Tetris AI Learning Journey — Understanding Features, Learning, and Functions
By Kelvin-Air Erayanmen (SDT-201-A)

When I started working on my Tetris AI project, I honestly thought it was just about making the game play itself using some clever rules. But as I kept going, I realized it was actually about understanding how machine learning systems represent and learn from the world.

At first, I was really confused about the evaluation function. In the code, there was this line that looked like:
score = w1*f1* + *w2*f2 + ... + w8*f8
I didn't understand how this was related to learning. It looked like simple math, not intelligence.

Then in the lecture slides, I saw those functions f1(x), f2(x), …, fn(x) and some equation f_j : X → D_j.
At that time, it didn't click that this was exactly what I was already doing.
In my code, the features like aggregate height, holes, bumpiness, complete lines, etc., were those same f(x) functions.
Each one described the Tetris board in a numerical way.
Together, they made the feature vector (f1(x), f2(x), …, f8(x)).

When I saw the matrix F = [f_j(x_i)] in the slides, I thought it was just theory.
Later I realized it was basically what I was generating during simulation — every board was a row, every feature a column.
That was my data matrix.
It suddenly made sense that my project wasn't just coding — I was actually creating data and analyzing it.

The next confusing part was about the algorithm a = μ(Xℓ, Yℓ).
I didn't get what μ was supposed to do.
Then after implementing the genetic algorithm (GA), I understood that μ was the method that finds the best weights w1...w8 based on how well the AI performs.
The GA was "training" my model — it was optimizing the same linear function a(x) = sum of w_j * f_j(x).

That realization completely changed my understanding of machine learning.
It's not about magic or hidden intelligence. It's about adjusting parameters to match what we want — in this case, longer survival or more cleared lines.

I also used to misunderstand what features really were.
I thought they were just random values, but I learned that features are simply measurements of how good or bad a state is.
For example, when holes increase, performance drops; when complete lines increase, performance improves.
That's when it clicked that features are like opinions, and the weights decide which opinions matter more.

When I started tuning weights automatically using GA, I saw what "learning" actually means.
Instead of guessing values by hand, the system explored, failed, and evolved.
The learning process replaced my intuition with systematic optimization.

Then, when the lecture talked about different types of features — binary, categorical, ordinal, and real — I saw that all my Tetris features were real-valued.
It also mentioned regression, classification, and ranking tasks.

That helped me understand that my AI was doing a regression-like task: it predicts a number that represents how good the board is.

After connecting everything, I understood the three main stages clearly:

1. Feature engineering means defining f_j(x).
2. Learning means finding w_j.
3. Prediction means computing a(x) = w · f(x).

In the end, the formulas from the slides and the code in my project were describing the same process — I just didn't see it at first.

This project taught me that learning algorithms are not mysterious.
They are ways to gradually improve decisions by measuring and adjusting.
Even a simple Tetris AI demonstrates perception, evaluation, and adaptation.

At first, I coded without understanding.
Then I read the math and got lost.
After experimenting, everything connected — the equations, the game, and the behavior of the AI.
Now when I look at those slides, I actually understand how they apply to real code.

This was one of the first times I felt like theory and practice finally met each other.

After finishing the genetic algorithm part, my next task was to implement beam search.
At first, I thought it was just another version of A* or brute-force search. I didn't understand why we needed it when the GA was already giving decent results.
The name "beam search" confused me because I imagined something to do with light beams or directions.

Once I started researching it, I realized beam search is not about physics — it's a smarter way to search through many possible future states.
Instead of only keeping the single best move like my heuristic agent did, beam search keeps several of the best candidates at once.
It's like saying, "Let's not just go with the top idea, let's keep a few good ones and look deeper from there."

When I tried implementing it, I saw it was the same pattern as the evaluation process I had already built.
For each possible move, I used my feature-based evaluation function to give it a score.
But now, instead of only choosing the highest score, I sorted all the moves and kept the top k ones.
Then I repeated the process, exploring from those k states again.

That's when I realized beam search connects directly with the same function a(x) = w · f(x).
The function still measures "how good" a board is, but beam search changes *how widely we explore*.
So even though the equation stayed the same, the algorithmic logic above it got more powerful.

It also connected back to the lecture part where we discussed approximation — how a(x) tries to approximate the real target y.
Beam search helped me understand that sometimes you don't just want the single best prediction

right now.
You want to look a few steps ahead, keeping multiple hypotheses, and that's also a kind of approximation process.

In a way, beam search felt like the bridge between reinforcement learning and pure heuristic search. It wasn't random or genetic; it was structured but open-minded.
That made me appreciate how search algorithms and learning algorithms both rely on the same ideas:
define good features, evaluate possible actions, and keep improving based on what the function says.

By the end, I realized that the GA and beam search were not separate worlds — they were two sides of the same coin.
The GA helps us learn better weights, and beam search helps us use those weights more intelligently during decision making.

This experience helped me connect all the dots: features, weights, search, and learning.
Everything the lectures mentioned — f(x), y, a(x), and μ(X,Y) — showed up in real code, one step at a time.

End of Report — Kelvin-Air Erayanmen