Programming in MATLAB

Chapter 3: Multi Layer Perceptron

Gp.Capt.Thanapant Raicharoen, PhD
Outline

- Limitation of Single layer Perceptron
- Multi Layer Perceptron (MLP)
- Backpropagation Algorithm
- MLP for non-linear separable classification problem
- MLP for function approximation problem
### Limitation of Perceptron (XOR Function)

<table>
<thead>
<tr>
<th>No.</th>
<th>P1</th>
<th>P2</th>
<th>Output/Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2.</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3.</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4.</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

The XOR function is a classic example of a logical operation that cannot be implemented with a single layer of perceptrons. The table above shows the limitations of a single-layer perceptron in accurately representing the XOR function. The graph illustrates the decision boundary of a perceptron, which struggles to separate the data points into distinct classes due to the nature of the XOR operation.
Multilayer Feedforward Network Structure

Output of each node

\[ y_i^{(h)} = f \left( \sum_j w_{i,j}^{(h)} y_j^{(h-1)} + \theta_i^{(h)} \right) \]

where

\[ y_j^{(0)} = x_j = \text{input } j \quad \text{and} \quad y_i^{(N)} = o_i = \text{Output } i \]
Multilayer Perceptron: How it works

Function XOR

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Layer 1

$y_1 = f(w_{1,1}^{(1)}x_1 + w_{1,2}^{(1)}x_2 + \theta_1^{(1)})$

$y_2 = f(w_{2,1}^{(1)}x_1 + w_{2,2}^{(1)}x_2 + \theta_2^{(1)})$

Layer 2

$o = f(w_{1,1}^{(2)}y_1 + w_{1,2}^{(2)}y_2 + \theta_1^{(2)})$

$f( ) = \text{Activation (or Transfer) function}$
Multilayer Perceptron: How it works (cont.)

Outputs at layer 1

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$y_1$</th>
<th>$y_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Line $L_2 \Rightarrow w_{2,1}^{(1)}x_1 + w_{2,2}^{(1)}x_2 + \theta_2^{(1)} = 0$

Line $L_1 \Rightarrow w_{1,1}^{(1)}x_1 + w_{1,2}^{(1)}x_2 + \theta_1^{(1)} = 0$
Inside layer 1

Class 0

Class 1

$y_1-y_2$ space

$x_1-x_2$ space

Linearly separable!

Line $L_1 \Rightarrow w_{1,1}^{(1)} x_1 + w_{1,2}^{(1)} x_2 + \theta_1^{(1)} = 0$

Line $L_2 \Rightarrow w_{2,1}^{(1)} x_1 + w_{2,2}^{(1)} x_2 + \theta_2^{(1)} = 0$
Inside output layer

Space $y_1$-$y_2$ is linearly separable. Therefore the line $L_3$ can classify (separate) class 0 and class 1.
How hidden layers work

- Try to map data in hidden layer to be a linearly separable, before transferring these data into output layer.

- Finally the data in hidden layer should be linearly separable.

- There may be more than one hidden layer in order to map data to be linearly separable.

- Generally Activation function of each layer is not necessary to be a Hard limit (Thresholding) function and not to be the same function.
How can we adjust weights?

Assume we have a function \[ y = x_1 + 2x_2 \]

And we want to use a single layer perceptron to approximate this function.

Output is:

\[ \hat{y} = w_1 x_1 + w_2 x_2 \]

In this case: activation function is identity function (Linear function) \( f(x) = x \)

- We need to adjust \( w_1 \) and \( w_2 \) in order to obtain \( \hat{y} \) is close to \( y \) (or equal to)
Delta Learning Rule (Widrow-Hoff Rule)

Consider the Mean Square Error, MSE

\[ \varepsilon^2 = \left\langle (y - \hat{y})^2 \right\rangle \]

\[ = \left\langle (y - w_1 x_1 - w_2 x_2)^2 \right\rangle \]

\(\left\langle \right\rangle\) means average

\(\varepsilon^2\) is a function of \(w_1\) and \(w_2\) as see on this below graph

This graph called error surface (parabola)
Mean square error $\varepsilon^2$ as a function of $w_1$ and $w_2$

The minimum point is $(1,2)$. Because $\text{MSE} = 0$

Therefore, $w_1$ and $w_2$ must be adjusted in order to reach the minimum point in this error surface
$w_1$ and $w_2$ are adjusted to the minimum point like this:
Gradient Descent Method

What is the direction of steepest descent?
In what direction will the function decrease
Most rapidly.

1. calculate gradient of error
   surface in the current position
   \((w_1, w_2)\), gradient direction is steepest
   (Go to Hill Direction)

2. Walk to the opposite site of gradient
   (adjust \(w_1, w_2\))

3. Go to Step 1 until reach the minimum point
Backpropagation Algorithm

2-Layer case

\[
\hat{y}_j = f\left(\sum_i w_{j,i}^{(1)} x_i + \theta_j^{(1)}\right) = f(h_j^{(1)})
\]

\[
h_m^{(n)} = \text{weighted sum of input of Node } m \text{ in Layer } n
\]

\[
\hat{\sigma}_k = f\left(\sum_j w_{k,j}^{(2)} y_j + \theta_k^{(2)}\right) = f(h_k^{(2)})
\]
Backpropagation Algorithm (cont.)

2-Layer case

\[ \varepsilon^2 = \sum_k (o_k - \hat{o}_k)^2 \]  \hspace{1cm} (2.1)

\[ = \sum_k (o_k - f(\sum_j w_{k,j}^{(2)} y_j + \theta_k^{(2)}))^2 \] \hspace{1cm} (2.2)

\[ = \sum_k (o_k - f(\sum_j w_{k,j}^{(2)} \cdot f(\sum_i w_{j,i}^{(1)} x_i + \theta_j^{(1)}) + \theta_k^{(2)}))^2 \] \hspace{1cm} (2.3)

The derivative of \( \varepsilon^2 \) with respect to \( w_{k,j}^{(2)} \)

\[ \frac{\partial \varepsilon^2}{\partial w_{k,j}^{(2)}} = -2 \cdot (o_k - \hat{o}_k) \cdot f'(h_k^{(2)}) \cdot y_j \]

The derivative of \( \varepsilon^2 \) with respect to \( \theta_k^{(2)} \)

\[ \frac{\partial \varepsilon^2}{\partial \theta_k^{(2)}} = -2 \cdot (o_k - \hat{o}_k) \cdot f'(h_k^{(2)}) \]
Backpropagation Algorithm (cont.)

2-Layer case

\[ \varepsilon^2 = \sum_k (o_k - f(\sum_j w_{k,j}^{(2)} y_j + \theta_k^{(2)}))^2 \]  
(2.2)

\[ = \sum_k (o_k - f(\sum_j w_{k,j}^{(2)} \cdot f(\sum_i w_{j,i}^{(1)} x_i + \theta_j^{(1)}) + \theta_k^{(2)}))^2 \]  
(2.3)

The derivative of \( \varepsilon^2 \) with respect to \( w_{j,i}^{(1)} \)

\[ \frac{\partial \varepsilon^2}{\partial w_{j,i}^{(1)}} = -2 \cdot \sum_k (o_k - \hat{o}_k) \cdot \frac{\partial}{\partial w_{j,i}^{(1)}} (f(\sum_j w_{k,j}^{(2)} y_j + \theta_k^{(2)})) \]

\[ = -2 \cdot \sum_k (o_k - \hat{o}_k) \cdot f'(\sum_j w_{k,j}^{(2)} y_j + \theta_k^{(2)}) \cdot w_{k,j}^{(2)} \cdot f'(\sum_j w_{j,i}^{(1)} x_i + \theta_j^{(1)}) \cdot x_i \]

\[ = -2 \cdot \sum_k (o_k - \hat{o}_k) \cdot f'(h_k^{(2)}) \cdot w_{k,j}^{(2)} \cdot f'(h_j^{(1)}) \cdot x_i \]
Taking the derivative of $e^2$ with respect to $w^{(1)}_{j,i}$ in order to adjust the weight connecting the Node $j$ of current layer (Layer 1) with Node $i$ of Lower Layer (Layer 0).

\[
\frac{\partial e^2}{\partial w_{j,i}^{(1)}} = -2 \cdot \left( \sum_k (o_k - \hat{o}_k) \cdot f'(h_k^{(2)}) \cdot w_{k,j}^{(2)} \right) \cdot f'(h_j^{(1)}) \cdot x_i
\]

This part is the back propagation of error to Node $j$ at current layer.
Backpropagation Algorithm (cont.)

Derivative of $\varepsilon^2$ with $w^{(2)}_{k,j}$

$$\frac{\partial \varepsilon^2}{\partial w^2_{k,j}} = -2 \cdot (o_k - \hat{o}_k) \cdot f'(h^{(2)}_k) \cdot y_j$$

Error at current node

Derivative of current node

Input from lower node

Derivative of $e^2$ with $w^{(1)}_{j,i}$

$$\frac{\partial \varepsilon^2}{\partial w^{(1)}_{j,i}} = -2 \cdot \left( \sum_k (o_k - \hat{o}_k) \cdot f'(h^{(2)}_k) \cdot w^{(2)}_{k,j} \right) f'(h^{(1)}_j) \cdot x_i$$

Artificial Neural Network 3.19 Gp.Capt.Thanapant Raicharoen, PhD
Updating Weights: Gradient Descent Method

\[ \Delta w_{j,i}^{(n)} = -\alpha \cdot \frac{\partial \varepsilon^2}{\partial w_{j,i}^{(n)}} = \alpha \cdot \Delta_j^{(n)} \cdot f'(h_j^{(n)}) \cdot x_i^{(n-1)} \]

\[ \Delta \theta_j^{(n)} = -\alpha \cdot \frac{\partial \varepsilon^2}{\partial \theta_j^{(n)}} = \alpha \cdot \Delta_j^{(n)} \cdot f'(h_j^{(n)}) \]

Updating weights and bias

\[ w_{j,i}^{(n)}(new) = w_{j,i}^{(n)}(old) + \Delta w_{j,i}^{(n)} \]

\[ \theta_j^{(n)}(new) = \theta_j^{(n)}(old) + \Delta \theta_j^{(n)} \]
Adjusting Weights for a Nonlinear Function (Unit)

calculation $f'$, in case of nonlinear (function) unit

1. Sigmoid function  
   
   $$ f(x) = \frac{1}{1 + e^{-2\beta x}} $$

   We get  
   
   $$ f'(x) = 2\beta \cdot f(x) \cdot (1 - f(x)) $$

2. Function tanh(x)  
   
   $$ f(x) = \tanh(\beta x) $$

   We get  
   
   $$ f'(x) = \beta \cdot (1 - f(x)^2) $$

Special case of $f'$  
It’s easy to calculate $f'$
Backpropagation Calculation Demonstration

Neural Network DESIGN Backpropagation Calculation

Input: \( p = 1.0 \)
Target: \( t = \text{sigmoid}(p/4) = 1/10 \)
Simulate:
- \( a1 = 7 \)
- \( a2 = 7 \)
- \( e = 7 \)
Backpropagate:
- \( s2 = 7 \)
Update:
- \( W1 = 7 \)
- \( b1 = 7 \)
- \( W2 = 7 \)
- \( b2 = 7 \)

Last Error: 1.261

Backpropagation Calculation

Input: \( p = 1.0 \)
Target: \( t = \text{sigmoid}(p/4) = 1/10 \)
Simulate:
- \( a1 = \text{logsig}(W1*p+b1) = 0.321 \)
- \( a2 = \text{pseudolin}(W2*a1+b2) = 0.446 \)
- \( e = a2 - 1.261 \)
Backpropagate:
- \( s2 = \text{pseudolin}(W2*a1+b2) = 0.446 \)
Update:
- \( W1 = 7 \)
- \( b1 = 7 \)
- \( W2 = 7 \)
- \( b2 = 7 \)

Last Error: 1.261

Artificial Neural Network Design and Backpropagation Calculation

Gp. Capt. Thanapant Raicharoen, PhD
Example: Run_XOR_MLP_Newff.m

```matlab
% Run_XOR_MLP_Newff.m
P = [0 0 1 1; 0 1 0 1];  % XOR Function
T = [0 1 1 0]
plotpv(P,T,[-1, 2, -1, 2]); % plot data
PR = [min(P(1,:)) max(P(1,:));
     min(P(2,:)) max(P(2,:))];
S1 = 2;
S2 = 1;
TF1 = 'logsig';
TF2 = 'logsig';
PF = 'mse';

net = newff(PR,[S1 S2],[TF1 TF2]);
net.trainParam.epochs = 100;
net.trainParam.goal = 0.001;
net = train(net,P,T);
```
Example: Application of MLP for classification

Example: Run_MLP_Random.m
Matlab command : Create training data

\[ x = \text{randn([2 200])}; \]
\[ o = (x(1,:).^2+x(2,:).^2)<1; \]

Input pattern \( x_1 \) and \( x_2 \) generated from random numbers

Desired output \( o \):
if \( (x_1,x_2) \) lies in a circle of radius 1 centered at the origin
then
\[ o = 1 \]
else
\[ o = 0 \]
Example: Application of MLP for classification (cont.)

Matlab command: Create a 2-layer network

\[
PR = [\min(x(1,:)) \; \max(x(1,:)); \\
\min(x(2,:)) \; \max(x(2,:))]; \quad \text{Range of inputs}
\]

\[
S1 = 10; \quad S2 = 1; \quad \text{No. of nodes in Layers 1 and 2}
\]

\[
TF1 = 'logsig'; \quad \text{Activation functions of Layers 1 and 2}
\]

\[
TF2 = 'logsig';
\]

\[
BTF = 'traingd'; \quad \text{Training function}
\]

\[
BLF = 'learngd'; \quad \text{Learning function}
\]

\[
PF = 'mse'; \quad \text{Cost function}
\]

\[
net = \text{newff}(PR, [S1 \; S2], {TF1 \; TF2}, BTF, BLF, PF); \quad \text{Command for creating the network}
\]
Matlab command: Train the network

```matlab
net.trainParam.epochs = 2000; % No. of training rounds
net.trainParam.goal = 0.002; % Maximum desired error
net = train(net,x,o); % Training command
y = sim(net,x); % Compute network outputs (continuous)
netout = y>0.5; % Convert to binary outputs
```
Example: Application of MLP for classification (cont.)

Network structure

Input nodes

Hidden nodes (sigmoid)

Output node (Sigmoid)

Threshold unit (for binary output)
Initial weights of the hidden layer nodes (10 nodes) displayed as Lines $w_1x_1+w_2x_2+\theta = 0$
Example: Application of MLP for classification (cont.)

Training algorithm: Gradient descent method

MSE vs training epochs

Performance is 0.151511, Goal is 0.002
Example: Application of MLP for classification (cont.)

Results obtained using the Gradient descent method

Classification Error: 40/200
Example: Application of MLP for classification (cont.)

Training algorithm: Levenberg-Marquardt Backpropagation

MSE vs training epochs (success with in only 10 epochs!)
Results obtained using the Levenberg-Marquardt Backpropagation Method

Only 6 hidden nodes are adequate!

Classification Error: 0/200
Summary: MLP for Classification Problem

- Each lower layer (hidden) Nodes of Neural Network create a local boundary decision.

- The upper layer Nodes of Neural Network combine all local boundary decisions to a global boundary decision.
Example: Application of MLP for function approximation

Example: Run_MLP_SinFunction.m
Matlab command: Create a 2-layer network

\[
PR = [\min(x) \ \max(x)] \quad \{ \text{Range of inputs} \}
\]

\[
S1 = 6; \quad \{ \text{No. of nodes in Layers 1 and 2} \}
S2 = 1;
\]

\[
TF1 = 'logsig'; \quad \{ \text{Activation functions of Layers 1 and 2} \}
TF2 = 'purelin';
\]

\[
BTF = 'trainlm'; \quad \rightarrow \text{Training function}
BLF = 'learngd'; \quad \rightarrow \text{Learning function}
PF = 'mse'; \quad \rightarrow \text{Cost function}
\]

\[
\text{net} = \text{newff}(PR, [S1 \ S2], \{TF1 \ TF2\}, BTF, BLF, PF); \quad \rightarrow \text{Command for creating the network}
\]
Example: Application of MLP for function approximation

Network structure

Input nodes

Hidden nodes (sigmoid)

Output node (Linear)
Example: Application of MLP for function approximation

Example: Run_MLP_SinFunction.m

```matlab
% Run_MLP_SinFunction.m
p=0:0.25:5;
t = sin(p);
figure;
plot(p,t,'+b'); axis([-0.5 5.5 -1.5 1.5]);

net = newff([0 10],[6,1],{'logsig','purelin'},'trainlm');
net.trainParam.epochs = 50; net.trainParam.goal = 0.01;
net = train(net,p,t);
a = sim(net,p);
hold on;
plot(p,a,'.r');
```

...
Example: Application of MLP for function approximation

Matlab command: Create a 2-layer network

\[
\begin{align*}
PR &= [\text{min}(x) \text{ max}(x)] \quad \{\text{Range of inputs}\}
S1 &= 3; \\
S2 &= 1; \quad \{\text{No. of nodes in Layers 1 and 2}\}
TF1 &= \text{'logsig'}; \\
TF2 &= \text{'purelin'}; \quad \{\text{Activation functions of Layers 1 and 2}\}
BTF &= \text{'trainlm'}; \quad \rightarrow \text{Training function}
BLF &= \text{'learngd'}; \quad \rightarrow \text{Learning function}
PF &= \text{'mse'}; \quad \rightarrow \text{Cost function}
\end{align*}
\]

\[
\text{net} = \text{newff}(PR,[S1 \ S2],\{TF1 \ TF2\},BTF,BLF,PF); \quad \rightarrow \text{Command for creating the network}
\]
Example: Application of MLP for function approximation

Function to be approximated

\[
x = 0:0.01:4;
\]

\[
y = (\sin(2\pi x)+1) \cdot \exp(-x^2);
\]
Example: Application of MLP for function approximation

Network structure

No. of hidden nodes is too small!

Function approximated using the network
Example: Application of MLP for function approximation

Matlab command: Create a 2-layer network

\[
\text{PR} = [\text{min}(x) \ \text{max}(x)] \quad \{\text{Range of inputs}\}
\]

\[
S1 = 5; \quad \{\text{No. of nodes in Layers 1 and 2}\}
\]

\[
S2 = 1;
\]

\[
\text{TF1} = '\text{radbas}';\quad \{\text{Activation functions of Layers 1 and 2}\}
\]

\[
\text{TF2} = '\text{purelin}';
\]

\[
\text{BTF} = '\text{trainlm}'; \quad \rightarrow \text{Training function}
\]

\[
\text{BLF} = '\text{learngd}'; \quad \rightarrow \text{Learning function}
\]

\[
\text{PF} = '\text{mse}'; \quad \rightarrow \text{Cost function}
\]

\[
\text{net} = \text{newff}(\text{PR},[S1 \ S2],\{\text{TF1} \ \text{TF2}\},\text{BTF},\text{BLF},\text{PF});\quad \rightarrow \text{Command for creating the network}
\]
Example: Application of MLP for function approximation

Function approximated using the network
Summary: MLP for Function Approximation Problem

- Each lower layer (hidden) nodes of Neural Network create a local (short) approximated function.

- The upper layer Nodes of Neural Network combine all local approximated function to global approximated function cover all input range.
Summary

- **Backpropagation** can train multilayer feed-forward networks with differentiable transfer functions to perform function approximation, pattern association, and pattern classification.
- The term backpropagation refers to the process by which derivatives of network error, with respect to network weights and biases, can be computed.
- The number of inputs and outputs to the network are constrained by the problem. However, the number of layers between network inputs and the output layer and the sizes of the layers are up to the designer.
- The two-layer sigmoid/linear network can represent any functional relationship between inputs and outputs if the sigmoid layer has enough neurons.
Programming in MATLAB Exercise

- Exercise:

  1. Write MATLAB to solve the question 1 in Exercise 4.

  2. Write MATLAB to solve the question 2 in Exercise 4.