A computer program is said to learn from **experience** **E** with respect to some **class of tasks T** and **performance measure P**, *if its performance at tasks in T, as measured by P, improves with experience E*.

The above definition is one of the most well known definitions of Machine Learning given by Tom Mitchell. For any learning problem, we must be knowing the factors **T (Task)**, **P (Performance Measure)**, and **E (Training Experience)**. Let's take a few examples to understand these factors.

Problem 1: Handwriting recognition learning problem

For handwriting recognition learning problem, TPE would be,

**Task T**: To recognize and classify handwritten words within the given images.

**Performance measure P**: Total percent of words being correctly classified by the program.

**Training experience E**: A set of handwritten words with given classifications/labels.

Problem 2: Spam Mail detection learning problem

For a system being designed to detect spam emails, TPE would be,

**Task T**: To recognize and classify mails into 'spam' or 'not spam'.

**Performance measure P**: Total percent of mails being correctly classified as 'spam' (or 'not spam' ) by the program.

**Training experience E**: A set of mails with given labels ('spam' / 'not spam').

Problem 3: Checkers learning problem

For a checkers learning problem, TPE would be,

**Task T**: To play checkers

**Performance measure P**: Total percent of the game won in the tournament.

**Training experience E**: A set of games played against itself

If we are able to find the factors T, P, and E of a learning problem, we will be able to decide the following three key components:

1. The exact type of knowledge to be learned (Choosing the Target Function)
2. A representation for this target knowledge (Choosing a representation for the Target Function)
3. A learning mechanism (Choosing an approximation algorithm for the Target Function)

Choose Target Function

Let's take the example of a checkers-playing program that can generate the legal moves (M) from any board state (B). The program needs only to learn how to choose the best move from among these legal moves. Let's assume a function NextMove such that:

NextMove: B -> M

Here, **B** denotes the set of board states and **M** denotes the set of legal moves given a board state. NextMove is our target function.

Choose the Representation of Target Function

We need to choose a representation that the learning algorithm will use to describe the function NextMove. The function NextMove will be calculated as a linear combination of the following board features:

* **xl**: the number of black pieces on the board
* **x2**: the number of red pieces on the board
* **x3**: the number of black kings on the board
* **x4**: the number of red kings on the board
* **x5**: the number of black pieces threatened by red (i.e., which can be captured on red's next turn)
* **x6**: the number of red pieces threatened by black

NextMove = u0 + u1x1 + u2x2 + u3x3 + u4x4 + u5x5 + u6x6

Here u0, u1 up to u6 are the coefficients that will be chosen(learned) by the learning algorithm.

Choose a Function Approximation Algorithm

To learn the target function NextMove, we require a set of training examples, each describing a specific board state **b** and the training value (Correct Move ) **y** for **b**. The training algorithm learns/approximate the coefficients u0, u1 up to u6 with the help of these training examples by estimating and adjusting these weights.

We will explore the different ways to find the coefficient u0, u1 up to u6 in the next blog. In the meanwhile think of any learning problem and try to find out a suitable Target function Representation for that. How about a chess game?