Lecture 12: Learning Architectures

AI For Traditional Games
Prof. Nathan Sturtevant
Winter 2011

UNIVERSITY OF DENVER

Learning Intro

- Lots of types of learning
  - We focus on two of many learning architectures
  - Will provide the basis for learning as used in:
    - Backgammon
    - Othello
    - Go
    - Hearts

Learning possibilities:

- Could learn a classifier
  - Usually a binary decision:
    - Is or is not in a particular group
- Could learn an evaluation function
  - Predict the outcome of a game
  - Real-valued output
- We will focus on evaluation functions

Learning types

- Online:
  - Agent is updating beliefs while acting the world
- Offline:
  - Agent trains and learns completely separately from test scenarios

- Regret was both online (ucb) and offline (CFR) learning
- Focus on offline learning

UNIVERSITY OF DENVER

Learning: input & output

- Input:
  - Could be the full state of the game
    - Corresponds to solving a game (too large)
  - Instead, choose set of representative features
    - Choosing features is an art
    - During training: target value
- Output:
  - Evaluation of the current game state

UNIVERSITY OF DENVER

Simple learning model: the linear perceptron

- Supervised learning
- Given:
  - Inputs **x**: $x_0 \ldots x_n$ [x_0 fixed at 1]
  - Weights **w**: $w_0 \ldots w_n$
- Output is:
  - $xw^t = \sum x_i w_i$
  - Classify as true if output >= 0

UNIVERSITY OF DENVER

Linear Perceptron learning:

- Gradient descent: move in the direction that reduces the error (minimize squared error)
  - $E = 1/2\ Err^2$ (by definition for least-squared error)
  - Err = (y - **wx**) (y is the target output)
  - Take the partial derivative to get the gradient with respect to a single weight
  - $\partial E/\partial w_i = \partial/\partial w_i\ (1/2\ Err^2) = Err \cdot \partial/\partial w_i\ (Err) =$

    $Err \cdot \partial/\partial w_i\ (y - \mathbf{wx}) = Err \cdot (-\partial/\partial w_i\ \mathbf{wx}) = -Err\ x_i =$

    $-(y - \mathbf{wx})\ x_i$

UNIVERSITY OF DENVER

Linear perceptron learning

- We have assumed a linear model
  - Generic model has g(**wx**) *activation* function
    - More on this in neural networks
  - This would add a g'(**wx**) term to gradient

UNIVERSITY OF DENVER

Linear Perceptron Learning

- Instead of following the 'full' gradient, just update the weights one step in that direction
  - $w_i = w_i + \alpha (y - \mathbf{wx}) x_i$
  - for $0 < \alpha < 1$
- Try this with 2 inputs and the AND function

UNIVERSITY OF DENVER

Linear Perceptron notes

- Corresponds to a hyper-plane in *d* dimensions
- In two dimensions, the two weights on the inputs control the slope
  - The bias term is needed to move the line off the origin
- Perceptron can only learn linearly separable functions as a classifier
  - This is a key "limitation" to what perceptrons can and cannot do
- This is "linear regression"

UNIVERSITY OF DENVER

Slightly Better Perceptron / Logistic Regression

- Given:
  - Inputs **x**: $x_0 \ldots x_n$ [x_0 fixed at 1]
  - Weights **w**: $w_0 \ldots w_n$
- Output is:
  - $f(xw^t) = f(\sum x_i w_i)$ where
    - $f(x) = 1/(1+e^{-x})$
  - Classify as true if output >= 0

UNIVERSITY OF DENVER

Logistic Regression

- Has the same drawbacks in terms of representing linear functions (as a classifier)
  - Stronger interpretation as probabilities
- Slightly different gradient descent
  - $\partial E/\partial w_i = \partial/\partial w_i \, (1/2 \, Err^2) = Err \cdot \partial/\partial w_i \, (Err) =$

    $Err \cdot \partial/\partial w_i \, (y - f(\mathbf{wx})) = Err \cdot (- \partial/\partial w_i \, f(\mathbf{wx})) =$

    $-Err \, f'(\mathbf{wx}) \, x_i = - (y - \mathbf{wx}) \, f(\mathbf{wx})(1-f(\mathbf{wx})) \, x_i$

UNIVERSITY OF DENVER

Derivative of Logistic Function

- $f(x) = 1/(1+e^{-x}) = (1+e^{-x})^{-1}$
- $f'(x) = -1\ (1+e^{-x})^{-2}\ \partial/\partial x\ (1+e^{-x})$

  $= -1\ (1+e^{-x})^{-2}\ e^{-x}\ (-1) = e^{-x} / (1+e^{-x})^2 = a / (1+a)^2\ [a = e^{-x}]$

  $= (1+a-1)/(1+a)^2 = (1+a)/(1+a)^2 - 1/(1+a)^2$

  $= 1/(1+a)(1-1/(1+a)) = f(x)(1 - f(x))$

UNIVERSITY OF DENVER

Neural Networks

- Add an additional layer onto the network
  - Given enough hidden units can represent *any* function
  - Learning and convergence is a bit tougher
    - Some functions converge easier than others
    - Depends on features

UNIVERSITY OF DENVER

Neural networks - learning

- The learning gets a bit more complicated
  - The learning for the output units stays the same
  - However, inputs are now hidden units
- Hidden units feed into all output units
  - Must propagate errors from all output units
- Also allow for multiple output units
  - In perceptrons output are completely independent

UNIVERSITY OF DENVER

Neural Networks

- Gradient descent used an error term:
  - $\partial E/\partial w_i = -\ (y - \mathbf{wx})\ f(\mathbf{wx})(1-f(\mathbf{wx}))\ x_i = -\ \Delta\ x_i$
  - Now, replace the error $(y - \mathbf{wx})$ with the combined error over all output units
  - $f(h(\mathbf{wx}))(1-f(h(\mathbf{wx})))\ \sum w_{ki}\Delta_i$
  - where h(x) is the output of the hidden unit
- The (hidden) update rule becomes: $w_i = w_i + \alpha\ \ x_i$
- The (output) update rule replaces x_i with h(x)