Towards an Automated Sportscasting System for Chess

Charles Chen
UC Berkeley
Berkeley, CA 94720
charleschen@berkeley.edu

Richard Shin
UC Berkeley
Berkeley, CA 94720
ricshin@berkeley.edu

Jinghao Yan
UC Berkeley
Berkeley, CA 94720
jinghao@berkeley.edu

Abstract

We investigate applications of statistical learning to automated sportscasting for chess. In particular, this project studies two methods to automatically identify high level structures in chess games using support vector machines. Our first classifier is based on manual feature selection, while our second classifier employs sparse auto-encoding to automatically generate a set of feature vectors for a chess game. We then compare the results of the two different approaches and showed improvements using auto-encoding techniques. Finally, we conclude with results, analysis of errors, and ideas for potential future work.

1 Introduction

Chess has a long history of intriguing enthusiasts due to its complexity. Even after years of play, each game is new challenge, a new problem to solve. Historically, chess has been prominent as a sport where the battles of grandmasters engulf thousands; given the highly intricate nature of chess (and similar abstract strategy games), experts have scrutinized these games to devise standard high-level structures like openings, strategies, tactics, and time advantage, which they use to explain maneuvers and blunders that occur in the games. Devotees of chess also attempt to improve their own performance in terms of these high-level structures, reading chess theorists' comments about games by high-ranking players.

In this paper, with the end goal of automatically producing textual commentary similar to those produced by experts for grandmaster games, we investigate whether we might automatically identify such high-level patterns in real games. Specifically, we want to see how statistical machine learning techniques can help us automatically identify such structures using a large number of real chess games as training data, where some of the games may be already annotated with these high-level structures, while others may not be.

While others have created highly rigid and rule-based approaches to annotating chess games (Love, 2003), we specifically sought to avoid such approaches which require a lot of manual engineering and chess-specific expertise in order to obtain good results. In particular, for chess, there exists a large number of computer chess tools like chess engines, which search the game tree to examine the breadth of options available and identify the outcomes of each. We would like our approaches to have general applicability to other domains, so we chose to specifically avoid using such tools as much as possible.

To that end, we investigate the use of unsupervised feature learning and deep learning techniques on transcripts of chess games to see if they could automatically identify the high-level structures that make up a chess game. Deep learning techniques, like the sparse stacked auto-encoder, learns a more efficient way to represent some input given a large set of examples. In other fields such as computer vision, researchers have found that these techniques can give state-of-the-art results rivaling or exceeding features hand-engineered by experts. While learning a more compact way to represent parts of chess games, auto-encoders may be able to discover high-level structures similar to those used by human experts.

As a comparison, we also implement a small number of chess-specific, hand-engineered features based on information from a chess engine, such as the number of moves available from a specific board position. Given feature vectors made from hand-labeled games, we train SVM classifiers to predict the label of an unlabeled feature vector. We compare its performance to automatically learned features to see which one can more successfully predict high-level structures.

To conduct our experiments, we collect several thousand "tactical training positions" organized by theme, intended for use by human chess players to learn how to recognize such situations and improve their playing skills. We also collect a large number of unannotated chess games (about 2.5 million) from various sources on the web for use with semi-supervised and unsupervised learning techniques. After training our models, for each of the positions in the training data labeled with a theme (or high-level structure), we attempt to predict the correct theme(s) for a position using the models.

Unfortunately, we found that our approaches did not deliver accurate results in our experiments. In particular, the auto-encoder's output did not help significantly as a useful criterion for distinguishing between different types of themes. We examine the types of correct predictions made by our classifiers and discuss how we may improve our results in the future.

2 Related Work

Deep learning. These techniques attempt to capture complex relationships between input variables in order to transform the input into a different representation which better expresses the relevant structures that make up the input. Specifically, they compose many layers of non-linearities in order to compactly represent complex functions which can model these relationships. They have been used successfully for many different types of tasks, especially in computer vision (Vincent et al., 2010; Lee et al., 2009; Socher et al., 2010).

Automated sportscasting. Chen and Mooney (2008) present a commentator system that learns language from sportscasts of simulated RoboCup games. Their system, which is trained on ambiguous supervision in the form of textual human commentaries and simulation states of the soccer games, simultaneously tries to establish correspondences between the commentaries and the simulation states. Their work focuses on generating meaningful English commentary from relatively simple topics such as "Player 1 kicked the ball to Player 2", while our work focuses less on text generation and more on identifying complex topics.

Automatic chess commentary. There exist techniques for automatic annotation of chess games which are heavily dependent on rule-based logic and use of a chess-engine to determine which parts of a specific chess game are salient (Love, 2003). As a departure from the rule-based approach to automatic chess commentary, Guid et al. (2008) discuss the use of argument based machine learning, which analyzes reasoning where arguments for and against a certain claim are produced and evaluated (in the context of chess, why a certain move was played over others.) Their system, much like the one presented in this paper, identifies complex positional features of a given chess game.

However, we identified some shortcomings of the AMBL system. First, it relies on experts to provide arguments as to why a certain position exemplifies (or doesn't) a particular positional feature. While this is an improvement from defining a classification rule manually, it is infeasible to repeat this procedure multiple times. Second, the arguments that the experts are allowed to provide rely on a preset list of features.

3 Data Sources and Processing

We obtained both annotated and unannotated records of chess games from various web sites in the PGN format (Edwards, 1994), a widely-used format for compactly representing chess games. A large number of games as well as software and tools that use the PGN format are freely available.

The already-annotated games come from

Name	Description
attackPossible	The current player has opportunity to attack.
backrankWeakness	There exists a future checkmate by a rook or queen on the edges of the board.
bishopPair	There exists a decisive strategic advantage for the player with two bishops.
clearance	There exists a maneuver to break the pawn structure.
closure	There exists a maneuver to fix the pawn structure.
decoy	There exists a maneuver to distract a crucial piece.
defensePressure	There exists a maneuver to break the opposing Kings defense.
deflection	There exists a maneuver to deflect an attack.
discoveredAttack	There exists a maneuver to allow for a discovered attack (an attack that was blocked by a friendly piece.)
doubleAttack	There exists a move that simultaneously attacks at least two crucial squares.
drawCombinationsStalemate	There exists a combination that allows for decisive material advantage or immediate checkmate.
enclosedKings	There exists a maneuver to trap the opposing King.
freeingBlockingSquares	There exists a move to free a crucial square.
goodEndgameChances	There exists a move to allow for a favorable endgame.
inBetweenMove	There exists an interrupting move that allows for decisive material advantage of immediate checkmate.
interference	There exists a maneuver to block a crucial piece.
¡¡¡¡¡¡¡ HEAD interruption	There exists a maneuver that interrupts the opponents tactics.
kingInCenter	There exists a decisive strategic advantage for the player with the King in the center of the board.
kingToBadSquares	There exists a combination that forces the opposing King to move to a bad square.
longDiagonals	There exists a strategic advantage for the player that controls squares on the long diagonal.
matingAttack	There exists a decisive mating attack.
openingClosingDiagonals	There exists a maneuver to clear a diagonal.
openingClosingFiles	There exists a maneuver to clear a file.
overload	There exists a combination that makes a defending piece defend too many squares at once.
passedPawn	There exists a combination that allows for a passed pawn (a pawn with no opposing pawns in front of it.)
pawnPromotion	There exists a combination that allows for a pawn promotion.
perpetualCheck	There exists a combination that forces a perpetual check/draw.
pin	There exists a combination that allows for a pin.
pullingPiecesToBadSquares	There exists a combination that forces an opposing piece to move to a bad square.
queenCapture	There exists a combination to capture the opposing Queen.
quietMates	There exists a quiet move (one that looks inconsequential on the surface) leading to mate.
removalOfDefence	There exists a maneuver to remove an opponents defender.
unpinning	There exists a maneuver to unpin a piece.
wrongBishop	The opponents only bishop is limited in use because of the pawn structure of the game.

Table 1: List of high-level themes we attempt to automatically identify, along with explanations

databases of short snippets of chess games,[1] [2] intended for chess players to learn from by example, which are organized by particular themes that the students are to identify. These provided a useful repository of both interesting high-level concepts we could identify (we provide a listing in Table 1), as well as boards and moves corresponding to each one.

We also looked for as many chess games as possible from a variety of web sites, such as `chessgames.com`. We easily obtained over five million untagged games spread over 600,000 files and 5 GB of data. The number of annotated game snippets we obtained was slightly over 5,000.

3.1 Data Pre-Processing

In order to ensure consistency and integrity over the games obtained from disparate sources, we sanitized them in multiple ways. We parsed each game in the data set and removed information that would confuse the parser. For example, some games had English or German comments by humans which we did not plan to use, and sometimes resulted in parsing errors due to specific characters present in the comments. Others had alternate branches, sometimes up to multiple levels (alternate branches within alternate branches), which the parser could not handle properly. Still others represented games in a non-standard way (for example, pawn promotions were represented without the "="), which we had to fix.

Since we obtained the games from many sources, there was significant overlap in games (approximately 50% of the unlabeled games collected were redundant). Therefore, we also went through all of the downloaded games and eliminated duplicate games: multiple games that had the same initial position and move sequence were considered redundant and only one of each were kept.

3.2 Parsing

We tried various open source PGN parsers, each succumbing to flaws like mishandling uncommon cases (such as promotions and special moves like *en passant*). We based our system on Chesspresso,[3] a Java library for writing chess programs, which we fixed and modified for our needs. Chesspresso takes a PGN file as input and returns a data structure that represents the initial position of the board and the subsequent moves (as well as all metadata necessary to recreate the PGN file). We also extended it to be more robust to variations in input format, in addition to being able to export PGNs in a canonical form, which was crucial for removing duplicates.

3.3 Preparing the Data

After parsing each game, we filtered out the tagged games that had more than 10 plies because it will be very difficult to identify the tags that apply to a complex sequence of moves. Among the games that we kept, we also kept only the first few positions and moves (the "prefix" of each game, as described earlier), a parameter that can be adjusted; for our experiments, we kept just the initial board specified in each game. We then generate labels for each game based on the information from the source websites. For example, games in a file that corresponds to a lesson called "Double Attack" are labeled `doubleAttack`. In addition, some of those games may appear in other lessons, and consequently will have multiple labels. We also segment the games further by the color of the player that makes the first move in the game, in addition to the labels of the game, since most of the labels in Table 1 only refer to a specific player; for instance, some of the games labeled with `backrankWeakness` may have the white player with the weakness, while others show the black player with the weakness.

4 Feature Extraction

4.1 Sparse auto-encoder

The sparse auto-encoder is one of many methods in use for unsupervised feature learning (Coates et al., 2010). It takes the form of a feedforward neural network, which consists of an input layer of N nodes, some number of hidden layers, and output layer of M nodes. The value of each layer $\mathbf{l_i}$ is determined with a weight matrix W_i and a bias vector b_i:

$$\mathbf{l_i} = f(W_i \cdot \mathbf{l_{i-1}} + b_i),$$

[1] `http://wwwu.uni-klu.ac.at/gossimit/c/tactic.htm`

[2] `http://webplaza.pt.lu/public/ckaber/Chess.htm#Chesstraining`

[3] `http://www.chesspresso.org/`

where $f(\cdot)$ is a non-linearity like the sigmoid $f(x) = \frac{1}{1+e^{-z}}$ applied component-wise.

An auto-encoder is such a neural network trained so as to minimize the error (specifically, the sum-squared difference) between the output layer and the input layer, over some set of input examples. If we have fewer nodes in the hidden layers than in the input or output layer, the neural network is forced to learn a compressed representation of the data (represented by the activations, or outputs, of the smaller number of hidden nodes). Instead, we keep a larger number of hidden nodes while imposing a sparsity constraint that the average activation of each one be small (e.g., 0.05). Other work has shown that such a constraint better forces the auto-encoder to learn interesting structures from the data (Bengio, 2009).

A normal auto-encoder only has one layer of hidden nodes, but we can also add more hidden layers to form a stacked auto-encoder. Per Bengio et al. (2007), we use a greedy strategy to train the extra hidden layers in the auto-encoder. We train an auto-encoder with one hidden layer as usual; once this completes, we obtain the output of the hidden layer over all the examples in the input set, and use that as the new set of examples. This allows us to learn higher-level representations of the data (Vincent et al., 2010).

Other strategies for unsupervised feature learning include restricted Boltzmann machines (which are undirected), deep belief nets (stacked RBMs, akin to stacked auto-encoders), and denoising auto-encoders (which try to reconstruct some input from a corrupted version of it). We only evaluated sparse stacked auto-encoders in this paper.

The auto-encoder requires training data in the form of vectors where all the elements are between 0 and 1, or the range of the sigmoid function. We generated the training vectors by, for some chess board and for every type of piece (knight, rook, etc.), whether a white one (valued 1.0), a black one (valued 0.0), or none at all (valued 0.5) was present in every square of the chessboard.

4.2 Manually-generated features

As a point of comparison to auto-encoders, we extracted chess-specific features manually from a given chess position (configuration of pieces on the board) and the move that resulted in that position; Table 2 contains a list. The position-based features include information such as where the pieces are on the board and potential moves, while the move-based features involve binary information such as if the move that resulted in this position was a capturing move or resulted in a check. The resulting feature vector was then L2-normalized to fit the specifications of the support vector machine.

5 Experimental methodology

We processed all of the labeled training data to obtain a mapping from each unique game prefix (for most of the experiments, just the initial specified configuration of the board in the labeled game) to the labels that apply to it; most of the prefixes only had one label, while others had two or three. We then segmented the data (consisting of about 3000 board configurations) into three disjoint sets: 90% for training, 5% as the validation set (for use with tuning our classifiers), and 5% as the test set. The segmenting was done by random selection from the set of all labeled games.

In our experiments, we first use the labeled data in the training set (and possibly also unlabeled data) to train the classifiers. Then, for every test input chess game (or a prefix of a game), we use our classifiers to apply some number of labels to it that come from Table 1. We measure the performance of the predictions made by our classifiers by counting the number of true positives (a predicted label was among the set of gold labels for a test input), true negatives (a label in Table 1 was neither in the set of predicted labels nor the set of gold labels), false positives (a predicted label was not in the set of gold labels), and false negatives (a gold label was not predicted). We compute and report the precision, recall, and F1 score using these measures of accuracy.

5.1 Support vector machine classifier

We note that any number of labels may apply to some test input that we want to classify; a straightforward use of a binary (or even multiclass) classifier, as provided by most implementations of support vector machines, will not suffice as it can only assign one label. Therefore, we take the one-versus-all strategy of training many binary SVMs; for each label that we consider, we train a binary classifier

Category	Description
Position	Piece location, potential moves per piece, material count, square control.
Previous move	Capturing move, recapturing move, *en passant* move, long castling move, short castling move, check move, stalemate move, checkmate move, promotion move.

Table 2: List of chess-specific manual features per position.

with all training games that have the given label as positive examples, and all other games as negative examples. We used `SVMlight` (Joachims, 1999), a popular third-party implementation of SVMs, for our experiments, using the manually-generated features extracted from each labeled data point.

5.2 Classifying with the auto-encoder

There exists several methods for using the auto-encoder for the task at hand. First, we can consider using the output of the last hidden layer as the feature vector for input into the SVM classifier, just as we did with the manually-generated features. To do this, we first perform unsupervised fine-tuning; we combine all the greedily-trained auto-encoders to make a many-layered neural network intended to reconstruct the input layer in its output layer, after which we again minimize the sum-squared error between the input and the output layers. We can then use the output of the hidden layer (i.e., not the one that reconstructs the input, but the middle layer which provides the most high-level representation of the input) as the feature vector.

We can also use semi-supervised learning methods, in which after greedily training the auto-encoders, we combine all the hidden layers of the auto-encoders and add an output layer at the end. For every labeled input game, we construct input-output vectors such that the output encodes the gold labels of the input. We then train the resulting multi-layer neural network to minimize the sum-squared error; this acts similarly to a multiclass logistic regression, except with a different objective at the last layer.

To train these neural networks, we used batch learning over all the examples at once. The loss function contained components for the sum-squared error as well as an L2 regularization term and a sparsity penalty based on the Kullback-Leibler divergence between the desired average activation of the hidden nodes and the actual average activation. We used backpropagation to compute the gradient of the loss function and L-BFGS to optimize the parameters as to minimize the loss function (Ng, 2011). Running serially, the training takes a long time to complete, even on simple inputs. Therefore, we examined ways to parallelize the task in order to fully leverage the capacity of modern many-core machines; we achieved a significant speedup by parallelizing the loss function evaluation to 16 execution threads.

5.3 Random baseline

To use as a baseline and ensure that our classifiers are giving sensible results, we randomly made predictions for every test game, where each one assigned each possible label at random, with the possibility of the label being given based on the prior probability of the label, or the proportion of the training games which have that label. The random predictions were made 1,000 times, with the predictions made independently at random, and the error rates were averaged.

6 Experimental results

6.1 SVM classifier

We first describe the results achieved when we trained using the manually-specified feature extractors. Since each game has very few labels, most of the binary classification results are negative. Therefore, the model is heavily biased toward negative results, as shown in Table 3.

This classification method had relatively high precision compared to the random baseline. However, the recall is much lower than that of the random classifier. As a result, the F1 score (a combination of the precision and recall scores) of the SVM-based model is no better than that of the random model.

Type	TP	FP	TN	FN	Precision	Recall	F1
Unsegmented	0.21%	0.00%	96.89%	2.90%	100%	6.71%	12.57%
Baseline (Random)	0.42%	2.76%	94.12%	2.69%	13.32%	13.65%	13.48%
Segmented	0.13%	0.02%	98.34%	1.51%	86.67%	7.65%	14.05%
Baseline (Random)	0.12%	1.49%	96.87%	1.51%	7.65%	7.55%	7.60%

Table 3: Classification results for the games that have not been segmented by the color of the active player, and those that have been. In the former, precision clearly outperforms a random baseline, but recall falls short, resulting in a lower F1 score. In the latter, precision has fallen relative to the unsegmented version, but recall has risen, resulting in a slightly higher F1 score. In contrast, the random model became much worse in both ways.

Tag	Error rate
`queenCapture`	0.6452%
`kingInCenter`	0.6452%
`perpetualCheck`	0.6452%
`drawCombinationsStalemate`	3.8710%
`bishopPair`	0.6452%
`doubleAttack`	0.6452%
`clearance`	2.5806%
`openingClosingDiagonals`	0.6452%
`openingClosingFiles`	0.6452%
`enclosedKings`	1.2903%
`discoveredAttack`	2.5806%
`passedPawn`	1.2903%
`quietMates`	3.8710%
`pin`	5.1613%
`removalOfDefence`	22.5806%
`deflection`	25.1613%
`decoy`	9.6774%
`interference`	3.8710%
`backrankWeakness`	3.2258%
`freeingBlockingSquares`	1.9355%
`pawnPromotion`	6.4516%
`pullingPiecesToBadSquares`	0.6452%

Table 4: Percent of incorrect classification (false negatives and false positives) for various tags.

When we extended the labeling to also incorporate the first mover's color information, the SVM model's recall was improved at a minor cost to precision, resulting in a higher F1 score. In contrast, the random predictions became much worse: Both the precision and recall scores fell as a result of this change, which reduced the evenness of the label distribution among the training games, and reduced the amount of games with each label (since games are now spread over twice as many tags).

Given the high precision but low recall, we delved deeper to analyze the sources of errors and true positives. As we show in Table 4, we found that `deflection` and `removalOfDefence` are much more likely to cause false negatives than other tags. In addition, we found that all of the true positives are for the `quietMates` tag. 9 of the 11 true positive `quietMates` games were classified correctly using only the features that describe the contents of the board. On the other hand, 5 true positive `quietMates` games were correctly classified when using only features about the possible moves in that position. Interestingly, disabling the "checkmate move" feature did not affect the number of true positives.

6.2 Classifying with the auto-encoder

We first tried to use the unsupervised fine-tuned stacked auto-encoder's hidden outputs as the input to the SVM classifiers, as with the manually-generated feature vectors. However, we found that the resulting feature vectors were not very useful for SVM, as the classifiers did not predict any labels as true for all training or test input games.

Instead, we used the semi-supervised classification method as described in Section 5.2. First, we greedily train each hidden layer as an auto-encoder on the training, test, and validation sets. We trained the combined neural network composed of greedily-trained auto-encoders to, for each input, predict an output vector corresponding to the labels that apply to the input—0 if a label did not apply, and 1 if it did. From the output vectors computed with the elements in the validation set, we tuned the threshold we use in considering whether a label is present or not present to maximize the F1 score. In Table 5, we report the performance achieved on the test set when

Type	Hidden layers	TP	FP	TN	FN	Precision	Recall	F1
Unsegmented	0	1.49%	3.62%	93.26%	1.61%	29.26%	48.17%	36.40%
Unsegmented	1	1.35%	3.83%	93.06%	1.76%	26.01%	43.29%	32.49%
Unsegmented	2	1.40%	4.48%	92.41%	1.71%	23.87%	45.12%	31.22%
Unsegmented	3	1.40%	4.48%	92.41%	1.71%	23.87%	45.12%	31.22%
Segmented	0	0.38%	1.01%	97.35%	1.26%	27.08%	22.94%	24.84%
Segmented	1	0.49%	2.90%	95.46%	1.15%	14.49%	30.00%	19.54%
Segmented	2	0.22%	1.27%	97.09%	1.42%	14.84%	13.53%	14.15%
Segmented	3	0.59%	3.89%	94.47%	1.05%	13.12%	35.88%	19.21%

Table 5: Auto-encoder classification results for the games that have not been segmented by the color of the active player, and those that have been, as the number of hidden layers varies.

using the threshold tuned to maximize the F1 score on the validation set.

These results show higher F1 score than the SVM classifier, with significantly lower precision but also much higher recall. Unfortunately, the results indicate that the representation of the board learned by the auto-encoder does not correlate very well with the concepts we wanted to identify, given that having no hidden layers resulted in higher scores. Adding extra hidden layers seems to not have helped improve the classification results, except in the last instance, where an extra layer improved the F1 score to original levels. It is hard to tell whether some slight change in the auto-encoder training procedure will be able to drastically improve the results, or whether significant changes in the architecture will be required. We also were not able to use the unlabeled chess games we collected with the auto-encoder, due to the long training time required and our relatively slow implementation of its training.

6.3 Analysis

Given the nature of labeled training data and their assembly from disparate sources, there exists some redundancy among the labels, and games may not have all the labels that they should have in our chess-description ontology. Furthermore, many of the labels had very few (fewer than twenty) games as positive examples, and performance may improve greatly if we had a larger training set to use.

7 Future Work

We identify three areas in which future work can make progress on automated sportscasting. Using a higher-performance auto-encoder training procedure and training it on a larger data set could help improve the results. We could also extend our system with an automated text-generation layer. In addition, leveraging chess-specific tools and features like chess engines and evaluation tools could help improve the quality of the output. Alternatively, an orthogonal path of applying similar general approaches to other games and attempting to learn higher-level constructs, including complex ones like StarCraft and simpler ones like Checkers, merits exploration.

7.1 Performance

One of the main limitations of our work is performance. Due to the slow performance of the auto-encoder, we were not able to fully exploit the millions of games that we have available to generate features for our unsupervised system. Therefore, extensions to our work should investigate ways to improve the performance so that the auto-encoder can process more inputs and thus generate more relevant feature functions.

Distributed training. Our auto-encoder takes advantage of some of the parallel processing capacity of the host system through a local MapReduce scheme. Extending that with, for example, a distributed MapReduce framework like Hadoop may improve performance if the additional cost of network I/O are outweighed by the benefits of parallelization.

Using the GPU. Another option that we partially explored is the exploitation of the processing power of the graphics processing unit (GPU). Function optimization, as used in training the auto-encoder, is a computation-heavy task, and the GPU is specifically designed for massively-parallel floating point computations. We investigated using JavaCL, an interface to using the GPU for general purpose computation, to parallelize the auto-encoder, and demonstrated that it is definitely superior at parallelizable and computation-intensive tasks like matrix multiplication, because the benefits of greater throughput through greater parallelization outweigh the costs of memory transfer.

7.2 Text Generation and Linguistic Grounding

One further avenue of exploration is a system that automatically generates text given a chess game with tagged positions. The naive approach would be to build a template-based text generation system, which could be as simple as generating a formatted sentence highlighting hand-selected salient features for each tag or something more complicated such as a context-free grammar. In the context of this project, we found that some chess positions were positive examples of multiple tags. As such, another possible avenue of research could be to conglomerate the annotations for each specific position and generate an annotation that makes sense in its context.

While we did use high-level structures as defined by experts and discussed in detailed commentary of chess games, we did not attempt to learn and mimic the actual structures used in the natural language commentary. To have a useful commentating system for an intricate game like chess, we need to both discover the high-level structures in the game, as well as know how to talk about the high-level structures. Extending the approach taken by some of the automated sportscasting work discussed in the Related Works section, it may be possible to learn both high-level concepts that are present in a game, and how to realize descriptions of them in language.

7.3 Improving Results

In pursuing this project, we attempted to maintain an approach that is as general as possible so that it can be applied to other contexts, and to minimize the amount of engineering required for components that are very specific to chess. Given the complexity of chess, a relatively straightforward, but chess-specific, way to improve the performance would be to incorporate tools like chess engines. Computer chess has been an active field of research, and much work has gone into game engines and weight functions which excel at the game.

For example, a chess-specific rule-based approach should be able to easily identify the more precisely-described themes like `doubleAttack` and `queenCapture` by analyzing the future branches of the game and looking for specific outcomes. However, this approach would require that each theme must have a specific heuristic function written for that express purpose to identify certain properties in future states in order to effectively identify tags using a chess engine. In addition, it may be difficult to identify more subjective themes like `clearance` and `kingToBadSquares` that are based more on human judgement.

This suggests a combined approach which can leverage the strengths of each approach by using the chess-specific features when they are available, and combining that with results from statistical learning. For instance, while the chess engine can easily identify imminent mate combinations, it would require a coordination of both the engine and the statistical learning tool to identify threshold functions like `kingToBadSquares`. Machine learning would be used to optimize weights and thresholds (e.g. for whether a king's position is bad) while the engine would be used to explore future paths. For example, a deep traversal of many future paths by a chess engine may reveal that an attack is possible (`attackPossible`) or that the player can force a pawn promotion (`pawnPromotion`).

7.4 Generalization to Other Games

One goal of this project was to develop a statistical machine learning approach that was general enough to be easily adapted to other applications like StarCraft or Checkers. Given the light dependence on chess-specific concepts of the approach described in this paper, specifically with the auto-encoder, we should be able to easily adapt this to other turn-based games.

8 Conclusion

We built two statistical learning systems to classify high-level positional concepts for chess, and described various issues involved in implementing these systems. Both classifiers are feature driven and classified by a support vector machine. The first system uses manually generated features for a chessboard configuration and the move that led to it, while the second system uses a sparse auto-encoder to automatically discover high-level structures (and hence features) for a given chessboard configuration in an unsupervised fashion. We found that the auto-encoder-based classifier outperformed the manual feature vector-based classifier, which outperformed the random baseline classifier: on the same segmented data set, the random baseline classifier had an F1 score of $\mathbf{7.60}$, the best manual feature vector-based classifier had an F1 score of $\mathbf{14.05}$, and the best auto-encoder-based classifier had an F1 score of $\mathbf{24.84}$.

9 Acknowledgements

We would like to thank Dan Klein for teaching us statistical machine learning techniques, Darren Kuo for helping with initial planning and game deduplication, and Soda Hall for accommodating us for many days and nights.

References

Yoshua Bengio, Pascal Lamblin, Dan Popovici, Hugo Larochelle, Universit De Montral, and Montral Qubec. 2007. Greedy layer-wise training of deep networks. In *In NIPS*. MIT Press.

Yoshua Bengio. 2009. *Learning Deep Architectures for AI*. Now Publishers.

David L. Chen and Raymond J. Mooney. 2008. Learning to sportscast: A test of grounded language acquisition. In *Proceedings of 25th International Conference on Machine Learning (ICML-2008)*, Helsinki, Finland, July.

Adam Coates, Honglak Lee, and Andrew Y. Ng. 2010. An analysis of single-layer networks in unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*.

Steven J. Edwards. 1994. Standard: Portable game notation specification and implementation guide.

Matej Guid, Martin Mozina, Jana Krivec, Aleksander Sadikov, and Ivan Bratko. 2008. Learning positional features for annotating chess games: A case study. In *Computers and Games*, pages 192–204.

Thorsten Joachims. 1999. Making large-scale support vector machine learning practical. pages 169–184.

Honglak Lee, Peter Pham, Yan Largman, and Andrew Ng. 2009. Unsupervised feature learning for audio classification using convolutional deep belief networks. In Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 1096–1104. The MIT Press.

Tim Love. 2003. Computers and chess annotation, January.

Andrew Ng. 2011. Cs294a lecture notes, January.

Richard Socher, Christopher Manning, and Andrew Ng. 2010. Learning Continuous Phrase Representations and Syntactic Parsing with Recursive Neural Networks. In *NIPS*2010 Workshop on Deep Learning and Unsupervised Feature Learning*.

Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. 2010. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. Mach. Learn. Res.*, 9999:3371–3408, December.