

1.1 Markov Decision Processes - Dice Game


The possible states of this game include ["play", "end"] which represent the states where the game is still ongoing or the end of the game respectively. In the "play" state, the possible actions are ["quit", "stay"]. The "quit" action will transition the current state to the "end" state and provide a reward of \$10 with a transition probability of 1. The "stay" action will transition back to the "play" state and provide a reward of \$10 with a transition probability $\frac{2}{3}$. The transition probability from the "end" to the "end" state is 1 and the transition probability of "end" to "play" is 0.

2. Hidden Markov Models

To calculate the probability that an agent's actions included hitting a wall given the final score is 7, the conditional probability formula is used. $P(A|B) = P(A \& B) / P(B)$ where A is the event where the agent hit a wall and B is the event that the final score of the agent is 7. There are two paths where the agent could end up with a final score of 7 which are [UP, RIGHT, UP, RIGHT, RIGHT] and [RIGHT, RIGHT, UP, UP]. The first sequence contains an action where the agent hits a wall, and the second sequence avoids the wall. The probability of the first sequence occurring $P(A \& B) = (.4 \cdot .8 + .6 \cdot .2) \cdot (.5 \cdot .8 + .5 \cdot .2) \cdot (.8 \cdot .8) \cdot (.9 \cdot .8) = 0.101376$. The first two actions are a sum of probabilities because every action has a probability for successfully completing the action and a probability to perform the undesired action. Following the same logic, the probability of the second sequence of actions occurring is $P((\text{NOT } A) \& B) = (.6 \cdot .8 + .4 \cdot .2) \cdot (.7 \cdot .8 + .3 \cdot .2) \cdot (.9 \cdot .8) \cdot (.9 \cdot .8) = 0.17998848$. $P(B) = P(A \& B) + P((\text{NOT } A) \& B) = 0.101376 + 0.17998848 = 0.28136448$. Following the conditional probability formula, the probability that the agent took an action hitting a wall while is $P(A|B) = P(A \& B) / P(B) = 0.101376 / 0.28136448 = 0.02852360552$.

3.12 Estimate Error Rates

If the probability of corruption is 0.01 and 0.1, I think that the error rate will greatly increase if the probability of corruption increases tenfold. This is because our bigram model is trained on the probability that specific characters will come after any character, so it can get seriously messed up from incorrect characters. After testing the higher corruption probability, it looks like my intuition was correct were we saw around a tenfold increase in the error rate.

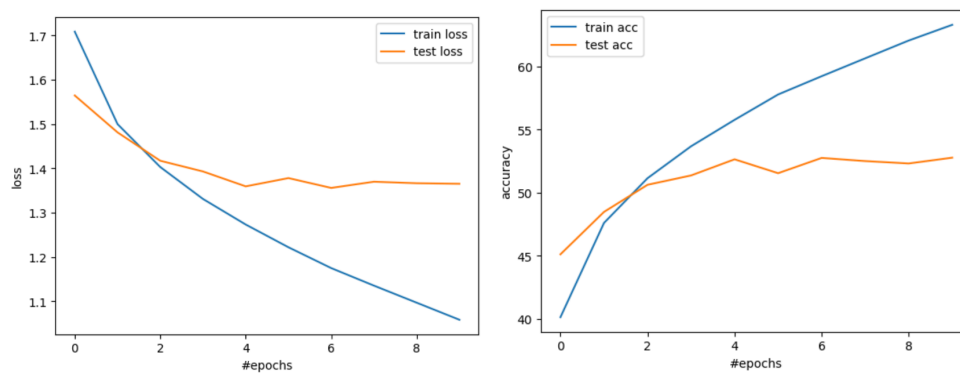
Error rate with $P_c = 0.01$:  Error rate: 0.86%

Error rate with $P_c = 0.1$:  Error rate: 8.70%

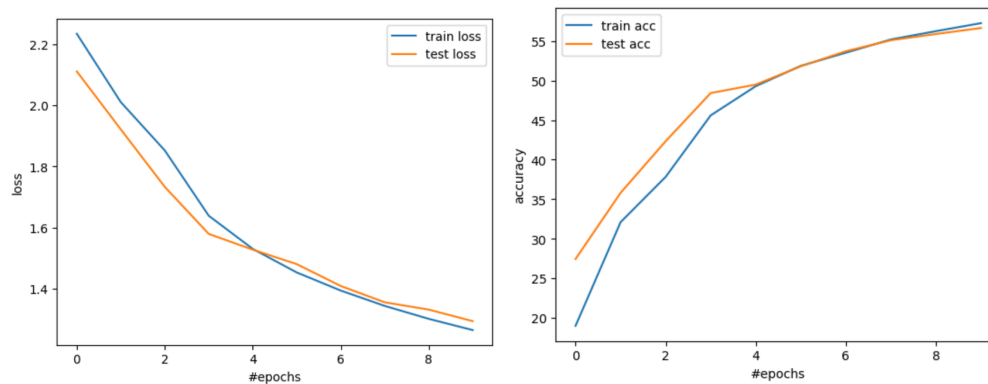
6.2 MLP vs CNN

I believe that the CNN model learns much better on the image dataset. This is because it seems like the MLP easily overfits to the data where training loss & accuracy further improve while testing loss and accuracy stop improving quickly. The MLP's final testing loss and accuracy are also worse than the final testing loss and accuracy of the CNN. Looking at the graphs below, it looks like the CNN is able to better learn key features about the data, strongly improving both the training and testing loss and accuracy as time goes on. Therefore, I believe that the CNN model learns better on the image dataset and is probably a better architecture to use for images.

MLP:



CNN:



6.3 CNN Architecture

The ResidualCNN architecture that I developed utilized three convolutional layers each followed by a relu activation layer and a max pooling layer. The output of the first convolutional layer had a residual sum connection with the output of the third convolutional layer before being passed to three fully connected linear layers that each have their own relu activation function. I tried different dimensions of hidden layers between the convolutional layers before settling on the current model due to its strong performance. This network architecture with residual connections performs better than the CNN without residual connections.

When training this model without the residual connections, the model does perform similarly to the original CNN model without residual connections. When adding significantly more layers, the residual connections have a stroger impact on model performance.

```
class ResidualCNN(nn.Module):
    """
    Feel free to experiment on CNN model.
    You only need to report the model that has the best performance on the dataset.
    """

    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 16, kernel_size=5)
        self.conv2 = nn.Conv2d(16, 32, kernel_size=5)
        self.conv3 = nn.Conv2d(32, 16, kernel_size=5, padding=10)
        self.relu = nn.ReLU()
        self.pool = nn.MaxPool2d(kernel_size=(2,2), stride=2)

        self.fc1 = nn.Linear(3136, 64)
        self.fc2 = nn.Linear(64, 32)
        self.fc3 = nn.Linear(32, 10)

    def forward(self, x):
        # residual + block 1
        residual_x = self.conv1(x)
        x = self.relu(residual_x)
        x = self.pool(x)

        # block 2
        x = self.conv2(residual_x)
        x = self.relu(x)
        x = self.pool(x)

        # residual + block 3
        x = self.conv3(x)
        x += residual_x
        x = self.relu(x)
        x = self.pool(x)

        # linear connections
        x = x.reshape(x.shape[0], -1)
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        x = self.relu(x)
        x = self.fc3(x)

        return x
```

```
training 1 epoch...: 100% |██████████| 3125/3125 [04:27<00:00, 11.69it/s]
Epoch: 1 | Runtime: 4.79[m] | train_loss: 1.743 | train_acc: 35.968 | test_loss: 1.397 | test_acc: 49.210
training 1 epoch...: 100% |██████████| 3125/3125 [04:43<00:00, 11.01it/s]
Epoch: 2 | Runtime: 9.81[m] | train_loss: 1.317 | train_acc: 52.698 | test_loss: 1.195 | test_acc: 57.890
training 1 epoch...: 100% |██████████| 3125/3125 [04:29<00:00, 11.59it/s]
Epoch: 3 | Runtime: 14.62[m] | train_loss: 1.131 | train_acc: 59.996 | test_loss: 1.090 | test_acc: 62.020
training 1 epoch...: 100% |██████████| 3125/3125 [04:27<00:00, 11.67it/s]
Epoch: 4 | Runtime: 19.40[m] | train_loss: 0.995 | train_acc: 64.802 | test_loss: 0.992 | test_acc: 65.600
training 1 epoch...: 100% |██████████| 3125/3125 [04:23<00:00, 11.87it/s]
Epoch: 5 | Runtime: 24.11[m] | train_loss: 0.896 | train_acc: 68.562 | test_loss: 0.981 | test_acc: 65.540
training 1 epoch...: 100% |██████████| 3125/3125 [04:26<00:00, 11.72it/s]
Epoch: 6 | Runtime: 28.85[m] | train_loss: 0.815 | train_acc: 71.510 | test_loss: 0.950 | test_acc: 67.640
training 1 epoch...: 100% |██████████| 3125/3125 [04:30<00:00, 11.54it/s]
Epoch: 7 | Runtime: 33.66[m] | train_loss: 0.752 | train_acc: 73.732 | test_loss: 0.900 | test_acc: 69.020
training 1 epoch...: 100% |██████████| 3125/3125 [04:27<00:00, 11.69it/s]
Epoch: 8 | Runtime: 38.41[m] | train_loss: 0.689 | train_acc: 75.790 | test_loss: 0.921 | test_acc: 68.920
training 1 epoch...: 100% |██████████| 3125/3125 [04:38<00:00, 11.20it/s]
Epoch: 9 | Runtime: 43.41[m] | train_loss: 0.643 | train_acc: 77.208 | test_loss: 0.924 | test_acc: 69.110
training 1 epoch...: 100% |██████████| 3125/3125 [04:36<00:00, 11.32it/s]
Epoch: 10 | Runtime: 48.31[m] | train_loss: 0.594 | train_acc: 79.098 | test_loss: 0.940 | test_acc: 69.310
```

```
... training 1 epoch...: 100% |██████████| 3125/3125 [04:26<00:00, 11.74it/s]
Epoch: 1 | Runtime: 4.81[m] | train_loss: 1.993 | train_acc: 25.772 | test_loss: 1.627 | test_acc: 39.360
training 1 epoch...: 100% |██████████| 3125/3125 [04:33<00:00, 11.43it/s]
Epoch: 2 | Runtime: 9.67[m] | train_loss: 1.484 | train_acc: 45.936 | test_loss: 1.432 | test_acc: 48.800
training 1 epoch...: 100% |██████████| 3125/3125 [04:28<00:00, 11.64it/s]
Epoch: 3 | Runtime: 14.44[m] | train_loss: 1.263 | train_acc: 54.670 | test_loss: 1.168 | test_acc: 58.700
training 1 epoch...: 100% |██████████| 3125/3125 [04:27<00:00, 11.69it/s]
Epoch: 4 | Runtime: 19.20[m] | train_loss: 1.099 | train_acc: 60.912 | test_loss: 1.057 | test_acc: 62.210
training 1 epoch...: 100% |██████████| 3125/3125 [04:27<00:00, 11.70it/s]
Epoch: 5 | Runtime: 23.97[m] | train_loss: 0.986 | train_acc: 65.226 | test_loss: 0.970 | test_acc: 65.610
training 1 epoch...: 100% |██████████| 3125/3125 [04:27<00:00, 11.70it/s]
Epoch: 6 | Runtime: 28.75[m] | train_loss: 0.903 | train_acc: 68.344 | test_loss: 0.937 | test_acc: 67.390
training 1 epoch...: 100% |██████████| 3125/3125 [04:28<00:00, 11.62it/s]
Epoch: 7 | Runtime: 33.56[m] | train_loss: 0.838 | train_acc: 70.698 | test_loss: 0.909 | test_acc: 69.170
training 1 epoch...: 100% |██████████| 3125/3125 [04:26<00:00, 11.71it/s]
Epoch: 8 | Runtime: 38.34[m] | train_loss: 0.782 | train_acc: 72.464 | test_loss: 0.921 | test_acc: 68.820
```