Machine Learning Intro

CompSci 370
Ronald Parr
Department of Computer Science
Duke University

Why Study Learning?

• Considered a hallmark of intelligence

• Viewed as way to reduce programming burden
  – Not enough programmers in the world to produce custom solutions to all problems – even if we knew how
  – Programmers are expensive!

• Many algorithms assume parameters that are difficult to determine exactly a priori
  – What is the right formula to filter spam?
  – When should your smart thermostat turn on the heat?
Examples

• SPAM classification
• Computational Biology/medicine
  – Distinguish healthy/diseased tissue (e.g., skin/colon cancer)
  – Find structure in biological data (regulatory pathways)
• Financial events
  – Predict good/bad credit risks
  – Predict price changes
  – Response to marketing
• Object/person recognition
• Natural language processing
• Document categorization and user preferences
• Recommend products to users
• Learn to play games, e.g., go, chess, etc.
• Learn to control systems, e.g., robots or helicopters
• Public database of (old) benchmark learning problems:

What is Machine Learning?

• Learning Element
  – The thing that learns

• Performance Element
  – Objective measure of progress

• Learning is simply an increase in the ability of the learning element over time (with data) to achieve the task specified by the performance element
ML vs. Statistics?

- Machine learning is:
  - Younger
  - More empirical
  - More algorithmic
  - (arguably) More practical
  - (arguably) More decision theoretic

- Statistics is:
  - More mature
  - (arguably) More formal and rigorous

Look at this cool result! Maybe somebody can explain why it works later?

Let’s model this situation and prove that we converge to a consistent answer!

ML vs. Data Mining

- Machine Learning is:
  - (Arguably) more formal
  - (Arguably) more task driven/decision theoretic

- Data Mining is:
  - More constrained by size of data set
  - More closely tied to database techniques
Feedback in Learning

- **Supervised Learning**
  - Given examples of correct behavior
  - Example input: Labeled x-rays
  - Example use: Cancer diagnosis

- **Unsupervised Learning**
  - No external notion of what is correct
  - Example: Unlabeled x-rays
  - Example use: Clustering based on appearance

- **Reinforcement Learning**
  - Indirect indication of effectiveness
  - Example use: PacMan, go, chess

Learning Methodology

- Distinction between training and testing is crucial

- Correct performance on training set is just memorization!

- Researcher should *never* look at the test data (but in practice always does)

- Raises issues for “benchmark” learning problems
Types of Supervised Learning

- **Training input:**
  - Feature vector for each datum: $x_1 \ldots x_n$
  - Target value: $y$

- **Classification** – assigning labels/classes
- **Regression** – assigning real numbers

Features and Targets

- **Features** can be anything
  - Images, sounds, text
  - Real values (height, weight)
  - Integers, or binaries

- **Targets can be discrete classes:**
  - Safe mushrooms vs. poisonous
  - Malignant vs. benign
  - Good credit risk vs. bad
  - Label of image

- **Or numbers**
  - Selling price of house
  - Life expectancy
How Most Supervised Learning Algorithms Work

• Main idea: Minimize error on training set
• How this is done depends on:
  – Hypothesis space
  – Type of data

• Big Question: What is the “right” hypothesis space?

What is the Best Choice of Polynomial?

Noisy Source Data
Observations

- Degree 3 is the best match to the source
- Degree 9 is the best match to the samples
- We call this **over-fitting**
- Performance on test data:

What went wrong?

- Is the problem a bad choice of polynomial?
- Is the problem that we don’t have enough data?
- Answer: Yes
How to pick our hypothesis space?

• Learning theory (a rich subarea) gives some guidance on this, though it is often more abstract than directly applicable to real world applications

• Practical approaches:
  – Regularizer or prior to trade off training set error vs. hypothesis space complexity
  – Cross validation uses one or more mini test sets to help inform hypothesis space selection

Classification vs. Regression

• Regression tries to hit the target values with the function we are fitting

• Classification tries to find a function that separates the classes
Decision Boundaries

- A classifier can be viewed as partitioning the input space or feature space $X$ into decision regions.

- A linear threshold unit always produces a linear decision boundary. A set of points that can be separated by a linear decision boundary is linearly separable.

What can be expressed?

- Examples of things that can be expressed (Assume $n$ Boolean $0/1$ features)
  - Conjunctions:
    - $x_1 \land x_3 \land x_4$: $1 \cdot x_1 + 0 \cdot x_2 + 1 \cdot x_3 + 1 \cdot x_4 \geq 3$
    - $x_1 \land \lnot x_3 \land x_4$: $1 \cdot x_1 + 0 \cdot x_2 + 1 \cdot x_3 + 1 \cdot x_4 \geq 2$
  - at-least-m-of-n
    - at-least-2-of$(x_1, x_2, x_4)$
      - $1 \cdot x_1 + 1 \cdot x_2 + 0 \cdot x_3 + 1 \cdot x_4 \geq 2$

- Examples of things that cannot be expressed:
  - Non-trivial disjunctions:
    - $(x_1 \land x_3) + (x_3 \land x_4)$
  - Exclusive-Or
    - $(x_1 \land \lnot x_2) + (\lnot x_1 \land x_2)$
Limitations of Linearly Separable Functions

Is red linearly separable from green?
Are the circles linearly separable from the squares?

Feature Engineering

• All data are represented in “feature space” - the space spanned by all possible values of all features
• Feature space is largely a choice, like the degree of your polynomial, i.e., feature space engineering = hypothesis space engineering
• If you don’t like your performance, you can change your feature space – but don’t forget peril of overfitting
Suppose we’re in 1-dimension

Easy to find a linear separator

Harder 1-dimensional dataset

What can be done about this?
Remember how permitting non-linear features (higher degree polynomials) made linear regression so much more powerful?

Let’s permit them here too

\[ \Phi = (x, x^2) \]

Now linearly separable in the new feature space

But, what if the right feature set isn’t obvious

\[ \Phi = (x, x^2) \]
Motivation for non-linear Classifiers

• Linear methods are “weak”
  – Make strong assumptions
  – Can only express relatively simple functions of inputs

• Coming up with good features can be hard
  – Requires human input
  – Knowledge of the domain

• Role of neural networks
  – Neural networks started as linear models of single neurons
  – Combining ultimately led to non-linear functions that don’t necessarily need careful feature engineering

Neural Network Motivation

• Human brains are only known example of actual intelligence
• Individual neurons are slow, boring
• Brains succeed by using massive parallelism
• Idea: Copy what works

• Raises many issues:
  – Is the computational metaphor suited to the computational hardware?
  – How do we know if we are copying the important part?
  – Are we aiming too low?
Why Neural Networks?

Maybe computers should be more brain-like:

<table>
<thead>
<tr>
<th></th>
<th>Computers</th>
<th>Brains</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computational Units</td>
<td>$10^{10}$ transistors/CPU</td>
<td>$10^{11}$ neurons/brain</td>
</tr>
<tr>
<td>Storage Units</td>
<td>$10^{11}$ bits RAM</td>
<td>$10^{11}$ neurons</td>
</tr>
<tr>
<td></td>
<td>$10^{13}$ bits HD</td>
<td>$10^{14}$ synapses</td>
</tr>
<tr>
<td>Cycle Time</td>
<td>$10^{-9}$ S</td>
<td>$10^{-3}$ S</td>
</tr>
<tr>
<td>Bandwidth</td>
<td>$10^{10}$ bits/s*</td>
<td>$10^{14}$ bits/s</td>
</tr>
<tr>
<td>Compute Power</td>
<td>$10^{10}$ Ops/s</td>
<td>$10^{14}$ Ops/s</td>
</tr>
</tbody>
</table>

Comments on Summit

(world’s fastest supercomputer as of 10/19)

- 149 Petaflops
- $\sim 10^{18}$ Ops/s (Summit) vs. $10^{24}$ Ops/s (brain)
- 2.4M cores (conflicting reports)
- 2.8 PB RAM ($10^{17}$ bits)
- 10 Megawatts power($\sim $10M/year in electricity [my estimate])
- $\sim $200M cost

Note: recently surpassed by Fugaku – 3x more cores, 3x more energy, 3x performance, 5x cost
Fugaku expected to replaced by Frontier this year, 2x Fugaku performance, same energy, 60% cost
More Comments on Summit

- What is wrong with this picture?
  - Weight
  - Size
  - Power Consumption

- What is missing?
  - Still can’t replicate human abilities
    (though vastly exceeds human abilities in many areas)
  - Are we running the wrong programs?
  - Is the architecture well suited to the programs we might need to run?

Artificial Neural Networks

- Develop *abstraction* of function of actual neurons

- Simulate large, massively parallel artificial neural networks on conventional computers – note that even supercomputers have very low connectivity compared to a brain

- Some have tried to build the hardware too

- Try to approximate human learning, robustness to noise, robustness to damage, etc.
Neural Network Lore

• Neural nets have been adopted with an almost religious fervor within the AI community – several times
  – First coming: Perceptron
  – Second coming: Multilayer networks
  – Third coming (present): Deep networks

• Sound science behind neural networks: gradient descent
• Unsound social phenomenon behind neural networks: HYPE!

Artificial Neurons

\[ a_i = h(\sum w_{ji}x_j) \]

h can be any function, but usually a smoothed step function
Threshold Functions

- $h(x) = \tanh(x)$ or $\frac{1}{1+\exp(-x)}$ (logistic sigmoid)
- $h(x) = \text{sgn}(x)$ (perceptron)

Feedforward Networks

- We consider acyclic networks
- One or more computational layers
- Entire network can be viewed as computing a complicated non-linear function
- Typical uses in learning:
  - Classification (usually involving complex patterns)
  - General continuous function approximation
- Many other variations possible
Special Case: Perceptron

h is a simple step function (sgn)

Perceptron is a Linear Classifier

\[ y = f(x, w) = h \left( \sum_{i=1}^{n} w_i x_i \right) \]
Good News/Bad News

• Good news
  – *Perceptron learning rule* can learn to distinguish any two classes that are linearly separable
  – *If* classes are separable, perceptron learning rule will converge for any learning rate

• Bad news
  – Linear separability is a strong assumption
  – Failure to appreciate this led to excessive optimism and first neural network crash

Multilayer Networks

• Once people realized how simple perceptrons were, they lost interest in neural networks for a while
• Multilayer networks turn out to be much more expressive (with a smoothed step function)
  – Use sigmoid, e.g., \( h = \tanh(w^T x) \) or logistic sigmoid
  – With 2 layers, can represent any continuous function
  – With 3 layers, can represent many discontinuous functions

• Tricky part: How to adjust the weights

Play with it at: http://playground.tensorflow.org
Smoothing Things Out

- Idea: Do gradient descent on a smooth error function
- Error function is sum of squared errors
- Consider a single training example first

\[ E = 0.5 \text{error}(x^{(i)}, w)^2 \]

\[
\frac{\partial E}{\partial w_j} = \frac{\partial E}{\partial a_j} \frac{\partial a_j}{\partial w_j}
\]

\[
\frac{\partial E}{\partial a_j} = \delta_j \quad \text{Notation}
\]

\[
\frac{\partial a_j}{\partial w_j} = z_i \quad \text{Calculus}
\]

\[
\frac{\partial E}{\partial w_j} = \delta_j z_i
\]

Calculus Reminder

- Chain rule for one variable: \( \frac{\partial f \circ g}{\partial x} = \frac{\partial f}{\partial g} \frac{\partial g}{\partial x} \)
- Chain rule for: \( f : \mathbb{R}^n \rightarrow \mathbb{R}^k, g : \mathbb{R}^m \rightarrow \mathbb{R}^n \)

\( J_x (f \circ g) = J_{g(x)} (f) J_x (g) = (k \times n)(n \times m) \)

- For \( k=1, m=1 \)

\( J_x (f \circ g) = \sum_{i=1}^{n} \frac{\partial f}{\partial g(x)_i} \frac{\partial g(x)_i}{\partial x} \)
Propagating Errors

- For output units (assuming no weights on outputs)

\[
\frac{\partial E}{\partial a_j} = \delta_j = y - t \\
a_j = \sum_i w_i z_i
\]

- For hidden units

\[
\frac{\partial E}{\partial a_i} = \delta_i = \sum_k \frac{\partial E}{\partial a_k} \frac{\partial a_k}{\partial a_i} = \sum_k \frac{\partial E}{\partial a_k} w_{ik} \frac{\partial h_j}{\partial a_i} = h'(a_i) \sum_k w_{ik} \delta_k
\]

All upstream nodes from i  
Error gradient of upstream nodes

Differentiating h

- Recall the logistic sigmoid:

\[
h(x) = \frac{e^x}{1 + e^x} = \frac{1}{1 + e^{-x}} \]

\[
1 - h(x) = \frac{e^{-x}}{1 + e^{-x}} = \frac{1}{1 + e^x}
\]

- Differentiating:

\[
h'(x) = \frac{e^{-x}}{(1 + e^{-x})^2} = \frac{1}{(1 + e^{-x})} \cdot \frac{e^{-x}}{(1 + e^{-x})} = h(x)(1 - h(x))
\]
Putting it together

• Apply input $x$ to network (sum for multiple inputs)
  – Compute all activation levels
  – Compute final output (forward pass)
• Compute $\delta$ for output units
  \[ \delta = y - t \]
• Backpropagate $\delta$s to hidden units
  \[ \delta_j = \sum_k \frac{\partial E}{\partial a_k} \frac{\partial a_k}{\partial a_j} = h'(a_j) \sum_k w_k \delta_k \]
• Compute gradient update:
  \[ \frac{\partial E}{\partial w_{ij}} = \delta_j a_i \]

Summary of Gradient Update

• Gradient calculation, parameter updates have recursive formulation
• Decomposes into:
  – Local message passing
  – No transcendental:
    • $h'(x)=1-h(x)^2$ for tanh$(x)$
    • $H'(x)=h(x)(1-h(x))$ for logistic sigmoid
• Highly parallelizable
• Biologically plausible(?)

• Celebrated backpropagation algorithm
Propagate forward, computing activation levels, outputs to next layer

Compute the output of the final layer
Compute the error (\( \delta \)) for the final layer

\[
\frac{\partial E}{\partial w_{ij}} = \delta_j a_i
\]

Compute the error \( \delta \)'s and gradient updates for earlier layers:

\[
\frac{\partial E}{\partial w_{ij}} = \delta_i a_j
\]
Complete training for one datum – now repeat for entire training set

Good News

- Can represent any continuous function with two layers (1 hidden)
- Can represent essentially any function with 3 layers
- (But how many hidden nodes?)

- Multilayer nets are a universal approximation architecture with a highly parallelizable training algorithm
Early Successes of Multilayer Nets

• Trained to pronounce English
  – Training set: Sliding window over text, sounds
  – 95% accuracy on training set
  – 78% accuracy on test set
• Trained to recognize handwritten digits
  – >99% accuracy
• Trained to drive (Pomerleau et al. no-hands across America 1995)

Backprop Issues

• Backprop = gradient descent on an error function
• Function is nonlinear (= powerful)
• Function is nonlinear (= local minima)
• Big nets:
  – Many parameters
  • Many optima
  • Slow gradient descent
  • Risk of overfitting
  – Biological plausibility ≠ Electronic plausibility
• Many NN experts became experts in numerical analysis (by necessity)
NN History Through the Second Coming

- Second wave of interest in neural networks lost research momentum in the 1990s – though still continued to enjoy many practical applications
- Neural network tricks were not sufficient to overcome competing methods:
  - Support vector machines
  - Clever feature selection methods wrapped around simple or linear methods
- 2000-2010 was an era of linear + special sauce
- What changed?

Deep Networks

- Not a learning algorithm, but a family of techniques
  - Improved training techniques (though still essentially gradient descent)
  - Clever crafting of network structure – convolutional nets
  - Some new activation functions

- Exploit massive computational power
  - Parallel computing
  - GPU computing
  - Very large data sets (can reduce overfitting)
Deep Networks Today

- Still on the upward swing of the hype pendulum
- State of the art performance for many tasks:
  - Speech recognition
  - Object recognition
  - Playing video games
- Controversial but increasingly accepted in practice:
  - Hype, hype, hype! (but it really does work well in many cases!)
  - Theory lags practice
  - Collection of tricks, not an entirely a science yet
  - Results are not human-interpretable

Conclusions

- Supervised learning = successful way to take training (input, output pairs) and induce functions that generalize to test data drawn from the same distribution as the training data.

- Methods for learning linear functions are well understood and perform well with good features

- Non-linear methods, such as neural networks are more powerful and require less feature engineering but are more computationally expensive and less predictable in practice
  - Historically wild swings in popularity
  - Currently on upswing due to clever changes in training methods, use of parallel computation, and large data sets