Text Categorization and Neural Network Learning
Categorization

• Given:
  – A description of an instance, \(x \in X\), where \(X\) is the *instance language* or *instance space*.
  – A fixed set of categories:
    \[C = \{c_1, c_2, \ldots, c_n\}\]

• Determine:
  – The category of \(x\): \(c(x) \in C\), where \(c(x)\) is a categorization function whose domain is \(X\) and whose range is \(C\).
Learning for Categorization

- A training example is an instance $x \in X$, paired with its correct category $c(x)$: $<x, c(x)>$ for an unknown categorization function, $c$.
- Given a set of training examples, $D$.
- Find a hypothesized categorization function, $h(x)$, such that:

$$\forall <x, c(x)> \in D : h(x) = c(x)$$

*Consistency*
Sample Category Learning Problem

- Instance language: \(<\text{size}, \text{color}, \text{shape}>\)
  - size $\in \{\text{small, medium, large}\}$
  - color $\in \{\text{red, blue, green}\}$
  - shape $\in \{\text{square, circle, triangle}\}$
- $C = \{\text{positive, negative}\}$

- $D$:

<table>
<thead>
<tr>
<th>Example</th>
<th>Size</th>
<th>Color</th>
<th>Shape</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>small</td>
<td>red</td>
<td>circle</td>
<td>positive</td>
</tr>
<tr>
<td>2</td>
<td>large</td>
<td>red</td>
<td>circle</td>
<td>positive</td>
</tr>
<tr>
<td>3</td>
<td>small</td>
<td>red</td>
<td>triangle</td>
<td>negative</td>
</tr>
<tr>
<td>4</td>
<td>large</td>
<td>blue</td>
<td>circle</td>
<td>negative</td>
</tr>
</tbody>
</table>
General Learning Issues

• Many hypotheses are usually consistent with the training data.

• Bias
  – Any criteria other than consistency with the training data that is used to select a hypothesis.

• Classification accuracy (% of instances classified correctly).
  – Measured on independent test data.

• Training time (efficiency of training algorithm).
• Testing time (efficiency of subsequent classification).
Generalization

• Hypotheses must generalize to correctly classify instances not in the training data.
• Simply memorizing training examples is a consistent hypothesis that does not generalize.
• *Occam’s razor:*
  – Finding a *simple* hypothesis helps ensure generalization.
Text Categorization

• Assigning documents to a fixed set of categories.

• Applications:
  – Web pages
    • Recommending
    • Hierarchical classification for browsing
  – Newsgroup Messages
    • Recommending
    • spam filtering
  – News articles
    • Personalized newspaper
  – Email messages
    • Routing
    • Prioritizing
    • Folderizing
    • spam filtering
Learning for Text Categorization

• Manual development of text categorization functions is difficult.

• Learning Algorithms:
  – Neural network
  – Bayesian (naïve)
  – Relevance Feedback (Rocchio classifier)
  – Rule based
  – Nearest Neighbor (case based)
  – Support Vector Machines (SVM)
Neural Network Learning

• Learning approach based on modeling adaptation in biological neural systems.

• **Perceptron**: Initial algorithm for learning simple neural networks (single layer) developed in the 1950’s.

• **Backpropagation**: More complex algorithm for learning multi-layer neural networks developed in the 1980’s.
Real Neurons

- Cell structures
  - Cell body
  - Dendrites
  - Axon
  - Synaptic terminals
Neural Communication

- Electrical potential across cell membrane exhibits spikes called action potentials.
- Spike originates in cell body, travels down axon, and causes synaptic terminals to release neurotransmitters.
- Chemical diffuses across synapse to dendrites of other neurons.
- Neurotransmitters can be excitatory or inhibitory.
- If net input of neurotransmitters to a neuron from other neurons is excitatory and exceeds some threshold, it fires an action potential.
Real Neural Learning

• Synapses change size and strength with experience.

• **Hebbian learning**: When two connected neurons are firing at the same time, the strength of the synapse between them increases.

• “Neurons that fire together, wire together.”
Artificial Neuron Model  
(Linear Threshold Unit)

- Model network as a graph with cells as nodes and synaptic connections as weighted edges from node $i$ to node $j$, $w_{ji}$

- Model net input to cell as

$$net_j = \sum_i w_{ji} o_i$$

- Cell output is:

$$o_j = \begin{cases} 
0 & \text{if } net_j < T_j \\
1 & \text{if } net_j \geq T_j
\end{cases}$$

($T_j$ is threshold for unit $j$)
Neural Computation

• McCollough and Pitts (1943) showed how such model neurons could compute logical functions and be used to construct finite-state machines.

• Can be used to simulate logic gates:
  – AND: Let all $w_{ji}$ be $T_j/n$, where n is the number of inputs.
  – OR: Let all $w_{ji}$ be $T_j$
  – NOT: Let threshold be 0, single input with a negative weight.

• Can build arbitrary logic circuits, sequential machines, and computers with such gates.

• Given negated inputs, two layer network can compute any boolean function using a two level AND-OR network.
Perceptron Training

• Assume supervised training examples giving the desired output for a unit given a set of known input activations.

• Learn synaptic weights so that unit produces the correct output for each example.

• Perceptron uses iterative update algorithm to learn a correct set of weights.
Perceptron Learning Rule

• Update weights by:
  \[ w_{ji} = w_{ji} + \eta(t_j - o_j) o_i \]
  where \( \eta \) is the “learning rate”
  \( t_j \) is the teacher specified output for unit \( j \).

• Equivalent to rules:
  – If output is correct do nothing.
  – If output is high, lower weights on active inputs
  – If output is low, increase weights on active inputs

• Also adjust threshold to compensate:
  \[ T_j = T_j - \eta(t_j - o_j) \]
Perceptron Learning Algorithm (Rosenblatt, 1957)

- Iteratively update weights until convergence.

Initialize weights to random values
Until outputs of all training examples are correct
For each training pair, $E$, do:
  - Compute current output $o_j$ for $E$ given its inputs
  - Compare current output to target value, $t_j$, for $E$
  - Update synaptic weights and threshold using learning rule

- Each execution of the outer loop is typically called an *epoch*.
Perceptron as a Linear Separator

- Since perceptron uses linear threshold function, it is searching for a linear separator that discriminates the classes.

\[ w_{12}o_2 + w_{13}o_3 > T_1 \]

\[ o_3 > -\frac{w_{12}}{w_{13}} o_2 + \frac{T_1}{w_{13}} \]

Or hyperplane in n-dimensional space
Concept Perceptron Cannot Learn

• Cannot learn exclusive-or, or parity function in general.
Perceptron Limits

- System obviously cannot learn concepts it cannot represent.
- Minsky and Papert (1969) wrote a book analyzing the perceptron and demonstrating many functions it could not learn.
- These results discouraged further research on neural nets; and symbolic AI became the dominate paradigm.
Perceptron Convergence and Cycling Theorems

• **Perceptron convergence theorem**: If the data is linearly separable and therefore a set of weights exist that are consistent with the data, then the Perceptron algorithm will eventually converge to a consistent set of weights.

• **Perceptron cycling theorem**: If the data is not linearly separable, the Perceptron algorithm will eventually repeat a set of weights and threshold at the end of some epoch and therefore enter an infinite loop.
  
  By checking for repeated weights+threshold, one can guarantee termination with either a positive or negative result.
Perceptron as Hill Climbing

- The hypothesis space being search is a set of weights and a threshold.
- Objective is to minimize classification error on the training set.
- Perceptron effectively does hill-climbing (gradient descent) in this space, changing the weights a small amount at each point to decrease training set error.
- For a single model neuron, the space is well behaved with a single minima.

![Graph showing training error against weights](image-url)
Perceptron Performance

• Linear threshold functions are restrictive (high bias) but still reasonably expressive; more general than:
  – Pure conjunctive
  – Pure disjunctive
  – M-of-N (at least M of a specified set of N features must be present)

• In practice, converges fairly quickly for linearly separable data.

• Can effectively use even incompletely converged results when only a few outliers are misclassified.

• Experimentally, Perceptron does quite well on many benchmark data sets.
Multi-Layer Networks

- Multi-layer networks can represent arbitrary functions, but an effective learning algorithm for such networks was thought to be difficult.

- A typical multi-layer network consists of an input, hidden and output layer, each fully connected to the next, with activation feeding forward.

- The weights determine the function computed. Given an arbitrary number of hidden units, any boolean function can be computed with a single hidden layer.
Hill-Climbing in Multi-Layer Nets

- Since “greed is good” perhaps hill-climbing can be used to learn multi-layer networks in practice although its theoretical limits are clear.
- However, to do gradient descent, we need the output of a unit to be a differentiable function of its input and weights.
- Standard linear threshold function is not differentiable at the threshold.
Differentiable Output Function

- Need non-linear output function to move beyond linear functions.
  - A multi-layer linear network is still linear.
- Standard solution is to use the non-linear, differentiable sigmoidal “logistic” function:

\[
o_j = \frac{1}{1 + e^{-(net_j - T_j)}}
\]

Can also use tanh or Gaussian output function
Gradient Descent

• Define objective to minimize error:

\[ E(W) = \sum_{d \in D} \sum_{k \in K} (t_{kd} - o_{kd})^2 \]

where \( D \) is the set of training examples, \( K \) is the set of output units, \( t_{kd} \) and \( o_{kd} \) are, respectively, the teacher and current output for unit \( k \) for example \( d \).

• The derivative of a sigmoid unit with respect to net input is:

\[ \frac{\partial o_j}{\partial net_j} = o_j (1 - o_j) \]

• Learning rule to change weights to minimize error is:

\[ \Delta w_{ji} = -\eta \frac{\partial E}{\partial w_{ji}} \]
Backpropagation Learning Rule

- Each weight changed by:
  \[ \Delta w_{ji} = \eta \delta_j o_i \]

  \[ \delta_j = o_j (1 - o_j) (t_j - o_j) \quad \text{if } j \text{ is an output unit} \]
  \[ \delta_j = o_j (1 - o_j) \sum_k \delta_k w_{kj} \quad \text{if } j \text{ is a hidden unit} \]

  where \( \eta \) is a constant called the learning rate
  \( t_j \) is the correct teacher output for unit \( j \)
  \( \delta_j \) is the error measure for unit \( j \)
Error Backpropagation

• First calculate error of output units and use this to change the top layer of weights.

  Current output: \( o_j = 0.2 \)
  Correct output: \( t_j = 1.0 \)
  Error \( \delta_j = o_j(1-o_j)(t_j-o_j) \)
  \( 0.2(1-0.2)(1-0.2) = 0.128 \)

  Update weights into \( j \)

  \[ \Delta w_{ji} = \eta \delta_j o_i \]
Error Backpropagation

- Next calculate error for hidden units based on errors on the output units it feeds into.

\[ \delta_j = o_j(1-o_j) \sum_k \delta_k w_{kj} \]
Error Backpropagation

- Finally update bottom layer of weights based on errors calculated for hidden units.

\[ \delta_j = o_j (1 - o_j) \sum_k \delta_k w_{kj} \]

Update weights into j

\[ \Delta w_{ji} = \eta \delta_j o_i \]
Create the 3-layer network with $H$ hidden units with full connectivity between layers.
Set all weights to small random real values.
Until all training examples produce the correct value (within $\varepsilon$), or mean squared error ceases to decrease, or other termination criteria:
   Begin epoch
   For each training example, $d$, do:
       Calculate network output for $d$’s input values
       Compute error between current output and correct output for $d$
       Update weights by backpropagating error and using learning rule
   End epoch
Comments on Training Algorithm

• Not guaranteed to converge to zero training error, may converge to local optima or oscillate indefinitely.

• However, in practice, does converge to low error for many large networks on real data.

• Many epochs (thousands) may be required, hours or days of training for large networks.

• To avoid local-minima problems, run several trials starting with different random weights (random restarts).
  – Take results of trial with lowest training set error.
  – Build a committee of results from multiple trials (possibly weighting votes by training set accuracy).
Hidden Unit Representations

• Trained hidden units can be seen as newly constructed features that make the target concept linearly separable in the transformed space.
• On many real domains, hidden units can be interpreted as representing meaningful features such as vowel detectors or edge detectors, etc..
• However, the hidden layer can also become a distributed representation of the input in which each individual unit is not easily interpretable as a meaningful feature.
NNs for Text Categorization

• Represent documents as vectors in the standard way, using:
  – Binary word vectors
  – Bag of words normalized frequency vectors
  – TF/IDF weighted bag of words vectors

• Train a Perceptron or a multi-layer Perceptron (MLP) (typically a 3-layer backprop net) to classify these document vectors.
Summary

- Automated text categorization uses machine learning to generate a classifier function by training on labeled training data.
- There are many approaches to machine learning that incorporate various inductive biases.
- Neural networks have demonstrated recent success on many problems particularly those involving text and language.