Learning by your Mistakes: How to Teach a Computer

In many real-life situations it is not possible to devise an algorithmic solution to a problem – that is, one cannot write a computer program which will follow through a predetermined sequence of steps to a definite solution. There are many examples in the world of finance which fall into this category: determining an individual's credit-worthiness, for example, relies in the last resort on judgement based on often incomplete evidence. In such cases it may be best to 'train' a computer to give the same decisions as an experienced human. This project shows how such training may take place, in the highly simplified world of a game called Hexapawn.[1]

Hexapawn is a simple enough game that all the possible positions and moves can be written down, and this is the method we shall use to teach our machine. In more complicated games such as chess or go such an exhaustive approach is not possible, and positions have to be analysed using concepts such as 'control of the centre' in chess. Nevertheless, programs to play such games can benefit from the learning approach.

The rules of hexapawn are simple. It is played on a 3 by 3 board, and starts with ranks of three black and three white pawns facing each other with an empty row between. The moves are those of normal chess:
A pawn may move one square directly forwards onto an empty square
A pawn may move one square diagonally onto a square occupied by an opposing piece and capture that piece.
The en passant move does not exist (if you don't play chess, ignore this – the two moves above are the only legal ones).
The game is won
By advancing a pawn onto the third row
By capturing all the enemy pieces
By achieving a position in which the enemy cannot move (n.b. this is a win, not stalemate).

Draw up a set of diagrams showing the possible moves for black after a white opening. If you like, simplify by only treating the cases in which white moves the left-hand or the centre pawn, as a move of the right-hand pawn just gives rise to a mirror-reflected situation. Repeat for the possibilities on White's next move, and so on.

A learning machine which does not use a computer may be made as follows: take a set of matchboxes and label each with the possible position at the start of each of Black's moves. Assign a number to each of Black's N possible legal moves at that point, and insert in the box N slips of paper, each bearing a number from 1 to N. Then play as follows. You play the White pieces. Make your first move. Take the box that is labelled with the position that results. Shake the matchbox, take out a slip of paper, make the corresponding move, and place the piece of paper on top of the box. Make your move, then make Black's next move as above. At the end of the game, if the matchboxes have won (how humiliating is it to be beaten by a pile of matchboxes?), replace all the slips of paper in their appropriate matchboxes and start again. If the

---

[1] Hexapawn is described by Martin Gardner in Further Mathematical Diversions (Penguin, 1977), and Gardner's scheme for teaching a machine to play Hexapawn is based on a Noughts-and-Crosses learning machine described by Donald Michie.

matchboxes lose, remove the slip of paper that corresponds to their last move, replace all the others, and play again. If you find an empty box, it means that the matchboxes have no legal moves left and have lost: you should then remove the slip of paper corresponding to the last move made.

Your task is to translate this scheme into an implementation in Python: a random number generator will 'select the slips', and the 'matchboxes' will be labelled with lists showing the positions. You can choose whether to play against the matchboxes yourself, or whether to analyse the game, devise a perfect player within Python, and train your 'matchboxes' against that.

Keep a record of the wins and losses, and plot the learning curve over, say, 50 games: the curve goes up one for a win, down one for a loss.

If you can persuade somebody else to play the machine, compare performances. You should find that the machine learns better against a stronger opponent.

Try other training regimes. For example, you could add a slip corresponding to the last move when the machine wins as well as 'punishing' it by removing a slip.

Try training the machine against itself, with two sets of 'matchboxes' alternating which has first move. Try two different machines with different reward/punishment schemes against each other.

If all this goes well, you might like to try teaching a program to play noughts and crosses. To keep this manageable, it would be wise to keep only patterns which are distinct, by removing those that are reflections or rotations of others.

A.H. Harker
September 2005.
Updated David Bowler, 2019