Neural Networks

COMP 135 Intro to Machine Learning
Liping Liu

slides modified from Roni Khardon’s with permission

Real Neural Networks

Drawing of neurons in the pigeon cerebellum, by Spanish neuroscientist Santiago Ramón y Cajal in 1899.

Diagram of a typical myelinated vertebrate motor neuron

From Wikipedia

A neuron

Simplified plot of a neuron cell

Linear Sigmoid Units

• Signal in: \( x = \{ x_j : j = 1, ..., d \} \)
• Signal out: \( \hat{y} \)
\[
\hat{y} = \sigma(w^T x) = \sigma \left( \sum_{j=1}^{d} w_j x_j \right)
\]

- This conveniently satisfies
\[
\sigma'(a) = \frac{-\sigma(a)^2}{(1-\sigma(a)^2)} = \sigma(a)(1 - \sigma(a))
\]

• Other “activation functions” \( \sigma(a) \) later.
• To emphasize the generality we will refer to any activation function \( \sigma(a) \)

Multi-Layer Networks

• Stack linear sigmoid units to get multilayer networks
Multi-Layer Networks

- Write down in math ...

\[ h^0 = x \]
\[ h^\ell = \sigma(W^\ell h^{\ell-1}), \ell = 1, ..., L \]
\[ \hat{y} = h^L \]

\( \sigma() \) is applied to vector element-wise

Activation function of the last layer can be adapted to real application.

Question: what activation function to use for binary classification and regression?

An Example on XOR problem

\[ z = \text{XOR}(x, y) \]

\[ \text{Output} \]

\[ \text{Input} \]

\( \sigma() = [a \geq t_x] \) for red circles
\( t_x \) is the number in the red circle
\( \sigma() = a \) for blue circle

From Wikipedia (feed forward network)

NN as Function Approximator

Theorem 10 (Two-Layer Networks are Universal Function Approximators). Let \( F \) be a continuous function on a bounded subset of \( \mathbb{D} \)-dimensional space. Then there exists a two-layer neural network \( \hat{F} \) with a finite number of hidden units that approximate \( F \) arbitrarily well. Namely, for all \( x \) in the domain of \( F \), \( |\hat{F}(x) - F(x)| < \epsilon \).

Learn MLN – Back Propagation

- Regression

\[ \min_{W,\ell=1, \ldots, L} \text{error } E = \frac{1}{Z} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2 \]

Note: \( \hat{y}_i \) is a function of connecting weights

- Classification
  - What objective to use?

Learn MLN – Back Propagation

- Back propagation
  - back propagation = gradient descent + chain rule

- Gradient update

\[ W^\ell = W^\ell - \eta \frac{\partial E}{\partial W^\ell} \]

\( \eta \) is step length

Learn MLN – Gradient Calculation

- Two layer network

\[ h = \sigma(W^1 x), \]
\[ \hat{y} = \sigma(W^2 h) \]

- Gradient

\[ \frac{\partial E}{\partial w_{jk}} = \frac{\partial E}{\partial y} \cdot \frac{\partial y}{\partial W^2} = (\hat{y} - y) \cdot \sigma'(W^2 h) \cdot h \]

\[ \frac{\partial E}{\partial w_{j}} = \sum_{k=1,...,K} \left( \frac{\partial E}{\partial w_{jk}} \right) \]

\[ \frac{\partial E}{\partial h} = \left( (\hat{y} - y) \cdot \sigma'((W^2)^T h) \cdot w^2 \right) \cdot \left( \sigma'(W^2 h) \cdot x \right) \]
Learn MLN – Gradient Calculation

- Multiple layers
  \[ h^l = W^l h^{l-1} \]
  \[ \hat{y} = h^L \]
- Gradient
  Enough if we can calculate
  \[ \frac{\partial E}{\partial x} = \frac{\partial E}{\partial h^L} \frac{\partial h^L}{\partial x} = \frac{\partial E}{\partial h^L} \frac{\partial h^L}{\partial W^L} \frac{\partial W^L}{\partial x} = \frac{\partial E}{\partial h^L} \left( \sigma'(h_{L-1}) \cdot W^L \right) \]

Learn MLN – Back Propagation

- Train the neural network with gradient descent method
  - Not easy to optimize; the error surface has local minima & saddle points
  - Solution 1: Momentum:
    \[ W^l = W^l - \frac{\partial E}{\partial W^l} + \alpha \times \text{previous update} \]
  - Solution 2: Use multiple restarts and pick one with lowest training set error
  - … many more recent techniques

Multiple Output Nodes

- All outputs share the same hidden layers
- Network identifies representations that are useful for all outputs
- Exactly same algorithm applies replacing the error on \( y \) with summation of errors on all units
- Forward pass identical
- Backward pass: back-propagate error from each output unit

What does the hidden layer do?

- Example: self-encoders

Learned hidden layer representation:

<table>
<thead>
<tr>
<th>Input</th>
<th>Hidden</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0000</td>
<td>0.04</td>
<td>0.08</td>
</tr>
<tr>
<td>0.011</td>
<td>0.11</td>
<td>0.58</td>
</tr>
<tr>
<td>0.022</td>
<td>0.35</td>
<td>0.02</td>
</tr>
<tr>
<td>0.033</td>
<td>0.07</td>
<td>0.21</td>
</tr>
<tr>
<td>0.044</td>
<td>0.08</td>
<td>0.18</td>
</tr>
<tr>
<td>0.055</td>
<td>0.09</td>
<td>0.08</td>
</tr>
<tr>
<td>0.066</td>
<td>0.04</td>
<td>0.04</td>
</tr>
</tbody>
</table>

Images from Mitchell's textbook
What does the hidden layer do?

[Images from Mitchell’s textbook]

Other Activation Functions

- hyperbolic tangent
  \[ f(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)} \]

- Softplus function
  \[ f(x) = \log(1 + \exp(x)) \]

- ReLU (rectified linear unit)
  \[ f(x) = \max(0, x) \]

Practical issues – Initialization

- Starting values
  - Start with small random values instead of zeros

- Scaling inputs
  - Better to standardize all inputs to have mean zero and standard deviation 1

Practical Issues – Network Structure

- How to pick network size (and shape)?
- Similar to model selection in other models
  - cross validation
  - Combine fit + penalty

- How many updates?
  - Large number of updates \( \rightarrow \) often overfit
  - Often can handle large network by limiting number of updates
Practical Issues - Overfitting

• Early stopping
  - Stop training before training converges
• Regularization
  - Add regularization to weights

Neural Networks

• Renewed interest in Deep Networks in last decade
• Several schemes for special network structure and special node functions
• Several schemes for training
• Combination of these ideas with BigData yields impressive improvements in performance in vision, NLP and other applications