HW 4

Solution 1

What is a Bayesian Network (BN)? What are its advantages over a tabular representation of the joint distribution of the system variables? Explain both the representational and the computational advantages. What is a Dynamic BN (DBN)? Describe Markov Models (MMs) and Hidden Markov Models (HMMs). Pick a problem of interest in Data Science that can be modeled using HMMs. Describe the problem and the application of HMM techniques on it.

Bayesian Networks

- A directed graphical model that represents probabilistic relationships among a set of variables.
- Consists of 2 parts:
  - Graph – Qualitative Specification
  - Conditional Probabilities – Quantitative Specification

Advantages over Tabular Representation of the joint distribution of the system variables

- Representational Advantages:
  - **Compact Representation:** BNs offer a more compact representation compared to tabular representations of joint distributions, especially for systems with many variables.
  - **Modularity:** BNs enable modular representations of dependencies, allowing the separation of complex systems into manageable components.
  - **Intuitive Structure:** Graphical structure of BNs provides an intuitive understanding of the conditional dependencies among variables.
- Computational Advantages:
  - **Efficient Computation:** BNs facilitate efficient computation of probabilities by exploiting conditional independence relationships, reducing computational complexity.
  - **Space and Time Efficiency:** They require less memory and computation time, particularly for inference tasks, compared to storing and manipulating the entire joint distribution table.

	BELIEF NETWORKS	**JOINT DISTRIBUTION**
REPRESENTATIONAL	$O(N2^K)$	$O(2^N)$
COMPUTATIONAL	$O(N2^K)$ is equivalent to $O(N2^t)$	$O(2^N)$

Where “t” is treewidth, “k” is size of largest family layout, and “N” is number of variables.

$$k < t \leq N$$

Dynamic BNs

- Extension of BNs where model systems evolve over time by incorporating temporal dependencies between variables.
- It represents temporal relationships among variables using a sequence of BNs, enabling modeling of time-varying systems.

Markov Models (MMs) and Hidden Markov Models (HMMs):

- Markov Models (MMs): MMs are stochastic models that describe a sequence of events where the probability of each event depends only on the state attained in the previous event.

- Hidden Markov Models (HMMs): HMMs are a type of MM where the system being modeled is assumed to be a Markov process with unobserved states (hidden states). Observations (visible states) are indirectly influenced by these hidden states.

Application of HMM in Data Science

Problem: Gene Prediction in Bioinformatics

- **Description:** Hidden Markov Models (HMMs) are extensively applied in bioinformatics for gene prediction, where the hidden states correspond to different parts of a gene (e.g., exons, introns), and observed states are nucleotide sequences.
- **Application:** In the context of gene prediction, HMMs help model the sequential nature of DNA sequences, allowing for the identification of potential genes and their structures. The hidden states represent the various components of a gene, and the observed nucleotide sequences serve as input data.
- **Techniques:** Training an HMM for gene prediction involves estimating transition probabilities between hidden states, representing gene components, and emission probabilities from hidden states to observed nucleotides. This enables the model to learn the patterns associated with gene structures.
- **Use Case:** Gene prediction using HMMs has significant applications in deciphering the genomic code. It aids in identifying the locations of genes in DNA sequences, understanding their structures, and predicting coding regions. This information is crucial for tasks like understanding genetic diseases, drug discovery, and the broader field of genomics.

Solution 2

What is a Weighted Constraint Satisfaction Problem (WCSP)? Pick two problems of interest in Graph Theory and explain how they can be modeled as WCSPs. Explain how the MAP query on a BN is equivalent to a WCSP. How can a WCSP on Boolean variables be translated to the Minimum Weighted Vertex Cover (MWVC) problem? In what cases can the substrate MWVC problem be solved efficiently?

Weighted Constraint Satisfaction Problems (WCSP)

- Extension of traditional Constraint Satisfaction Problems (CSPs) where each constraint has an associated weight or cost.
- Goal: Find an assignment of values to variables that minimize the sum of weights of violated constraints.

Graph Theory Problems Modeled as WCSPs

1. **Graph Coloring as WCSP:**
    - Variables: Nodes in the graph.
    - Domain: Different colors.
    - Constraints: Adjacent nodes must have different colors,
    - Weights: The weight associated with a constraint represents the cost of having adjacent nodes with the same color.
    - Objective: Minimize total cost of conflicting color assignments.
2. **Maximum Clique Problem as WCSP:**
    - Variables: Vertices in the graph.
    - Domain: Boolean values (in or not in the clique).
    - Constraints: Subset forms a clique.
    - Objective: Maximize the size of the clique.
    - Weights: The weights associated with constraints may represent the cost of excluding certain vertices from the clique.

MAP Query on a Bayesian Network (BN) as WCSP

- The Maximum A Posteriori (MAP) query on a Bayesian Network involves finding the most probable assignment of values to a subset of variables given observed evidence. MAP query on a BN is equivalent to a WCSP.
- Constraints are negative log-probabilities associated with observed evidence and BN structure.
- Goal: Minimize negative log-probabilities, equivalent to maximizing joint probability

Translating WCSP on Boolean Variables to Minimum Weighted Vertex Cover (MWVC) Problem

- A WCSP on Boolean variables can be translated to the Minimum Weighted Vertex Cover (MWVC) problem in graph theory.
- Given a WCSP with Boolean variables, where:
  - Each variable corresponds to a node.
  - Each constraint corresponds to an edge.
- **Goal**: Find a minimum-weight subset of nodes (vertices) such that each constraint (edge) is satisfied.

Efficient Solution of Substrate MWVC Problem

The substrate MWVC problem, where the goal is to find the minimum-weight vertex cover in a graph, can be efficiently solved in special cases, such as:

1. **Bipartite Graphs:** For bipartite graphs, the MWVC problem can be solved efficiently using algorithms like the Hopcroft–Karp algorithm.
2. **Special Graph Structures:** In certain graph structures, such as trees or graphs with low treewidth, specialized algorithms exist to find the minimum-weight vertex cover efficiently.
3. **Sparse Graphs:** When the graph is sparse, it has relatively few edges compared to the number of vertices, algorithms like approximation algorithms or dynamic programming approaches can provide efficient solutions.

In general MWVC is NP Hard while efficiency in solving the MWVC problem often depends on the specific characteristics of the graph, and exploiting these characteristics can lead to faster algorithms for finding the minimum-weight vertex cover.

Solution 3

What is the Variable Interaction (VI) graph associated with a combinatorial problem? Explain how the Variable Elimination (VE) algorithm uses the VI graph to solve the combinatorial problem via Dynamic Programming. In this context, explain the notion of the treewidth of the VI graph. Is computing the treewidth NP-hard? In what cases is it easy?

Variable Interaction (VI) Graph

- Represents dependencies among variables in a combinatorial problem.
- **Nodes**: Variables.
- **Edges**: Indicate interactions or dependencies between variables.

Variable Elimination using VI Graph

- Construct VI graph based on common participation among variables with a complete factorized form.
- Choose an elimination order depending on the VI graph, then eliminate variable one-by-one, whilst updating the factorized form.
- Dynamic programming is used here to store and compute the intermediate results and reused when encountered similar sub-problem.

- The process repeated until left with only one edge.

Tree Width in VI Graph

- Measures how close the VI graph has a "tree-like" structure.
- $Treewidth = size\ of\ the\ largest\ clique\ (subgraph) - 1$
  in the graph obtained by adding edges to form cliques in VI graph.

Computing Treewidth

In the context of combinatorial problems, the VE algorithm exploits the structure of the problem represented by a factor graph or a Bayesian network. The efficiency of VE is closely related to the treewidth of the Variable Interaction (VI) graph associated with the problem.

- Low Treewidth:
  - VE algorithm tends to be more efficient.
  - It allows for more effective dynamic programming strategies.
  - Reduces the computational complexity of the algorithm.
- High Treewidth:
  - The efficiency of the VE algorithm decreases.
  - The computation may become more challenging, and in the worst case, when the treewidth is high, VE might exhibit exponential complexity.

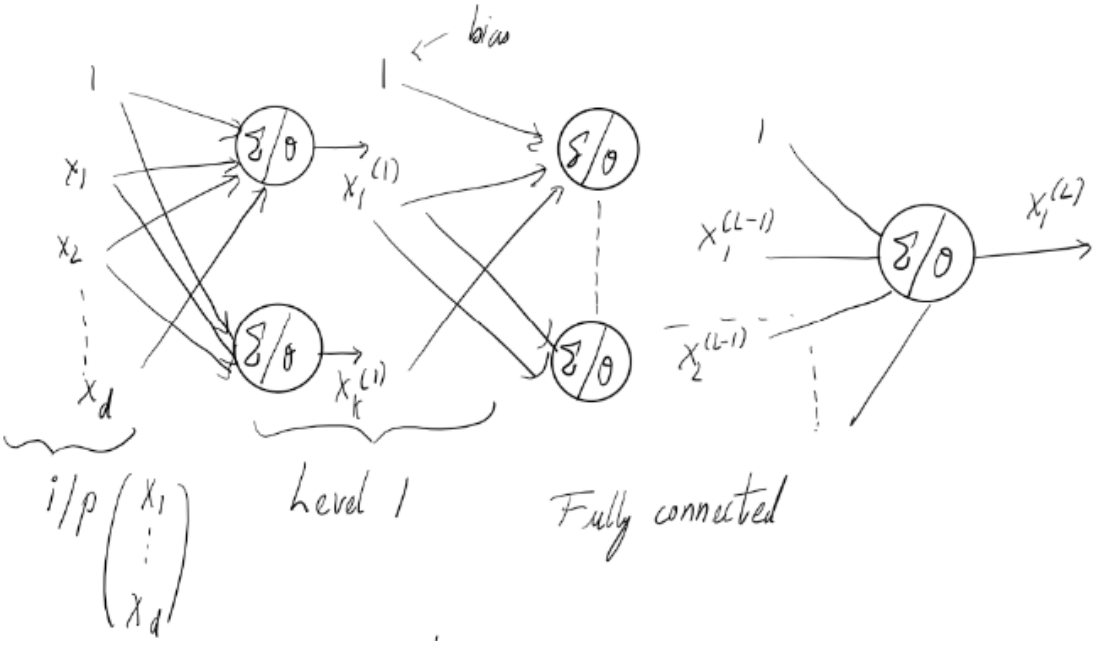
In general, it is known to be NP-hard problem but there exist a few cases where it can be handled efficiently:

- Sparse graphs – graph with relatively few edges
- Graph with known structure – Since graph structure is known, it is comparatively easier to find the treewidth would be manageable.
- Bounded treewidth cases – These graphs have a treewidth that is constant.
- Graphs with Low-Degree Nodes – When the maximum degree of nodes in the graph is small, algorithms for tree width computation might perform better.

Solution 4

Describe the architecture of a Feed-Forward Neural Network (NN). Explain the Backpropagation (BP) algorithm used to train it. What are the key algorithmic principles used in BP? Pick a problem of interest in Data Science that can be solved efficiently using a Feed-Forward NN and BP. Describe the problem and the application of these techniques on it.

Architecture



A Feed-Forward Neural Network consists of an input layer, one or more hidden layers, and an output layer. Each layer contains neurons, and connections between neurons are represented by weighted edges.

- **Input Layer:** Neurons representing input features.
- **Hidden Layers:** Layers between the input and output layers, each comprising multiple neurons that apply non-linear transformations to the input.
- **Output Layer:** Neurons representing the network's final output.

Backpropagation Algorithm

Trains the feed forward neural network by adjusting the weights to minimize the error difference between true and predicted outputs.

1. Forward pass
    a. Input is fed forwards through the network, and transformations are applied at every layer to generate predictions.
    b. Neurons in each layer applies an activation function to weight summation of inputs from the previous layer.
2. Backward pass
    a. Error calculated at the output layers is propagated backwards through the network.
    b. Contribution on each to error is computed using chain rule calculus.
    c. Using an optimization technique, weights are adjusted based on computed errors to reduce the overall error.

Key Algorithmic Principles

1. Chain rule – Used to calculate the gradient of the error function with respect to network weights by backpropagation.
2. Gradient Descent (Optimization technique) – Adjusts the weights in the direction that minimizes the error by iteratively updating the weights based on the negative gradient of the error function.

Application

Problem: Predicting Video Game High Scores

Application: Predicting a player's high score in a video game based on gameplay data.

- **Description:** The problem involves predicting a player's high score in a video game based on various gameplay-related features. Features might include playing time, number of levels completed, power-ups collected, enemies defeated, etc.
- **Application of FFNN and BP:** Use a Feed-Forward Neural Network with multiple hidden layers to learn the complex relationship between gameplay features and high scores.
- **Training:** The network is trained using gameplay data, where each instance consists of features (playing time, levels completed, etc.) and the corresponding high score.
- **Backpropagation:** The Backpropagation algorithm adjusts the weights and biases of the network to minimize the difference between predicted high scores and actual high scores.
- **Evaluation:** The trained network's performance is evaluated on a separate dataset to assess its ability to predict high scores accurately based on gameplay features.

Solution 5

What is Polynomial Identity Checking (PIC)? Describe the randomized algorithm standardly used for PIC. Pick a problem of interest in Data Science that can be solved efficiently using PIC. Describe the problem and the application of PIC on it. What are heavy-tailed distributions and where do they occur? Describe a situation where the runtime behavior of an algorithm on a combinatorial problem can exhibit a heavy-tailed distribution. In such a situation, what can be done to boost the performance of the algorithm?

Polynomial Identity Checking (PIC)

- Is a computational problem concerned with determining the equality of two given polynomials *p* and *q* in *n* variables. The goal is to ascertain if,

  $$p(x_1, x_2, \ldots, x_n) = q(x_1, x_2, \ldots, x_n)$$

  holds true for all potential values of $x_1, x_2, \ldots, x_n$.

- PIC arises frequently in scenarios where verifying the equality of polynomials is pivotal.
- The challenge lies in determining this equivalence without explicitly expanding or simplifying the polynomials, especially when dealing with high-dimensional and high-degree polynomials where direct computation may lead to an exponential number of terms.
- PIC encompasses the fundamental task of confirming whether two polynomials express identical relationships across their variables, addressing complexities inherent in verifying equality in multivariate polynomial expressions without resorting to exhaustive computation.

Randomized Algorithm for PICs

Univariate Case

- **Approach:** Randomize the algorithm by picking an element uniformly at random from a set S $(subset\ of\ field\ F\ with\ size\ \geq\ 2d)$.

- **Evaluation:** Plug the random element into the polynomial's black box.
- **Output Logic:** If the output is 0, return "≡ 0"; otherwise, output "not ≡ 0".
- **Probability Analysis:**
  - For $p\ not\ \equiv\ 0$, Probability of outputting "≡0" is $\leq\ d/|\ S\ |\leq\ 1/2$.
  - If $p \equiv 0$, the algorithm outputs "≡0" with high probability.
  - Mitigation for Unlikely Cases: In rare cases where *p* is identically 0 in a specific field but not in *Z*, using multiple fields can address this issue.

Multivariate Case (Schwartz Zippel Lemma)

- **Observation:** Multivariate polynomials may have infinitely many roots over infinite-sized fields.
- **Algorithm:** Works effectively for large enough fields ($|\ F\ | \geq 2d$).
- **Steps:**
  1. Arbitrarily pick *S* with $|\ S\ | \geq 2d$ from field *F*.
  2. Randomly choose $x_1, x_2, \ldots, x_n$ from *S*.
  3. If $p(x_1, x_2, \ldots, x_n) = 0$, output "≡0"; otherwise, output "not ≡ 0".
- **Complexity:** Needs only $O(1)$ evaluations of *p* on numbers of size $O(log\ d)$.
- **Claim Verification:** Proven through induction, ensuring efficient evaluation based on the degree of polynomials.

Behavior of Algorithm

- For $p \equiv 0$: Algorithm outputs "≡0".
- For $p\ not \equiv 0$: Probability of outputting "not ≡0" is ≥ 1/2.
- **Improving Probability**: Enhanced by repeating the algorithm or choosing a larger field size to minimize false identifications for non-zero polynomials.

Application in Data Science

Problem: Determining Functional Dependencies in Datasets

Application: Using PIC to verify functional dependencies between attributes in a dataset.

- **Description:** In databases and data analysis, functional dependencies determine relationships between attributes. For example, in a dataset, knowing that "A determines B" helps in understanding correlations.
- **Application of PIC:** Represent functional dependencies as polynomial identities and use PIC to check if these identities hold true for various attribute values.
- **Procedure:** Convert functional dependencies into polynomial equations and apply the randomized algorithm for PIC to verify if these equations hold for different sets of attribute values.
- **Outcome:** Confirmation of consistent relationships between attributes helps in data preprocessing, normalization, and optimization in databases.

Heavy-Tailed Distributions

- "Heavy Tailed" Distributions are those whose tails are exponentially bounded – they decay more slowly than those of normal distributions.
- Are characterized by higher probability values for extreme events or outliers.
- Many real-world phenomena have been found to follow heavy tailed distributions.
  - Earthquakes
  - Stock market fluctuations
  - Distribution of wealth. One percent of the population owns 40% of wealth.
  - File sizes in computer systems
  - Connection durations
  - Network traffic

- Web pages sizes

Runtime Behavior of Algorithm on Combinatorial Problems

- Example Situation: In graph algorithms like random walks or Monte Carlo methods for solving graph problems, the runtime behavior can exhibit a heavy-tailed distribution.
- Scenario: Random walks might encounter rare, but long, paths in a graph, causing certain executions to take significantly longer due to encountering rare events or large components.

Boosting Algorithm Performance

To enhance algorithm performance in situations with heavy-tailed distributions:

- **Sampling Techniques**: Use sophisticated sampling methods that reduce the probability of encountering extreme events.
- **Heuristics**: Employ heuristic methods or intelligent sampling strategies that prioritize exploring regions of the problem space likely to yield more informative or representative samples.
- **Algorithm Modification**: Tailor the algorithm to mitigate the impact of extreme events or outliers by adapting the sampling strategy dynamically.