# Report on DQN Architecture and Hyperparameters

## Architecture

The architecture used in the DQN (Deep Q-Network) model is a simple feedforward neural network with three fully connected (linear) layers. Here are the details:

1. 1. Input Layer:

The input layer size is determined by the input\_dim, which corresponds to the number of state features. For example, in the FlappyAgent.py file, state\_dim is set to 12.

1. 2. Hidden Layer 1:

The first hidden layer (fc1) has 256 neurons. Uses the ReLU (Rectified Linear Unit) activation function to introduce non-linearity. This helps the network learn complex patterns.

1. 3. Hidden Layer 2:

The second hidden layer (fc2) has 32 neurons. Also uses the ReLU activation function.

1. 4. Output Layer:

The output layer (fc3) has a size equal to the output\_dim, which corresponds to the number of possible actions. For example, in the FlappyAgent.py file, action\_dim is set to 2.

The forward pass through the network involves applying the ReLU activation function to the outputs of the first two hidden layers and then passing the result through the output layer.

```python  
import torch  
from torch import nn  
  
class DQN(nn.Module):  
 def \_\_init\_\_(self, input\_dim, output\_dim, hidden\_dim1=256, hidden\_dim2=32):  
 super(DQN, self).\_\_init\_\_()  
  
 self.fc1 = nn.Linear(input\_dim, hidden\_dim1)  
 self.fc2 = nn.Linear(hidden\_dim1, hidden\_dim2)  
 self.fc3 = nn.Linear(hidden\_dim2, output\_dim)  
  
 def forward(self, x):  
 x = torch.relu(self.fc1(x))  
 x = torch.relu(self.fc2(x))  
 return self.fc3(x)  
```

## Hyperparameters

The following hyperparameters are used in the Agent class for training the DQN model:

1. 1. Replay Memory Size:

The maximum number of experiences stored in the replay memory. This helps in breaking the correlation between consecutive experiences.

1. 2. Minibatch Size:

The number of experiences sampled from the replay memory for each training step. This helps in stabilizing the training process.

1. 3. Epsilon (Exploration Rate) Initialization:

The initial value of epsilon for the epsilon-greedy policy. This controls the exploration-exploitation trade-off.

1. 4. Epsilon Decay:

The rate at which epsilon decays after each episode. This gradually reduces the exploration over time.

1. 5. Minimum Epsilon:

The minimum value of epsilon to ensure some level of exploration even in later stages of training.

1. 6. Network Sync Rate:

The number of steps after which the target network is updated with the policy network's weights. This helps in stabilizing the training by reducing the variance.

1. 7. Device:

The device used for computation (GPU if available, otherwise CPU).

1. 8. Learning Rate:

The learning rate for the optimizer. This controls the step size during the gradient descent optimization.

1. 9. Discount Factor (Gamma):

The discount factor for future rewards. This determines the importance of future rewards.

1. 10. Best Reward Stop:

The reward threshold to stop training if achieved. This is used as an early stopping criterion.

1. 11. Loss Function:

The loss function used to compute the difference between predicted and target Q-values. Mean Squared Error (MSE) is used here.

1. 12. Optimizer:

The optimizer used for training the network. RMSprop is used here to adapt the learning rate for each parameter.

```python  
class Agent:  
 def \_\_init\_\_(self):  
 self.replay\_memory\_size = 10000  
 self.minibatch\_size = 32  
 self.epsilon\_init = 1.0  
 self.epsilon\_decay = 0.9995  
 self.epsilon\_min = 0.05  
 self.network\_sync\_rate = 10  
 self.device = 'cuda' if torch.cuda.is\_available() else 'cpu'  
 self.learning\_rate = 0.005  
 self.discount\_factor\_g = 0.99  
 self.best\_reward\_stop = 1000  
  
 self.loss\_fn = torch.nn.MSELoss()  
 self.optimizer = None  
  
 self.MODEL\_FILE = os.path.join(os.path.dirname(\_\_file\_\_), "flappy\_model.pth")  
 self.GRAPH\_FILE = os.path.join(os.path.dirname(\_\_file\_\_), 'flappy\_model.png')  
```