CS420 Introduction to Artificial Intelligence
Assignment 1

Bharat Gangwani

Question 1

1. True

$A \rightarrow C \rightarrow E$ Blocked by cascade

$A \rightarrow B \rightarrow D \rightarrow C \rightarrow E$ Blocked by cascade

$A \rightarrow B \rightarrow D \rightarrow G \leftarrow E$ Blocked by v-structure

2. False

The third trail in 1. becomes active.

3. True

$F \rightarrow D \rightarrow G \leftarrow E$ Blocked by v-structure

$F \rightarrow D \rightarrow C \rightarrow E$ Blocked by cascade

$F \rightarrow D \leftarrow B \leftarrow A \rightarrow C \rightarrow E$ Blocked by cascade

4. True

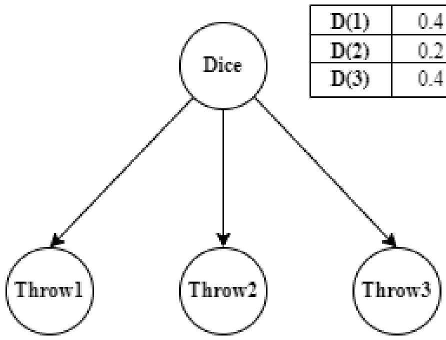
$B \leftarrow A \rightarrow C \rightarrow E$ Blocked by common cause

$B \rightarrow D \rightarrow C \rightarrow E$ Blocked by cascade

$B \rightarrow D \rightarrow G \leftarrow E$ Blocked by cascade and v-structure

Question 2

(i)



D(1)	0.4
D(2)	0.2
D(3)	0.4

i=[1,3]	D(1)	D(2)	D(3)
T-i(1)	0.3333	0.2	0.25
T-i(2)	0.3333	0.4	0.5
T-i(3)	0.3333	0.4	0.25

Variable Name	Domain	Interpretation
Dice (D)	1, 2, 3	The dice picked from the box
Throw1 (T-1)	1, 2, 3	Outcome of the first dice throw
Throw2 (T-2)	1, 2, 3	Outcome of the second dice throw
Throw3 (T-3)	1, 2, 3	Outcome of the third dice throw

(ii)

Table 1: Probability distribution for $Dice$

Dice(1)	**Dice(2)**	**Dice(3)**
0.4	0.2	0.4

Table 2: Conditional probability distribution $Throw1|Dice$

	Dice(1)	**Dice(2)**	**Dice(3)**
Throw1(1)	0.3333	0.2	0.25
Throw1(2)	0.3333	0.4	0.5
Throw1(3)	0.3333	0.4	0.25

Table 3: Conditional probability distribution $Throw2|Dice$

	Dice(1)	**Dice(2)**	**Dice(3)**
Throw2(1)	0.3333	0.2	0.25
Throw2(2)	0.3333	0.4	0.5
Throw2(3)	0.3333	0.4	0.25

Table 4: Conditional probability distribution $Throw3|Dice$

	Dice(1)	**Dice(2)**	**Dice(3)**
Throw3(1)	0.3333	0.2	0.25
Throw3(2)	0.3333	0.4	0.5
Throw3(3)	0.3333	0.4	0.25

(iii)

Dice 3 (D3) is the most likely dice. We can show this by calculating $P(D_i|T_1 = 2 \cap T_2 = 1 \cap T_3 = 2) \forall i$. I'll shorten $T_i = k$ to T_i for conciseness.

$$
\begin{aligned}
P(D_i|T_1 = 2 \cap T_2 = 1 \cap T_3 = 2) &= \frac{P(D_i \cap T_1 \cap T_2 \cap T_3)}{P(T_1 \cap T_2 \cap T_3)} \\
P(T_1 \cap T_2 \cap T_3) &= 0.4(0.3333)^3 + 0.2^2 \cdot 0.4^2 + 0.4(0.5^2 \cdot 0.25) \\
&= 0.0462 \\
\implies P(D_1|T_1 \cap T_2 \cap T_3) &= \frac{0.4 \cdot (0.3333)^3}{0.0462} = 0.3206 \\
\implies P(D_2|T_1 \cap T_2 \cap T_3) &= \frac{0.4^2 \cdot 0.2^2}{0.0462} = 0.1385 \\
\implies P(D_3|T_1 \cap T_2 \cap T_3) &= \frac{0.4 \cdot 0.5^2 \cdot 0.25)}{0.0462} = 0.5410
\end{aligned}
$$

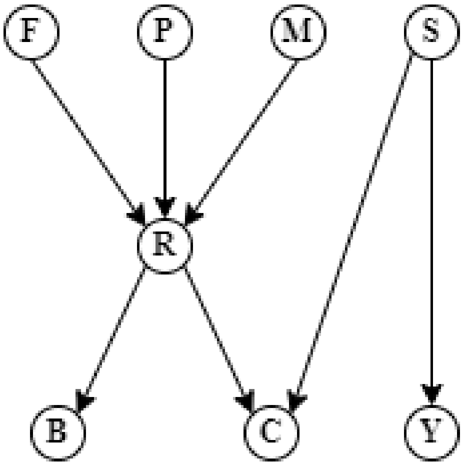
Question 3

```
model = BayesianNetwork([("S", "Y"), ("S", "C"), ("F", "R"),
    ("M", "R"), ("P", "R"), ("R", "C"), ("R", "B")])

pdP = TabularCPD("P", 2, [[0.001], [0.999]])
pdS = TabularCPD("S", 2, [[0.9], [0.1]])
pdF = TabularCPD("F", 2, [[0.999], [0.001]])
pdM = TabularCPD("M", 2, [[0.2], [0.8]])
cpdB = TabularCPD("B", 2, [[0.99, 0.1], [0.01, 0.9]], ["R"],
    [2])
cpdR = TabularCPD("R", 2, [[0.99, 0.8, 0.7, 0.5, 0.8, 0.6, 0.7,
    0.4], [0.01, 0.2, 0.3, 0.5, 0.2, 0.4, 0.3, 0.6]], ["F",
    "M", "P"], [2, 2, 2])
cpdC = TabularCPD("C", 2, [[0.9, 0.4, 0.8, 0.1], [0.1, 0.6,
    0.2, 0.9]], ["S", "R"], [2, 2])
cpdY = TabularCPD("Y", 2, [[0.9, 0.1], [0.1, 0.9]], ["S"], [2])

model.add_cpds(pdS, pdF, pdM, pdP, cpdB, cpdR, cpdC, cpdY)
```

1



2

```python
infer = VariableElimination(model)
dist = infer.query(["R"], {"C": 1}, show_progress=False)
print(dist)
```

R	P(R)
R(0)	0.1819
R(1)	0.8181

3

```python
dist = infer.query(["C"], {"S": 1}, show_progress=False)
print(dist)
```

C	P(C)
C(0)	0.4921
C(1)	0.5079

4

Yes, since there is only a single trail from S to P: $S \to C \leftarrow R \leftarrow P$ which is blocked at C as a result of v-structure.

5

```
dist = infer.query(["C"], {"P": 0}, show_progress=False)
print(dist)
```

$P(C = 1|P = 0) = 0.2359$

Question 4

```
from matplotlib import pyplot as plt
import tensorflow as tf
from tensorflow.keras import layers
```

a)

```
import tensorflow_datasets as tfds

## write your code here
[train, test], info = tfds.load("binary_alpha_digits", split =
    ["train[:60%]", "train[60%:]"], with_info=True)

## write your code here to split data into 60% train 40% test
import numpy as np

train_images = np.array(list(map(lambda x:x["image"].numpy(),
    train)))
train_labels = np.array(list(map(lambda x:x["label"].numpy(),
    train)))
test_images = np.array(list(map(lambda x:x["image"].numpy(),
    test)))
```

```
test_labels = np.array(list(map(lambda x:x["label"].numpy(),
    test)))
```

b)

```
model = tf.keras.Sequential()

## write your code here to build your dense ANN
model = tf.keras.Sequential()
model.add(layers.Flatten(input_shape=(20, 16, 1)))
model.add(layers.Dense(1000, activation=tf.nn.relu))
model.add(layers.Dense(1000, activation=tf.nn.relu))
model.add(layers.Dense(36, activation=tf.nn.softmax))
```

c)

36 layers and softmax activation function

d)

sparse_categorical_crossentropy loss function

```
### write your code here to compile model
model.compile(optimizer="Adam",
             loss='sparse_categorical_crossentropy',
             metrics=['accuracy'])

### write your code here to train your model
epochs = 50
history = model.fit(train_images, train_labels, epochs=epochs,
    verbose = 0)
```

e)

```python
### write your code to plot training loss (hint: use history)

fig = plt.figure()
ax = fig.gca()
ax.plot([i for i in range(1, 51)], history.history["loss"])
ax.set_title("Loss over Epochs")
ax.set_xlabel("Epochs")
ax.set_ylabel("Loss")

plt.show()
```

```python
#### write your code to report overall accuracy on test set

result = model.evaluate(test_images, test_labels, verbose = 0)
print(f"Overall accuracy: {result[1]:.2%}")

### write your code to report per-class accuracy
### you have a list where index is the class label with value
    corresponding to accuracy for that class label
test_pred = model.predict(test_images)
output = np.array(list(map(lambda x, y: (np.where(x == max(x))
    == y)[0, 0], test_pred, test_labels)))
classAccuracy = []

for i in range(36):
    classArray = output[np.where(test_labels == i)[0]]
    accuracy = sum(classArray)/classArray.size
    classAccuracy.append(accuracy)
```

f)

Overall accuracy: 74.20%

Class accuracies: [0.1765, 0.8235, 0.7692, 0.9286, 0.7647, 0.7143, 0.8889, 0.9091, 0.75, 0.9333, 0.9167, 0.8182, 0.9375, 0.7333, 0.8235, 0.8261, 0.25, 0.8125, 0.8333, 0.8333, 0.5625, 0.8125, 0.8, 0.5556, 0.3529, 0.8235, 0.75, 0.7619, 0.5455, 0.8462, 0.7, 0.7895, 0.6667, 0.9167, 0.7059, 0.8636]

Question 5

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from keras import Model

from keras.applications.mobilenet_v2 import preprocess_input
from keras.applications.mobilenet_v2 import decode_predictions
from keras.datasets import cifar10

import cv2

import sys
import numpy as np
import csv
import math

import matplotlib.pyplot as plt

# Import dataset

# Class names for different classes
class_names = ['airplane', 'automobile', 'bird', 'cat',
    'deer','dog', 'frog', 'horse', 'ship', 'truck']

# Load training data, labels; and testing data and their true
    labels
(train_images, train_labels), (test_images, test_labels) =
    cifar10.load_data()
print ('Training data size:', train_images.shape, 'Test data
    size', test_images.shape)

# Visualise dataset
%matplotlib inline
#Show first 25 training images below
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i], cmap=plt.cm.binary)
```

```
    plt.xlabel(class_names[train_labels[i][0]])

plt.show()

# Preprocess images
# Normalize pixel values between -1 and 1

train_images = preprocess_input(train_images)
test_images = preprocess_input(test_images)

# Resize images
# Upsize all training and testing images to 96x96 for use with
    mobile net
from configparser import Interpolation

truncateSize = 35000
minSize = 96 #minimum size requried for mobileNetV2
# You may use cv2 package. Look for function:
#"cv2.resize(<originalImage>, dsize=(minSize, minSize),
    interpolation=cv2.INTER_AREA)"
# resize train image: You can first initialize a numpy array
    resized_train_images to store all the resized training
    images
resized_train_images = np.zeros((truncateSize, minSize,
    minSize, 3), dtype=np.float32)
# <Write code for resizing>
for i, image in enumerate(train_images[0:truncateSize]):
    resized_train_images[i] = cv2.resize(image, (minSize,
        minSize), interpolation = cv2.INTER_LANCZOS4)

# resize test image: You can first initialize a numpy array
    resized_test_images to store all the resized test images
resized_test_images = np.zeros((10000, minSize, minSize, 3),
    dtype=np.float32)
# <Write code for resizing>
for i, image in enumerate(test_images):
    resized_test_images[i] = cv2.resize(image, (minSize,
        minSize), interpolation = cv2.INTER_LANCZOS4)

# a), b) Download base_model
#<Write code for downloading MobileNetV2>

base_model = tf.keras.applications.MobileNetV2(
```

```
    input_shape=[minSize, minSize, 3],
    alpha=1.0,
    include_top=False,
    weights="imagenet",
    input_tensor=None,
    pooling=None
)

base_model.trainable = False

# c) Add custom layers
#<Write code for adding custom layers>

inputs = tf.keras.Input(shape = [minSize, minSize, 3])
model = base_model(inputs, training = False)
model = layers.Flatten()(model)
model = layers.Dense(10, activation='softmax')(model)
model = tf.keras.Model(inputs=inputs, outputs= model)

model.summary()

# d)

# Training, compiling and fitting the model
model.compile(optimizer=tf.optimizers.Adam(learning_rate=0.001),
loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

# Run the stochastic gradient descent for specified epochs
epochs = 25
batch_size = 64
history = model.fit(resized_train_images,
    train_labels[0:truncateSize], batch_size=batch_size,
    epochs=epochs)

# Save the model
model_json = model.to_json()
with open("model.json", "w") as json_file:
    json_file.write(model_json)
# serialize weights to HDF5
model.save_weights("model.h5")

# load json and create model
```

```python
json_file = open('model.json', 'r')
loaded_model_json = json_file.read()
json_file.close()
loaded_model =
    tf.keras.models.model_from_json(loaded_model_json)
# load weights into new model
loaded_model.load_weights("model.h5")

loaded_model.compile(optimizer=tf.optimizers.Adam(learning_rate=0.001),
         loss='sparse_categorical_crossentropy',
         metrics=['accuracy'])

test_loss, test_acc =
    loaded_model.evaluate(resized_test_images, test_labels)
print(f'Test accuracy: {test_acc:.2%}')

# Loss over epochs
### write your code to plot training loss (hint: use history)

fig = plt.figure()
ax = fig.gca()
ax.plot([i for i in range(1, 26)], history.history["loss"])
ax.set_title("Loss over Epochs")
ax.set_xlabel("Epochs")
ax.set_ylabel("Loss")

plt.show()
```

---

1) I added a single flatten layer followed by the output layer on top of the MobileNetV2. Within the output layer, I used the softmax activation function with 10 nodes.

2) I trained the parameters of the output layer for 25 epochs as that seemed to give me a sufficient accuracy for the training and test datasets. I tested batch sizes 32 and 64 with 64 giving me better results as well. I tried increasing the batch size further but was met with RAM failures. The learning rate was held constant at 0.001.

3) 84.41%