CSCE 580: Artificial Intelligence
Coding Homework 3: Machine Learning
Due: 4/18./2022 at 11:59pm

**Your code must run in order to receive credit.**

**Do not change the signature (including the name) of the functions provided in the file. Your code must accept the exact arguments and return exactly what is specified in the documentation.**

# Installation

We will be using the same `conda` environment as in Homework 0.

The entire GitHub repository should be downloaded again as changes were made to other files. You can download it with the green "Code" button and click "Download ZIP".

# 1 Supervised Learning (50 pts)

The MNIST dataset contains 28 by 28 dimensional images of digits with 60,000 training images and 10,000 validation images. The data has a value of either zero or one. Implement `train_nnet` to create and train a neural network on the MNIST dataset. The data will be given as a flattened 28 by 28 image (a 784 dimensional input). You can implement any neural network with any hyperparameters you would like. If you want to implement a convolutional neural network, you can reshape the flattened input to again be a 28 by 28 image. The neural network you implement must be your own (not from someone else's code, such as code online).

Do `python run_mnist_classification.py` to run the training of the neural network. After the neural network is trained it will be validated on the validation set. **Your neural network should obtain a validation accuracy of 98% or more and must train in 5 minutes or less on a CPU.**

Hint: To train your neural network for classification, see the PyTorch `CrossEntropyLoss` documentation. To use this, your neural network should have a output layer of size 10 with just a linear activation function.

## Helper Functions

`evaluate_nnet(nnet, data_input_np, data_labels_np)`: Returns the loss and accuracy on the given data. Feel free to use or modify for your purposes. This function puts the neural network in evaluation mode (`nnet.eval()`). Remember to put the neural network back in training mode with `nnet.train()`. This function shows how the input Tensor to the neural network should be float and the labels given to the criterion should be long.

## Extra Credit (0-20 pts)

Those with the highest classification accuracy with receive extra credit. Note that, to be considered, your neural network must train within the time limit.

# 2 Reinforcement Learning

**Key building blocks:**

- `env.state_action_dynamics(state, action)`: returns, in this order, the expected reward $r(s, a)$, all possible next states given the current state and action, their probabilities. Keep in mind, this only returns states that have a non-zero state-transition probability.
- `env.get_actions(state)` function that returns a list of all possible actions for that state
- `state_values`: you can obtain $V(s)$ with `state_values[state]`
- `policy`: you can obtain $\pi(a|s)$ with `policy[state][action]`
- `viz`: you can use `update_dp(viz, state_values, policy)` to visualize algorithms that use a state-value function. You can use `update_model_free(viz, state, action_values)` to visualize algorithms that use an action-value function. You can modify these functions however you like for your debugging purposes. However, when you turn in your code, please remove all calls to these functions for efficiency of grading.
- `env.sample_transition(state, action)`: returns, in this order, the next state and reward
- `env.sample_start_states(N)` function that returns N start states
- `action_vals`: you can obtain $Q(s, a)$ with `action_vals[state][action]`

Switches:

- `--env`, environment name. AI farm is `aifarm_<prob>` where `<prob>` is the probability that the wind blows you to the right. For example, `aifarm_0` is deterministic and `aifarm_0.1` is stochastic.
- `--discount`, to change the discount (default=1.0)
- `--epsilon`, to change the $\epsilon$ for the $\epsilon$-greedy policy (default=0.1)
- `--learning_rate`, to change the learning rate (default=0.1)
- `viz`: you can use `update_model_free(viz, state, action_values)` to visualize your algorithm.

## 2.1 Policy Iteration (25 pts)

Policy iteration can be used to compute the optimal value function. Policy iteration starts with a given value and policy function and iterates between policy evaluation and policy improvement until the policy function stops changing. For more information, see the lecture slides and Chapter 4 of Sutton and Barto. In this exercise, we will be finding the optimal value function using policy iteration. Terminate policy iteration when the policy stops changing. In your implementation, return the state values and the policy (in that order) found by policy iteration.

Implement the `policy_iteration` function.

**Running the code:**

```
python run_rl.py --env aifarm_<prob> --algorithm policy_iteration --discount <discount>
```

The policy starts as a uniform random policy and a value function that is zero at all states. Check your code for different values of `--discount` and `<prob>`.

## 2.2 SARSA (25 pts)

SARSA is an on-policy model-free reinforcement learning algorithm. For more information, see the lecture slides and Chapters 5 and 6 of Sutton and Barto.

**Important:** Since this is a model-free reinforcement learning algorithm, we will assume that we do not have access to the dynamics of the MDP. Therefore, in your implementation of SARSA, you cannot use `env.state_action_dynamics(state, action)`. Instead, use `env.sample_transition(state, action)`.

Implement the `sarsa` function.

**Running the code:**
`python run_rl.py --env aifarm_<prob> --algorithm sarsa --learning_rate <lr>`
`--epsilon <epsilon> --discount <discount>`

In this setting, we will be running SARSA with for 10,000 episodes. Each episode should end when the agent reaches the goal or when the number of steps taken reaches 50. Check your code for different values of `--discount`, `<prob>`, `--learning_rate`, and `--epsilon`. Your agent should perform well with a learning rate of 0.5, a discount of 1.0, and epsilon of 0.01.

# What to Turn In

Turn in your implementation of `coding_hw/coding_hw4.py` to Blackboard.