```
import keras
from keras.models import Sequential
from keras.models import load_model
from keras.layers import Dense
from keras.optimizers import Adam

import numpy as np
import random
from collections import deque

class Agent:
  def __init__(self, state_size, is_eval=False, model_name=""):
    self.state_size = state_size # normalized previous days
    self.action_size = 3 # sit, buy, sell
    self.memory = deque(maxlen=1000)
    self.inventory = []
    self.model_name = model_name
    self.is_eval = is_eval

    self.gamma = 0.95  # discount factor
    self.epsilon = 1.0   # initial epsilon for greedy policy
    self.epsilon_min = 0.01  # minimum attainable epsilon
    self.epsilon_decay = 0.995  # time decrease in each episode

    self.model = load_model("models/" + model_name) if is_eval else self._model()

  def _model(self):
    model = Sequential() # load model
    model.add(Dense(units=64, input_dim=self.state_size, activation="relu")) # inpu
    model.add(Dense(units=32, activation="relu"))  # 1,0 function  # hidden layer
    model.add(Dense(units=8, activation="relu"))  # 2nd hidden layer
    model.add(Dense(self.action_size, activation="linear"))  # output layer - 3 un
    model.compile(loss="mse", optimizer=Adam(lr=0.001))  # cost function

    return model

  def act(self, state):
    if not self.is_eval and np.random.rand() <= self.epsilon:
      return random.randrange(self.action_size)  # send any random action

    options = self.model.predict(state)
    return np.argmax(options[0])   # send action having maximum value after neurat

  def expReplay(self, batch_size):
    mini_batch = []
    l = len(self.memory)
    for i in xrange(l - batch_size + 1, l):
      mini_batch.append(self.memory[i])

    for state, action, reward, next_state, done in mini_batch:
      target = reward
      if not done:
        target = reward + self.gamma * np.amax(self.model.predict(next_state)[0])
      target_f = self.model.predict(state)
      target_f[0][action] = target
      self.model.fit(state, target_f, epochs=1, verbose=0)

    if self.epsilon > self.epsilon_min:
      self.epsilon *= self.epsilon_decay
```

```
Using TensorFlow backend.
```