



Published in Towards Data Science

You have **2** free member-only stories left this month. [Sign up for Medium and get an extra one](#)



Ziad SALLOUM

[Follow](#)

Dec 28, 2020 · 6 min read · ✨ · [Listen](#)

[Save](#)



## Counterfactual Regret Minimization

Introduction to CFR and implementation of the Tic-Tac-Toe game



Photo by [Brett Jordan](#) on [Unsplash](#)

“We should regret our mistakes and learn from them, but never carry them forward into the future with us.”

# Lucy Maud Montgomery

Learning from regrets is what Counter Factual Minimization is all about.

The notion of “regret” is introduced in the article “[Introduction to Regret in Reinforcement Learning](#)”. However, it considers scenarios or games composed of a single step or action. Certainly, this is not realistic enough, because most scenarios, in reality, are composed of multiple steps.

It is clear that in every aspect of life, each decision might have a long term impact, and its effect might not be apparent on the spot, but later on.

Counter Factual Regret Minimisation, is a method that deals with scenarios composed of multiple steps, and how to detect errors (thus estimating regret) at every step.

This article provides a general overview of the CFR algorithm and a working example of a tic-tac-toe game.

## How to assess regret every step of the way

As we said, most of the time games or scenarios are multi-step.

So it is important to be able to know what is the regret of not taking a certain action at each step.

After each turn, each player assesses the situation and the value of the state she is in. The assessment takes into consideration whether there is a possibility of winning, losing, etc...

## Simple Value Case

Similarly to Value State in MDP, each node of the game in CFR has a value, called counterfactual. The bottom line is the average of future rewards weighed by their probabilities of really occurring.

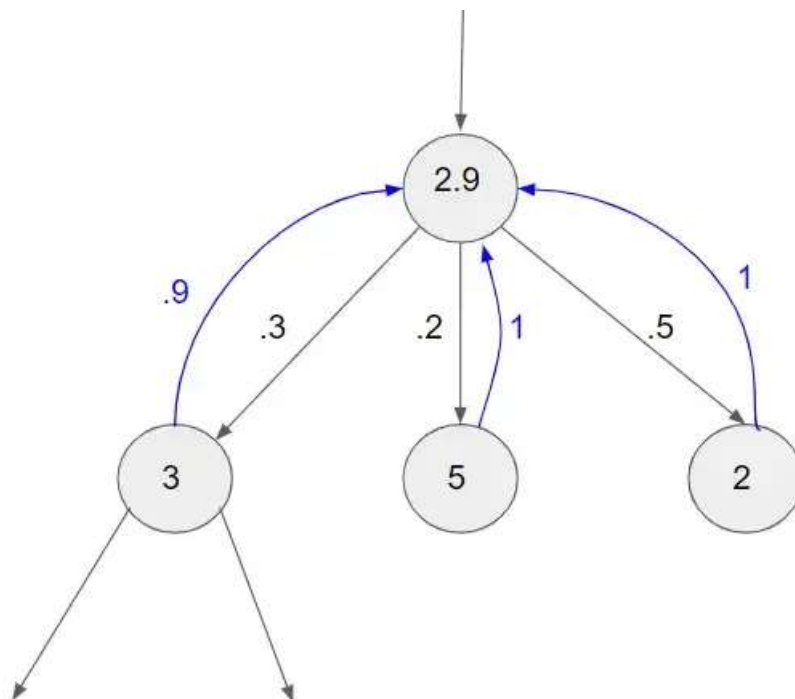


Image by the author

The image above shows a node directly leading to three other nodes, each one with a different probability (.3, .2, .5). Each of the destination nodes has a value (3, 5, 2). The value of the parent node will be equal to

the sum of each of the child node multiplied by its probability of happening:

$$3 \cdot .3 + 5 \cdot .2 + 2 \cdot .5 = 2.9$$

Recursively the parent node communicates its value to its parent multiplied by the probability of happening.

### Simple Regret Case

Since we have computed the value of each node, let's compute the regret of not taking each action. We define regret by the value of the child minus the value of the parent. A quick computation will give us these results:

$$3 - 2.9 = .1$$

$$5 - 2.9 = 2.1$$

Open in app ↗

Sign up

Sign In



Search Medium



recomputed to reflect this fact.

The probabilities (also called strategies) will be computed as follows:

Strategy(i) = Regret(i)/sum of positive regrets

Sum of positive regrets is  $2.1 + .1 = 2.2$

The strategy of negative regrets is given a value of zero.

$$\text{Strategy1} = .1 / 2.2 = .05$$

$$\text{Strategy2} = 2.1 / 2.2 = .95$$

$$\text{Strategy3} = 0$$

With these new strategies, we compute the new value of the parent node as seen in the image below.

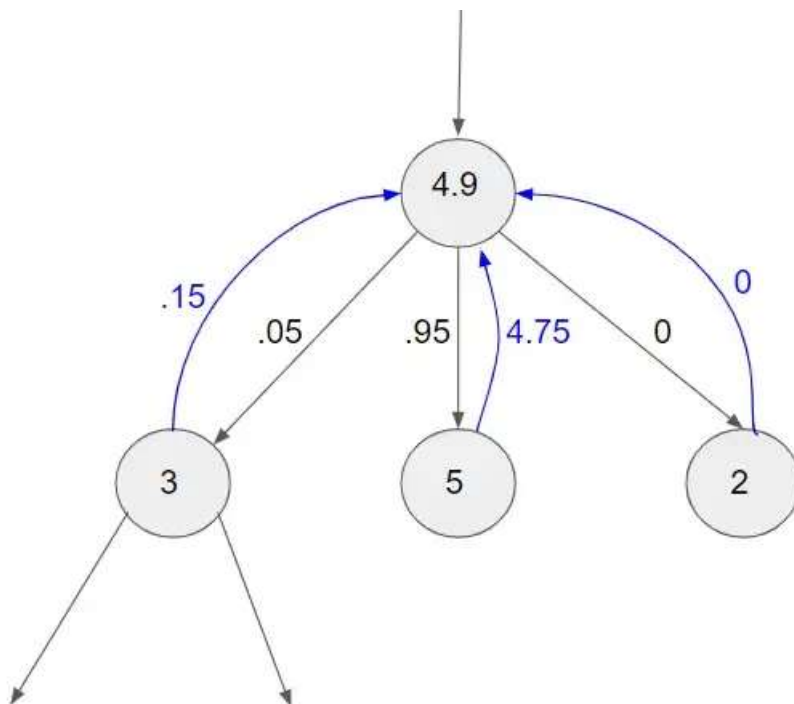


Image by the author

### Adding Some Complexity

So far we have established the value and the regret for each node, however, there is a catch!

The value of each node will be used by its parent combined with the probability that that node will be reached.

But this is not the case for the regret. Considering that the regret (in our example) of the middle action is  $5 - 2.9 = 2.1$ , we still don't know that this node will be reached and this regret will be computed?!

So the regret should also be multiplied by the probability of reaching its appropriate node.

Assuming that the probability of reaching the node in our example is .2, then the regret of that node will be  $2.1 * .2 = 0.42$

**Important.** This will not change the values of the computed strategies because we are multiplying all its components by the same (reaching) probability. on the current iteration (more on that below).

 22 |  1

### Cumulative Computations

As you most probably already know, all AI/Machine Learning algorithms are based on iterative computations. Thousands or even millions of iterations are done to reach acceptable results of behavior.

CFR is no exception! The CFR tree is computed many times, and on each iteration, the regret and the strategies are updated.

The regret is computed cumulatively for each node and each action, then the strategy is derived from these regret values.

Suppose for node  $n$ , there are 2 actions  $a1, a2$ . Since the tree will be iterated  $N$  times, on each iteration, on node  $n$ , we have a new value for **regret** and the **reaching probability**.

$$Regret[n, a1] = Regret[n, a1, 1] * P[1] + Regret[n, a1, 2] * P[2] + \dots + Regret[n, a1, N] * P[N]$$

Same for Regret  $a2$ :

$$Regret[n, a2] = Regret[n, a2, 1] * P[1] + Regret[n, a2, 2] * P[2] + \dots + Regret[n, a2, N] * P[N]$$

Sum of regrets:

$$Regret[n] = Regret[n, a1] + Regret[n, a2]$$

The strategies are computed as usual:

$$Strategy[n, a1] = Regret[n, a1] / Regret[n]$$

$$Strategy[n, a2] = Regret[n, a2] / Regret[n]$$

### The implication of Two Players

When computing CFR for a two-player zero-sum game, there are some important details to be aware of.

- The game tree is organized by layers, alternating between player 1 and 2

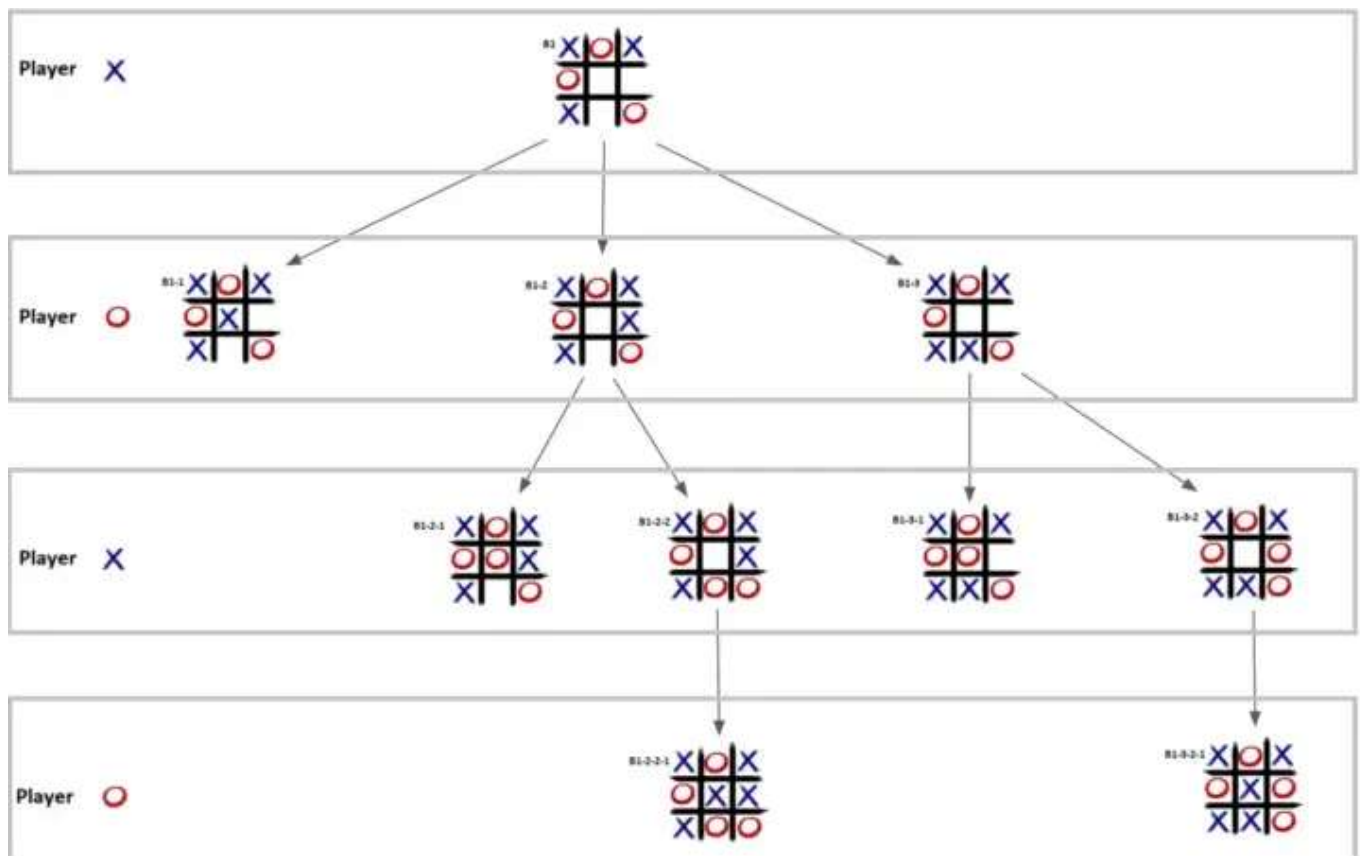


Image by the author

- What is positive for one, is negative for the other and vice-versa. As each node reports its value to the parent, the sign flips to reflect that what is beneficial for one player is a loss for the other.

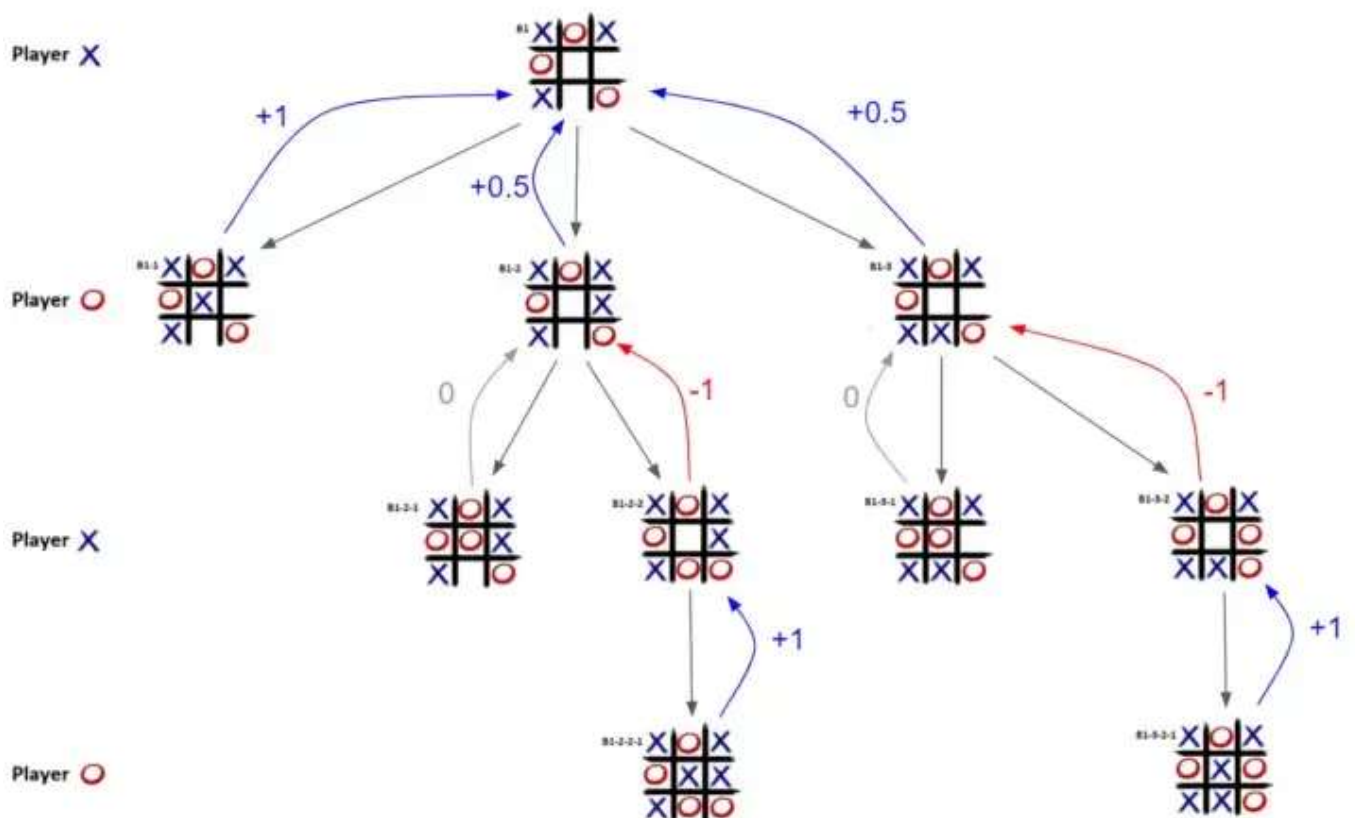


Image by the author

- The reaching probability directly before a node depends on the action of the player in the layer above the node.

### The Code

The code shared in Google Colab is an implementation of CFR of the Tic-Tac-Toe game.

To use it, you should first create a copy of your own, set the iterations that you want, train it by running the first cell, then start playing by running the second cell.

Important! the training time is exponentially proportional to the number of iterations.

*1 iteration takes approx 30sec*

*10 iterations take approx 5min*

*100 iterations take approx 30min*

*1000 iterations take approx 4h30min*

<b>Google Colaboratory</b> Edit description colab.research.google.com	
---	--

It is advisable to train with 1 iteration first then play against the algorithm, then train it with 10 or 100 iterations then play and see the difference.

**Update:** If you are looking to learn and practice Reinforcement Learning you might be interested in <http://rl-lab.com>

### Related Articles

[Introduction to Regret in Reinforcement Learning](#)

[Introduction to Fictitious Play](#)

[Fictitious Self Play](#)

[Neural Fictitious Self-Play](#)

[Neural Fictitious Self-Play in Practice](#)

Reinforcement Learning

Counterfactual

Regret Minimisation

Artificial Intelligence

Machine Intelligence


---

## Sign up for The Variable

By Towards Data Science

Every Thursday, the Variable delivers the very best of Towards Data Science: from hands-on tutorials and cutting-edge research to original features you don't want to miss. [Take a look.](#)

By signing up, you will create a Medium account if you don't already have one. Review our [Privacy Policy](#) for more information about our privacy practices.

 [Get this newsletter](#)

[About](#) [Help](#) [Terms](#) [Privacy](#)

Get the Medium app

