9.1

LSTM contains four different types of gates, the forget gate, the input gate, the output gate, the gate gate. The forget gate decides which information should be discarded from the cell state. It looks at the current input and the previous hidden state and outputs a number from 0 and 1 for each number in the cell state. The input gate decides what values in the cell state are updated. The gate gate decides which information is carried across. The output gate decides what the next hidden state should be, it looks at the current input and the previous hidden state and decides what to output between -1 and 1.

9.2a.

ResNet each path bypasses n nodes and connects the input and output of n sequential layers.

The total depth of the network is N = mn + 2.

Base Case: L_0 to L_{mn+1} The entire network is a single feedforward network without any skipping

With skip connections, each skip connection creates an alternative path bypassing n nodes and since there are m skip connections to either skip or not then we get 2^m different paths.

9.2b.

Path length is determined by the number nodes it passes.
If a path uses no skips that then length is mn + 2, which fits the formula 2 + kn for k = 0
If all identity paths are used, the length of the feedforward network is 2, which fits the formula 2 + kn when k = m
If some of the identity paths are used, it bypasses kn nodes. The total number of nodes is mn + 2, the length being so the length of such a network will be mn + 2 - kn, which simplifies to (m - k)n + 2. And since m - k is just another way of stating the number of skips not used, it is consistent with the formula 2 + kn

Therefore, for k = 0, 1, … m there are $\binom{m}{k}$ feedforward networks of length 2 + kn.

9.2c.

To find the average we sum the lengths of all paths and divide by the number of paths. Each length 2 + kn has $\binom{m}{k}$ paths. So the total length summed over all paths is

$$\sum_{k=0}^{m} \binom{m}{k}(2+kn)$$

which we divide the sum by the total number of paths to

$$\frac{\sum_{k=0}^{m} \binom{m}{k}(2+kn)}{2^m}$$

get

With m = 3 and n = 2, we then get 5 nodes as the average length.

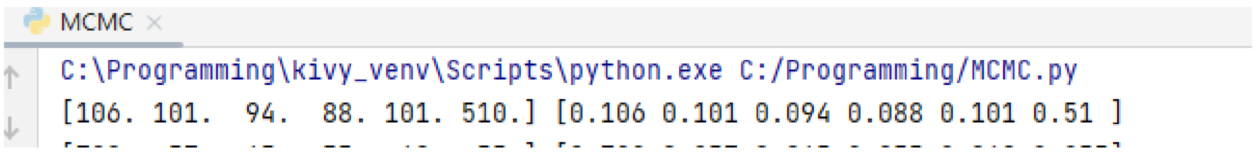
9.7.

So we first define the parameters θd for the discriminative model λd and θg for the generative model λg which include the initial, transition and observation probabilities. For several iterations we train the λd by using the set of observation sequences, some form training data and some generated by λg. The objective is for λd to maximize the probability of classifying real and generated sequences performing gradient ascent on the log likelihood of the correct classifications.

After updating λd, we update λg by generating observation sequences using the current parameters. λg is trained to maximize the probability that the λd incorrectly classifies the generated sequences as real by performing gradient ascent on the log likelihood of the incorrect classifications.

We set up a stopping condition for the training process by using either iterations or the performance or the convergence of the models' parameters. We finally return the final parameters of θd and θg

9.9a.

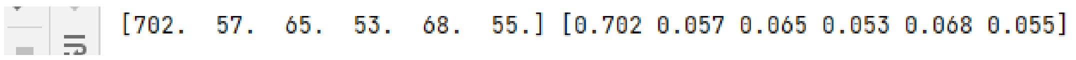


```
MCMC
C:\Programming\kivy_venv\Scripts\python.exe C:/Programming/MCMC.py
[106. 101.  94.  88. 101. 510.] [0.106 0.101 0.094 0.088 0.101 0.51 ]
```

Yes, this looks correct since we except 1-5 to be similar in probabilities while probability of the 6th die would be around 50%

Code in Folder

9.9b.



```
[702.  57.  65.  53.  68.  55.] [0.702 0.057 0.065 0.053 0.068 0.055]
```

Yes, this looks correct since we except 2-6 to be similar in probabilities while probability of the 1th die would be around 75%

Code in Folder

9.10a.

P(Cx = 2 |xt = 1) = 1
P(Cx = 5 |xt = 6) = 1
P(Cx = k |xt = k + 1) = ½ for k = 1,2,...,5
P(Cx = k-1 |xt = k) = ½ for k = 2,3…,6

9.10b.

P(xt = 1 |Cx = 2) = 1
P(xt = 6 |Cx = 5) = 1
P(xt = k + 1 |Cx = k) = ½ for k = 1,2,...,5
P(xt = k |Cx = k -1) = ½ for k = 3,4,5,6

P(xt = 2 |Cx = 1) = 1 can’t go down from 1

9.10c.

Given that we're sampling from a fair die, the stationary probabilities are equal for all states, so P(Cx) = P(xt) = 1/6 for any x. So we just need consider the cases for xt within {2,3,4,5} and {1,6}

So For xt ∈ {2,3,4,5}
Here, the proposal can be to either go up or down by one. For example, if xt = 3, then Cx can be either 2 or 4, and the proposal probabilities are symmetric

P(Cx= 2 |xt = 3) = ½
P(xt= 3 |Cx = 2) = ½

u <= ½ / ½ = 1

Since *u* is a uniform random number between 0 and 1, it will always be less than or equal to 1. Therefore, for xt ∈ {2,3,4,5}, we always accept the proposal Cx.

And For xt ∈ {1,6}

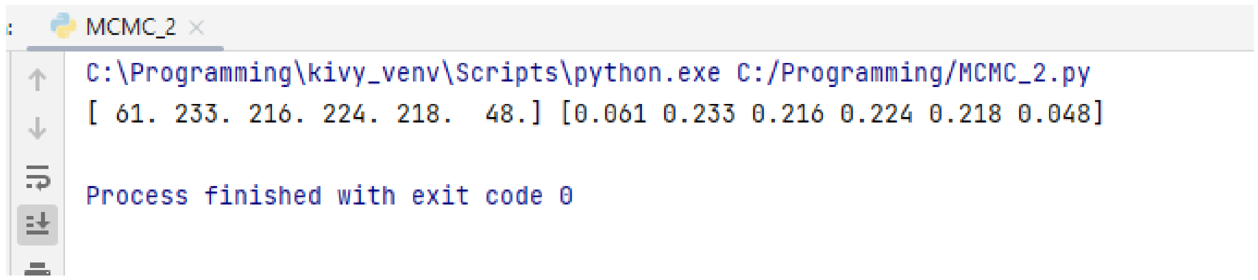
Here, the proposals are not symmetric because you can't go down from 1 or up from 6.
If xt = 1, then Cx is always 2, so P(Cx = 2 |xt = 1) = 1, but P(xt = 1 | Cx = 2) = ½

If xt = 6, then Cx is always 5, so P(Cx = 5 |xt = 6) = 1, but P(xt = 6 | Cx = 5) = ½

u <= ½ / 1 = ½

This means that for xt ∈ {1,6}, we only accept the proposal Cx with probability ½ because *u* will be less than or equal to ½ half of the time.

9.10d.



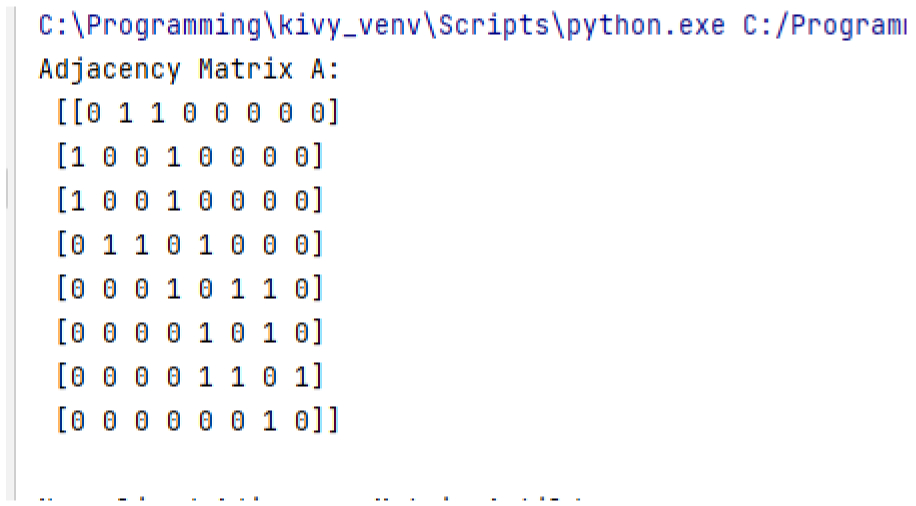
```
C:\Programming\kivy_venv\Scripts\python.exe C:/Programming/MCMC_2.py
[ 61. 233. 216. 224. 218.  48.] [0.061 0.233 0.216 0.224 0.218 0.048]

Process finished with exit code 0
```

Yes, this looks correct since we expect 2-5 to be similar in probabilities while probability of the 1th die and the 6th die would have similar probabilities. The probabilities of 1 and 6 should be about ¼ of the other probabilities which are shown above.

Code in Folder

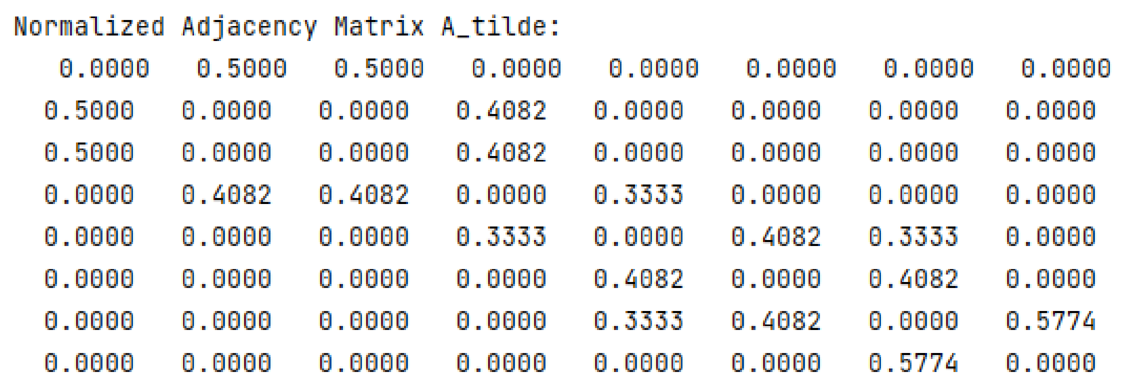
9.14a.



```
C:\Programming\kivy_venv\Scripts\python.exe C:/Program
Adjacency Matrix A:
 [[0 1 1 0 0 0 0 0]
 [1 0 0 1 0 0 0 0]
 [1 0 0 1 0 0 0 0]
 [0 1 1 0 1 0 0 0]
 [0 0 0 1 0 1 1 0]
 [0 0 0 0 1 0 1 0]
 [0 0 0 0 1 1 0 1]
 [0 0 0 0 0 0 1 0]]
```

Code in Folder

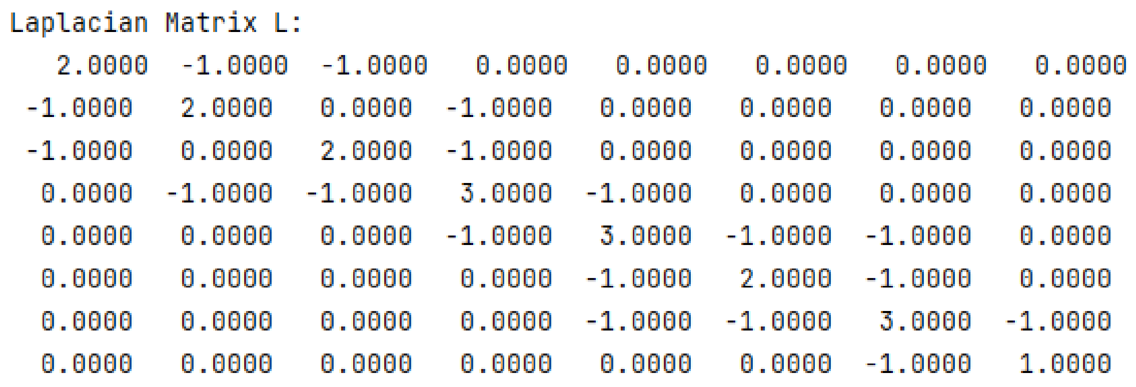
9.14b.



```
Normalized Adjacency Matrix A_tilde:
    0.0000   0.5000   0.5000   0.0000   0.0000   0.0000   0.0000   0.0000
   0.5000   0.0000   0.0000   0.4082   0.0000   0.0000   0.0000   0.0000
   0.5000   0.0000   0.0000   0.4082   0.0000   0.0000   0.0000   0.0000
   0.0000   0.4082   0.4082   0.0000   0.3333   0.0000   0.0000   0.0000
   0.0000   0.0000   0.0000   0.3333   0.0000   0.4082   0.3333   0.0000
   0.0000   0.0000   0.0000   0.0000   0.4082   0.0000   0.4082   0.0000
   0.0000   0.0000   0.0000   0.0000   0.3333   0.4082   0.0000   0.5774
   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.5774   0.0000
```

Code in Folder

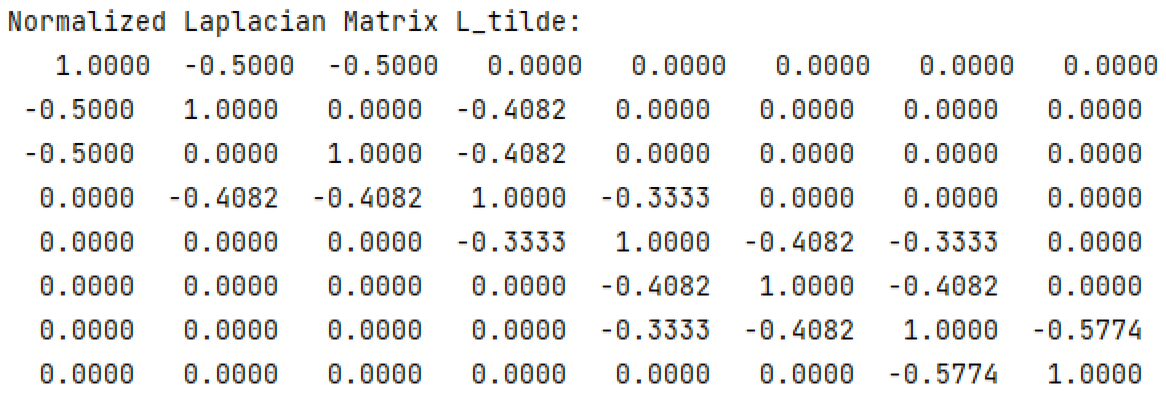
9.14c.



```
Laplacian Matrix L:
   2.0000  -1.0000  -1.0000   0.0000   0.0000   0.0000   0.0000   0.0000
  -1.0000   2.0000   0.0000  -1.0000   0.0000   0.0000   0.0000   0.0000
  -1.0000   0.0000   2.0000  -1.0000   0.0000   0.0000   0.0000   0.0000
   0.0000  -1.0000  -1.0000   3.0000  -1.0000   0.0000   0.0000   0.0000
   0.0000   0.0000   0.0000  -1.0000   3.0000  -1.0000  -1.0000   0.0000
   0.0000   0.0000   0.0000   0.0000  -1.0000   2.0000  -1.0000   0.0000
   0.0000   0.0000   0.0000   0.0000  -1.0000  -1.0000   3.0000  -1.0000
   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000  -1.0000   1.0000
```

Code in Folder

9.14d.



```
Normalized Laplacian Matrix L_tilde:
   1.0000  -0.5000  -0.5000   0.0000   0.0000   0.0000   0.0000   0.0000
  -0.5000   1.0000   0.0000  -0.4082   0.0000   0.0000   0.0000   0.0000
  -0.5000   0.0000   1.0000  -0.4082   0.0000   0.0000   0.0000   0.0000
   0.0000  -0.4082  -0.4082   1.0000  -0.3333   0.0000   0.0000   0.0000
   0.0000   0.0000   0.0000  -0.3333   1.0000  -0.4082  -0.3333   0.0000
   0.0000   0.0000   0.0000   0.0000  -0.4082   1.0000  -0.4082   0.0000
   0.0000   0.0000   0.0000   0.0000  -0.3333  -0.4082   1.0000  -0.5774
   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000  -0.5774   1.0000
```

Code in Folder

9.15a.

ResNet: It’s an architecture famous for its skip connection, which helps get rid of the vanishing gradient problem by allowing an alternate path for the gradient to flow. It’s an architecture with 152 layers, and the layers are arranged in an CNN, known for its performance on ImageNet.

9.15b.

VGG-19: It’s characterized by its simplicity using only 3x3 convolutional layers stacked on top of each other. It has 19 layers each with weights; this includes 16 convolutional layers and 3 fully connected layers. It’s known for its robustness and is widely used as feature extractor for image processing tasks.

9.15c.

Inceptionv3: Is a convolutional neural network architecture, it is 48 layers deep and uses a multi-level feature extraction method by applying kernels of different sizes on the same input map. The architecture is known for its high accuracy on the ImageNet dataset.

9.15d.
EfficientNet: Is an architecture that scales up CNNs in a structured manner. It scales each dimension with a set of fixed scaling coefficients. It uses a compound coefficient to uniformly scale network width, depth and resolution.

9.15d.

Xception: Extreme Inception is an architecture that extends Inception by replacing Inception modules with depth wise separable convolutions. The model has 71 layers and is shown to outperform Inception models on large scale image classification dataset.