A Deep Dive into Monte Carlo Tree Search

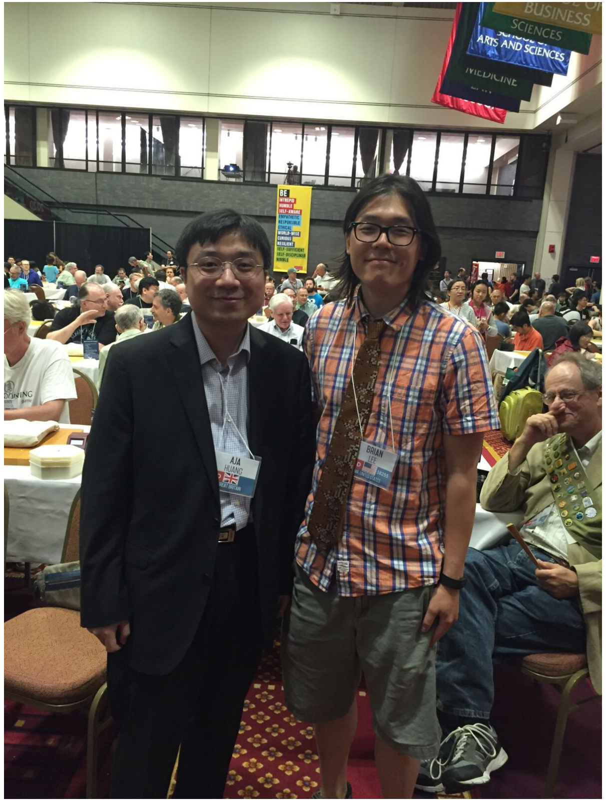
Brian Lee (briankle@google.com)

www.moderndescartes.com

Pycon 2018

About Me

- Software engineer at Verily Life Sciences (an Alphabet company).
- Local organizer for Go events.
- First programming project was a Django website to manage Go tournaments (2012).
- More recently: Go AI (2016)



AlphaGo

(paper)[1]

MuGo

(github.com/brilee/MuGo)

[1] David Silver, et. al. Mastering the game of Go with deep neural networks and tree search. *Nature,* 529(7587):484–489, January 2016

AlphaGo

Zero

(paper)[1]

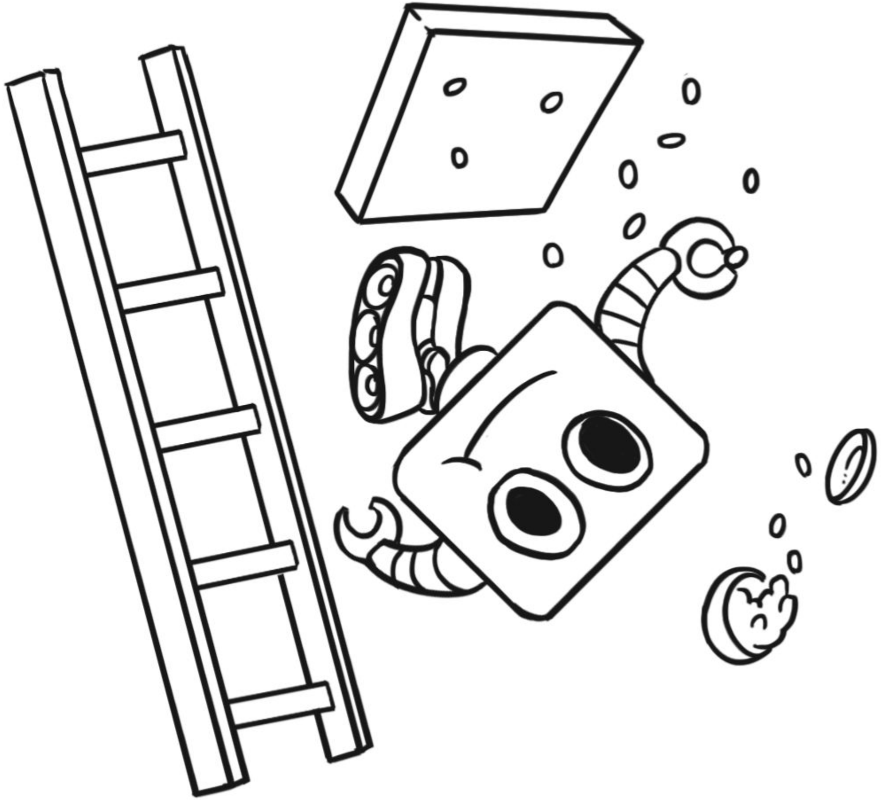
+

MuGo

(github.com/brilee/MuGo)

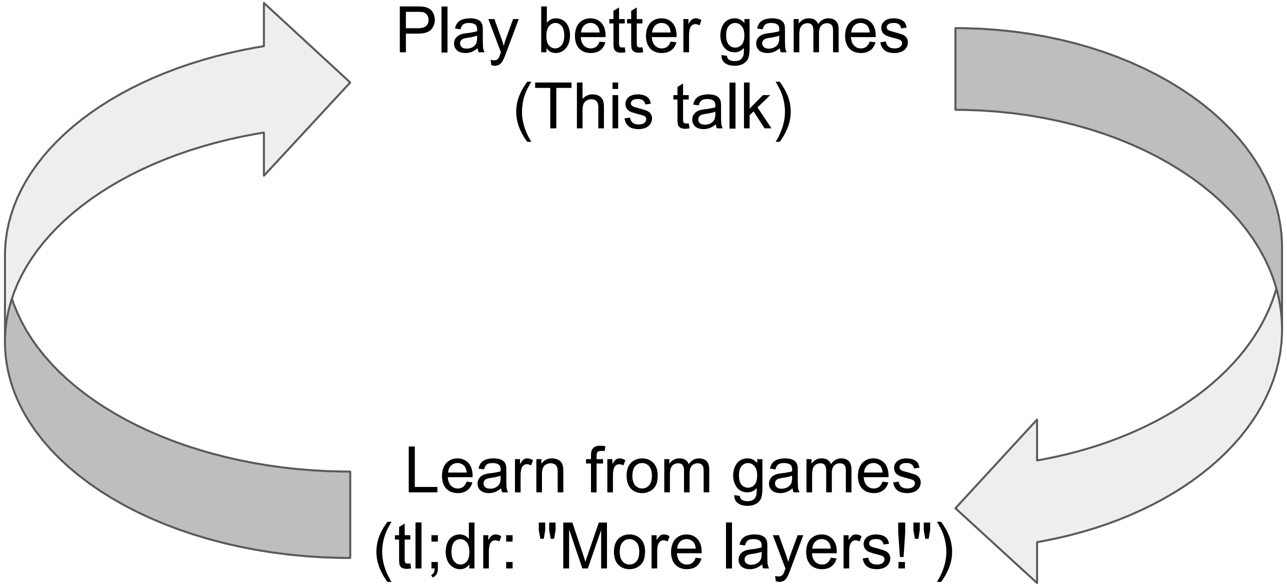
[1] David Silver, et. al. Mastering the game of go without human knowledge. *Nature*, 550:354– 359, 2017.

=



github.com/tensorflow/minigo

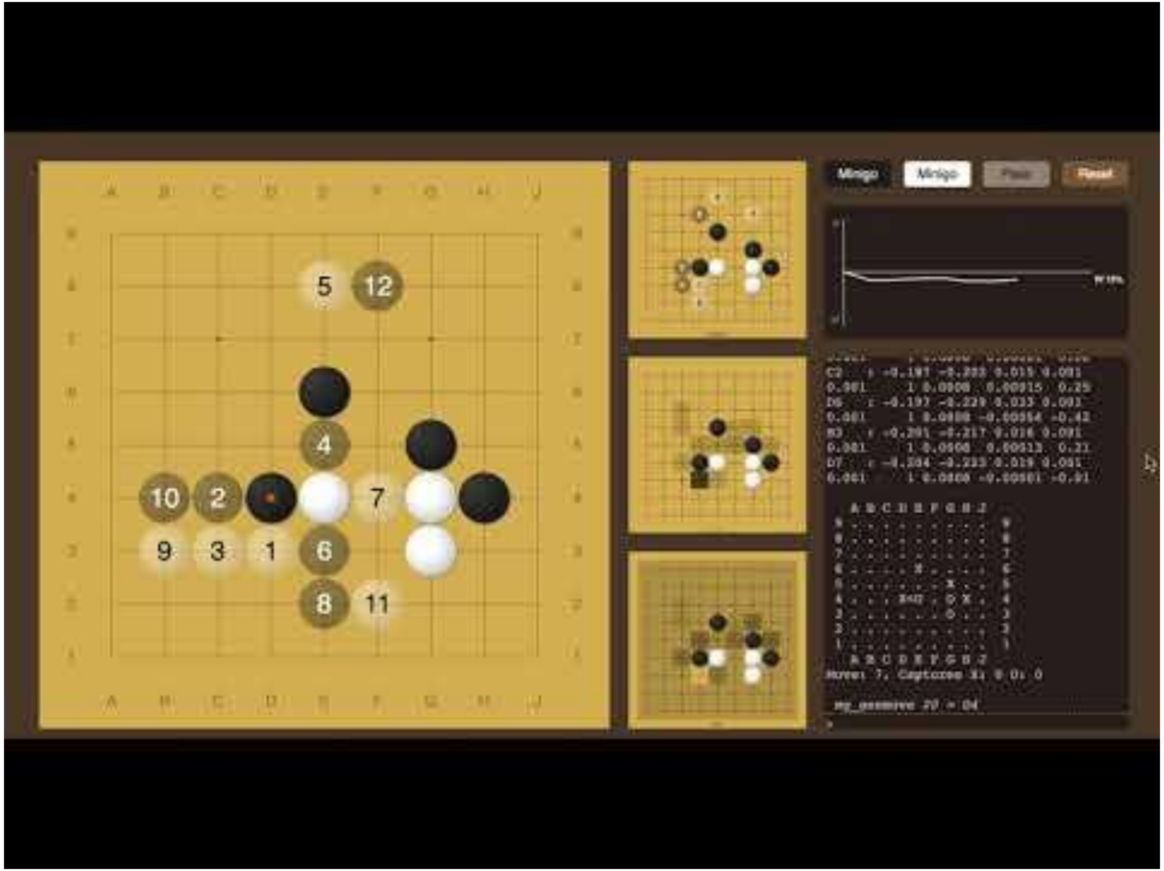
AlphaGoZero, summarized



Play better games
(This talk)

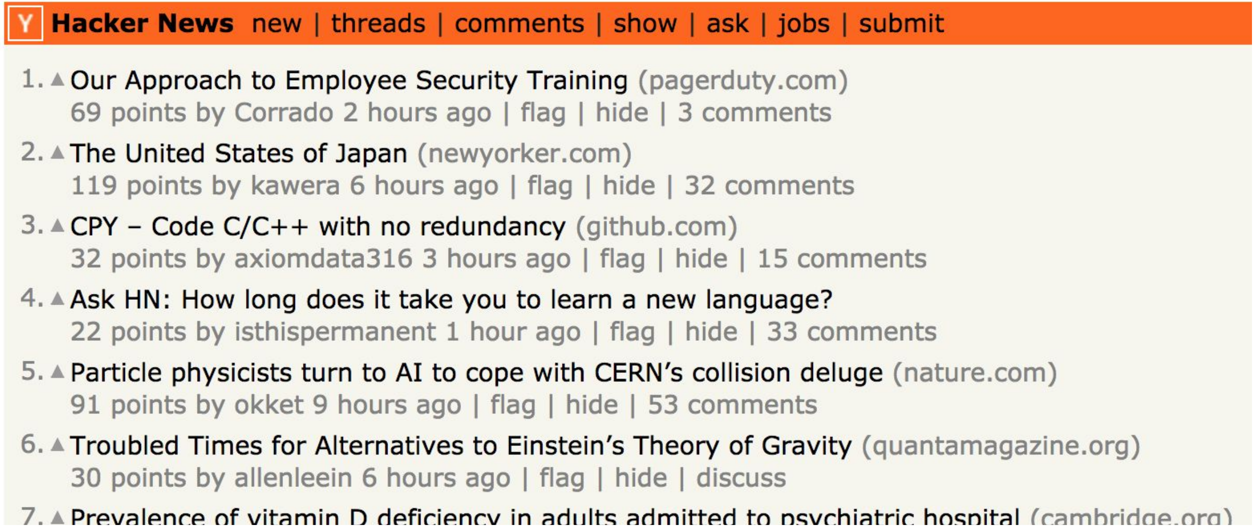
Learn from games
(tl;dr: "More layers!")

Come see MiniGo in action!



```
C2   : -0.197 -0.203 0.015 0.001
0.001     1 0.0000  0.00015  0.25
D6   : -0.197 -0.229 0.033 0.001
0.001     1 0.0000 -0.00354 -0.42
H3   : -0.201 -0.217 0.016 0.001
0.001     1 0.0000  0.00013  0.21
D7   : -0.204 -0.233 0.019 0.001
0.001     1 0.0000 -0.00001 -0.01

   A B C D E F G H J
 9 . . . . . . . . .  9
 8 . . . . . . . . .  8
 7 . . . . . . . . .  7
 6 . . . . X . . . .  6
 5 . . . . . . X . .  5
 4 . . . X)O . O X .  4
 3 . . . . . . O . .  3
 2 . . . . . . . . .  2
 1 . . . . . . . . .  1
   A B C D E F G H J
Move: 7, Captures X: 0 O: 0

 mg_genmove 20 = D4
>
```

A simpler problem

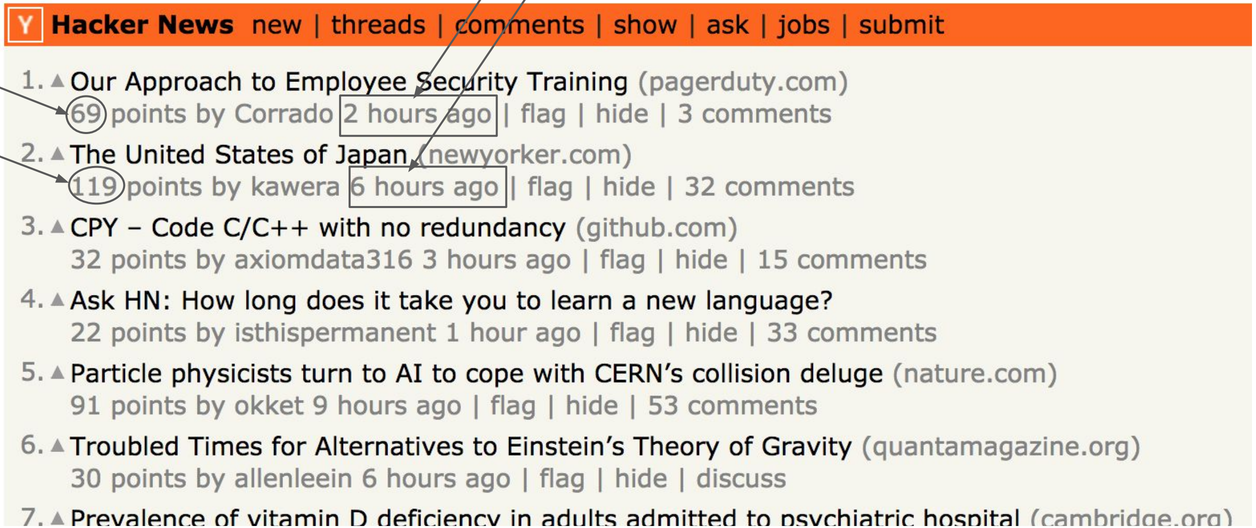

Y
Hacker News new | threads | comments | show | ask | jobs | submit

1. ▲ Our Approach to Employee Security Training (pagerduty.com)
69 points by Corrado 2 hours ago | flag | hide | 3 comments
2. ▲ The United States of Japan (newyorker.com)
119 points by kawera 6 hours ago | flag | hide | 32 comments
3. ▲ CPY – Code C/C++ with no redundancy (github.com)
32 points by axiomdata316 3 hours ago | flag | hide | 15 comments
4. ▲ Ask HN: How long does it take you to learn a new language?
22 points by isthispermanent 1 hour ago | flag | hide | 33 comments
5. ▲ Particle physicists turn to AI to cope with CERN's collision deluge (nature.com)
91 points by okket 9 hours ago | flag | hide | 53 comments
6. ▲ Troubled Times for Alternatives to Einstein's Theory of Gravity (quantamagazine.org)
30 points by allenleein 6 hours ago | flag | hide | discuss
7. ▲ Prevalence of vitamin D deficiency in adults admitted to psychiatric hospital (cambridge.org)

Solution:
Upper Confidence Bounds

Uncertainty

Best guess at quality



Hacker News new | threads | comments | show | ask | jobs | submit

1. ▲ Our Approach to Employee Security Training (pagerduty.com)
   69 points by Corrado 2 hours ago | flag | hide | 3 comments
2. ▲ The United States of Japan (newyorker.com)
   119 points by kawera 6 hours ago | flag | hide | 32 comments
3. ▲ CPY – Code C/C++ with no redundancy (github.com)
   32 points by axiomdata316 3 hours ago | flag | hide | 15 comments
4. ▲ Ask HN: How long does it take you to learn a new language?
   22 points by isthispermanent 1 hour ago | flag | hide | 33 comments
5. ▲ Particle physicists turn to AI to cope with CERN's collision deluge (nature.com)
   91 points by okket 9 hours ago | flag | hide | 53 comments
6. ▲ Troubled Times for Alternatives to Einstein's Theory of Gravity (quantamagazine.org)
   30 points by allenleein 6 hours ago | flag | hide | discuss
7. ▲ Prevalence of vitamin D deficiency in adults admitted to psychiatric hospital (cambridge.org)

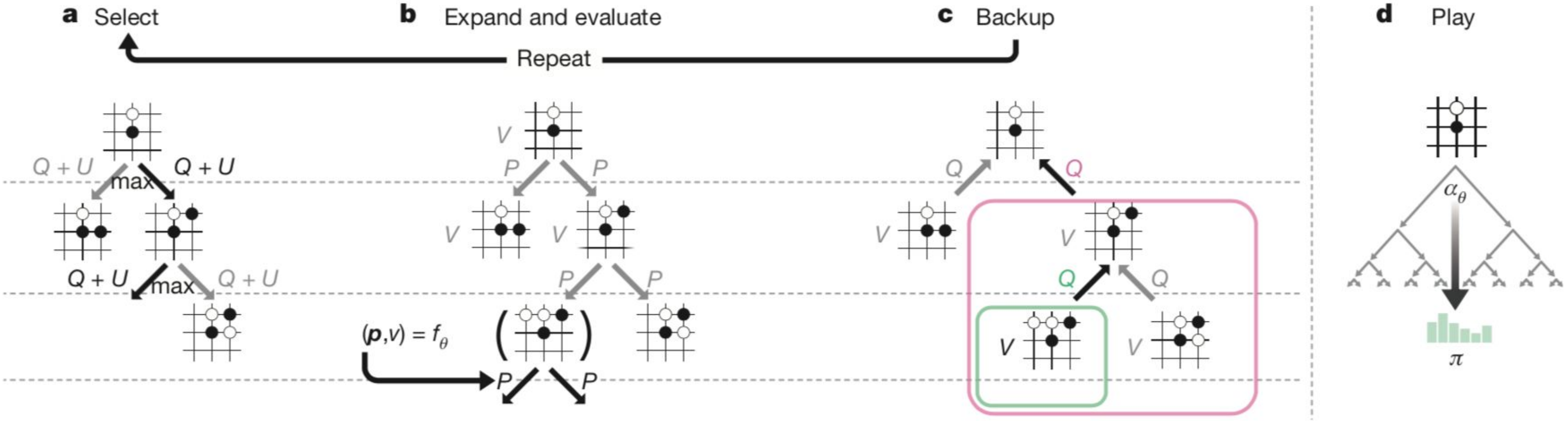
Ranking = quality + upper confidence bound

To learn more:

<u>multi armed bandit</u>

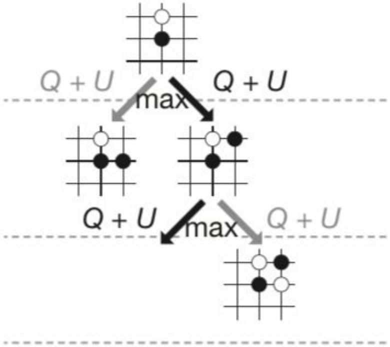
UCT = Upper Confidence bounds (applied to Trees)

Fig. 2 from AlphaGoZero paper



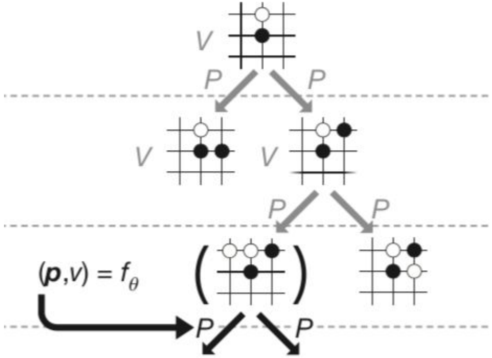
David Silver, et. al. Mastering the game of go without human knowledge. *Nature*, 550:354– 359, 2017.

Step 1: Select



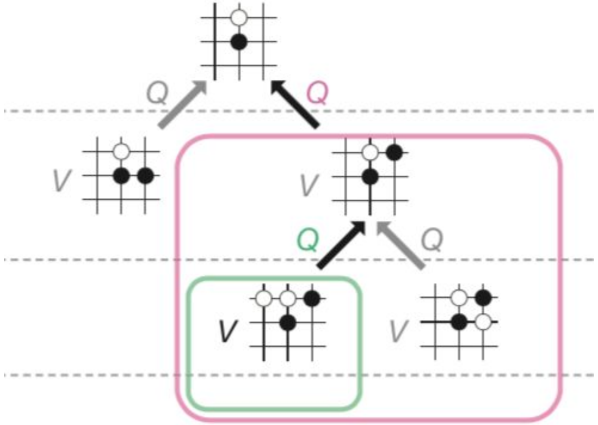
Q: Our best guess at the move's value
U: The upper confidence bound

Step 2: Evaluate, Expand



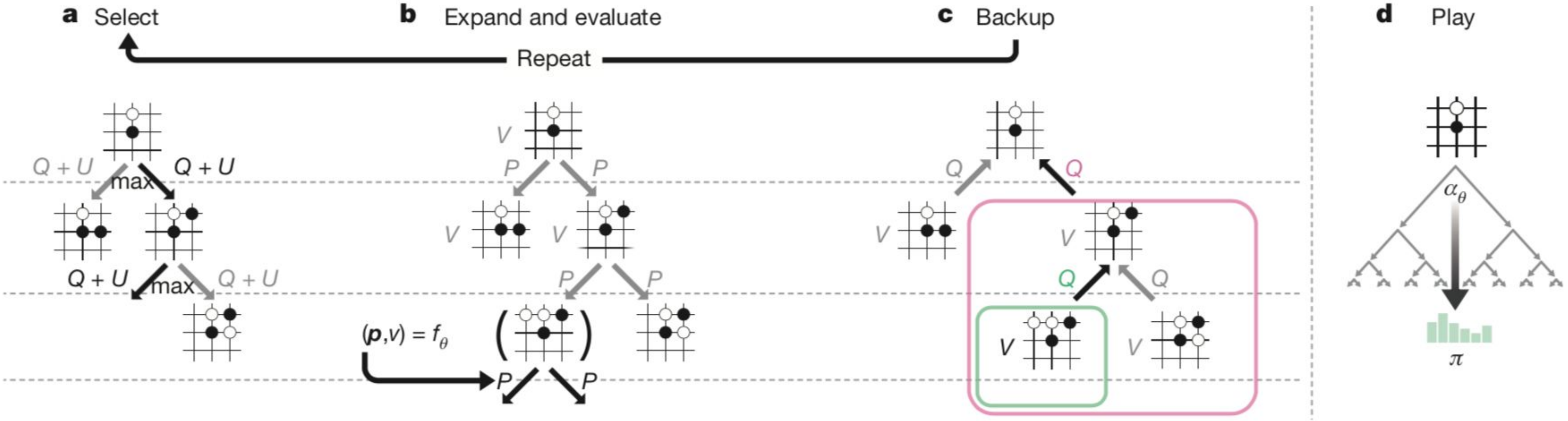
P: Neural net's move probabilities (Array of floats)
V: Neural net's position evaluation (Float)

Step 3: Backup



Q = Average of all children's evaluation V

Step 4: repeat



David Silver, et. al. Mastering the game of go without human knowledge. *Nature*, 550:354– 359, 2017.

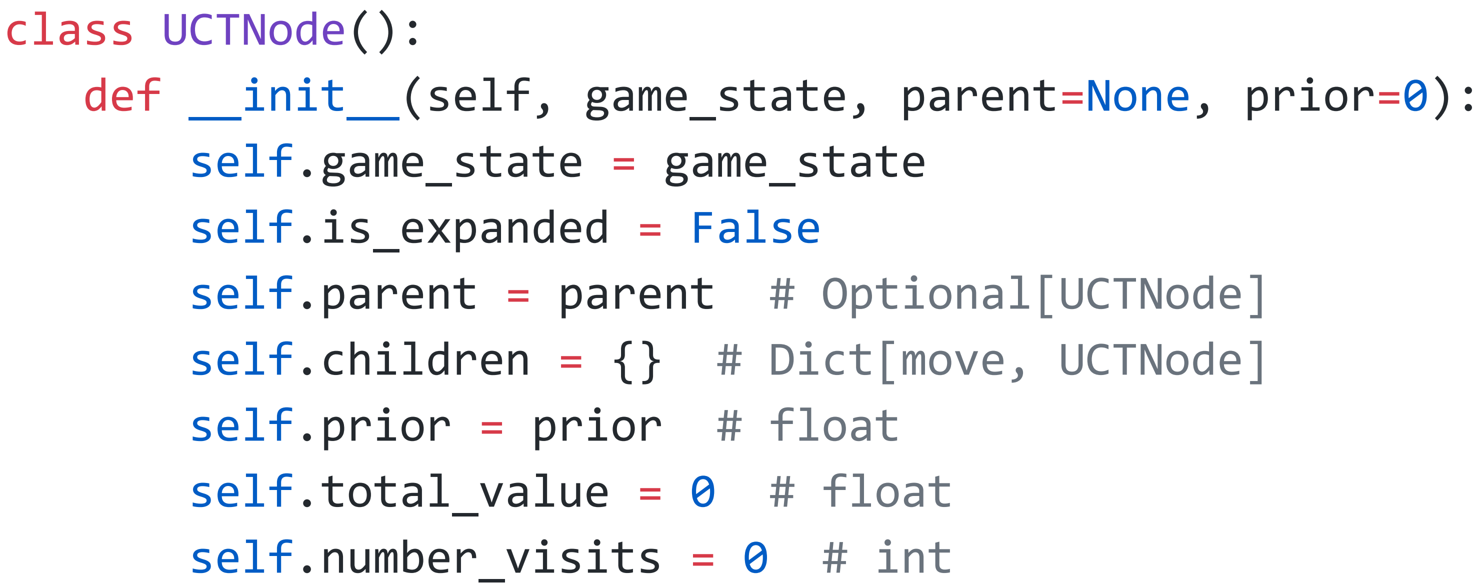
Implementing UCT in Python

All code is online at
github.com/brilee/python_uct

The UCT Algorithm

```
def UCT_search(game_state, num_reads):
   root = UCTNode(game_state)
   for _ in range(num_reads):
       leaf = root.select_leaf()
       child_priors, value_estimate = NeuralNet.evaluate(leaf.game_state)
       leaf.expand(child_priors)
       leaf.backup(value_estimate)
   return max(root.children.items(), key=lambda item: item[1].number_visits)
```

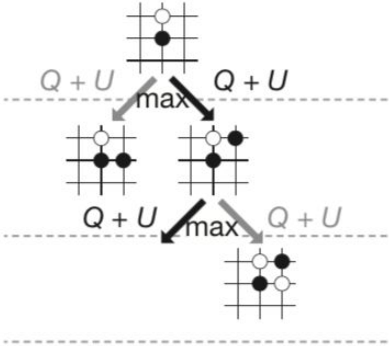
class UCTNode



```python
class UCTNode():
    def __init__(self, game_state, parent=None, prior=0):
        self.game_state = game_state
        self.is_expanded = False
        self.parent = parent  # Optional[UCTNode]
        self.children = {}  # Dict[move, UCTNode]
        self.prior = prior  # float
        self.total_value = 0  # float
        self.number_visits = 0  # int
```

Step 1: Select

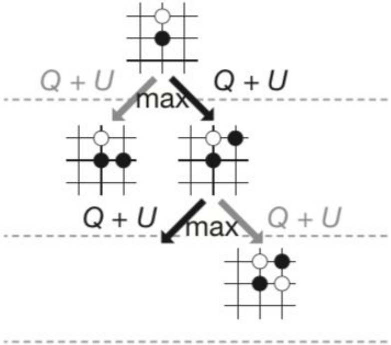
```
def Q(self):  # returns float
    return self.total_value / (1 + self.number_visits)

def U(self):  # returns float
    return (math.sqrt(self.parent.number_visits) * self.prior /
            (1 + self.number_visits))
```



```
def UCT_search(game_state, num_reads):
    root = UCTNode(game_state)
    for _ in range(num_reads):
        leaf = root.select_leaf()
        child_priors, value_estimate = NeuralNet.evaluate(leaf.game_state)
        leaf.expand(child_priors)
        leaf.backup(value_estimate)
    return max(root.children.items(), key=lambda item: item[1].number_visi
```

Step 1: Select

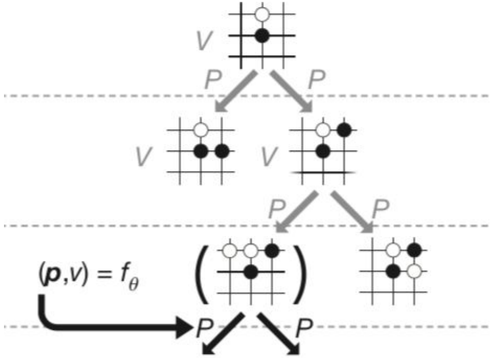
```
def best_child(self):
    return max(self.children.values(),
               key=lambda node: node.Q() + node.U())

def select_leaf(self):
    current = self
    while current.is_expanded:
        current = current.best_child()
    return current
```



```
def UCT_search(game_state, num_reads):
    root = UCTNode(game_state)
    for _ in range(num_reads):
        leaf = root.select_leaf()
        child_priors, value_estimate = NeuralNet.evaluate(leaf.game_state)
        leaf.expand(child_priors)
        leaf.backup(value_estimate)
    return max(root.children.items(), key=lambda item: item[1].number_visi
```

Step 2: Evaluate, expand

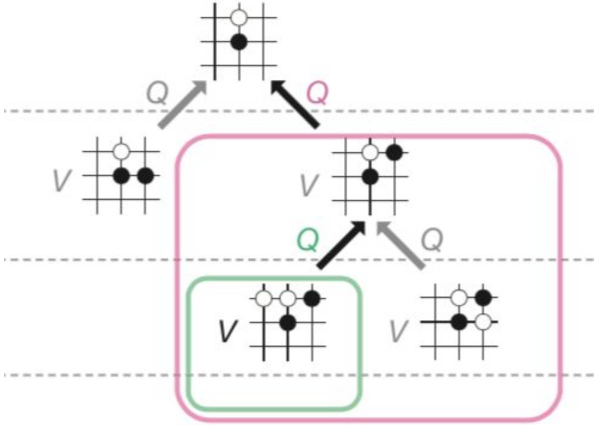
```
def expand(self, child_priors):
    self.is_expanded = True
    for move, prior in enumerate(child_priors):
        self.add_child(move, prior)

def add_child(self, move, prior):
    self.children[move] = UCTNode(
        self.game_state.play(move), parent=self, prior=prior)
```



```
def UCT_search(game_state, num_reads):
    root = UCTNode(game_state)
    for _ in range(num_reads):
        leaf = root.select_leaf()
        child_priors, value_estimate = NeuralNet.evaluate(leaf.game_state)
        leaf.expand(child_priors)
        leaf.backup(value_estimate)
    return max(root.children.items(), key=lambda item: item[1].number_visi
```

Step 3: Backup

```
def backup(self, value_estimate: float):
    current = self
    while current.parent is not None:
        current.number_visits += 1
        current.total_value += (value_estimate
            * self.game_state.to_play)
        current = current.parent
```



```
def UCT_search(game_state, num_reads):
    root = UCTNode(game_state)
    for _ in range(num_reads):
        leaf = root.select_leaf()
        child_priors, value_estimate = NeuralNet.evaluate(leaf.game_state)
        leaf.expand(child_priors)
        leaf.backup(value_estimate)
    return max(root.children.items(), key=lambda item: item[1].number_visi
```

Performance of simple code

For num_reads = 10000...

Memory usage: 1.8GB
Time: 30 sec

Numpy:
SIMD In Python

```
>>> nodes = [(0.7, 0.1), (0.3, 0.3),
(0.4, 0.2)]
>>> q_plus_u = [_1 + _2 for _1, _2 in
nodes]
>>> q_plus_u
[0.8, 0.6, 0.6]
>>> max(range(len(q_plus_u)),
key=lambda i: q_plus_u[i])
0
```

```
>>> import numpy as np
>>> q = np.array([0.7, 0.3, 0.4])
>>> u = np.array([0.1, 0.3, 0.2])
>>> q_plus_u = q + u
>>> q_plus_u
array([0.8, 0.6, 0.6])
>>> np.argmax(q_plus_u)
0
```

```
class UCTNode():
  def __init__(self, game_state,
               parent=None, prior=0):
    self.game_state = game_state
    self.is_expanded = False
    self.parent = parent  # Optional[UCTNode]
    self.children = {}  # Dict[move, UCTNode]
    self.prior = prior  # float
    self.total_value = 0  # float
    self.number_visits = 0  # int
```

```
class UCTNode():
  def __init__(self, game_state,
               move, parent=None):
    self.game_state = game_state
    self.move = move
    self.is_expanded = False
    self.parent = parent  # Optional[UCTNode]
    self.children = {}  # Dict[move, UCTNode]
    self.child_priors = np.zeros(
        [362], dtype=np.float32)
    self.child_total_value = np.zeros(
        [362], dtype=np.float32)
    self.child_number_visits = np.zeros(
        [362], dtype=np.float32)
```

```python
@property
def number_visits(self):
    return self.parent.child_number_visits[self.move]

@number_visits.setter
def number_visits(self, value):
    self.parent.child_number_visits[self.move] = value

@property
def total_value(self):
    return self.parent.child_total_value[self.move]

@total_value.setter
def total_value(self, value):
    self.parent.child_total_value[self.move] = value
```

```
def Q(self):  # returns float
  return self.total_value /
    (1 + self.number_visits)

def U(self):  # returns float
  return (math.sqrt(self.parent.number_visits)
    * self.prior / (1 + self.number_visits))

def best_child(self):
  return max(self.children.values(),
    key=lambda node: node.Q() + node.U())
```

```
def child_Q(self):  # returns np.array
  return self.child_total_value /
    (1 + self.child_number_visits)

def child_U(self):  # returns np.array
  return math.sqrt(self.number_visits) *
    (self.child_priors / (1 + self.child_number_visits))

def best_child(self):
  return np.argmax(self.child_Q() + self.child_U())
```

Performance of optimized code

For num_reads = 10000...

Memory usage: 90MB (20x improvement)
Time: 0.8 sec (40x improvement)

Performance of optimized code

For time = 30 sec

Memory usage: 1.7GB (Same as before)
Num_reads: 300,000 (30x improvement)

Other components of a battle-tested UCT implementation...

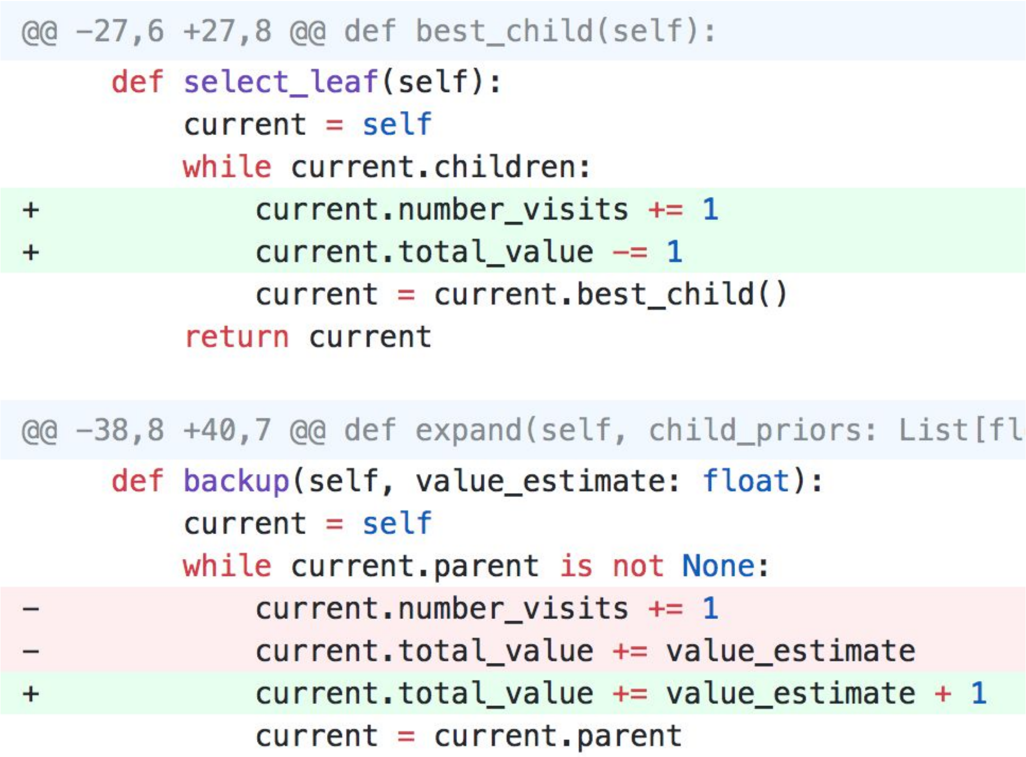
- Handle illegal moves
- When game end condition is triggered, use actual scoring, not network
- Impose a move limit to prevent long games
- Subtree reuse
- Pondering
- Parent-Q initialization
- Tuning relative weighting of Q, U
- Virtual Losses

What about the GPU?

The UCT Algorithm

```
def UCT_search(game_state, num_reads):
   root = UCTNode(game_state)
   for _ in range(num_reads):
       leaf = root.select_leaf()
       child_priors, value_estimate = NeuralNet.evaluate(leaf.game_state)
       leaf.expand(child_priors)
       leaf.backup(value_estimate)
   return max(root.children.items(), key=lambda item: item[1].number_visits)
```

Virtual losses to the rescue!



```
@@ -27,6 +27,8 @@ def best_child(self):
     def select_leaf(self):
         current = self
         while current.children:
+            current.number_visits += 1
+            current.total_value -= 1
             current = current.best_child()
         return current

@@ -38,8 +40,7 @@ def expand(self, child_priors: List[fl
     def backup(self, value_estimate: float):
         current = self
         while current.parent is not None:
-            current.number_visits += 1
-            current.total_value += value_estimate
+            current.total_value += value_estimate + 1
             current = current.parent
```

Virtual losses:
Up to 50x faster with right hardware

Brian Lee
(briankle@google.com)
www.moderndescartes.com