Machine Learning & Subsymbolic Computing

Chapters 10 (to end of 10.2), 11 (to end of 11.2)

(disclaimer: most images are not mine)

Machine Learning

- Obviously learning is important to any domain
- Machine learning is often associated with lower-level systems where learning is most of what the system does: e.g. neural networks
- But learning components are important in any agent designed to work in a changing environment
- Learning allows such agents to adapt as the domain changes around them

Symbolic vs. Subsymbolic Learning

- Learning can be done at both the symbolic and subsymbolic levels
- I'm first going to talk about learning in general, then learning in subsymbolic systems, and finally learning in symbolic systems
- Because you haven't seen any subsymbolic systems yet, this will also introduce you to this topic

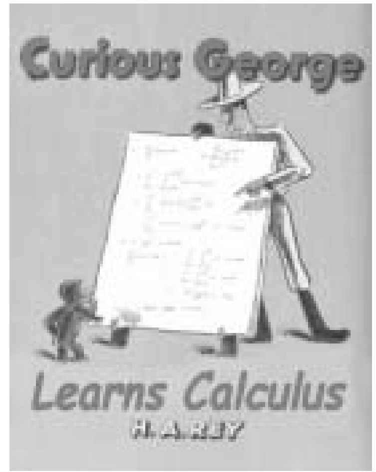
Learning

- The number of variables in learning is HUGE
- We can talk about WHAT is being learned for example: concepts? goodness of concepts? experiences to repeat or not to repeat?
- As you'd imagine, there's specialized techniques on that
- There's also ways of teaching, and indeed, degrees to which an instructor is even involved…

Learning Method

- We have a spectrum of learning mechanisms based on how involved an external instructor is in the process
- Highest to lowest involvement:
- Rote learning
  - Direct implantation of knowledge without inference on the part of the learner
  - i.e. I tell you the answer, you remember it and recall it at the right time later on

Learning Method



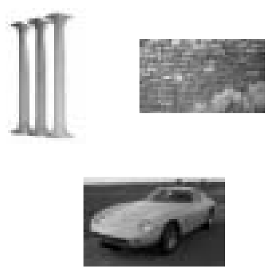
- Learning from Instruction
  - transformation of external instruction into an internal representation and integration with prior knowledge and skills

Learning Method

- Learning from Example
  - Extraction and refinement of knowledge from specific cases – e.g. an arch

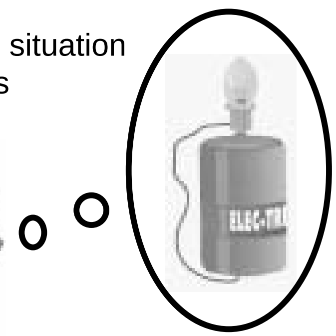
Positive Examples

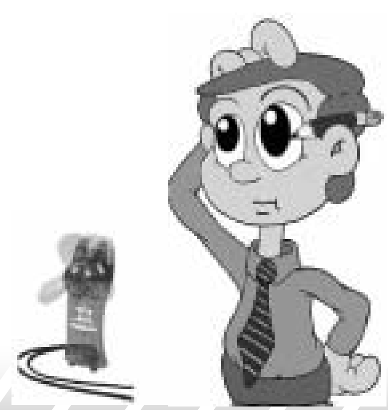
Negative Examples



Learning Method

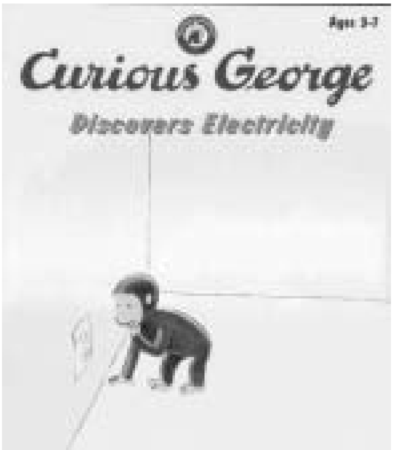
- Learning by Analogy
  - Transferring a solution from one situation to a new one with novel aspects





(this is a very weak analogy!)

Learning Method



- Learning by Discovery
  - Gathering new knowledge and skills from observation and experience
  - easy to see that this is well beyond the other levels!

Learning Feedback

- Feedback indicates performance level achieved so far. We can differentiate learning based on the type of feedback given to an agent as well:
  - Supervised Learning: Feedback specifies desired activity of learner and objective of learning is to match this
  - Reinforcement Learning: Feedback only specifies utility of actual activity of the learner and objective is to maximize this utility
  - Unsupervised Learning: No explicit feedback provided, objective is to find useful and desired activities based on trial and error and self-organization

Types of Reasoning

- You're well-used to deductive reasoning by now! We have some axiom, and use logical reasoning to DEDUCE some new axiom
- This deduction is certain and solid provided we have everything purely FOL based
- This isn't the only type of reasoning we can perform though
- We can and do use non-deductive reasoning

Types of Reasoning

- **Abduction**

b	conclusion
$a \rightarrow b$	rule
______	
$\therefore a$	premise

- A could have caused B
- Recall from implication that this is not necessarily so – but we CAN reason about the possibility it caused it
- This is a major part of diagnostic reasoning

Types of Reasoning

- **Induction**

  a premise

  b conclusion

  $\therefore$ a $\rightarrow$ b rule

- This is a major part of reasoning in learning: associating concepts!
- Again, this MIGHT be wrong (not sound!), but we do it all the time!

Symbolic Vs. Subsymbolic Learning

- In the case of symbolic learning, we also have to choose a representation for what's to be learned
- This is not the case in subsymbolic learning
- Why?
- There IS no representation
- Recall subsymbolic reasoning: it exists below the symbol level and does not construct symbolic representations

Symbolic vs. Subsymbolic

- Instead of building and manipulating symbols:

  bird(tweety)

  If bird(X) Then hasWings(X)

  $\therefore$hasWings(tweety)

- We recognize (match) patterns in data without using symbols or maintaining symbol structures

Cognitive Processing

- This is very fast (we recognize faces in <100ms, and this is not an easy task!)
- But also unexplainable – we can't say *WHY* we recognize a face when we're doing it subsymbolicly , it just happens
- The lack of explainability emphasizes the lack of representation
- The overall SYSTEM recognizes faces, but no single part or aspect is stored anywhere in particular
  - this is why subsymbolic approaches can be very robust, but also very hard to understand

Connectionism

- Most subsymbolic implementations involve connectionist approaches
- A connectionist approach is one that is centered around the connections between simple elements
- Connectionist approaches don't HAVE to be subsymbolic
  - For example, if I connect behaviours together so the stimulus of one leads to another, that's a connectionist concept…

Subsymbolic AI

- Connectionist approaches are most commonly associated with low level subsymbolic problems, however
- This paradigm has been used very successfully to develop intelligent systems that perform low-level cognitive tasks such as speech recognition
- Neural networks are the most well known form of connectionist approach (there are many others!)
- Most artificial neural networks operate in a manner that is significantly simpler than that of actual biological neurons, but are based on the same concepts

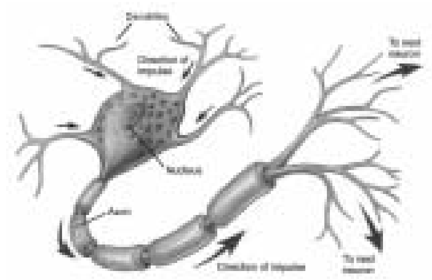
Neural Networks

- Neural Networks are pattern matching systems. Some researchers believe that we do not perform explicit symbol manipulation when solving many of the tasks that are considered to require intelligence
- Instead, we perform a pattern matching process in which we react to stimuli at a subconscious level instead of explicitly processing the stimuli at a conscious level

Neural Networks

- Neural networks are models of intelligent processing that consist of large numbers of simple processing units that collectively are able to perform very complex pattern matching tasks
- The processor can perform only one type of operation; the processors are quite slow
- Doesn't sound like much but: Connections link the processors together
- This massive parallelism lets them do very interesting things

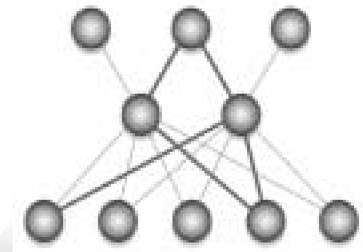
Human Neural Cells



- There are approximately 10^{11} neurons in the brain
- There are approximately 10^{15} connections in the brain
- The brain performs about 10^4 operations per second

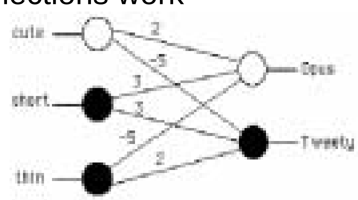
Neural Networks

- Artificial neural networks attempt to duplicate the processing of neural cells: they're a physical model of intelligence rather than a logical one
- A unit "turns on" (dark connections in image below) if it receives sufficient stimuli (from the outside or from other neurons) and in turn stimulates others or generates output



Neural Networks

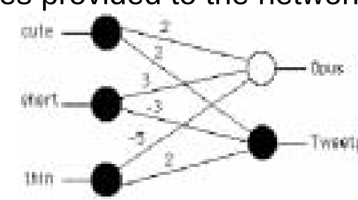
- Neural networks are trained by presenting stimulus-response patterns and adjusting weights (strengths between connections) until the desired response is generated for each stimulus
- In this example, the network below associates short and thin with tweety – it's small enough you can see how the connections work



- but in any non-trivial network we could never artificially weight (i.e. hand construct) the connections to our desired outputs this way!

Neural Networks

- Instead, we design learning algorithms that cause the weights in a network to be adjusted based on training cases provided to the network



- We could take patterns and get the network to adjust itself to the network on the previous slide. A different set of patterns could be used to train the network to associate cute, short, and thin with Tweety
- Clearly not just the patterns, but the information we give the network when it is incorrect (supervised learning)

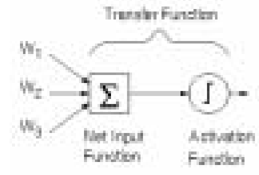
Processing

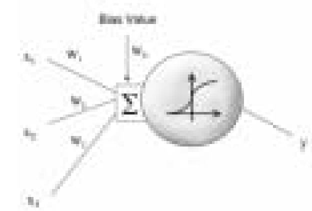
- The inputs to each node are multiplied by the strength (weight) of the connection to the node and then summed to produce the net input (or stimulus) to the node

  Net = $\sum I_i * W_i$

Processing

- The net input is then sent through an activation or transfer function that generates the output value (or response) for the node

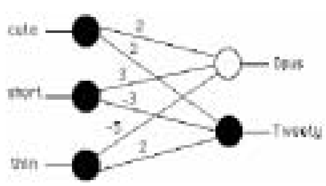




- A typical transfer function is the threshold function – at a certain input threshold an output is given, and none is given if that threshold is not reached
- Using a Sigmoid function for transfer is also popular – what effect would this have?

Training

- A neural network is trained by presenting patterns to the network
- For units that do not generate the desired output, the weights coming into that unit are adjusted slightly by a learning algorithm



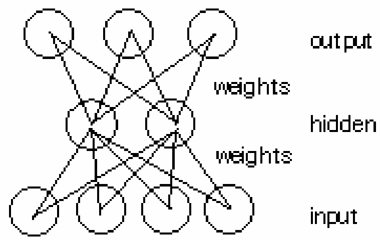
- An error term is computed for each output unit
- The error term is a function of the difference between *the value actually generated by the unit* and the *value that was supposed to have been generated (supervised learning)*

Single-Layer Networks

- Networks with direct connections from the input layer to the output layer are referred to as single-layer networks because there is only one layer of processing units
- Single-layer networks are very limited in their ability to perform stimulus-response mappings
- For example, single-layer networks cannot learn an exclusive-or mapping
- This was why perceptrons were initially abandoned

Multi-Layer Networks

- Adding one or more additional layers of hidden units between the input layer and the output layer permits the network to perform **MUCH** more complex stimulus-response mappings



Backpropagation Networks

- Backpropagation neural networks are the most popular type of multi-layer network
- Until relatively recently, it was not known how to train backpropagation networks
- Finding error and using it to alter weights on output nodes is easy – just difference between desired and actual outputs
- But what about hidden layers? How can we find out how much error some hidden node is responsible for? No way to know for sure…

Backpropagation

- The error is instead approximated, then *propagated back* through hidden layers
- I won't state the approximation formula used, or even its characteristics here – to even discuss those would require getting into low level learning functions beyond the level of 3190 – You will see this in 4360 next term
- Error ends up being propagated back to the input nodes, and each node can adjust connection weights as it goes back

Example: NetTalk (p. 471-3)

- Terry Sejnowski of Johns Hopkins developed a system that can pronounce words of text
- The system consists of a backpropagation network with 203 input units (29 text characters, 7 characters at a time), 80 hidden units, and 26 output units
  - The system was developed over a year
- The DEC-talk system consists of hand-coded linguistic rules for speech pronunciation
  - developed over approximately 10 years
- DEC-talk outperforms NETtalk but DEC-talk required significantly more development time

NetTalk

- "This exemplifies the utility of neural networks; they are easy to construct and can be used even when a problem is not fully understood. However, rule-based algorithms usually out-perform neural networks when enough understanding is available"
  - Hertz, *Introduction to the Theory of Neural Networks*, p. 133

Remember

- What we're doing is strictly supervised learning here
- The system is learning patterns that ***we*** are organizing for it and presenting in a rational way, and we're telling it whether a new item fits the pattern or not
- The system can't explain WHY it feels something matches or not
  - again, typical of subsymbolic approaches
- This means the system can't explain what it's learning either…how are we satisfied that it is doing what we want in the end?

Tank Recognition System

- Researchers trained a neural network to differentiate scenes that contained one or more tanks from scenes that did not contain tanks
- The network was able to correctly recognize new scenes that the network had not been trained with
- The network was finally tested with additional scenes and was not able to recognize them correctly
- diagnosis?

Tank Recognition System

- The network actually recognized that the pictures taken of the tanks were taken on sunny days while the pictures taken without tanks were taken on cloudy days
- The system WAS learning – just not what we expected
  - This is mainly a problem with US not recognizing the features and not differentiating what is intended to be learned (a faulty teacher)
  - also really a common-sense reasoning problem (think of a person doing this)

Symbolic Machine Learning

- We can (and do!) use learning at a level above the subsymbolic as well
- One common method is through the use of *Version Space Search.* The name comes from the fact that we are searching through the space of different versions of a concept as we see positive and negative examples
- Version space search uses two basic operations: generalization and specialization
- Generalization makes more general statements: color(ball,red) can become color(X,red)
  - size(small)^color(red) can become
    size(small) ^ (color(red)vcolour(blue))

Version Space Search

- Similarly, specialization takes those more general statements and applies them more specifically (e.g. the inverse of the previous two examples!)
- What version space search does is attempt to find symbolic expressions that accurately describe a set of examples
- For example, various examples of an arch
- We can give a symbolic description of several arches and ask the system to search for a common description

Version Space Search

- That search, with generalization, will allow it to make a general statement true of all arches it knows about
- as we add more examples, that description may prove too general
  - i.e. cover things that aren't arches
- the system is then expected to use specialization to find coverings that deal with the positive cases but exclude the negative ones

Version Space Search

- One such approach to search: Specific to General
- We maintain a set of candidate concept definitions, and ensure they are maximally specific generalizations
- This means that a concept covers all positive examples, no negative examples, and is more specific than any other concept that does
- We start out extremely specific (just this one case is an arch!), and gradually generalize to include all valid candidates

Algorithm

- S=set of candidate hypotheses; N is the set of all negative instances seen so far
- initialize S to be the 1st positive training instance (i.e. S starts off extremely specific – one case!)
- For every training instance {
  if the instance is a Positive instance, P {
  for every element s in S: {if s doesn't match P,
  replace s with its most specific generalizations
  that do match P} (there may be several of these)
  Delete from S any hypothesis more general than
  some other hypothesis in S
  Delete from S all hypotheses that match a
  previously observed negative instance in N } //if

(continued…)

Algorithm

- If the instance is a negative instance, N {
  Delete all members of S that match n
  add n to N to check future hypotheses for
  overgeneralization } } //if, for
- This requires both positive and negative examples – negative examples keep the algorithm from overgeneralizing
- How else could we have done this?
- we could also have gone the other way – started very general, then got more specific
- i.e. start by saying EVERYTHING is an arch!
- Then go through negative examples making our definition of an arch more and more specific!

Candidate Elimination

- What we'd be keeping track of is a set of concepts that are maximally general and still cover all positive instances and no negative ones
- We'd need positive instances here to avoid getting too specific
- Taken together, these form a BIDIRECTIONAL search that we call Candidate Elimination
- We maintain TWO sets of candidate concepts: the maximally general set and the maximally specific set
- we specialize G and generalize S until we converge on the target!

Symbolic Machine Learning

- There are many other types of machine learning at the symbolic level
- For example, we can learn decision trees based on cases
- e.g. a loan manager looking at specific questions that were asked, and from that whether people later defaulted on loans
- We can integrate cases one at a time and gradually learn a tree that lets us decide a risk assessment for different categories

Symbolic Machine Learning

- this is analogous to our version space scenario
- we're attempting to induce a decision tree using cases at hand, and alter it based on new positive and negative cases
- we add new branches or merge branches into similar decisions (specialize, generalize)
- We can again do this top down, bottom up, or bidirectionally (similar issues)
- There are still different types of learning yet –

Reinforcement Learning

- In our robotics situation, we could learn a set of reactions based on experiences in an environment
- The previous situations were all examples of SUPERVISED LEARNING – the most basic type, where a teacher organizes the cases and presents them in a coherent fashion
- In robotics, we want them be more independent from the actions of a teacher – we're going to do REINFORCEMENT LEARNING
- i.e. they just get good/bad (from a teacher or something in the world, such as a score)

Q-Learning

- We're going to look at a simplified version of Q-learning, a form of reinforcement learning
- Think of the robot having a big reaction table
- rows: Actions I can perform; cols: situations I can recognize (it's a mapping, doesn't really have to be a table)
- The table cells can contain RANKINGS of how good each potential action is for all situations
- we call this table Q (our Q-table)
- We could fill this with random values to start with
- it would then be a (very poor) approximation of Q*, the optimal table for the environment
- A learning algorithm would allow the agent to improve its table

Q-learning

- When it does something good (think scoring a goal in soccer) we'll give it positive reinforcement
  - Yay….
- when it does something bad, we'll give it negative reinforcement
  - Booo….
- and the learning algorithm will adjust the table's "goodness" entry for that action down or up
- (What kind of algorithm is this?)
  - Iterative Improvement!
- What action do we pick on each turn?

The Best?

- No – we'll have local maximum problems
- Suppose my highest ranked action initially (i.e. randomly) is 8, and the next highest is 5
- We do the action, goodness gets adjusted to 7
- Which do we do next time?
- The 7, if we're choosing by goodness
- but what's the 5?
  - a *GUESS*..could really be a 9 (even worse, what if all untested actions were 0?)
- We need to be able to find out whether that 5 is *really* a 5 or not…so what do we do?

Exploration vs. Exploitation

- We choose the best action *sometimes*, and some other choice at other times
- most often a RANDOM choice
  - Randomness is extremely important in learning, this is a good example
  - Being able to try anything randomly also assumes that all actions are safe to try – may want to initially bias/restrict some
- This also considers only the immediate reaction in dealing with reinforcement
- What's the problem with reinforcing individual reactions?
- Not often accurate!

Delayed Reinforcement

- You might get delayed reinforcement
  - I work today, I get paid @ the end of the month…
  - reinforcement is naturally delayed sometimes – how can we learn from that?
- In multi-agent settings the reinforcement is delayed in other ways
  - You kick a ball, somebody else scores a goal – how do you know what you did was good?
  - or you screw up, somebody else scores a goal – how do you know what you did was bad?
- These are instances of the Credit Assignment Problem (former: *temporal* credit assignment; latter: *inter-agent* credit assignment)

Dealing with the CAP

- need to deal with the fact that your immediate action didn't necessarily cause what's going on
- we do this in Q-learning by passing a portion of the blame/praise *back to previous actions*
- Divide reinforcement up into two parts, I and P – I reinforces the immediate action, P reinforces the previous one
- And we keep doing this recursively – over time this helps us to learn sequences!
- btw, this is still not completely describing Q-learning, you'll cover more in 4360

Example

- Mataric's *Reinforcement Learning in the Multi-Robot Domain* (paper in Stuff of the Week – paper is not examinable, but you should understand this example at the level at which these notes present it!)
- This paper nicely discusses some of the issues we've already brought up
- Uses several behaviour-based robots in a puck-gathering setting (think of waste cleanup)
- Individuals able to deal with the following behaviours:

Behaviours

- Safe-wandering: moving about without colliding with objects (including other robots)
- Dispersion: achieve and maintain distance between all robots within sensing range of one another
- Resting: keeps robot parked for fixed period of time (intended for recharging)
- Homing: go to particular location

General idea

- Learning the appropriate conditions for triggering these behaviours
- Can perceive conditions:
  - Have puck?
  - At home?
  - Near intruder? (distance between robot and nearest neighbour – "personal space")
  - Night-time? (internal clock, desire to recharge)
  - These behaviours and predicates constitute the learning space. *Other behaviours are present but not learned*

Learning Approach

- It had been thought in the RL community that minimizing reinforcement as much as possible was good, because it diminishes experimenter bias
- In this work, Mataric was trying to illustrate that it's more important to get power out of learning by shaping reinforcement to suit the task
- Makes sense for human learning anyway!
- So she compared learning with a simple reinforcement function to some more complex ones that were tailored more closely to the situation

Learning Approach

- similar in spirit to Q-learning
- we have a matrix representing a weighted mapping between percepts and behaviours
- A(c,b) entry in the matrix is the normalized sum of reinforcement R received for condition/behaviour pair over time T (i.e. *average goodness over time*)
- possible to give positive reinforcement for these events: grasping a puck, dropping puck at home, waking up at home
- Negative reinforcement for dropping puck away from home, waking up away from home

Events

- Selecting behaviours is induced by events
- These may be external: getting in the way of one another, dropping a puck
- These may be internal (clock signalling night-time, or a wake-up)
- May be triggered by two *progress indicator* functions
  - one to see if we are moving away from an intruder in our personal space
  - Another to tell if we are moving away from home or not
  - These can also be used give reinforcement under certain conditions (bad if we have a puck and are moving away from our goal, or if we have an intruder and aren't moving away from them)

Handling Events

- When an event is detected:
- Call the reinforcement function to deal with current behaviour-condition pair (deals out reinforcement based on current conditions)
- We terminate current behaviour
- We choose another behaviour:
  - Untried one if one is available, Otherwise the best
  - This first encourages exploration (ok in this domain, not so great in others!)
  - This is unusual in that most domains would divide it into best x% of the time, random y%
  - She argues that perception is so noisy that this alone gives us enough randomness

Learning

- Incremental and continuous over time
- Table is 64 entries: 2^4 conditions * 4 behaviours
- Compared reinforcement function to others:
  - Basic single goal reward function (puck home) using Q-learning
  - Heterogeneous reward function using multiple goals, summing reinforcement over time (the collection of all reinforceable elements described earlier, except for the progress estimators)
  - Heterogeneous reward function using this as well as the two progress estimators

Success in Learning

- Many ways to look at success in learning
- We could look at overall success at a task (e.g. if we're foraging for pucks, the more pucks we get in the better)
  - indirect measurement of learning
- There are more direct measures as well
- For example, if we know what the correct set of actions or decisions (policy) should be, we can look at how much of that policy is learned
  - learning should converge to the correct policy
  - if learning doesn't ever converge, something is not learnable given our learning method
- Also, how fast do we converge?

In This Example

- 60 trials, 20 of each strategy, 4 robots
- Hard to compare – we can't just talk about convergence because the amount of time required depends on external events
- looked at % of correct policy (constructed by hand from observation) robots learned in 15 minutes
- Q: ~28%
- R over time: ~52%
- R over time + PE = 89%