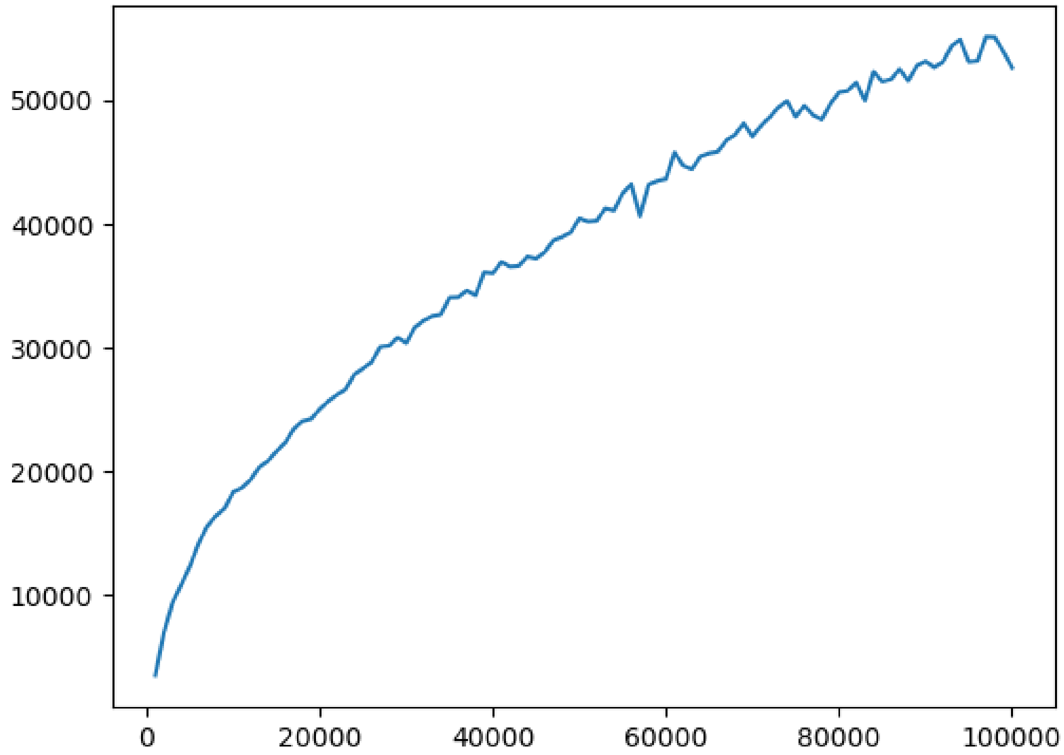
DLP Lab 2 Report

▼ 310551178 資科工碩 穆冠蓁

A plot shows scores (mean) of at least 100k training episodes



Describe the implementation and the usage of n-tuple network

因為要利用所有pattern的weighted sum 來代表當下盤面的value，而且上下左右mirror and rotate皆是isomorphic，所以我們會利用n-tuple來將各pattern的8個isomorphic pattern的weight值加總得到這個pattern的值。

- 取得pattern 所有的isomorphic pattern分別在盤面上的index

```
for (int i = 0; i < 8; i++) {
  board idx = 0xfedcba9876543210ull;
  if (i >= 4) idx.mirror();
  idx.rotate(i);
  for (int t : p) {
    isomorphic[i].push_back(idx.at(t));
  }
}
```

- 透過`indexof`可以取得isomorphic pattern在這個board得出來的一個value，並return index.

```
size_t indexof(const std::vector<int>& patt, const board& b) const {
  // TODO
  size_t index = 0;
  for (size_t i = 0; i < patt.size(); i++)
    index |= b.at(patt[i]) << (4 * i);
  return index;
}
```

在for loop裡使用 or的原因：每取得一個index並用`(4*i)`將位數往左移，在最右邊四位都為0的狀況下存到index裏，在下一輪就可以直接透過`or`將新的index放到末四位。

- 透過`indexof`取得對應的`index`，再透過`operator[]` 取得此盤面的weight，並將所有weight做加總。

```
float& operator[] (size_t i) { return weight[i]; }
float operator[] (size_t i) const { return weight[i]; }
```

```
virtual float estimate(const board& b) const {
  // TODO
  float value = 0;
  for  (int i = 0; i < iso_last; i++){
    size_t index = indexof(isomorphic[i], b);
    value += (*this)[index];
  }
  return value;
}
```

Explain the mechanism of TD(0)

TD(0) 是透過 TD-target: $R_{t+1} + \gamma V(S_{t+1})$來預測$V(S_t)$，而這兩者的差值叫做TD-error= $R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$，透過error*learning rate 對value更新

另外，TD(0)不需要等到episode完全做完，即可更新每個state value

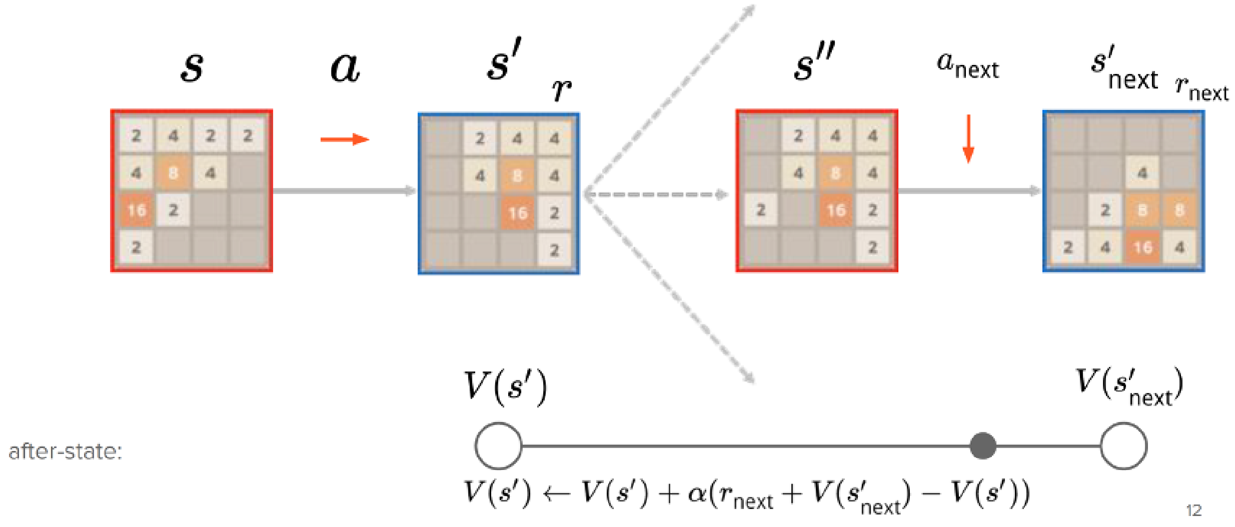
```
void update_episode(std::vector<state>& path, float alpha = 0.1) const {
  // TODO
  float exact = 0;
  path.pop_back();
  while(!path.empty())
  {
    auto &move = path.back();
    float error = move.reward() + exact - move.value();
    debug << "update error = " << error << " for after state" << std::endl << move.after_state();
    exact = move.reward() + update(move.before_state(), alpha * error);
    path.pop_back();
  }
}
```

Describe your implementation in detail including action selection and TD-backup diagram.

- After state

After state 代表做完action後還未pop新的tile的狀況，TD-target搖次透過下一個after state加上獲得的reward r_{next} 來預估真正的$V(S')$，並利用我們剛剛提到的TD-error來更新weight

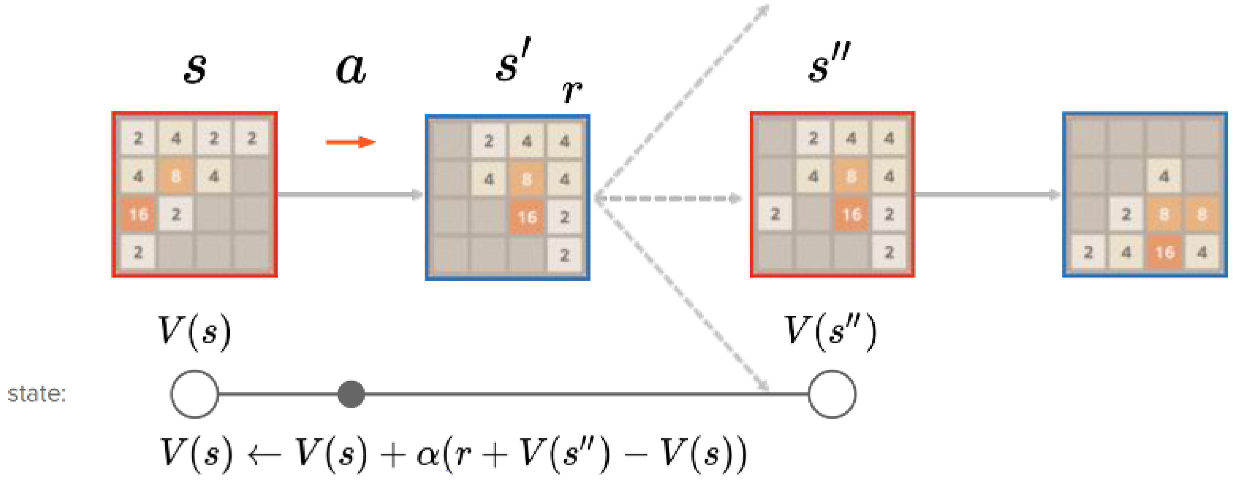
在after state選擇action的方式會透過計算每個action的期望值並選最大的那個當最好的action



```
if (move->assign(b)) {
  // TODO
  board after_state = move->after_state();
  int space[16], num = 0;
  for (int i = 0; i < 16; i++)
    if (after_state.at(i) == 0) {
        space[num++] = i;
    }

  // Set value
  float total = move->reward();
  for (int i = 0; i < num; i++) {
      board* temp = new board(uint64_t(after_state));
      temp->set(space[i], 1);
      total += 0.9f * estimate(*temp) / num;
      temp->set(space[i], 2);
      total += 0.1f * estimate(*temp) / num;
      delete temp;
  }
  move->set_value(estimate(move->before_state()));
  if (total > best_total) {
    best = move;
    best_total = total;
  }
}
```

- Before state



這邊的before state是還沒有滑過action的state，那他的做法會以每個move的before state作為value function，所以會以下一個move的reward加上before state的期望值作為TD target來更新現在這個move的before state期望值。

Demo

```
10000 mean = 18360.8  max = 60116
  128 100%  (0.2%)
  256 99.8% (3.2%)
  512 96.6% (15.4%)
  1024  81.2% (58.1%)
  2048  23.1% (22.6%)
  4096  0.5%  (0.5%)
```

由我們的結果顯示 我們可以在10000個step內玩到2048