Tic Tac Toe

Nearly everybody knows Tic Tac Toe. It is a very simple, easy-to-learn game. Shortly, two players play against each other, each one has a mark, the aim of this game is to place three of their own marks in a horizontal, vertical or diagonal row in a 3x3 grid.

Definition:

Task T: playing Tic Tac ToePerformance P: percent of games one against opponents Training experience E: playing practice games against itself

In the context of machine learning the last point is the most important. To develop a program to play Tic Tac Toe and to measure the performance of the games, no special knowledge is necessary, this is a plain programming task. The design of the latter one can have a huge impact on success or failure of a learner. We have some possibilities to design the training experience. We can provide the learner training examples consisting of game states and the correct move for each (direct feedback) or we can only provide each move sequences and the final outcome (indirect feed- back). The latter case is mostly the more complex one. There the learner has to infer the quality of a move from the result of the game (won/lost). This is called credit assignment. It is a very difficult task to determine, if a move in the sequence is positive or negative for the final outcome, because a game can be lost even when early moves were optimal.

But this is not the only design decision we have to make. It is possible, that a teacher selects board states and provides the correct moves. Or the learner can select board states and ask the teacher for the correct moves. Or the learner can play against itself with no teacher present. In this case the learner selects board states autonomous and evaluates the moves regarding the final outcome.

Finally the training experience should be similar like the real-world experience (over the real-world examples the performance P must be measured). That means the training experience should represent the real

world. We say, the distribution of training examples should follow the distribution similar to the test (real-world) examples.

For our learner we have decided that our system will train by playing games against it. So now we have to define what type of knowledge will be learned. Our Tic Tac Toe system can generate every legal move from any board state. So our system has to learn how to choose the best move from among the legal moves. These legal moves represent a search space and we need the best search strategy. We call a function, which choose the best move from a given board state, target function.

For Tic Tac Toe we define the target function this way: V: B -> R, where B is a legal board state and V maps a numeric score R to B. There, better board states get a higher score, worse board states lower score. So in our scenario, our learner has to learn this target function. To select the best move from a board state, the learner has to generate all possible successor board states and has to use V to choose the best board state (and so the best move).

Most real-world examples are to complex to learn V exactly. In general we are loo- king for a approximation of the target function. We call this function V`. There are many options for V`. For the Tic Tac Toe system we choose a linear combination for V`.

$$\grave{V}(b)=w_0 +w_1x_1 +w_2x_2 +w_3x_3 +w_4x_4 +w_5x_5 +w_6x_6$$

where w_0 through w_6 are weights and the x,s are so called features:

x_1: number of blue marks
x_2: number of red
x_3: number of two in one row/column/diagonal (blue)
x_4: number of two in one row/column/diagonal (red)
x_5: number of red in winning position
x_6: number of red in winning position

With this target function our learner has just to adjust the weights. This is our whole learning task. The weights will determine the importance of each feature.

So we complete our definition of the Tic Tac Toe learning task:

Definition:

Task T: playing Tic Tac ToePerformance P: percent of games one against opponents Training experience E: playing practice games against itself Target function: V: B -> R

Target function representation:

$$V'(b)=w_0 +w_1x_1 +w_2x_2 +w_3x_3 +w_4x_4 +w_5x_5 +w_6x_6$$

Note: With this definition we have reduced the learning of a Tic Tac Toe strategy to the adjusting of weights (w_0 through w_6) in the target function representation.

We have decided, that our learner will train by playing against it. So the only information the learner is able to access is, whether the game was won or lost. We have said, that the learner has to learn to choose the best move. Therefore the learner has to store every single board state and has to assign a score to each board state. The board with the best score represents the best move. It is very simple to assign a score to the last board (the board before the end of the game): If the game was won, we assign +100, if it was lost, we assign -100. So our next challenge is to assign a score to the intermediate board states. If a game was lost, it does not mean, that every intermediate board state is bad. It is i.e. possible, that a game was perfect and just the last move was fatally bad. A very surprising solution for this problem is presented:

$$V_{train}(b) < -V^{c}(\text{Successor}(b))$$

V^{c} is the current approximation of V, Successor denotes the next board state follo- wing b and V_{t}rain is the training value of the board b. So, to summarize, we use the successor board state of b to calculate the score of the training board state b.

The last thing we need is the learning algorithm to adjust the weights. We decide to use the least mean squares, the LMS training rule. LMS training rule helps us to minimize the error between the training values

and the values of the current approximation. The algorithm adjust the weights a small amount in the direction that reduces the error.

Note that there are other algorithms, but for our problem this algorithm is sufficient.

Now we can design the Tic Tac Toe system; • learner plays against itself

– calculates the features of every board state (x_i)– calculates the score of every board state using the features – uses the current weights to choose the current best move

• calculates the training scores for the boards (using the successor board state) – if game was won, set last training score to +100– if game was lost, set last training score to -100– if game was a tie, set last training score to 0

• for each training score– adjust weights using:

$$w_i < -w_i + n(V_{train}(b) - V^c(b)) * x_i$$

n is a small constant (e.g. 0.1). $V_{train}(b) - V^c(b)$ is the error, we can see, that we change the weights to reduce the error between the training examples. The implementation of the Tic Tac Toe-system contains three classes:

1. Game: represents one game

2. TicTacToeSimpleOpponent: A TicTacToe player, who randomly makes his mo- ves

3. TicTacToeLearner: uses the LMS training rule to learn playing TicTacToe (the training partner is TicTacToeSimpleOpponent)

After some experiments with the parameters (number of training loops, ...) I got following results: If two TicTacToeSimpleOpponents play against each other, the TicTacToeSimpleOpponent, who starts, will win around 59% of the games and loses 29%.

A TicTacToeLearner, who uses 70 rounds to train with a TicTacToeSimpleOpponent, wins about 70% of following games against TicTacToeSimpleOpponents.

If we increase the training rounds to 500, it will win approximately 86% of the games and will lose only 6% of it. We can increase the training rounds to 1500, but the result is just a marginal increase of won games.

References

[1] *Machine Learning*, Mitchell, Tom M. (1997), ISBN 0-07-115467-1
[2] *Tic Tac Toe*, http://en.wikipedia.org/wiki/Tic_tac_toe[3] *Source Code*, http://code.google.com/p/mindthegap/