Machine Learning and Data Mining

Multi-layer Perceptrons & Neural Networks: Basics

Prof. Alexander Ihler
Linear Classifiers (Perceptrons)

- **Linear Classifiers**
  - a linear classifier is a mapping which partitions feature space using a linear function (a straight line, or a hyperplane)
  - separates the two classes using a straight line in feature space
  - in 2 dimensions the decision boundary is a straight line

Linearly separable data

Linearly non-separable data

(c) Alexander Ihler
Perceptron Classifier (2 features)

\[ f = w_1 x_1 + w_2 x_2 + w_0 \]

Decision Boundary at \( f(x) = 0 \)

Solve: \( X_2 = -\frac{w_1}{w_2} X_1 - \frac{w_0}{w_2} \)  (Line)

(c) Alexander Ihler
Perceptron (Linear classifier)

\[ f = w_1 x_1 + w_2 x_2 + w_0 \]

Weighted sum of the inputs

Threshold Function

\[ T(f) \]

Output = class decision

\[ \{0, 1\} \]

Decision boundary = “x such that \( T( w_1 x + w_0 ) \) transitions”

(c) Alexander Ihler
Features and perceptrons

• Recall the role of features
  – We can create extra features that allow more complex decision boundaries
  – Linear classifiers
  – Features \([1,x]\)
    • Decision rule: \(T(ax+b) = ax + b \geq 0\)
    • Boundary \(ax+b = 0\) => point
  – Features \([1,x,x^2]\)
    • Decision rule \(T(ax^2+bx+c)\)
    • Boundary \(ax^2+bx+c = 0\) = ?

• What features can produce this decision rule?
Features and perceptrons

- Recall the role of features
  - We can create extra features that allow more complex decision boundaries
  - For example, polynomial features
    \[ \Phi(x) = [1 \ x \ x^2 \ x^3 \ldots] \]

- What other kinds of features could we choose?
  - Step functions?

Linear function of features
\[ a \ F1 + b \ F2 + c \ F3 + d \]

Ex: \[ F1 - F2 + F3 \]
Multi-layer perceptron model

- Step functions are just perceptrons!
  - “Features” are outputs of a perceptron
  - Combination of features output of another

**Linear function of features:**
\[ a F_1 + b F_2 + c F_3 + d \]

**Example:**
\[ F_1 - F_2 + F_3 \]

\[ W^1 = \begin{bmatrix} w_{10} & w_{11} \\ w_{20} & w_{21} \\ w_{30} & w_{31} \end{bmatrix} \]

\[ W^2 = w_1 \ w_2 \ w_3 \]

(c) Alexander Ihler
Multi-layer perceptron model

- Step functions are just perceptrons!
  - “Features” are outputs of a perceptron
  - Combination of features output of another

\[ \text{Linear function of features: } \ a \ F1 + b \ F2 + c \ F3 + d \]

Ex: \( F1 - F2 + F3 \)

\[ W^1 = \begin{bmatrix} w_{10} & w_{11} \\ w_{20} & w_{21} \\ w_{30} & w_{31} \end{bmatrix} \]

\[ W^2 = w_1 \ w_2 \ w_3 \]

(c) Alexander Ihler
Features of MLPs

• Simple building blocks
  – Each element is just a perceptron \( f' n \)

• Can build upwards

(c) Alexander Ihler
Features of MLPs

- Simple building blocks
  - Each element is just a perceptron \( f' x \)

- Can build upwards

2-layer:
  
  “Features” are now partitions
  All linear combinations of those partitions

(c) Alexander Ihler
Features of MLPs

• Simple building blocks
  – Each element is just a perceptron $f^n$

• Can build upwards

3-layer:
“Features” are now complex functions
Output any linear combination of those
Features of MLPs

- Simple building blocks
  - Each element is just a perceptron $f^n$

- Can build upwards

Current research: “Deep” architectures (many layers)
Features of MLPs

- Simple building blocks
  - Each element is just a perceptron function

- Can build upwards

- Flexible function approximation
  - Approximate arbitrary functions with enough hidden nodes
Neural networks

- Another term for MLPs
- Biological motivation

- Neurons
  - “Simple” cells
  - Dendrites sense charge
  - Cell weighs inputs
  - “Fires” axon
## Activation functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Formula</th>
<th>Derivative Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logistic</td>
<td>$\sigma(z) = \frac{1}{1 + \exp(-z)}$</td>
<td>$\frac{\partial \sigma}{\partial z}(z) = \sigma(z)(1 - \sigma(z))$</td>
</tr>
<tr>
<td>Hyperbolic Tangent</td>
<td>$\sigma(z) = \frac{1 - \exp(-2z)}{1 + \exp(-2z)}$</td>
<td>$\frac{\partial \sigma}{\partial z}(z) = 1 - (\sigma(z))^2$</td>
</tr>
<tr>
<td>Gaussian</td>
<td>$\sigma(z) = \exp(-z^2/2)$</td>
<td>$\frac{\partial \sigma}{\partial z}(z) = -z\sigma(z)$</td>
</tr>
<tr>
<td>Linear</td>
<td>$\sigma(z) = z$</td>
<td>$\frac{\partial \sigma}{\partial z}(z) = 1$</td>
</tr>
</tbody>
</table>

And many others...
Feed-forward networks

- Information flows left-to-right
  - Input observed features
  - Compute hidden nodes (parallel)
  - Compute next layer…

```
X1 = _add1(X);  # add constant feature
T  = X1.dot(W[0].T);  # linear response
H  = Sig( T );  # activation f’n

H1 = _add1(H);  # add constant feature
S  = H1.dot(W[1].T);  # linear response
H2 = Sig( S );  # activation f’n

% ...  
```

- Alternative: recurrent NNs…
Feed-forward networks

A note on multiple outputs:

• Regression:
  – Predict multi-dimensional $y$
  – “Shared” representation
    = fewer parameters

• Classification
  – Predict binary vector
  – Multi-class classification
    $y = 2 = [0 \ 0 \ 1 \ 0 \ … \ ]$
  – Multiple, joint binary predictions
    (image tagging, etc.)
  – Often trained as regression (MSE),
    with saturating activation

(c) Alexander Ihler
Machine Learning and Data Mining

Multi-layer Perceptrons & Neural Networks: Backpropagation

Prof. Alexander Ihler
Training MLPs

- Observe features “x” with target “y”
- Push “x” through NN = output is “ŷ”
- Error: \((y - ŷ)^2\) (Can use different loss functions if desired...)
- How should we update the weights to improve?

- Single layer
  - Logistic sigmoid function
  - Smooth, differentiable

- Optimize using:
  - Batch gradient descent
  - Stochastic gradient descent
Backpropagation

- Just gradient descent...
- Apply the chain rule to the MLP

\[
\frac{\partial J}{\partial w_{k,j}^2} = -2 \sum_{k'} (y_{k'} - \hat{y}_{k'}) (\partial \hat{y}_{k'}) \\
= -2(y_k - \hat{y}_k) \sigma'(s_k) h_j
\]

(Identical to logistic mse regression with inputs “h_j”)

Forward pass

Loss function

\[
J_i(W) = \sum_k (y_k^{(i)} - \hat{y}_k^{(i)})^2
\]

Output layer

\[
\hat{y}_k = \sigma(s_k) = \sigma(\sum_j w_{k,j}^2 h_j)
\]

Hidden layer

\[
h_j = \sigma(t_j) = \sigma(\sum_i w_{j,i}^1 x_i)
\]
Backpropagation

- Just gradient descent...
- Apply the chain rule to the MLP

\[
\frac{\partial J}{\partial w^2_{kj}} = -2 \sum_{k'} (y_{k'} - \hat{y}_{k'}) \left( \frac{\partial \hat{y}_{k'}}{\partial h_j} \right)
\]
\[
= -2(y_k - \hat{y}_k) \sigma'(s_k) h_j
\]
\[
= \sum_k -2(y_k - \hat{y}_k) \sigma'(s_k) w^2_{kj} \partial h_j
\]
\[
= \sum_k -2(y_k - \hat{y}_k) \sigma'(s_k) w^2_{kj} \sigma'(t_j) x_i
\]

Forward pass

Loss function
\[
J_i(W) = \sum_k (y^{(i)}_k - \hat{y}^{(i)}_k)^2
\]
Output layer
\[
\hat{y}_k = \sigma(s_k) = \sigma(\sum_j w^2_{kj} h_j)
\]
Hidden layer
\[
h_j = \sigma(t_j) = \sigma(\sum_i w^1_{ji} x_i)
\]

(Identical to logistic mse regression with inputs “\(h_j\)”)
Backpropagation

• Just gradient descent...
• Apply the chain rule to the MLP

\[
\frac{\partial J}{\partial w_{k,j}^2} = -2(y_k - \hat{y}_k) \sigma'(s_k) h_j \\
\frac{\partial J}{\partial w_{ji}^1} = \sum_k -2(y_k - \hat{y}_k) \sigma'(s_k) w_{k,j}^2 \sigma'(t_j) x_i
\]

B2 = (Y-Yhat) * dSig(S)  #(1xN3)
G2 = B2.T.dot( H )  # (N3x1)*(1xN2)=(N3xN2)
B1 = B2.dot(W[1])*dSig(T)# (1xN3).*(N3*N2)*(1xN2)
G1 = B1.T.dot( X )  # (N2 x N1+1)

Forward pass

Loss function
\[
J_i(W) = \sum_k (y_k^{(i)} - \hat{y}_k^{(i)})^2
\]
Output layer
\[
\hat{y}_k = \sigma(s_k) = \sigma(\sum_j w_{k,j}^2 h_j)
\]
Hidden layer
\[
h_j = \sigma(t_j) = \sigma(\sum_i w_{ji}^1 x_i)
\]
Example: Regression, MCycle data

- Train NN model, 2 layer
  - 1 input features => 1 input units
  - 10 hidden units
  - 1 target => 1 output units
  - Logistic sigmoid activation for hidden layer, linear for output layer

Data:
+ learned prediction f’n:

Responses of hidden nodes (= features of linear regression): select out useful regions of “x”

(c) Alexander Ihler
Example: Classification, Iris data

- Train NN model, 2 layer
  - 2 input features => 2 input units
  - 10 hidden units
  - 3 classes => 3 output units \( (y = [0 \ 0 \ 1], \text{etc.}) \)
  - Logistic sigmoid activation functions
  - Optimize MSE of predictions using stochastic gradient

(c) Alexander Ihler
MLPs in practice

- Example: Deep belief nets (Hinton et al. 2007)
  - Handwriting recognition
  - Online demo
  - 784 pixels \(\Leftrightarrow 500 \text{ mid} \Leftrightarrow 500 \text{ high} \Leftrightarrow 2000 \text{ top} \Leftrightarrow 10 \text{ labels} \)
MLPs in practice

- Example: Deep belief nets (Hinton et al. 2007)
  - Handwriting recognition
  - Online demo
  - 784 pixels $\leftrightarrow$ 500 mid $\leftrightarrow$ 500 high $\leftrightarrow$ 2000 top $\leftrightarrow$ 10 labels
MLPs in practice

- Example: Deep belief nets (Hinton et al. 2007)
  - Handwriting recognition
  - Online demo
  - 784 pixels ⇔ 500 mid ⇔ 500 high ⇔ 2000 top ⇔ 10 labels

Fix output, simulate inputs
Neural networks & DBNs

• Want to try them out?
• Matlab “Deep Learning Toolbox”
  https://github.com/rasmusbergpalm/DeepLearnToolbox
  rasmusbergpalm / DeepLearnToolbox

  Matlab/Octave toolbox for deep learning. Includes Deep Belief Nets, Stacked Autoencoders, Convolutional Neural Nets, Convolutional Autoencoders and vanilla Neural Nets. Each method has examples to get you started.

• PyLearn2
  https://github.com/lisa-lab/pylearn2

• TensorFlow
Summary

• Neural networks, multi-layer perceptrons

• Cascade of simple perceptrons
  – Each just a linear classifier
  – Hidden units used to create new features

• Together, general function approximators
  – Enough hidden units (features) = any function
  – Can create nonlinear classifiers
  – Also used for function approximation, regression, …

• Training via backprop
  – Gradient descent; logistic; apply chain rule