Neural Networks

CAP5610 Machine Learning
Instructor: Guo-Jun Qi
Recap: linear classifier

• Logistic regression
  • Maximizing the posterior distribution of class Y conditional on the input vector X

• Support vector machines
  • Maximizing the maximum margin, and
    • Hard margin: subject to the constraints that no training error shall be made
    • Soft margin: minimizing the slack variables that represent how much an associated training example violates the classification rule.
  • Extended to nonlinear classifier with kernel trick
    • Mapping input vectors to high dimensional space
    • linear classifier in high dimensional space, nonlinear in original space.
Building nonlinear classifier

• With a network of logistic units?
• A single logistic unit is linear classifier: $f: X \rightarrow Y$

$$f(X) = \frac{1}{1 + \exp(-W_0 - \sum_{n=1}^{N} W_n X_n)}$$
Graph representation of a logistic unit

- Input layer: An input $X=(X_1, \ldots, X_n)$
- Output: logistic function of the input features
A logistic unit as an neuron:

• Input layer: An input $X = (X_1, \ldots, X_n)$
• Activation: weighted sum of input features $a = W_0 + \sum_{n=1}^{N} W_n X_n$
• Activation function: logistic function $h$ applied to the weighted sum
• Output: $z = h(a)$
Neural Network: Multiple layers of neurons

- Output of a layer is the input into the upper layer
An example

• A three layer neural network

\[
\begin{align*}
    a_1^{(1)} &= w_{11}^{(1)} x_1 + w_{12}^{(1)} x_2 + w_{10}^{(1)} \\
    z_1 &= h(a_1^{(1)}) \\
    a_2^{(1)} &= w_{21}^{(1)} x_1 + w_{22}^{(1)} x_2 + w_{20}^{(1)} \\
    z_2 &= h(a_2^{(1)}) \\
    a_1^{(2)} &= w_{11}^{(2)} z_1 + w_{12}^{(2)} x_2 + w_{10}^{(2)} \\
    y_1 &= f(a_1^{(2)})
\end{align*}
\]
XOR Problem

- It is impossible to linearly separate these two classes
XOR Problem

- Two classes become separable by putting a threshold 0.5 to the output $y_1$.
Application: Drive a car

• Input: real-time videos captured by a camera
• Output: signals that steer a car
  • From the sharp left, straight to sharp right
Training Neural Network

• Given a training set of $M$ examples $\{(x^{(i)}, t^{(i)}) | i=1, \ldots, M\}$

• Training neural network is equivalent to minimizing the least square error between the network output and the true value

$$\min_w L(w) = \frac{1}{2} \sum_{i=1}^{M} (y^{(i)} - t^{(i)})^2$$

Where $y^{(i)}$ is the output depending on the network parameters $w$. 
Recap: Gradient decent Method

• Gradient descent method is an iterative algorithm
  • hill climbing method to find the peak point of a “mountain”
  • At each point, compute its gradient
    \[
    \nabla L = \left[ \frac{\partial L}{\partial w_0}, \frac{\partial L}{\partial w_1}, \ldots, \frac{\partial L}{\partial w_N} \right] 
    \]
  • Gradient is a vector that points to the steepest direction climbing up the mountain.
  • At each point, \( w \) is updated so it moves a size of step \( \lambda \) in the gradient direction
    \[
    w \leftarrow w + \lambda \nabla L(w) 
    \]
Stochastic Gradient Ascent Method

- Making the learning algorithm scalable to big data
- Computing the gradient of square error for only one example

\[ L(w) = \sum_{i=1}^{M} (y^{(i)} - t^{(i)})^2 \]

\[ \nabla L = \left[ \frac{\partial L}{\partial w_0}, \frac{\partial L}{\partial w_1}, \ldots, \frac{\partial L}{\partial w_N} \right] \]

\[ L^{(i)}(w) = (y^{(i)} - t^{(i)})^2 \]

\[ \nabla L^{(i)} = \left[ \frac{\partial L^{(i)}}{\partial w_0}, \frac{\partial L^{(i)}}{\partial w_1}, \ldots, \frac{\partial L^{(i)}}{\partial w_N} \right] \]
Boiling down to computing the gradient

Square loss: \( L = \frac{1}{2} (y_k - t_k)^2 \)

\[ y_k = f(a_k), a_k^{(2)} = \sum_j w_{kj} z_j \]

Derivative to the activation in the second layer:

\[ \delta_k^{(2)} \triangleq \frac{\partial L}{\partial a_k^{(2)}} = (y_k - t_k) \frac{\partial y_k}{\partial a_k^{(2)}} = (y_k - t_k) f'(a_k^{(2)}) \]

Derivative to the parameter in the second layer:

\[ \frac{\partial L}{\partial w_{ki}^{(2)}} = \frac{\partial L}{\partial a_k^{(2)}} \frac{\partial a_k^{(2)}}{\partial w_{ki}^{(2)}} = \delta_k^{(2)} z_i \]
Boiling down to computing the gradient

• Computing the derivatives to the parameters in the first layer

Relation between activations of the first and second layers

\[ a^{(2)}_k = \sum_j w^{(2)}_{kj} h(a^{(1)}_j) \]

By chain rule:

\[ \frac{\partial L}{\partial a^{(1)}_j} = \sum_k \frac{\partial L}{\partial a^{(2)}_k} \frac{\partial a^{(2)}_k}{\partial a^{(1)}_j} \]

\[ = h'(a^{(1)}_j) \sum_k \delta^{(2)}_k w^{(2)}_{kj} \equiv \delta^{(1)}_j \]

The derivative to the parameter in the first layer:

\[ a^{(1)}_j = \sum_n w^{(1)}_{jn} x_n \]

\[ \frac{\partial L}{\partial w^{(1)}_{jn}} = \frac{\partial L}{\partial a^{(1)}_j} \frac{\partial a^{(1)}_j}{\partial w^{(1)}_{jn}} = \delta^{(1)}_j x_n \]
Summary: Back propagation

• For each training example \((x,y)\),
  
  • For each output unit \(k\)
    \[
    \delta_k^{(2)} = (y_k - t_k) f'(a_k^{(2)})
    \]
  
  • For each hidden unit \(j\)
    \[
    \delta_j^{(1)} = h'(a_j^{(1)}) \sum_k \delta_k^{(2)} w_{kj}^{(2)}
    \]
Summary: Back propagation (2)

• For each training example \((x,y)\),

  • For each weight \(w_{ki}^{(2)}\):
    \[
    \frac{\partial L}{\partial w_{ki}^{(2)}} = \delta_k^{(2)} z_i
    \]
    • Update
    \[
    w_{ki}^{(2)} \leftarrow w_{ki}^{(2)} - \delta_k^{(2)} z_i
    \]

  • For each weight \(w_{jn}^{(1)}\):
    \[
    \frac{\partial L}{\partial w_{jn}^{(1)}} = \delta_j^{(1)} x_n
    \]
    • Update
    \[
    w_{jn}^{(1)} \leftarrow w_{jn}^{(1)} - \delta_j^{(1)} x_n
    \]
Regularized Square Error

• Add a zero mean Gaussian prior on the weights $w_{ij}^{(l)} \sim N(0, \sigma^2)$

• MAP estimate of $w$

$$L^{(i)}(w) = \frac{1}{2} (y^{(i)} - t^{(i)})^2 + \frac{\gamma}{2} \sum (w_{ij}^{(l)})^2$$
Summary: Back propagation (2)

• For each training example \((x,y)\),
  
  • For each weight \(w_{ki}^{(2)}\):
    \[
    \frac{\partial L}{\partial w_{ki}^{(2)}} = \delta_{k}^{(2)} z_{i} + \gamma w_{ki}^{(2)}
    \]
    
    • Update
    \[
    w_{ki}^{(2)} \leftarrow w_{ki}^{(2)} - \delta_{k}^{(2)} z_{i} - \gamma w_{ki}^{(2)}
    \]
  
  • For each weight \(w_{jn}^{(1)}\):
    \[
    \frac{\partial L}{\partial w_{jn}^{(1)}} = \delta_{j}^{(1)} x_{n} + \gamma w_{jn}^{(1)}
    \]
    
    • Update
    \[
    w_{jn}^{(1)} \leftarrow w_{jn}^{(1)} - \delta_{j}^{(1)} x_{n} - \gamma w_{jn}^{(1)}
    \]
Multiple outputs encoding multiple classes

- MNIST: ten classes of digits
- Encoding multiple classes as multiple outputs:
  - An output variable is set to 1 if the corresponding class is positive for the example
  - Otherwise, the output is set to 0.
- The posterior probability of an example belonging to class $k$

$$P(\text{Class}_k|x) = \frac{y_k}{\sum_{k'=1}^{K} y_{k'}}$$
Overfitting

- Tuning the number of update iterations on validation set
How expressive is NN?

• Boolean functions:
  • Every Boolean function can be represented by network with single hidden layer
  • But might require exponential number of hidden units

• Continuous functions:
  • Every bounded continuous function can be approximated with arbitrarily small error by neural network with one hidden layer
  • Any function can be approximated to arbitrary accuracy by a network with two hidden layers
Learning feature representation by neural networks

• A compact representation for high dimensional input vectors
  • A large image with thousands of pixels
  • High dimensional input vectors
    • might cause curse of dimensionality
      • Needs more examples for training (in lecture 1)
    • Not well capture the intrinsic variations
      • an arbitrary point in a high dimensional space probably does not represent a valid real object.
  • A meaningful low dimensional space is preferred!
Autoencoder

• Set output to input
• Hidden layers as feature representation, since it contains sufficient information to reconstruct the input in the output layer
An example

<table>
<thead>
<tr>
<th>Input Values</th>
<th>Hidden Values</th>
<th>Output Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000000</td>
<td>.89 .04 .08</td>
<td>10000000</td>
</tr>
<tr>
<td>01000000</td>
<td>.01 .11 .88</td>
<td>01000000</td>
</tr>
<tr>
<td>00100000</td>
<td>.01 .97 .27</td>
<td>00100000</td>
</tr>
<tr>
<td>00010000</td>
<td>.99 .97 .71</td>
<td>00010000</td>
</tr>
<tr>
<td>00001000</td>
<td>.03 .05 .02</td>
<td>00001000</td>
</tr>
<tr>
<td>00000100</td>
<td>.22 .99 .99</td>
<td>00000100</td>
</tr>
<tr>
<td>00000010</td>
<td>.80 .01 .98</td>
<td>00000010</td>
</tr>
<tr>
<td>00000001</td>
<td>.60 .94 .01</td>
<td>00000001</td>
</tr>
</tbody>
</table>

Sum of squared errors for each output unit.
Deep Learning: A Deep Feature Representation

• If you build multiple layers to reconstruct the input at the output layer
Summary

• Neural Networks: Multiple layers of neurons
  • Each upper layer neuron encodes weighted sum of inputs from the other neurons at a lower layer by an activation function

• BP training: a stochastic gradient descent method
  • From the output layer down to the hidden and input layers

• Autoencoder: feature representation