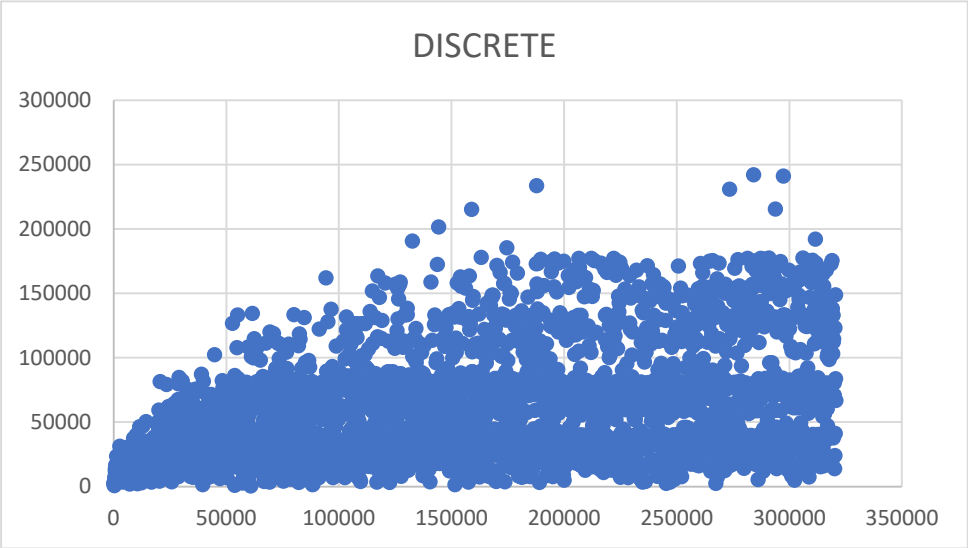
Lab3

A plot shows episode scores of at least 100,000 training episodes (10%)

In some case, 2048 could appear in first 1000 episode.

```
1000    mean = 6234.56   max = 25880
        64      100%     (1%)
        128     99%      (11.9%)
        256     87.1%    (30.3%)
        512     56.8%    (45.2%)
        1024    11.6%    (11.5%)
        2048    0.1%     (0.1%)
```

Random sample:



In the graph, we can find that the average and maximum are raising before about 200'000 episodes.

Describe the implementation and the usage of n-tuple network. (10%)

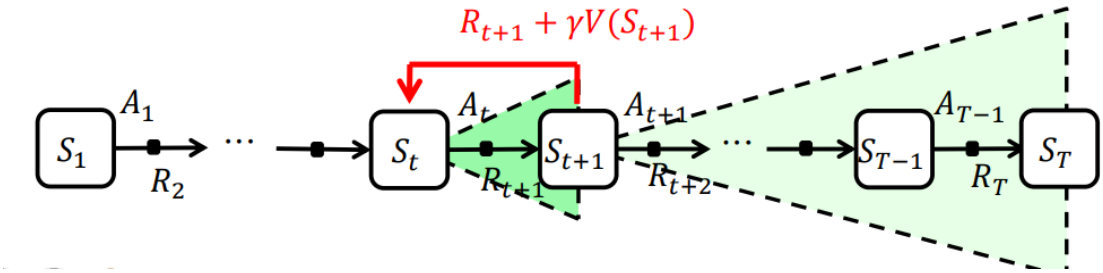
In the game, we can know that there are 16 possible value in one tile.

So that, if we generate all possible case to different state the network can't run on any computer in the world. So we use approximated way to consider some pattern as same one.

Explain the mechanism of TD(0). (5%)

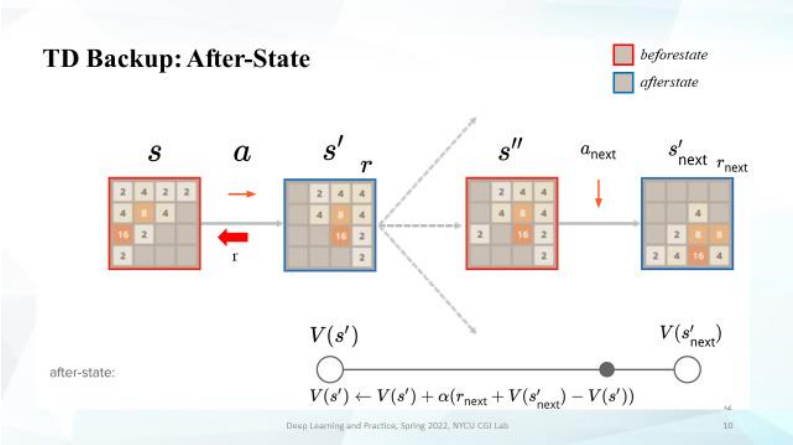
It name is "Temporal Difference Learning", it is a way we don't need to learning after an "episode", that is, it can learn at all time.

The tree graph in slide is:



It doesn't need to finish the game like MC if we have a good way to estimate the $V(S_n)$. The way is biased, but lower variance.

Explain the TD-backup diagram of V(after-state). (5%)

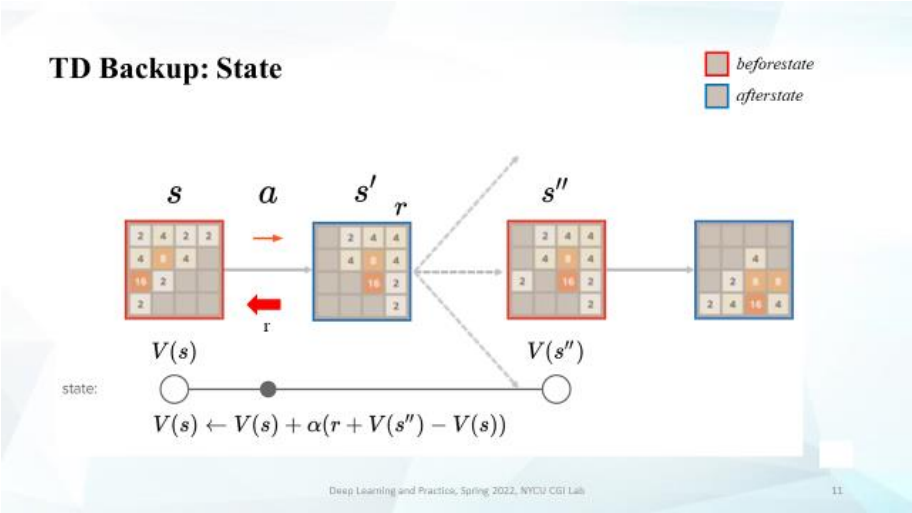


Using the page in slide to explain, it focuses on only s', that is, after state, and it mean how good this time we selected and it going to be.

Explain the action selection of V(after-state) in a diagram. (5%)

Because there is no random mechanism and the previous learning, we can easily use the reward and value to guess the best way.

Explain the TD-backup diagram of V(state). (5%)



function LEARN EVALUATION(s, a, r, s', s'')

$V(s) \leftarrow V(s) + \alpha(r + V(s'') - V(s))$

This step only compares to two state which are before-state, there are no obviously different to after-state case in backup step. Because the random mechanism in this step are finished and not need to mention.

Explain the action selection of V(state) in a diagram. (5%)

Because we had known that the rate happens between s' and s'', we should compute that each s'' could happen and use their value to estimate the best one. This way should cost a lot of time.

Describe your implementation in detail. (10%)

The main different is the upper question. And the following two function is the implement of those two part.

```
void update_episode(std::vector<state>& path, float alpha = 0.1) const {
    // TODO
    float vspp = estimate(path.back().before_state());
    path.pop_back();
    for(; path.size(); path.pop_back()){
        state& move = path.back();
        float err = move.reward() + vspp - estimate(move.before_state());
        vspp = update(move.before_state(), alpha * err);
    }

}
```

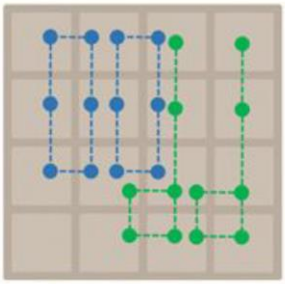
This function show how the (before)state works. As upper mention, the whole function not care about the after-state (including the value() function). And I pass the value from back to front, we can get the newest weights.

```
state* best = after;
for (state* move = after; move != after + 4; move++) {
    if (move->assign(b)) {
        // TODO
        float pi = 0;
        int count = 0;
        for(int k = 0; k < 16; k++){
            if(move->after_state().at(k) != 0) continue;
            board spp = move->after_state();
            spp.set(k, 1);
            pi += estimate(spp) * 0.9;
            spp = move->after_state();
            spp.set(k, 2);
            pi += estimate(spp) * 0.1;
            count++;
        }
        if(count) pi /= count;
        pi += move->reward();
        move->set_value(pi);


        if (move->value() > best->value())
            best = move;
```

In the game, we know that the new tile only happens in the tile label 0 (nothing) with 0.9 rate happen label 1(2), 0.1 rate happen label 2(4), and we compute all case could happen after the after-state. And this cost most time in the whole program. Then choose the best one.

$$V(s) = f_1(s) + f_2(s) + f_3(s) + f_4(s)$$



```
tdl.add_feature(new pattern({ 0, 1, 2, 3, 4, 5 }));
tdl.add_feature(new pattern({ 4, 5, 6, 7, 8, 9 }));
tdl.add_feature(new pattern({ 0, 1, 2, 4, 5, 6 }));
tdl.add_feature(new pattern({ 4, 5, 6, 8, 9, 10 }));
```

The pattern I choose is same to slide and sample. If we don’t do this part, it can’t learn anyway.

Other discussions or improvements. (5%)

Compare between two ways, (before)state obviously cost more time because each time it need to choose it would consider all the case it could be. So in the case that has mechanism let many random state could happen we should avoid to use the (before)state.