Homework 2

Kevin Yang - 50244152

October 20, 2021

Link to repo:
https://github.com/keviny2/CPSC532W-Assignments/tree/main/HW2

1 Code Snippets

1.1 evaluation_based_sampling.py

```python
functions = {}

def evaluate_program(orig_ast):
    """Evaluate a program as desugared by daphne, generate a sample from the prior
    Args:
        ast: json FOPPL program
    Returns: sample from the prior of ast
    """

    variable_bindings = {}

    if type(orig_ast[0]) is list and orig_ast[0][0] == 'defn':
        function_expression = orig_ast[0][2:]
        functions[orig_ast[0][1]] = function_expression
        ast = orig_ast[1]
    else:
        ast = orig_ast[0]
    return evaluate_program_helper(ast, variable_bindings)


def evaluate_program_helper(ast, variable_bindings):

    # print(ast)

    if type(ast) is not list:
        if ast in math_operations:
            return ast
        if ast in data_structure_operations:
            return ast
        if ast in matrix_operations:
            return ast
        if ast in complex_operations:
            return ast
        if ast in my_distributions:
            return ast
        if type(ast) is torch.Tensor:
            return ast
        if type(ast) in [int, float]:
            return torch.tensor(ast)
        if ast in list(variable_bindings.keys()):
            return variable_bindings[ast]
        if ast in list(functions.keys()):
            return functions[ast]
        if ast is None:
            return None
        else:
            raise RuntimeError('Invalid Function', ast)
```

```
    if type(ast) is list:
        if ast[0] == 'let':
            # evaluate the expression that the variable will be bound to
            binding_obj = evaluate_program_helper(ast[1][1], variable_bindings)

            # the variable name is found in let_ast[1][0]
            # update variable_bindings dictionary
            variable_bindings[ast[1][0]] = binding_obj

            # evaluate the return expression
            return evaluate_program_helper(ast[2], variable_bindings)
        if ast[0] in my_distributions:
            curr = [evaluate_program_helper(elem, variable_bindings) for elem in ast]
            return Distribution(dist_type=curr[0], params=curr[1:])
        if ast[0] in math_operations:
            curr = [evaluate_program_helper(elem, variable_bindings) for elem in ast]
            return evaluate_math_operation(curr)
        if ast[0] in data_structure_operations:
            curr = [evaluate_program_helper(elem, variable_bindings) for elem in ast]
            return evaluate_data_structure_operation(curr)
        if ast[0] in complex_operations:
            curr = [evaluate_program_helper(elem, variable_bindings) for elem in ast]
            return evaluate_complex_operation(curr)
        if ast[0] in matrix_operations:
            curr = [evaluate_program_helper(elem, variable_bindings) for elem in ast]
            return evaluate_matrix_operation(curr)
        if ast[0] in list(functions.keys()):
            inputs = [evaluate_program_helper(elem, variable_bindings) for elem in ast[1:]]
            body = functions[ast[0]]

            for idx, param in enumerate(body[0]):
                variable_bindings[param] = inputs[idx]

            return evaluate_program_helper(body[1], variable_bindings)
```

1.2 graph_based_sampling.py

```
def sample_from_joint(graph):
    "This function does ancestral sampling starting from the prior."

    g = Graph(graph[1]['V'])
    for key, values in graph[1]['A'].items():
        for child in values:
            g.addEdge(key, child)
    sampling_order = g.topologicalSort()

    for vertex in sampling_order:
        # substitute parent nodes with their sampled values
        raw_expression = graph[1]['P'][vertex]
        variable_bindings = graph[1]['Y']
        expression = substitute_sampled_vertices(raw_expression, variable_bindings)

        graph[1]['Y'][vertex] = deterministic_eval(expression)

    # substitute return nodes with sampled values
    raw_expression = graph[2]
    variable_bindings = graph[1]['Y']
    expression = substitute_sampled_vertices(raw_expression, variable_bindings)
    return deterministic_eval(expression)
```

```
import primitives
from graph import Graph
from tests import is_tol, run_prob_test,load_truth
from utils import load_ast, substitute_sampled_vertices

# Put all function mappings from the deterministic language environment to your
# Python evaluation context here:
env = {'normal': dist.Normal,
       'beta': dist.Beta,
       'exponential': dist.Exponential,
       'uniform': dist.Uniform,
       'discrete': dist.Categorical,
       '+': primitives.add,
       '*': primitives.multiply,
       '-': primitives.minus,
       '/': primitives.divide,
       'sqrt': torch.sqrt,
       'vector': primitives.vector,
       'sample*': primitives.sample,
       'observe*': primitives.observe,
       'hash-map': primitives.hashmap,
       'get': primitives.get,
       'put': primitives.put,
       'first': primitives.first,
       'second': primitives.second,
       'rest': primitives.rest,
       'last': primitives.last,
       'append': primitives.append,
       '<': primitives.less_than,
       '>': primitives.greater_than,
       'mat-transpose': primitives.mat_transpose,
       'mat-tanh': primitives.mat_tanh,
       'mat-mul': primitives.mat_mul,
       'mat-add': primitives.mat_add,
       'mat-repmat': primitives.mat_repmat,
       'if': primitives.conditional
       }


def deterministic_eval(exp):
    "Evaluation function for the deterministic target language of the graph based representation."
    if type(exp) is list:
        op = exp[0]
        args = exp[1:]
        return env[op](*map(deterministic_eval, args))
    elif type(exp) is int or type(exp) is float:
        # We use torch for all numerical objects in our evaluator
        return torch.tensor(float(exp))
    elif type(exp) is torch.Tensor:
        return exp
    else:
        raise("Expression type unknown.", exp)
```

2 Unit Tests

2.1 Evaluation based

```
C:\Users\kevin\miniconda3\envs\cpsc532w\python.exe "C:\Program Fi
Connected to pydev debugger (build 212.4746.96)
Test passed
Test passed
Test passed
Test passed
Test passed
Test passed
Test passed
Test passed
Test passed
Test passed
Test passed
C:\Users\kevin\OneDrive - The University Of British Columbia\Docu
  ast[i] = torch.tensor(ast[i])
C:\Users\kevin\OneDrive - The University Of British Columbia\Docu
  return torch.tensor(normal.Normal(self.params[0], self.params[1
Test passed
All deterministic tests passed
('normal', 5, 1.4142136)
p value 0.16500575029224607
Test passed
C:\Users\kevin\OneDrive - The University Of British Columbia\Docu
  return torch.tensor(beta.Beta(self.params[0], self.params[1]).s
('beta', 2.0, 5.0)
p value 0.6070512707743787
Test passed
C:\Users\kevin\OneDrive - The University Of British Columbia\Docu
  return torch.tensor(exponential.Exponential(self.params[0]).sam
('exponential', 0.0, 5.0)
p value 0.8017816954390596
Test passed
('normal', 5.3, 3.2)
p value 0.7885079698489363
Test passed
C:\Users\kevin\OneDrive - The University Of British Columbia\Docu
  return torch.tensor(uniform.Uniform(self.params[0], self.params
('normalmix', 0.1, -1, 0.3, 0.9, 1, 0.3)
p value 0.6140139772600066
Test passed
('normal', 0, 1.44)
p value 0.3383791169877566
Test passed
All probabilistic tests passed
```

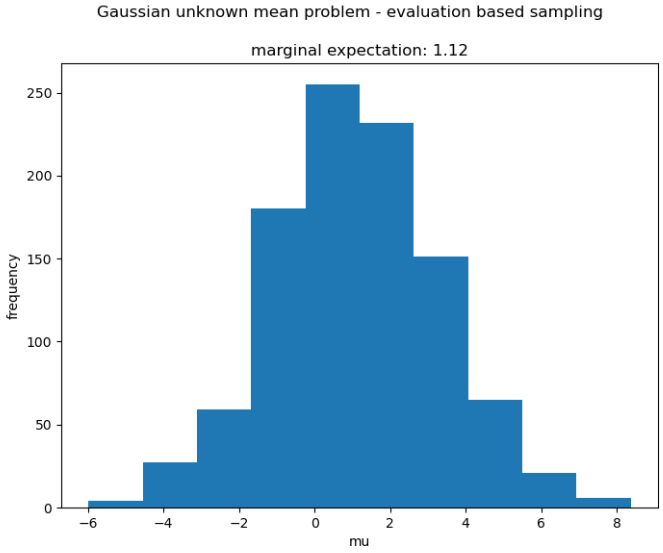
2.2 Graph based

```
C:\Users\kevin\miniconda3\envs\cpsc532w\python.exe
Connected to pydev debugger (build 212.4746.96)
Test passed
Test passed
Test passed
Test passed
Test passed
Test passed
Test passed
Test passed
Test passed
Test passed
Test passed
All deterministic tests passed
C:\Users\kevin\OneDrive - The University Of Britis
  ast[i] = torch.tensor(ast[i])
('normal', 5, 1.4142136)
p value 0.8735142546507202
Test passed
('beta', 2.0, 5.0)
p value 0.13524821964664047
Test passed
('exponential', 0.0, 5.0)
p value 0.017292256006750224
Test passed
('normal', 5.3, 3.2)
p value 0.40779311331076884
Test passed
('normalmix', 0.1, -1, 0.3, 0.9, 1, 0.3)
p value 0.17894277813922865
Test passed
('normal', 0, 1.44)
p value 0.9457802847585154
Test passed
All probabilistic tests passed
```

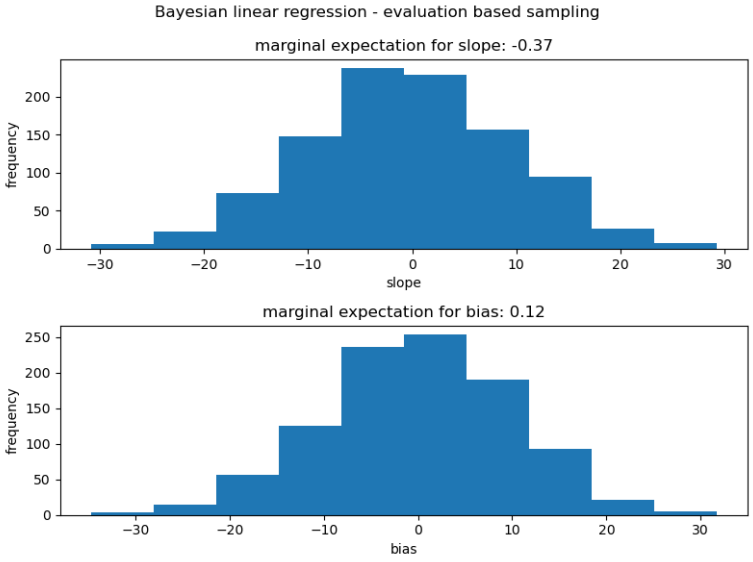
3 Plots

3.1 Evaluation based

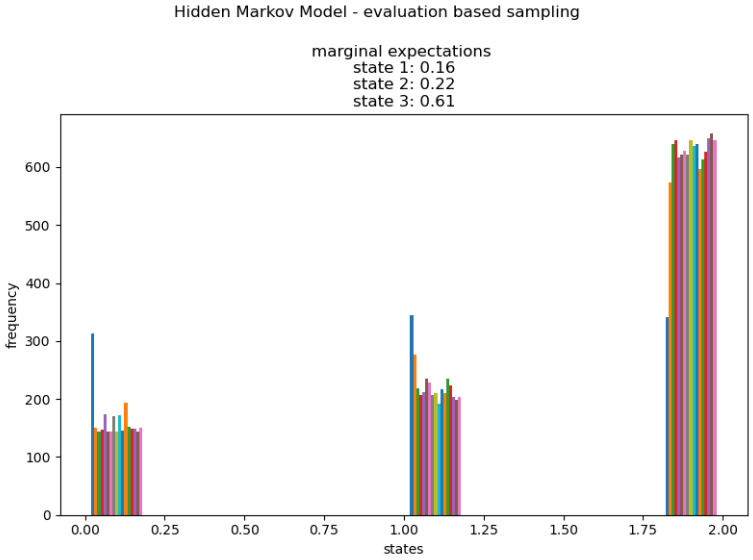
3.1.1 Gaussian unknown mean



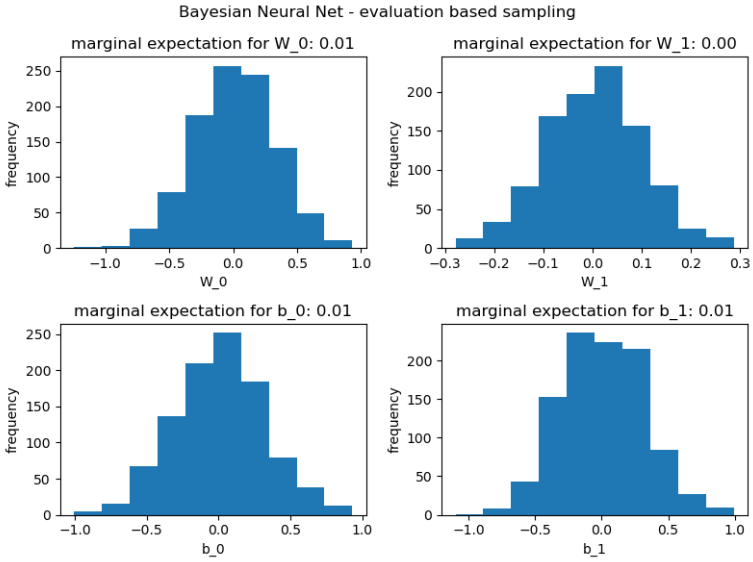
3.1.2 Bayesian linear regression problem



3.1.3 Hidden Markov Model

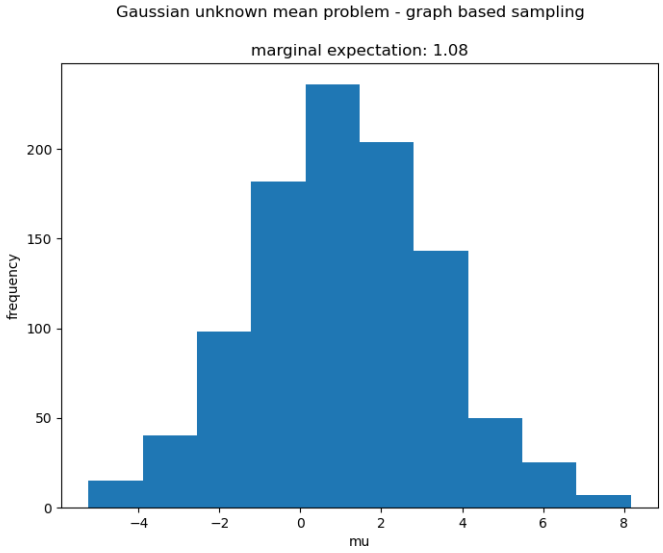


3.1.4 Bayesian Neural Network Learning

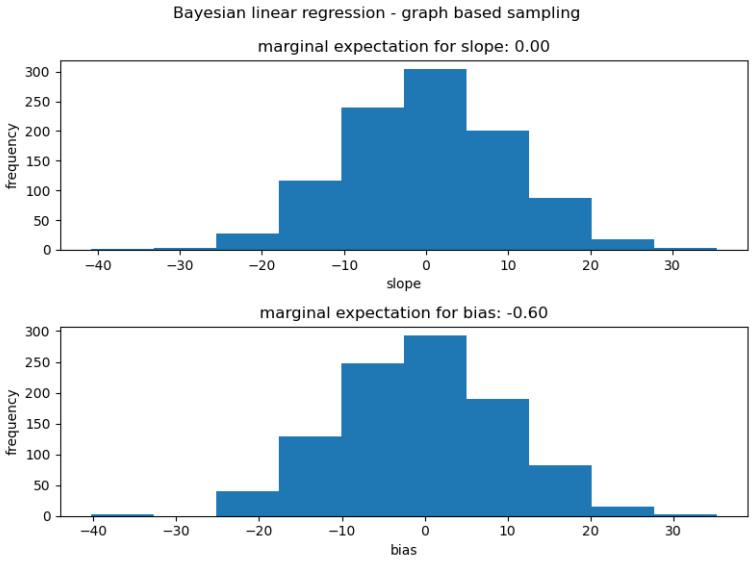


3.2 Graph based

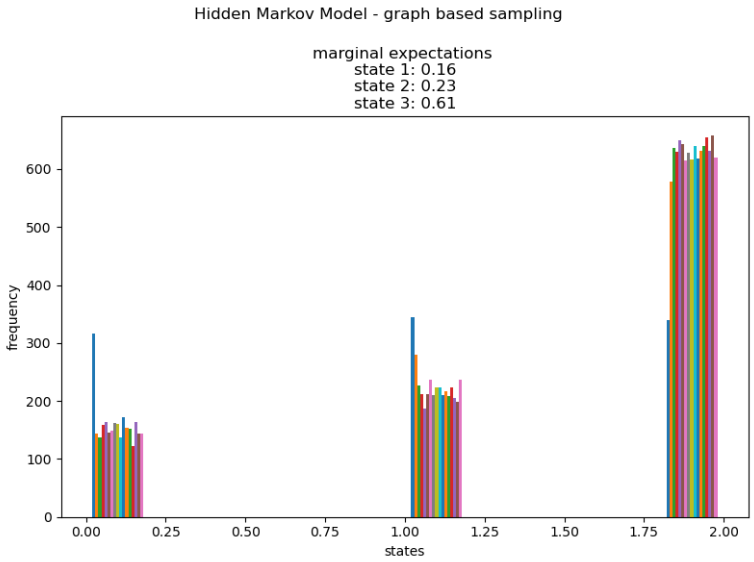
3.2.1 Gaussian unknown mean



3.2.2 Bayesian linear regression problem



3.2.3 Hidden Markov Model



3.2.4 Bayesian Neural Network Learning