CSC 417 Midterm Study Guide

1. Quiz 1
   1. A drawback of rote learning is that AI learns a good sequence of moves for a game, but does not understand how to win
   2. Generalized learning techniques do not result in an AI that plays strong endgames.
   3. Humans and AI tend to not use similar representations of board game states (e.g. visual representation).
   4. Human beings seem to be built to employ heuristics rather than algorithms
   5. In conjunction with Checkers, Arthur Samuel observed that developing a heuristic for a task can be harder than performing the task.
   6. When employing a Finite State machine, additional states are not to be created as needed during execution
   7. All Finite State Machines do not need to contain an ending state.
   8. Behavior Trees tend to be more transparent than Finite State Machines (that is, it is easier for a human to understand exactly what the AI is doing).
   9. A perceptron learns by adjusting the weights associated with specific inputs.
   10. A Convolutional Neural Network analyzes a group of pixels in a source image by means of one or more filter(s).
   11. Heuristics are most appropriate for analyzing the midgame of a board game (such as checkers or chess).
   12. A chess AI which employs a Shannon Type B approach will extend the searching depth for interesting moves.
   13. In a behavior tree, a leaf node specifies the actual action to be taken.
   14. The following is true of backpropagation in neural networks—they typically requires large training data sets, they distribute error across weights in the output layer and all hidden layers and they require a continuous activation function.
   15. It is not true of backpropagation in neural network that it allows networks to learn training data in a single epoch.
   16. One strength of Convolutional Neural Networks compared to “traditional” ANNs is that they can identify subject regardless of its position in an image.
2. Quiz 2
   1. In a Convolutional Neural Network, a tensor is not a one-dimensional array.
   2. The output from a convolutional layer is smaller than the input.
   3. The purpose of a pooling layer in a CNN is to reduce spatial size of features (and thus require the network to process less data).
   4. In order to achieve high accuracy, a Recurrent Neural Network must include more than one recurrent layer.
   5. In an RNN, the vanishing gradient problem refers to the fact that later inputs have a substantially larger impact on the output than early inputs.
   6. In an RNN, one purpose of the hidden state is not to preserve sequence relationship information.
   7. The goal of a Generative Adversarial Network is not to create an exact duplicate of some set of input data.
   8. Because the input is typically random noise, a GAN typically requires many epochs to create “good” output.
   9. Generative algorithms are not used to classify input data.
   10. A “standard” CNN would not be an appropriate choice for the generative component of a GAN.
   11. In a Convolutional Neural Network, max pooling is used to perform both de-noising and dimensionality reduction on the input data.
   12. Padding is used to address this shortcoming of CNNs—that information “on the edges” of the input is not captured well.
   13. In an RNN, processing all inputs in sequence is referred to as unfolding.
   14. In a GAN, the following is a job of the generative network—modle the distribution of data in a category, determine the probability of a feature existing in a given category, and produce a synthetic data item which fits a given category.
   15. In a GAN, to learn the decision boundary separating categories is not a job of the generative network.
   16. Within a GAN, no training data set is required, the training data set is passed into the discriminative network, and the training data is unrelated to the random noise input to the generative network.
3. Advanced Computer Games
   1. Checkers (Arthur Samuel)
      1. Why checkers?
         1. Non-deterministic – different outputs possible for identical inputs
         2. Clear goal – remove opponent’s pieces
         3. Clear rules – easily described
         4. Established knowledge – many human experts
         5. Understandable to even “causal” observers
      2. Minimax – good at representing a game like checkers
         1. System defaulted to looking 3 moves ahead
         2. Exceptions that increased search depth:
            1. Considering a jump
            2. Last move was a jump
            3. Piece exchange possible
      3. Notation – light colored squares are numbered (this allows for easy input into computers)
      4. Heuristics
         1. Piece advantage
         2. Denial of occupancy (board control)
         3. Mobility
         4. Hybrid measure of control of board center plus piece advancement
      5. Moves that were not referenced for a long period were “forgotten” – storage space was expensive
      6. Rote Learning
         1. Lacked sense of direction (*how* do we achieve the goal)
         2. Program could memorize a good sequence of moves but did not “understand” how to win
         3. Resulted in strong openings and endgames, but a poor midgame
         4. Good for situations requiring specific actions
      7. Generalized Learning
         1. Alpha program – updated evaluation of moves during a game
         2. Beta program – updated evaluation of moves after a game
         3. Programs played each other repeatedly to develop a collection of good moves
         4. Resulting program was above average, but had limitations:
            1. Easily fooled by deliberate bad play (opponent could make a non-optimal move to “throw off” the system)
            2. Evaluation function changed too quickly
            3. Overvalued “flashy plays” – favored large score swings over basic moves that set up later wins
         5. Strong midgame, but poor openings and endgames
         6. Useful when there are many possible actions – aka the midgames of chess and checkers
      8. Non-search-based techniques
         1. Turing Machine
            1. Components

Infinitely long tape with “boxes” (memory locations)

Each box may contain a symbol

Each box may be read, written, or erased by the read/write head

Instructions on specifying how to modify squares

Listing of states (which define actions) and transitions

* + - 1. Finite State Machines
         1. Consist of one or more states (finite number)
         2. One state active at a time
         3. FSM transitions between states
         4. States must be known in advance
  1. Chess
     1. Search Trees
        1. Typically minimax with alpha-beta pruning
        2. In order to search all possible move must be represented (there is a lot for chess)
     2. Shannon Type B
        1. Extend search depth beyond specified limit for “interesting” moves (captures, checks, exchanges, etc)
        2. Moves with significant impact on the state of the game require careful consideration
        3. Anticipate likely moves and consider follow-up move (progressive deepening)
     3. Computer require numeric values to represent squares – facilitates “pseudo-legal move lists”
     4. Openings/Position Evaluation
        1. Openings are difficult (there are a lot of general rules and a lot of exceptions to these rules)
        2. Computers – brute force openings (apply search to large databases)
        3. Midgames – Heuristic evaluation (general guidelines for determining how “good” a move is)
  2. Poker
     1. Cepheus
        1. AI system that plays heads-up limit Texas hold’em
        2. Decision space is small enough to essentially compute all possible outcomes
        3. Learned based on rules of the game without human strategy analysis
     2. Libratus
        1. AI system that plays heads-up no-limit Texas hold’em ( a lot more decision points
        2. Extremely high hand win rate
        3. General Approach
           1. Pre-compute solution to an abstraction of the game
           2. Subgame solving – recalculate based on opponent’s moves
           3. Self-improvement – add actions taken by opponent to abstractions and solve them
  3. Behavior Trees
     1. Advantages
        1. Provide for overall flow of decision making
        2. Facilitate “fallback tactics”
        3. Handle complexity better than Finite State Machines (Easier to debug)
     2. Disadvantages – can lead to AI “holes” if not carefully planned
     3. Definition
        1. Directed acyclic graphs (Connections are one way and there are no cycles)
        2. During each iteration of the game’s logic loop, the system traverses the tree to identify the appropriate node to process
        3. To avoid traversing the entire tree during each update, the program can store the currently active node
     4. Components
        1. Composite nodes – have one or more child nodes
           1. Sequence Node – indicates a sequence of actions (child nodes) to be performed (AND); actions may be tests/checks
           2. Selector Node – Perform actions specified by child nodes until one succeeds (OR)
        2. Decorator Nodes – have exactly one child node, can transform result from, terminate or repeat the child node
        3. Leaf Nodes – no children, define the actual actions to be taken, and can call another behavior tree

1. Deep Learning
   1. Mind vs. Machine
      1. Search in AI doesn’t work the same way that the human brain works
      2. Neural Networks are inspired by the human brain
   2. Artificial Neural Network
      1. Definition
         1. Collection of artificial neurons (processing units) arranged in some topology, has an input layer, some number of hidden layers and an output layer
      2. Function – adjust weights of connections between neurons
   3. Development of ANNs
      1. McCulloch-Pitts Network (1943)
         1. First artificial neuron
         2. Input and output are binary
         3. No learning due to lack of weights – MC-P was not adaptive
      2. Perceptron Learning Rule
         1. Weights are adjusted based on output from network
            1. Replace threshold θ with an activation function
            2. If output is incorrect, adjust weights
            3. Calculate new weight as wnew = wold + η(t – o)xi

η = learning rate

* + - 1. Network converges (learns correct output if)
         1. Training data is linearly separable
         2. Learning rate is sufficiently small
         3. Perceptrons are adaptable
      2. Classic shortcoming – single layer perceptron cannot learn XOR (not linearly separable)
    1. Backpropagation
       1. Method for adjusting weights in a multilayer network – attempts to determine “share” of error to assign to each weight
       2. Requires a lot of data & a lot of epochs
       3. Activation function must be continuous and differentiable
       4. Procedure
          1. Feed forward computation
          2. Backpropagation of error to output layer
          3. Backpropagation of error to hidden layer
          4. Update all weights
    2. Feature Extraction
       1. Deep neural network = network with two or more hidden layers
       2. Deep networks automatically perform feature extraction (identification of the best variables in the data) – a separate extraction step is not needed
  1. Advantages of Deep Networks
     1. Layers can learn different sets of features in the data
     2. Can find structures and patterns in unlabeled, unstructured data
     3. Can perform automatic feature extraction – determine what features that are key to the output

1. Convolutional Neural Networks
   1. Key concept – tensors
      1. Multidimensional arrays that may consist of many dimensions (similar to nested arrays)
   2. Analyze a group of pixels at once by applying a filter to a group of pixels to evaluate the destination pixel
   3. Reduce images to more easily processed forms without losing features
   4. Layers
      1. Convolution layer
         1. Filter is “moved across” pixels in an image while performing matrix multiplication
         2. Different filters may be applied to different layers
         3. First CV layer captures low-level features (edges, color, etc)
         4. Additional CV layers capture high-level features (e.g. components of a face)
      2. Pooling layer
         1. Reduces spatial size of features (less data to process)
         2. Max Pooling – new value is max value contained in pooled area
            1. Performs de-nosing and dimensionality reduction
         3. Average Pooling – new value is average of the pooled area
            1. Performs dimensionality reduction only
      3. May have multiple convolutional + pooling layer pairs
      4. Fully connected layer – “traditional” NN layers learn non-linear combinations of high-level features
   5. Activation
      1. Convolutional layers often use the ReLU activation function
      2. Advantages:
         1. Simpler computation – derivative always = 1 for positive input
         2. Representational sparsity – can output zero values
         3. Linearity – easier to optimize
      3. Disadvantages – “Dying ReLU” – once neuron goes negative, it is unlikely to "recover” because slope of any negative value is 0
   6. Padding
      1. Convolution layers reduce output volume (create a smaller image) – information on the border of an image is not capture well by the network
      2. Pooling also results in reduce output volume
      3. Padding addresses this problem by adding “extra” information around the edge of the data
         1. Same/Zero padding – adds zeros to the edge of layer output
         2. Constant padding – adds a user-specified constant value at the edges
         3. Reflection padding – adds “mirror” values in opposite direction
         4. Replication/symmetric padding – copies and mirrors values
         5. Zero/same padding change the distribution of the data and are generally not recommented
   7. Drawbacks of CNNs
      1. CNNs do not encode (learn) position or orientation of an object – can learn the components of a face, but not how they are arranged
2. Recurrent Neural Networks
   1. Introduction
      1. Traditional feed-forward networks accept an input and transform it into an output via a static model based on training data
      2. Each input in a traditional network is analyzed in isolation
         1. Relationships between different inputs are not considered
         2. Past input does not influence the classification of the current input
      3. Traditional networks have difficultly with sequences (Natural Language Processing is very hard)
   2. RNN Overview
      1. RNNs incorporate past input into analysis of the current input
         1. A “hidden state” captures information related to the sequence of inputs
      2. RNNs often have a single recurrent layer – network “depth” relates to the number of inputs processed in a sequence
      3. Example: asking chatbot “what time is it?”
         1. Input first word into network and obtain output
         2. Input second word and hidden state into the network and obtain an output
         3. Final output is created based on the input s and the hidden state – the hidden state preserves sequence relationship information
   3. RNN Process
      1. Initialize network layers – input, recurrent (often only 1), output (feed forward)
      2. Pass inputs and initial hidden state into network, receive output and modified hidden state
      3. Repeat until all inputs are processed
         1. “Unfolding” a network = processing all input in sequence
      4. Prediction made via feed-forward layer
      5. Adjustments made via backpropagation through time – adds series of calculations (linking time steps) to standard backpropagation algorithm
   4. Vanishing Gradients
      1. RNNs have trouble retaining sequence information over many steps (they have short term memory)
      2. Later inputs have a significantly larger impact on the output
      3. Addressing Vanishing Gradients
         1. LSTM – stores extra info to “learn” what inputs are important
         2. GTU – LSTM with fewer internal components