).

Matrix Notations

$Z(\ell)$, $\ell = 0, 1, \dots, L$: $N(\ell) \times K$ matrix. i^{th} column is the outputs of neurons corresponding to the k^{th} training sample in the ℓ^{th} layer if $\ell > 0$, and is the training feature vector if $\ell = 0$.

$W(\ell)$, $\ell = 1, 2, \dots, L$: $N(\ell) \times (N(\ell-1)+1)$ weight matrix. The $(i,j)^{\text{th}}$ element of $W(\ell)$ is $w_{ij}^{(\ell)}$. The indices of its columns starts with 0 so that $w_{i0}^{(\ell)}$ corresponds to the bias input of the i^{th} neuron of the ℓ^{th} layer.

$$U(\ell + 1) = W(\ell) \cdot \begin{bmatrix} 1 & \dots & 1 \\ Z(\ell) \end{bmatrix}, \quad \text{and} \quad Z(\ell) = f(U(\ell))$$

the activation function $f(\bullet)$ is applied to individual elements if its argument is a matrix

Matrix Notations (Cont'd)

D : $N(L) \times K$ matrix of the desired output

$\delta(L) = f'(U(L)) \cdot *(D - Z(L))$: $N(L) \times K$ output layer delta error matrix. “.” means element by element product of two matrices (not usual matrix-matrix product).

$\delta(\ell) = f'(U(\ell)) \cdot [W(\ell+1)(:, 2:N(\ell+1)+1)]^T \delta(\ell+1)$: $N(\ell) \times K$ matrix where $W(\ell+1)(:, 2:N(\ell+1)+1)$ is the last $N(\ell+1)$ columns of the $W(\ell+1)$ matrix, and $[\]^T$ is matrix transpose.

$$\Delta W(\ell, t) = \eta \cdot \delta(\ell) \cdot \begin{bmatrix} 1 \\ \vdots \\ Z^T(\ell) \\ 1 \end{bmatrix} + \mu \cdot \Delta W(\ell, t-1) + \varepsilon;$$

ε : noise.

$$W(\ell, t+1) = W(\ell, t) + \Delta W(\ell, t)$$

Matlab Implementation

```

% BP iterations begins
while not_converged==1,
    % start a new epoch
    % Randomly select K training samples from the
    % training set.
    [train,ptr,train0]=rsample(train0,K,Kr,ptr);
    % train is K by M+N
    z{1}=(train(:,1:M))';
    % input sample matrix M by K, layer# = 1
    d=train(:,M+1:MN)';
    % corresponding target value N by K

    % Feed-forward phase,
    % compute sum of square errors
    for l=2:L, % the l-th layer
        u{1}=w{1}*[ones(1,K);z{1-1}];
        % u{1} is n(l) by K
        z{1}=actfun(u{1},atype(1));
    end
    error=d-z{L}; % error is N by K
    E(t)=sum(sum(error.*error));
    % Error back-propagation phase,
    % compute delta error
    delta{L}=actfunp(u{L},atype(L)).*error;
    % N (=n(L)) by K

    if L>2,
        for l=L-1:-1:2,
            delta{1}=(w{1+1}(:,2:n(1)+1))'...
*delta{1+1}.*actfunp(u{1},atype(1));
        end
    end

    % update the weight matrix using gradient,
    % momentum and random perturbation
    for l=2:L,
        dw{1}=alpha*delta{1}*...
            [ones(1,K);z{1-1}]'+...
            mom*dw{1}+randn(size(w{1}))*0.005;
        w{1}=w{1}+dw{1};
    end

    % display the training error
    bpdisplay; % call mfile bpdisplay.m

    % Test convergence to see if the
    % convergence condition is satisfied,
    cvgtest; % call mfile cvgtest.m
    t = t + 1; % increment epoch count
end % while loop

```

bp.m