TRAINING DEEP Q-NETWORK FOR STOCHASTIC PROCESS ENVIRONMENTS

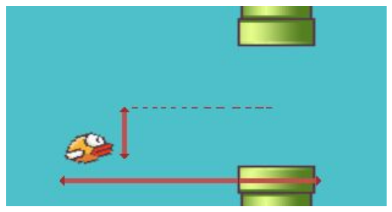
Gerald He

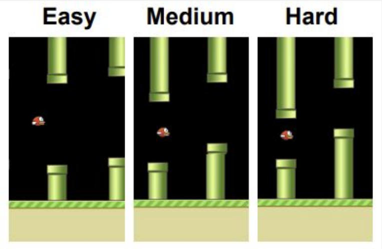
April 27th, 2023

PART 1: FLAPPY BIRD ENVIRONMENT

FLAPPY BIRD ENVIRONMENT

- 1 Observation: The position of the bird
- 2 Actions: The bird goes up and down
- Reward: +1 each time when the bird passes through the pipe
- Done: When the bird crashes at the pipe
- Note: The initial reward is 101 in the hard mode





Chen, Kevin. *2017*

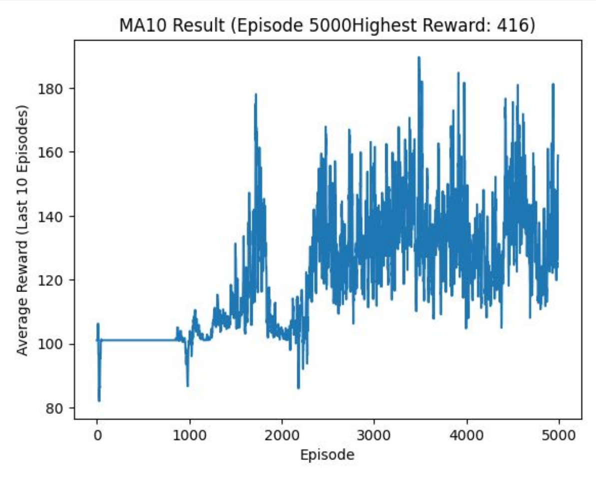
MY FIRST DQN

```python
class DQN(nn.Module):

    # DEFINE THE SPECIAL INIT METHOD
    def __init__(self, features, hidden, output):
        super(DQN, self).__init__()
        self.linear1 = nn.Linear(features, hidden[0])
        self.linear2 = nn.Linear(hidden[0], hidden[1])
        self.linear3 = nn.Linear(hidden[1], output)

    # DEFINE THE FORWARD METHOD
    def forward(self, x):
        out = F.relu(self.linear1(x))
        out = F.relu(self.linear2(out))
        y_pred = self.linear3(out)
        return y_pred
```

RESULT




```
BATCH_SIZE = 128
GAMMA = 0.99
EPS_START = 0.9
EPS_END = 0.05
EPS_DECAY = 1000
TAU = 0.005
LR = 1e-4
HIDDEN = 64
MEMORY = 50000
```

```
Episode 1: Cumulative Reward = 116
Episode 2: Cumulative Reward = 228
Episode 3: Cumulative Reward = 179
Episode 4: Cumulative Reward = 143
Episode 5: Cumulative Reward = 137
Episode 6: Cumulative Reward = 174
Episode 7: Cumulative Reward = 174
Episode 8: Cumulative Reward = 212
Episode 9: Cumulative Reward = 107
Episode 10: Cumulative Reward = 121
Episode 11: Cumulative Reward = 174
Episode 12: Cumulative Reward = 148
Episode 13: Cumulative Reward = 103
Episode 14: Cumulative Reward = 226
Episode 15: Cumulative Reward = 154
Episode 16: Cumulative Reward = 156
Episode 17: Cumulative Reward = 108
Episode 18: Cumulative Reward = 190
Episode 19: Cumulative Reward = 137
Episode 20: Cumulative Reward = 187
Average Reward over 20 Episodes = 158.70
```

The actual score is 57.70

COMPARE TO REFERENCE

Average Score

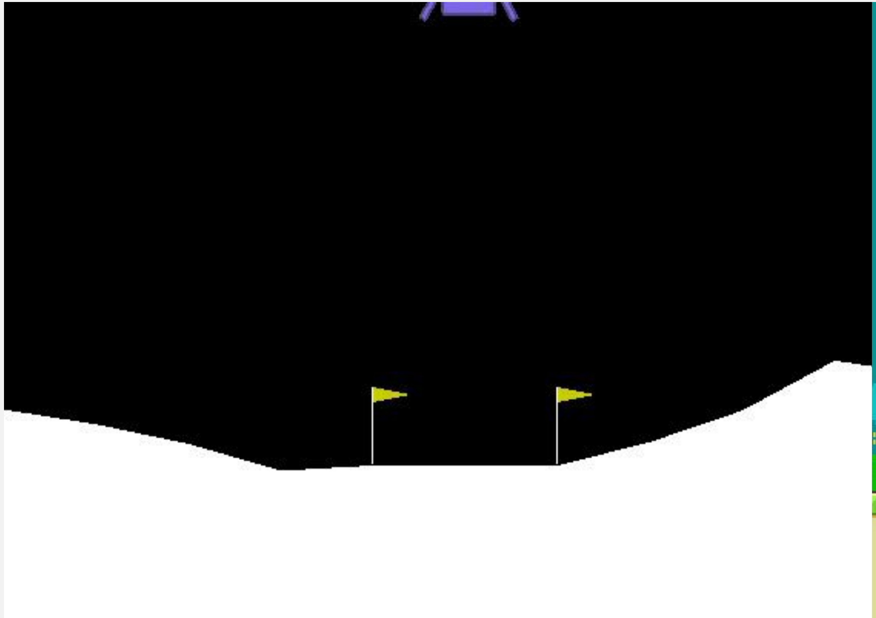
Game difficulty	Human	Baseline (flap every n)	DQN (easy)	DQN (medium)	DQN (hard)
Easy	Inf	Inf	**Inf**	Inf	Inf
Medium	Inf	Inf	0.7	**Inf**	Inf
Hard	21	0.5	0.1	0.6	**82.2**

Highest Score Achieved

Game difficulty	Human	Baseline (flap every n)	DQN (easy)	DQN (medium)	DQN (hard)
Easy	Inf	Inf	**Inf**	Inf	Inf
Medium	Inf	11	2	**Inf**	Inf
Hard	65	1	1	1	**215**

Chen, Kevin. *2017*

Any Differences Between Two Environments?



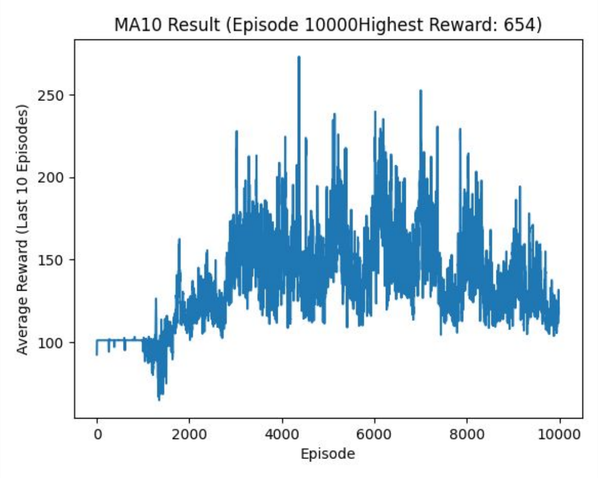
Enhanced DQN With Drop-out Layers

```python
class enhancedDQN(nn.Module):
    def __init__(self, n_features, hidden, n_actions, p=0.1):
        super(enhancedDQN, self).__init__()
        self.fc1 = nn.Linear(n_features, hidden[0])
        self.drop1 = nn.Dropout(p)
        self.fc2 = nn.Linear(hidden[0], hidden[1])
        self.drop2 = nn.Dropout(p)
        self.fc3 = nn.Linear(hidden[1], n_actions)

    def forward(self, x):
        x = F.relu(self.drop1(self.fc1(x)))
        x = F.relu(self.drop2(self.fc2(x)))
        x = self.fc3(x)
        return x

    # using the Kaiming He's initialization
    def init_weights(self):
        for m in self.modules():
            if isinstance(m, nn.Linear):
                nn.init.kaiming_uniform_(m.weight, nonlinearity='relu')
                nn.init.constant_(m.bias, 0.1)
```

BETTER OUTCOME




```
BATCH_SIZE = 256
GAMMA = 0.99
EPS_START = 0.99
EPS_END = 0.01
EPS_DECAY = 1000
TAU = 0.005
LR = 1e-4
HIDDEN = 256
MEMORY = 50000
```

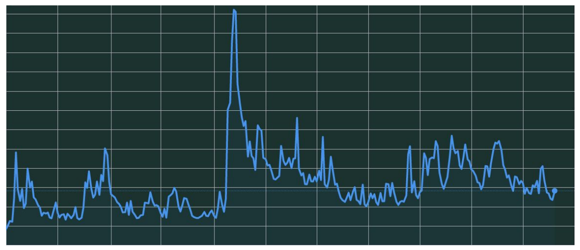
```
Episode 1: Cumulative Reward = 506
Episode 2: Cumulative Reward = 396
Episode 3: Cumulative Reward = 210
Episode 4: Cumulative Reward = 285
Episode 5: Cumulative Reward = 1394
Episode 6: Cumulative Reward = 358
Episode 7: Cumulative Reward = 1024
Episode 8: Cumulative Reward = 212
Episode 9: Cumulative Reward = 211
Episode 10: Cumulative Reward = 804
Episode 11: Cumulative Reward = 139
Episode 12: Cumulative Reward = 211
Episode 13: Cumulative Reward = 728
Episode 14: Cumulative Reward = 2320
Episode 15: Cumulative Reward = 1805
Episode 16: Cumulative Reward = 1098
Episode 17: Cumulative Reward = 876
Episode 18: Cumulative Reward = 876
Episode 19: Cumulative Reward = 1765
Episode 20: Cumulative Reward = 802
Average Reward over 20 Episodes = 801.00
```

The actual score is 701.00

PART 2: STOCHASTIC PROCESS WITH STOCK TRADING SIMULATION

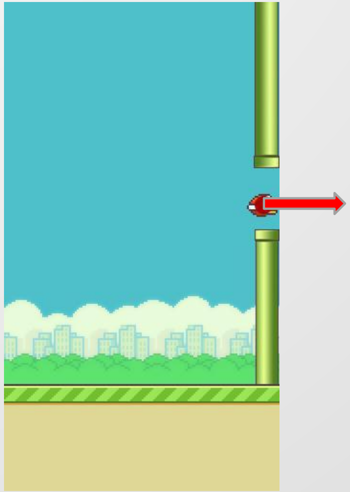
- What is the relationship between the Flappy Bird and stock price?
- A stochastic process is a mathematical model that describes the evolution of a random system over time. It involves the study of the probability distribution of a sequence of random variables.

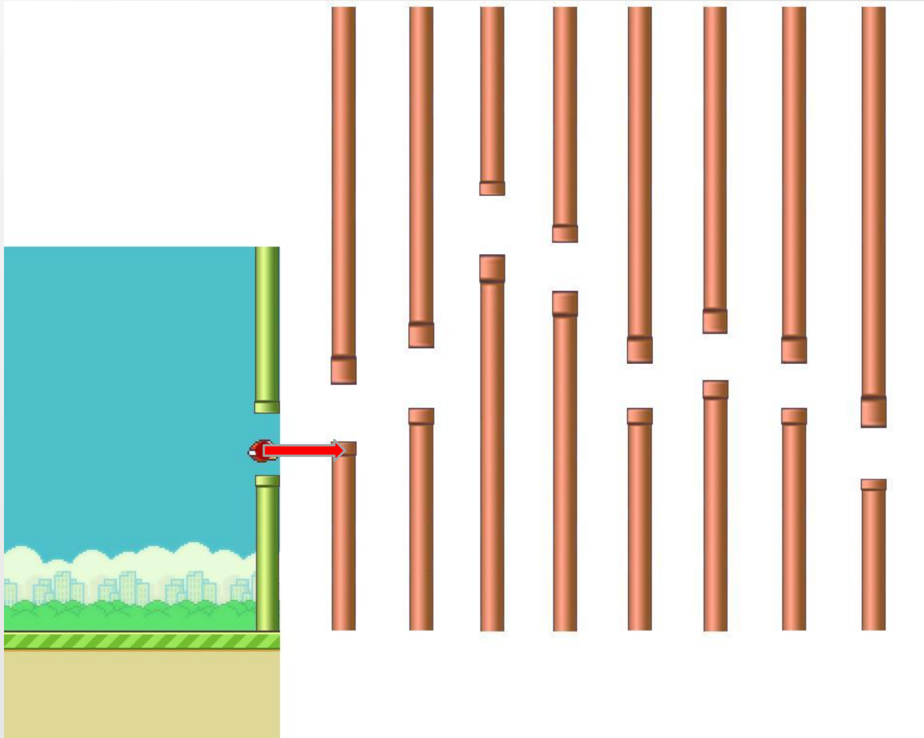
$$\frac{dS}{S} = \mu\, dt + \sigma\, dW_t.$$

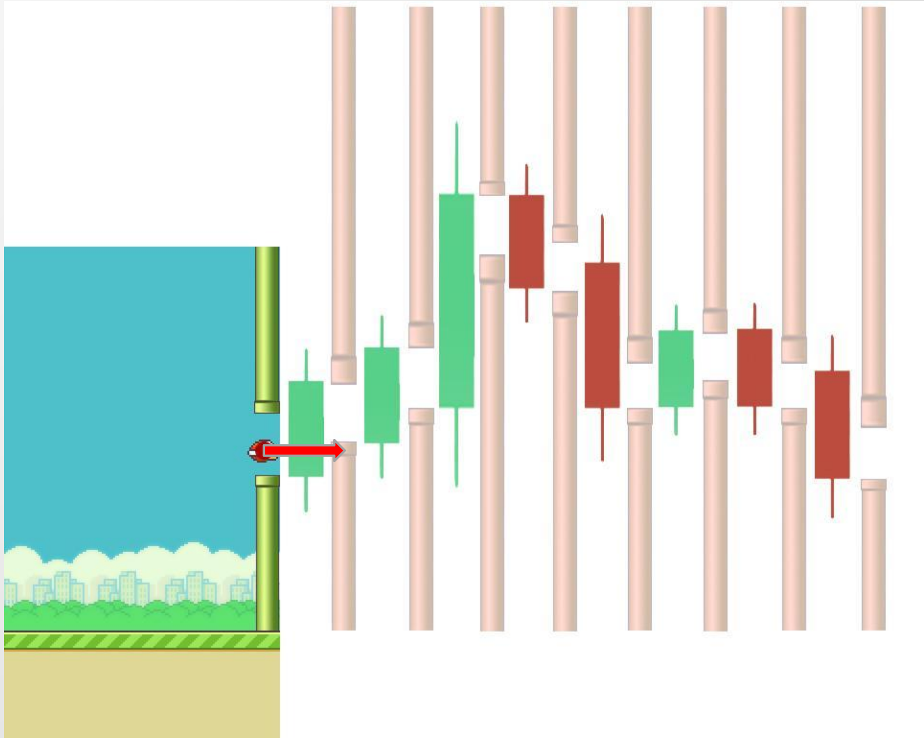


VIX Index from 2018 to Today

BACK TO FLAPPY BIRD ENVIRONMENT







FLAPPY ~~BIRD~~ STOCK ENVIRONMENT

```python
class StockBirdEnv(gym.Env):
    def __init__(self):
        # Define observation space (single continuous v
        self.observation_space = spaces.Box(low=0, high

        # Define action space (discrete with 3 actions)
        self.action_space = spaces.Discrete(3)

        # Set initial reward and step count
        self.reward = 100
        self.steps = 0

        # Initialize the figure for rendering
        self.fig = plt.figure(figsize=(5, 3), dpi=100)

    def reset(self):
        # Reset the environment to its initial state
        self.reward = 100
        self.steps = 0
        self.trajectory = np.random.normal(loc=0, scale

        # Return the initial state as an observation
        observation = np.array([0])
        self.state = observation
        return observation
```

```python
        # Calculate the distance from the ideal trajectory
        distance = np.abs(price_tick - self.trajectory[self.steps])

        # Calculate the reward based on the distance from the trajectory
        if price_tick > self.trajectory[self.steps]:
            reward = distance
        else:
            reward = -distance

        # Update the cumulative reward
        self.reward += reward

        # Update the state with the new observation
        observation = np.array([price_tick])
        self.state = observation

        # Check if the episode is over
        done = self.reward < 0 or self.steps >= 1000

        # Return the observation, reward, done flag, and info dictionary
        info = {'trajectory': self.trajectory}
        return observation, reward, done, info

    def render(self, mode='human'):
        # Clear the figure and plot the ideal trajectory and the bird's position
        self.fig.clf()
        plt.plot(self.trajectory[:self.steps+1], color='gray')
        plt.plot(self.steps, self.state[0], 'ro', markersize=10)

        # Add labels and formatting
        plt.title('Stock Bird Environment')
        plt.xlabel('Time Step')
        plt.ylabel('Price Tick')
        plt.ylim([-5, 5])
        plt.xlim([0, 1000])

        # Show the plot
        plt.show()
```

MY TRADING ENVIRONMENT

- Initial state: $1M cash balance
- 4 Observations:
  - The current price of the stock
  - The current value of the account
  - The current position of the account (how many stocks the account holds)
  - The current cash balance of the account
- 3 Actions: Buy/Sell stock or hold
- Reward: The total profit made by trading
- Done: When the time comes to an end (1700 steps total)
- Data: 20 stocks with the highest trading volume from 2017 to 2020

DQN DESIGN

```
class DQN(nn.Module):
    def __init__(self, lr, input_dims, fc1_dims, fc2_dims,  n_actions):
        super(DQN, self).__init__()
        self.lr = lr
        self.input_dims = input_dims
        self.fc1_dims = fc1_dims
        self.fc2_dims = fc2_dims

        self.n_actions = n_actions

        self.fc1 = nn.Linear(*self.input_dims, self.fc1_dims)
        self.BN1 = nn.BatchNorm1d(fc1_dims)
        self.drop1 = nn.Dropout(p)
        self.fc2 = nn.Linear(self.fc1_dims, self.fc2_dims)
        self.drop2 = nn.Dropout(p)
        self.BN2 = nn.BatchNorm1d(fc2_dims)
        self.fc3 = nn.Linear(self.fc2_dims, self.n_actions)

        self.optimizer = optim.Adam(self.parameters(), lr=lr)
        self.loss = nn.MSELoss()

        self.device = torch.device('cuda:0' if torch.cuda.is_available() else
'cpu')
        self.to(self.device)
```

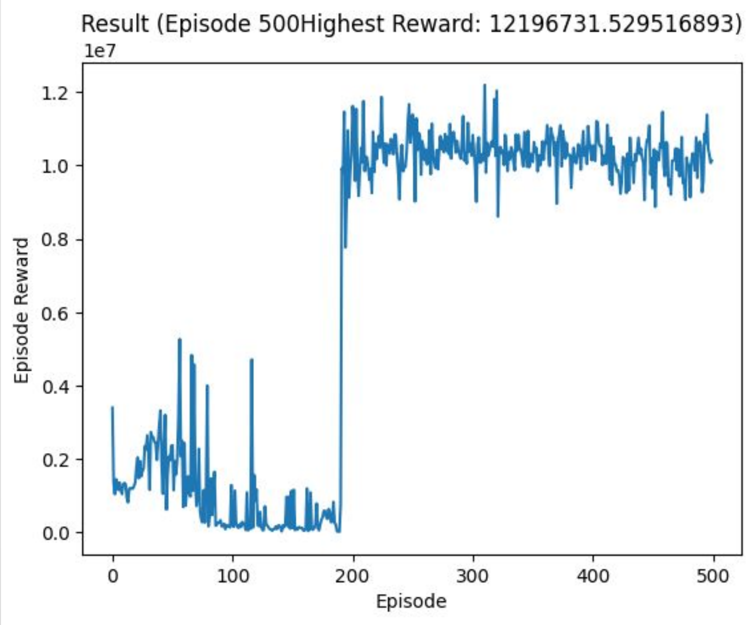
```
def forward(self, state):
    state.to(self.device)
    x = self.fc1(state.to(torch.float32))
    x = F.relu(x)
    x = self.fc2(x)
    x = F.relu(x)
    actions = self.fc3(x)
    return actions

  def init_weights(self):
    for m in self.modules():
        if isinstance(m, nn.Linear):
nn.init.kaiming_uniform_(m.weight,
nonlinearity='relu')
            nn.init.constant_(m.bias, 0.1)
```

RESULT

```
BATCH_SIZE = 256
GAMMA = 0.99
EPS_START = 0.9
EPS_END = 0.01
EPS_DECAY = 2000
TAU = 0.005
LR = 1e-4
HIDDEN = 256
MEMORY = 100000
```

```
20 episodes average profit:    10854882.156452134
```



BACKTEST CLASS

- I am still working on testing the policy in the real world using real-time data.
- Currently using the Alpha Vantage API to test the "strategy".

REFLECTION

- 1. Use convolutional layers to play the rendering environment like a human.
- 2. Improve the performance of the trading environment by exploring additional strategies and techniques.
- 3. Add more observations to the trading environment, such as trading volume, financial reports, and other relevant data.
- 4. Enable the agent to trade multiple stocks simultaneously with customized trading volumes. This will allow for more diverse and sophisticated trading strategies.

CONCLUSION

- Reinforcement learning enables researchers to create data freely and at low cost, instead of gathering and labeling data expensively as in supervised learning.
- Drop-layer, batchnorm, and enhanced activation functions make DQN more effective in high-precision environments, such as stochastic processes.
- Even in information-missing environments, a well-trained agent can still find a way to solve the problem.

CITATION AND SOURCE

- Chen, Kevin. *Deep Reinforcement Learning for Flappy Bird - Stanford University*. 2017, https://cs229.stanford.edu/proj2015/362_report.pdf.
- Chuchro, Robert, and Deepak Gupta. *Game Playing with Deep Q-Learning Using Openai Gym*. 2017, http://cs231n.stanford.edu/reports/2017/pdfs/616.pdf.
- He, Kaiming, et al. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification." *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, https://doi.org/10.1109/iccv.2015.123.

THANK YOU!

Any questions?