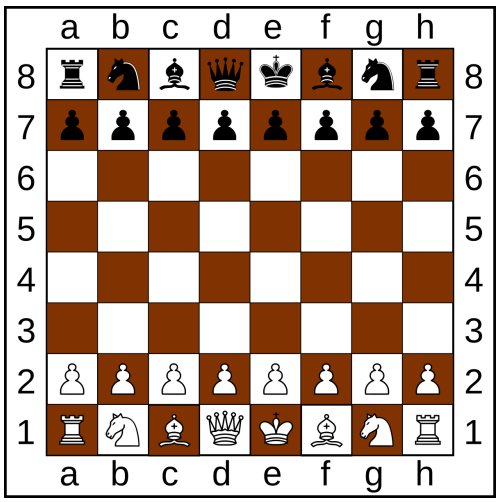
Chess (King-Rook vs. King)

Assignment 2



Dataset Description

An Inductive Logic Programming (ILP) or relational learning framework is assumed (Muggleton, 1992). The learning system is provided with examples of chess positions described only by the coordinates of the pieces on the board. Background knowledge in the form of row and column differences is also supplied. The relations necessary to form a correct and concise classifier for the target concept must be discovered by the learning system (the examples already provide a complete extensional definition). The task is closely related to Quinlan's (1983) application of ID3 to classify White King and Rook against Black King and Knight (KRKN) positions as lost 2-ply or lost 3-ply. The framework is similar in that the example positions supply only low-grade data. An important difference is that additional background predicates of the kind supplied in the KRKN study via hand-crafted attributes are not provided for this KRK domain.

Chess endgames are complex domains which are enumerable. Endgame databases are tables of stored game-theoretic values for the enumerated elements (legal positions) of the domain. The game-theoretic values stored denote whether or not positions are won for either side, or include also the depth of win (number of moves) assuming minimax-optimal play. From the point of view of experiments on computer induction such databases provide not only a source of examples but also an oracle (Roycroft, 1986) for testing induced rules. However a chess endgame database differs from, say, a relational database containing details of parts and suppliers in the following important respect. The combinatorics of computing the required game-theoretic values for individual position entries independently would be prohibitive. Therefore all the database entries are generated in a single iterative process using the 'standard backup' algorithm (Thompson, 1986).

A KRK database was described by Clarke (1977). The current database was described and used for machine learning experiments in Bain (1992; 1994). It should be noted that our database is not guaranteed correct, but the class distribution is the same as Clarke's database. In (Bain 1992; 1994) the task was classification of positions in the database as won for white in a fixed number of moves, assuming optimal play by both sides. The problem was structured into separate sub-problems by depth-of-win ordered draw, zero, one, ..., sixteen. When learning depth d all examples at depths > d are used as negatives. Quinlan (1994) applied Foil to learn a complete and correct solution for this task.

The typical complexity of induced classifiers in this domain suggest that the task is demanding when background knowledge is restricted.

Attribute Information

1. White King file (column)
2. White King rank (row)
3. White Rook file
4. White Rook rank
5. Black king file
6. Black king rank
7. Optimal depth-of-win for White in 0 to 16 moves, otherwise drawn {draw, zero, one, two, …, sixteen}.

Dataset

GENERAL

Train set: 25251 entries
Validation set: 2805 entries

SAMPLE (20 LINES FROM TRAIN SET)

d,1,d,5,f,5,fourteen
c,2,g,7,e,8,ten
b,1,f,7,h,7,thirteen
c,2,d,1,g,5,thirteen
d,1,d,7,g,2,twelve
d,4,f,7,f,2,eight
d,1,f,8,f,6,fifteen
c,2,f,5,b,4,thirteen
c,1,d,8,a,8,twelve
d,1,g,7,a,5,fourteen
b,1,a,8,g,7,fourteen
c,2,f,6,f,7,draw
c,1,h,3,a,3,thirteen
b,1,h,2,d,2,thirteen
c,2,a,6,g,6,thirteen
d,2,h,4,h,1,six
c,1,c,4,a,7,nine
b,1,a,4,a,3,draw
c,2,b,8,f,7,thirteen
d,4,g,5,a,1,seven

Task

You would be given the task to implement the network which predicts the depth of win (number of moves) assuming minimax-optimal play. Two files were provided with the assignment: **train.data**, **val.data**. You need to be able to get higher accuracies than **0.70** or **70%** over test set (**test.data**). Test set is not provided with the assignment and the solution would be graded over test set.

Subtasks

1. Implement **ChessDataset** class which gets a file path in the constructor and returns **X**, **Y** at a given index. Think about the input and output formats of the model. What do you think about the King moves and the Rook moves? (15pt)

2. The **ChessDataset** has to be applicable for the **DataLoader**. Set a reasonable number of items in a single batch for the data loader. (10pt)

3. Implement the **ChessModel** class. Think about the given task is either a classification problem or a regression problem. Therefore what inputs the model has to take and what outputs it has to generate in order to predict accordingly. For the given task take it as a classification problem. (20pt)

4. Implement the following functions: (25pt)

    1. **accuracy** - returns an accuracy of a batch between predictions and real values. (5pt)

    2. **loss** - returns a loss value of a batch between predictions and real values. (5pt)

    3. **train** - trains the model one step further given a model, a batch and an optimizer. *Hint: use **loss** function*. (5pt)

    4. **evaluate** - calculates an accuracy and a loss value for a given model and a batch. *Hint: use **accuracy** and **loss** functions*. (5pt)

    5. **predict** - predicts the depth of win given a model and a sample. (5pt)

5. Draw the confusion matrix (you can read more about it from: https://en.wikipedia.org/wiki/Confusion_matrix). (5pt)

6. Get accuracy higher than **70%** over test set after submission. (25pt)

Submission policy

You have to submit 5 files in a single folder named with your **student id**, for example 180102030. The folder should be compressed in ZIP format.

180102030/

1. **train.data**

2. **val.data**

3. **source.ipynb**

   Please write your name and surname on the top in addition

4. **source.html** *(you can miss this file, but I suggest you to upload it as well. If source code would have problems, then this file guarantees your work done)*

   This is HTML version of your source code. You can export the HTML version of your work in Jupyter Notebook.

5. **model.pt**

   Do not forget to save the state of you **WEIGHTS**!!!! You can read more about it from: https://pytorch.org/tutorials/beginner/saving_loading_models.html

You are free to use any hyper parameters, any predefined functions and any network of NN. Also, the formats of input and output of the model depends on you. Take reasonable input and output formats.