Learning to Win by Reading Manuals in a Monte-Carlo Framework

S.R.K. Branavan **David Silver *** **Regina Barzilay**

Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
{branavan, regina}@csail.mit.edu

* Department of Computer Science
University College London
d.silver@cs.ucl.ac.uk

Abstract

This paper presents a novel approach for leveraging automatically extracted textual knowledge to improve the performance of control applications such as games. Our ultimate goal is to enrich a stochastic player with high-level guidance expressed in text. Our model jointly learns to identify text that is relevant to a given game state in addition to learning game strategies guided by the selected text. Our method operates in the Monte-Carlo search framework, and learns both text analysis and game strategies based only on environment feedback. We apply our approach to the complex strategy game Civilization II using the official game manual as the text guide. Our results show that a linguistically-informed game-playing agent significantly outperforms its language-unaware counterpart, yielding a 27% absolute improvement and winning over 78% of games when playing against the built-in AI of Civilization II. [1]

1 Introduction

In this paper, we study the task of grounding linguistic analysis in control applications such as computer games. In these applications, an agent attempts to optimize a utility function (e.g., game score) by learning to select situation-appropriate actions. In complex domains, finding a winning strategy is challenging even for humans. Therefore, human players typically rely on manuals and guides that describe promising tactics and provide general advice about the underlying task. Surprisingly, such textual information has never been utilized in control algorithms despite its potential to greatly improve performance.

The natural resources available where a population settles affects its ability to produce food and goods. Build your city on a plains or grassland square with a river running through it if possible.

Figure 1: An excerpt from the user manual of the game Civilization II.

Consider for instance the text shown in Figure 1. This is an excerpt from the user manual of the game Civilization II.[2] This text describes game locations where the action "build-city" can be effectively applied. A stochastic player that does not have access to this text would have to gain this knowledge the hard way: it would repeatedly attempt this action in a myriad of states, thereby learning the characterization of promising state-action pairs based on the observed game outcomes. In games with large state spaces, long planning horizons, and high-branching factors, this approach can be prohibitively slow and ineffective. An algorithm with access to the text, however, could learn correlations between words in the text and game attributes – e.g., the word "river" and places with rivers in the game – thus leveraging strategies described in text to better select actions.

The key technical challenge in leveraging textual knowledge is to automatically extract relevant information from text and incorporate it effectively into a control algorithm. Approaching this task in a supervised framework, as is common in traditional information extraction, is inherently difficult. Since the game's state space is extremely large, and the states that will be encountered during game play cannot be known a priori, it is impractical to manually annotate the information that would be relevant to those states. Instead, we propose to learn text analysis based on a feedback signal inherent to the control application, such as game score.

[1] The code, data and complete experimental setup for this work are available at http://groups.csail.mit.edu/rbg/code/civ.

[2] http://en.wikipedia.org/wiki/Civilization_II

Our general setup consists of a game in a stochastic environment, where the goal of the player is to maximize a given utility function $R(s)$ at state s. We follow a common formulation that has been the basis of several successful applications of machine learning to games. The player's behavior is determined by an action-value function $Q(s, a)$ that assesses the goodness of an action a in a given state s based on the features of s and a. This function is learned based solely on the utility $R(s)$ collected via simulated game-play in a Monte-Carlo framework.

An obvious way to enrich the model with textual information is to augment the action-value function with word features in addition to state and action features. However, adding all the words in the document is unlikely to help since only a small fraction of the text is relevant for a given state. Moreover, even when the relevant sentence is known, the mapping between raw text and the action-state representation may not be apparent. This representation gap can be bridged by inducing a predicate structure on the sentence—e.g., by identifying words that describe actions, and those that describe state attributes.

In this paper, we propose a method for learning an action-value function augmented with linguistic features, while simultaneously modeling sentence relevance and predicate structure. We employ a multi-layer neural network where the hidden layers represent sentence relevance and predicate parsing decisions. Despite the added complexity, all the parameters of this non-linear model can be effectively learned via Monte-Carlo simulations.

We test our method on the strategy game Civilization II, a notoriously challenging game with an immense action space.[3] As a source of knowledge for guiding our model, we use the official game manual. As a baseline, we employ a similar Monte-Carlo search based player which does not have access to textual information. We demonstrate that the linguistically-informed player significantly outperforms the baseline in terms of number of games won. Moreover, we show that modeling the deeper linguistic structure of sentences further improves performance. In full-length games, our algorithm yields a 27% improvement over a language unaware baseline, and wins over 78% of games against the built-in, hand-crafted AI of Civilization II.[4]

2 Related Work

Our work fits into the broad area of grounded language acquisition where the goal is to learn linguistic analysis from a situated context (Oates, 2001; Siskind, 2001; Yu and Ballard, 2004; Fleischman and Roy, 2005; Mooney, 2008a; Mooney, 2008b; Branavan et al., 2009; Vogel and Jurafsky, 2010). Within this line of work, we are most closely related to reinforcement learning approaches that learn language by proactively interacting with an external environment (Branavan et al., 2009; Branavan et al., 2010; Vogel and Jurafsky, 2010). Like the above models, we use environment feedback (in the form of a utility function) as the main source of supervision. The key difference, however, is in the language interpretation task itself. Previous work has focused on the interpretation of instruction text where input documents specify a set of actions to be executed in the environment. In contrast, game manuals provide high-level advice but do not directly describe the correct actions for every potential game state. Moreover, these documents are long, and use rich vocabularies with complex grammatical constructions. We do not aim to perform a comprehensive interpretation of such documents. Rather, our focus is on language analysis that is sufficiently detailed to help the underlying control task.

The area of language analysis situated in a game domain has been studied in the past (Eisenstein et al., 2009). Their method, however, is different both in terms of the target interpretation task, and the supervision signal it learns from. They aim to learn the rules of a given game, such as which moves are valid, given documents describing the rules. Our goal is more open ended, in that we aim to learn winning game strategies. Furthermore, Eisenstein et al. (2009) rely on a different source of supervision – game traces collected a priori. For complex games, like the one considered in this paper, collecting such game traces is prohibitively expensive. Therefore our approach learns by actively playing the game.

[3]Civilization II was #3 in IGN's 2007 list of top video games of all time (http://top100.ign.com/2007/ign_top_game_3.html)

[4]In this paper, we focus primarily on the linguistic aspects of our task and algorithm. For a discussion and evaluation of the non-linguistic aspects please see Branavan et al. (2011).

3 Monte-Carlo Framework for Computer Games

Our method operates within the Monte-Carlo search framework (Tesauro and Galperin, 1996), which has been successfully applied to complex computer games such as Go, Poker, Scrabble, multi-player card games, and real-time strategy games, among others (Gelly et al., 2006; Tesauro and Galperin, 1996; Billings et al., 1999; Sheppard, 2002; Schäfer, 2008; Sturtevant, 2008; Balla and Fern, 2009). Since Monte-Carlo search forms the foundation of our approach, we briefly describe it in this section.

Game Representation The game is defined by a large Markov Decision Process $\langle S, A, T, R \rangle$. Here S is the set of possible states, A is the space of legal actions, and $T(s'|s,a)$ is a stochastic state transition function where $s, s' \in S$ and $a \in A$. Specifically, a state encodes attributes of the game world, such as available resources and city locations. At each step of the game, a player executes an action a which causes the current state s to change to a new state s' according to the transition function $T(s'|s,a)$. While this function is not known a priori, the program encoding the game can be viewed as a black box from which transitions can be sampled. Finally, a given utility function $R(s) \in \mathbb{R}$ captures the likelihood of winning the game from state s (e.g., an intermediate game score).

Monte-Carlo Search Algorithm The goal of the Monte-Carlo search algorithm is to dynamically select the best action for the current state s_t. This selection is based on the results of multiple *roll-outs* which measure the outcome of a sequence of actions in a simulated game – e.g., simulations played against the game's built-in AI. Specifically, starting at state s_t, the algorithm repeatedly selects and executes actions, sampling state transitions from T. On game completion at time τ, we measure the final utility $R(s_\tau)$.[5] The actual game action is then selected as the one corresponding to the roll-out with the best final utility. See Algorithm 1 for details.

The success of Monte-Carlo search is based on its ability to make a fast, local estimate of the action quality at each step of the roll-outs. States and actions are evaluated by an *action-value function* $Q(s,a)$, which is an estimate of the expected outcome of action a in state s. This action-value function is used to guide action selection during the roll-outs. While actions are usually selected to maximize the action-value function, sometimes other actions are also randomly explored in case they are more valuable than predicted by the current estimate of $Q(s,a)$. As the accuracy of $Q(s,a)$ improves, the quality of action selection improves and vice

---

procedure PlayGame ()

Initialize game state to fixed starting state
$s_1 \leftarrow s_0$

for $t = 1 \dots T$ **do**
  Run N simulated games
  for $i = 1 \dots N$ **do**
    $(a_i, r_i) \leftarrow \text{SimulateGame}(s)$
  end
  Compute average observed utility for each action
  $$a_t \leftarrow \arg\max_a \frac{1}{N_a} \sum_{i:a_i=a} r_i$$
  Execute selected action in game
  $s_{t+1} \leftarrow T(s'|s_t, a_t)$
end

---

procedure SimulateGame (s_t)

for $u = t \dots \tau$ **do**
  Compute Q function approximation
  $Q(s,a) = \vec{w} \cdot \vec{f}(s,a)$
  Sample action from action-value function in ϵ-greedy fashion:
  $$a_u \sim \begin{cases} \text{uniform}(a \in A) & \text{with probability } \epsilon \\ \arg\max_a Q(s,a) & \text{otherwise} \end{cases}$$
  Execute selected action in game:
  $s_{u+1} \leftarrow T(s'|s_u, a_u)$
  if game is won or lost **break**
end
Update parameters $\vec{w}$ of $Q(s,a)$
Return action and observed utility:
return $a_t, R(s_\tau)$

Algorithm 1: The general Monte-Carlo algorithm.

[5] In general, roll-outs are run till game completion. However, if simulations are expensive as is the case in our domain, roll-outs can be truncated after a fixed number of steps.

versa, in a cycle of continual improvement (Sutton and Barto, 1998).

In many games, it is sufficient to maintain a distinct action-value for each unique state and action in a large search tree. However, when the branching factor is large it is usually beneficial to approximate the action-value function, so that the value of many related states and actions can be learned from a reasonably small number of simulations (Silver, 2009). One successful approach is to model the action-value function as a linear combination of state and action attributes (Silver et al., 2008):

$$Q(s,a) = \vec{w} \cdot \vec{f}(s,a).$$

Here $\vec{f}(s,a) \in \mathbb{R}^n$ is a real-valued feature function, and $\vec{w}$ is a weight vector. We take a similar approach here, except that our feature function includes latent structure which models language.

The parameters $\vec{w}$ of $Q(s,a)$ are learned based on feedback from the roll-out simulations. Specifically, the parameters are updated by stochastic gradient descent by comparing the current predicted $Q(s,a)$ against the observed utility at the end of each roll-out. We provide details on parameter estimation in the context of our model in Section 4.2.

The roll-outs themselves are fully guided by the action-value function. At every step of the simulation, actions are selected by an ϵ-greedy strategy: with probability ϵ an action is selected uniformly at random; otherwise the action is selected greedily to maximize the current action-value function, $\arg\max_a Q(s,a)$.

4 Adding Linguistic Knowledge to the Monte-Carlo Framework

In this section we describe how we inform the simulation-based player with information automatically extracted from text – in terms of both model structure and parameter estimation.

4.1 Model Structure

To inform action selection with the advice provided in game manuals, we modify the action-value function $Q(s,a)$ to take into account words of the document in addition to state and action information. Conditioning $Q(s,a)$ on all the words in the document is unlikely to be effective since only a small

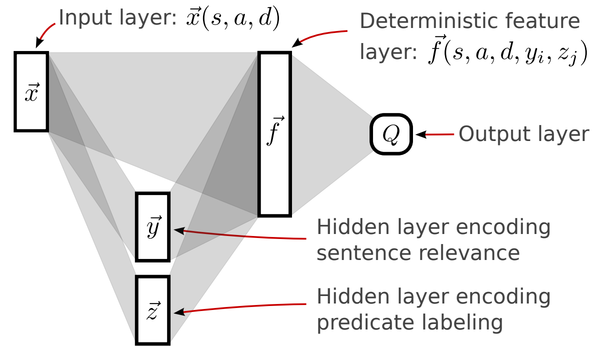

Figure 2: The structure of our model. Each rectangle represents a collection of units in a layer, and the shaded trapezoids show the connections between layers. A fixed, real-valued feature function $\vec{x}(s,a,d)$ transforms the game state s, action a, and strategy document d into the input vector $\vec{x}$. The first hidden layer contains two disjoint sets of units $\vec{y}$ and $\vec{z}$ corresponding to linguistic analyzes of the strategy document. These are softmax layers, where only one unit is active at any time. The units of the second hidden layer $\vec{f}(s,a,d,y_i,z_i)$ are a set of fixed real valued feature functions on s, a, d and the active units y_i and z_i of $\vec{y}$ and $\vec{z}$ respectively.

fraction of the document provides guidance relevant to the current state, while the remainder of the text is likely to be irrelevant. Since this information is not known a priori, we model the decision about a sentence's relevance to the current state as a hidden variable. Moreover, to fully utilize the information presented in a sentence, the model identifies the words that describe *actions* and those that describe *state attributes*, discriminating them from the rest of the sentence. As with the relevance decision, we model this labeling using hidden variables.

As shown in Figure 2, our model is a four layer neural network. The *input layer* $\vec{x}$ represents the current state s, candidate action a, and document d. The *second layer* consists of two disjoint sets of units $\vec{y}$ and $\vec{z}$ which encode the sentence-relevance and predicate-labeling decisions respectively. Each of these sets of units operates as a stochastic 1-of-n softmax selection layer (Bridle, 1990) where only a single unit is activated. The activation function for units in this layer is the standard softmax function:

$$p(y_i = 1 | \vec{x}) = e^{\vec{u}_i \cdot \vec{x}} \Big/ \sum_k e^{\vec{u}_k \cdot \vec{x}},$$

where y_i is the i^{th} hidden unit of $\vec{y}$, and $\vec{u}_i$ is the weight vector corresponding to y_i. Given this activation function, the second layer effectively models sentence relevance and predicate labeling decisions via log-linear distributions, the details of which are described below.

The third *feature layer* $\vec{f}$ of the neural network is deterministically computed given the active units y_i and z_j of the softmax layers, and the values of the input layer. Each unit in this layer corresponds to a fixed feature function $f_k(s_t, a_t, d, y_i, z_j) \in \mathbb{R}$. Finally the *output layer* encodes the action-value function $Q(s, a, d)$, which now also depends on the document d, as a weighted linear combination of the units of the feature layer:

$$Q(s_t, a_t, d) = \vec{w} \cdot \vec{f},$$

where $\vec{w}$ is the weight vector.

Modeling Sentence Relevance Given a strategy document d, we wish to identify a sentence y_i that is most relevant to the current game state s_t and action a_t. This relevance decision is modeled as a log-linear distribution over sentences as follows:

$$p(y_i|s_t, a_t, d) \propto e^{\vec{u} \cdot \phi(y_i, s_t, a_t, d)}.$$

Here $\phi(y_i, s_t, a_t, d) \in \mathbb{R}^n$ is a feature function, and $\vec{u}$ are the parameters we need to estimate.

Modeling Predicate Structure Our goal here is to label the words of a sentence as either *action-description*, *state-description* or *background*. Since these word label assignments are likely to be mutually dependent, we model predicate labeling as a sequence prediction task. These dependencies do not necessarily follow the order of words in a sentence, and are best expressed in terms of a syntactic tree. For example, words corresponding to *state-description* tend to be descendants of *action-description* words. Therefore, we label words in dependency order — i.e., starting at the root of a given dependency tree, and proceeding to the leaves. This allows a word's label decision to condition on the label of the corresponding dependency tree parent.

Given sentence y_i and its dependency parse q_i, we model the distribution over predicate labels $\vec{e}_i$ as:

$$\begin{aligned} p(\vec{e}_i \,|y_i, q_i) &= \prod_j p(e_j|j, \vec{e}_{1:j-1}, y_i, q_i), \\ p(e_j|j, \vec{e}_{1:j-1}, y_i, q_i) &\propto e^{\vec{v} \cdot \psi(e_j, j, \vec{e}_{1:j-1}, y_i, q_i)}. \end{aligned}$$

Here e_j is the predicate label of the j^{th} word being labeled, and $\vec{e}_{1:j-1}$ is the partial predicate labeling constructed so far for sentence y_i.

In the second layer of the neural network, the units $\vec{z}$ represent a predicate labeling $\vec{e}_i$ of every sentence $y_i \in d$. However, our intention is to incorporate, into action-value function Q, information from only the most relevant sentence. Thus, in practice, we only perform a predicate labeling of the sentence selected by the relevance component of the model.

Given the sentence selected as relevant and its predicate labeling, the output layer of the network can now explicitly learn the correlations between textual information, and game states and actions – for example, between the word "grassland" in Figure 1, and the action of building a city. This allows our method to leverage the automatically extracted textual information to improve game play.

4.2 Parameter Estimation

Learning in our method is performed in an online fashion: at each game state s_t, the algorithm performs a simulated game roll-out, observes the outcome of the game, and updates the parameters $\vec{u}$, $\vec{v}$ and $\vec{w}$ of the action-value function $Q(s_t, a_t, d)$. These three steps are repeated a fixed number of times at each actual game state. The information from these roll-outs is used to select the actual game action. The algorithm re-learns $Q(s_t, a_t, d)$ for every new game state s_t. This specializes the action-value function to the subgame starting from s_t.

Since our model is a non-linear approximation of the underlying action-value function of the game, we learn model parameters by applying non-linear regression to the observed final utilities from the simulated roll-outs. Specifically, we adjust the parameters by stochastic gradient descent, to minimize the mean-squared error between the action-value $Q(s, a)$ and the final utility $R(s_\tau)$ for each observed game state s and action a. The resulting update to model parameters θ is of the form:

$$\begin{aligned} \Delta\theta &= -\frac{\alpha}{2} \nabla_\theta \left[R(s_\tau) - Q(s, a) \right]^2 \\ &= \alpha \left[R(s_\tau) - Q(s, a) \right] \nabla_\theta Q(s, a; \theta), \end{aligned}$$

where α is a learning rate parameter.

This minimization is performed via standard error backpropagation (Bryson and Ho, 1969; Rumelhart et al., 1986), which results in the following online updates for the output layer parameters $\vec{w}$:

$$\vec{w} \leftarrow \vec{w} + \alpha_w \left[Q - R(s_\tau)\right] \vec{f}(s, a, d, y_i, z_j),$$

where α_w is the learning rate, and $Q = Q(s, a, d)$. The corresponding updates for the sentence relevance and predicate labeling parameters $\vec{u}$ and $\vec{v}$ are:

$$\vec{u}_i \leftarrow \vec{u}_i + \alpha_u \left[Q - R(s_\tau)\right] Q \, \vec{x} \left[1 - p(y_i|\cdot)\right],$$

$$\vec{v}_i \leftarrow \vec{v}_i + \alpha_v \left[Q - R(s_\tau)\right] Q \, \vec{x} \left[1 - p(z_i|\cdot)\right].$$

5 Applying the Model

We apply our model to playing the turn-based strategy game, Civilization II. We use the official manual [6] of the game as the source of textual strategy advice for the language aware algorithms.

Civilization II is a multi-player game set on a grid-based map of the world. Each grid location represents a tile of either land or sea, and has various resources and terrain attributes. For example, land tiles can have hills with rivers running through them. In addition to multiple cities, each player controls various units – e.g., settlers and explorers. Games are won by gaining control of the entire world map. In our experiments, we consider a two-player game of Civilization II on a grid of 1000 squares, where we play against the built-in AI player.

Game States and Actions We define the game state of Civilization II to be the map of the world, the attributes of each map tile, and the attributes of each player's cities and units. Some examples of the attributes of states and actions are shown in Figure 3. The space of possible actions for a given city or unit is known given the current game state. The actions of a player's cities and units combine to form the action space of that player. In our experiments, on average a player controls approximately 18 units, and each unit can take one of 15 actions. This results in a very large action space for the game – i.e., 10^{21}. To effectively deal with this large action space, we assume that given the state, the actions of a single unit are independent of the actions of all other units of the same player.

Utility Function The Monte-Carlo algorithm uses the utility function to evaluate the outcomes of

[6] www.civfanatics.com/content/civ2/reference/Civ2manual.zip

Map tile attributes:
- Terrain type (e.g. grassland, mountain, etc)
- Tile resources (e.g. wheat, coal, wildlife, etc)

City attributes:
- City population
- Amount of food produced

Unit attributes:
- Unit type (e.g., worker, explorer, archer, etc)
- Is unit in a city ?

$$f_1(s, a, v, \vec{e}) = \begin{cases} 1 \text{ if action=build-city} \\ \quad \text{\& tile-has-river=true} \\ \quad \text{\& action-words=\{build,city\}} \\ \quad \text{\& state-words=\{river,hill\}} \\ 0 \text{ otherwise} \end{cases}$$

$$\phi_1(v, s, a) = \begin{cases} 1 \text{ if action=build-city} \\ \quad \text{\& tile-has-river=true} \\ \quad \text{\& words=\{build,city,river\}} \\ 0 \text{ otherwise} \end{cases}$$

$$\psi_1(e_i, i, \vec{e}_{1:i-1}, v, z_v) = \begin{cases} 1 \text{ if label=action} \\ \quad \text{\& word-type='build'} \\ \quad \text{\& parent-label=action} \\ 0 \text{ otherwise} \end{cases}$$

Figure 3: Example attributes of the game (box above), and features computed using the game manual and these attributes (box below).

simulated game roll-outs. In the typical application of the algorithm, the final game outcome is used as the utility function (Tesauro and Galperin, 1996). Given the complexity of Civilization II, running simulation roll-outs until game completion is impractical. The game, however, provides each player with a *game score*, which is a noisy indication of how well they are currently playing. Since we are playing a two-player game, we use the ratio of the game score of the two players as our utility function.

Features The sentence relevance features $\vec{\phi}$ and the action-value function features $\vec{f}$ consider the attributes of the game state and action, and the words of the sentence. Some of these features compute text overlap between the words of the sentence, and text labels present in the game. The feature function $\vec{\psi}$ used for predicate labeling on the other hand operates only on a given sentence and its dependency parse. It computes features which are the Cartesian product of the candidate predicate label with word attributes such as type, part-of-speech tag, and dependency parse information. Overall, $\vec{f}$, $\vec{\phi}$ and $\vec{\psi}$ compute approximately 306,800, 158,500, and 7,900 features respectively. Figure 3 shows some examples of these features.

6 Experimental Setup

Datasets We use the official game manual for Civilization II as our strategy guide. This manual uses a large vocabulary of 3638 words, and is composed of 2083 sentences, each on average 16.9 words long.

Experimental Framework To apply our method to the Civilization II game, we use the game's open source implementation *Freeciv*.[7] We instrument the game to allow our method to programmatically measure the current state of the game and to execute game actions. The Stanford parser (de Marneffe et al., 2006) was used to generate the dependency parse information for sentences in the game manual.

Across all experiments, we start the game at the same initial state and run it for 100 steps. At each step, we perform 500 Monte-Carlo roll-outs. Each roll-out is run for 20 simulated game steps before halting the simulation and evaluating the outcome. For our method, and for each of the baselines, we run 200 independent games in the above manner, with evaluations averaged across the 200 runs. We use the same experimental settings across all methods, and all model parameters are initialized to zero.

The test environment consisted of typical PCs with single Intel Core i7 CPUs (4 hyper-threaded cores each), with the algorithms executing 8 simulation roll-outs in parallel. In this setup, a single game of 100 steps runs in approximately 1.5 hours.

Evaluation Metrics We wish to evaluate two aspects of our method: how well it leverages textual information to improve game play, and the accuracy of the linguistic analysis it produces. We evaluate the first aspect by comparing our method against various baselines in terms of the percentage of games won against the built-in AI of Freeciv. This AI is a fixed algorithm designed using extensive knowledge of the game, with the intention of challenging human players. As such, it provides a good open-reference baseline. Since full games can last for multiple days, we compute the percentage of games won within the first 100 game steps as our primary evaluation. To confirm that performance under this evaluation is meaningful, we also compute the percentage of full games won over 50 independent runs, where each game is run to completion.

[7]http://freeciv.wikia.com. Game version 2.2

Method	% Win	% Loss	Std. Err.
Random	0	100	—
Built-in AI	0	0	—
Game only	17.3	5.3	$\pm$ 2.7
Sentence relevance	46.7	2.8	$\pm$ 3.5
Full model	**53.7**	5.9	$\pm$ 3.5
Random text	40.3	4.3	$\pm$ 3.4
Latent variable	26.1	3.7	$\pm$ 3.1

Table 1: Win rate of our method and several baselines within the first 100 game steps, while playing against the built-in game AI. Games that are neither won nor lost are still ongoing. Our model's win rate is statistically significant against all baselines except *sentence relevance*. All results are averaged across 200 independent game runs. The standard errors shown are for percentage wins.

Method	% Wins	Standard Error
Game only	45.7	$\pm$ 7.0
Latent variable	62.2	$\pm$ 6.9
Full model	**78.8**	$\pm$ 5.8

Table 2: Win rate of our method and two baselines on 50 full length games played against the built-in AI.

7 Results

Game performance As shown in Table 1, our language aware Monte-Carlo algorithm substantially outperforms several baselines – on average winning 53.7% of all games within the first 100 steps. The dismal performance, on the other hand, of both the *random* baseline and the game's own *built-in AI* (playing against itself) is an indicator of the difficulty of the task. This evaluation is an underestimate since it assumes that any game not won within the first 100 steps is a loss. As shown in Table 2, our method wins over 78% of full length games.

To characterize the contribution of the language components to our model's performance, we compare our method against two ablative baselines. The first of these, *game-only*, does not take advantage of any textual information. It attempts to model the action value function $Q(s, a)$ only in terms of the attributes of the game state and action. The performance of this baseline – a win rate of 17.3% – effectively confirms the benefit of automatically extracted textual information in the context of our task. The second ablative baseline, *sentence-relevance*, is

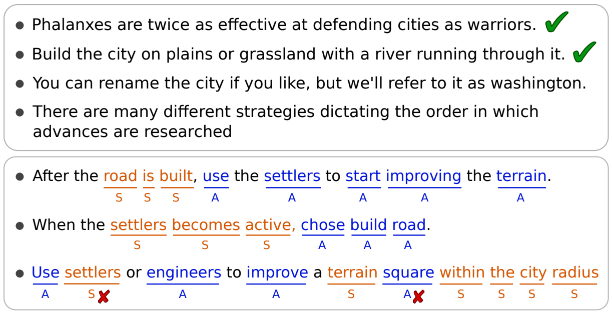

Figure 4: Examples of our method's sentence relevance and predicate labeling decisions. The box above shows two sentences (identified by check marks) which were predicted as relevant, and two which were not. The box below shows the predicted predicate structure of three sentences, with "S" indicating *state* description,"A" *action* description and *background* words unmarked. Mistakes are identified with crosses.

identical to our model, but lacks the predicate labeling component. This method wins 46.7% of games, showing that while identifying the text relevant to the current game state is essential, a deeper structural analysis of the extracted text provides substantial benefits.

One possible explanation for the improved performance of our method is that the non-linear approximation simply models game characteristics better, rather than modeling textual information. We directly test this possibility with two additional baselines. The first, *random-text*, is identical to our full model, but is given a document containing random text. We generate this text by randomly permuting the word locations of the actual game manual, thereby maintaining the document's overall statistical properties. The second baseline, *latent variable*, extends the linear action-value function $Q(s, a)$ of the *game only* baseline with a set of latent variables – i.e., it is a four layer neural network, where the second layer's units are activated only based on game information. As shown in Table 1 both of these baselines significantly underperform with respect to our model, confirming the benefit of automatically extracted textual information in the context of this task.

Sentence Relevance Figure 4 shows examples of the sentence relevance decisions produced by our method. To evaluate the accuracy of these decisions, we ideally require a ground-truth relevance annotation of the game's user manual. This however, is

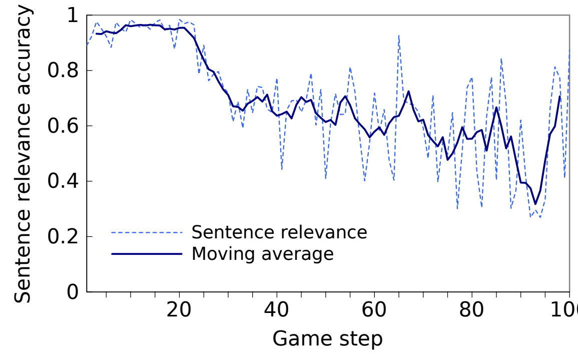

Figure 5: Accuracy of our method's sentence relevance predictions, averaged over 100 independent runs.

impractical since the relevance decision is dependent on the game context, and is hence specific to each time step of each game instance. Therefore, for the purposes of this evaluation, we modify the game manual by adding to it sentences randomly selected from the Wall Street Journal corpus (Marcus et al., 1993) – sentences that are highly unlikely to be relevant to game play. We then evaluate the accuracy with which sentences from the original manual are picked as relevant.

In this evaluation, our method achieves an average accuracy of 71.8%. Given that our model only has to differentiate between the game manual text and the Wall Street Journal, this number may seem disappointing. Furthermore, as can be seen from Figure 5, the sentence relevance accuracy varies widely as the game progresses, with a high average of 94.2% during the initial 25 game steps.

In reality, this pattern of high initial accuracy followed by a lower average is not entirely surprising: the official game manual for Civilization II is written for first time players. As such, it focuses on the initial portion of the game, providing little strategy advice relevant to subsequence game play.[8] If this is the reason for the observed sentence relevance trend, we would also expect the final layer of the neural network to emphasize game features over text features after the first 25 steps of the game. This is indeed the case, as can be seen from Figure 6.

To further test this hypothesis, we perform an experiment where the first 50 steps of the game are played using our full model, and the subsequent 50 steps are played without using any textual informa-

[8]This is reminiscent of *opening books* for games like Chess or Go, which aim to guide the player to a playable middle game.

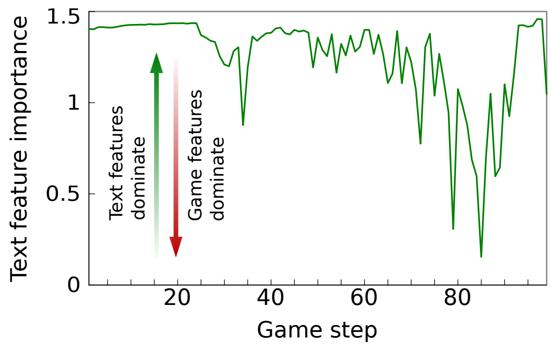

Figure 6: Difference between the norms of the text features and game features of the output layer of the neural network. Beyond the initial 25 steps of the game, our method relies increasingly on game features.

tion. This hybrid method performs as well as our full model, achieving a 53.3% win rate, confirming that textual information is most useful during the initial phase of the game. This shows that our method is able to accurately identify relevant sentences when the information they contain is most pertinent to game play.

Predicate Labeling Figure 4 shows examples of the predicate structure output of our model. We evaluate the accuracy of this labeling by comparing it against a gold-standard annotation of the game manual. Table 3 shows the performance of our method in terms of how accurately it labels words as *state*, *action* or *background*, and also how accurately it differentiates between *state* and *action* words. In addition to showing a performance improvement over the random baseline, these results display two clear trends: first, under both evaluations, labeling accuracy is higher during the initial stages of the game. This is to be expected since the model relies heavily on textual features only during the beginning of the game (see Figure 6). Second, the model clearly performs better in differentiating between state and action words, rather than in the three-way labeling.

To verify the usefulness of our method's predicate labeling, we perform a final set of experiments where predicate labels are selected uniformly at random within our full model. This random labeling results in a win rate of 44% – a performance similar to the *sentence relevance* model which uses no predicate information. This confirms that our method is able identify a predicate structure which, while noisy, provides information relevant to game play.

Method	S/A/B	S/A
Random labeling	33.3%	50.0%
Model, first 100 steps	45.1%	78.9%
Model, first 25 steps	48.0%	92.7%

Table 3: Predicate labeling accuracy of our method and a random baseline. Column "S/A/B" shows performance on the three-way labeling of words as *state*, *action* or *background*, while column "S/A" shows accuracy on the task of differentiating between state and action words.

game attribute	word
state: grassland	"city"
state: grassland	"build"
action: settlers_build_city	"city"
action: set_research	"discovery"

Figure 7: Examples of word to game attribute associations that are learned via the feature weights of our model.

Figure 7 shows examples of how this textual information is grounded in the game, by way of the associations learned between words and game attributes in the final layer of the full model.

8 Conclusions

In this paper we presented a novel approach for improving the performance of control applications by automatically leveraging high-level guidance expressed in text documents. Our model, which operates in the Monte-Carlo framework, jointly learns to identify text relevant to a given game state in addition to learning game strategies guided by the selected text. We show that this approach substantially outperforms language-unaware alternatives while learning only from environment feedback.

Acknowledgments

The authors acknowledge the support of the NSF (CAREER grant IIS-0448168, grant IIS-0835652), DARPA Machine Reading Program (FA8750-09-C-0172) and the Microsoft Research New Faculty Fellowship. Thanks to Michael Collins, Tommi Jaakkola, Leslie Kaelbling, Nate Kushman, Sasha Rush, Luke Zettlemoyer, the MIT NLP group, and the ACL reviewers for their suggestions and comments. Any opinions, findings, conclusions, or recommendations expressed in this paper are those of the authors, and do not necessarily reflect the views of the funding organizations.

References

R. Balla and A. Fern. 2009. UCT for tactical assault planning in real-time strategy games. In *21st International Joint Conference on Artificial Intelligence*.

Darse Billings, Lourdes Peña Castillo, Jonathan Schaeffer, and Duane Szafron. 1999. Using probabilistic knowledge and simulation to play poker. In *16th National Conference on Artificial Intelligence*, pages 697–703.

S.R.K Branavan, Harr Chen, Luke Zettlemoyer, and Regina Barzilay. 2009. Reinforcement learning for mapping instructions to actions. In *Proceedings of ACL*, pages 82–90.

S.R.K Branavan, Luke Zettlemoyer, and Regina Barzilay. 2010. Reading between the lines: Learning to map high-level instructions to commands. In *Proceedings of ACL*, pages 1268–1277.

S.R.K. Branavan, David Silver, and Regina Barzilay. 2011. Non-linear monte-carlo search in civilization ii. In *Proceedings of IJCAI*.

John S. Bridle. 1990. Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters. In *Advances in NIPS*, pages 211–217.

Arthur E. Bryson and Yu-Chi Ho. 1969. *Applied optimal control: optimization, estimation, and control*. Blaisdell Publishing Company.

Marie-Catherine de Marneffe, Bill MacCartney, and Christopher D. Manning. 2006. Generating typed dependency parses from phrase structure parses. In *LREC 2006*.

Jacob Eisenstein, James Clarke, Dan Goldwasser, and Dan Roth. 2009. Reading to learn: Constructing features from semantic abstracts. In *Proceedings of EMNLP*, pages 958–967.

Michael Fleischman and Deb Roy. 2005. Intentional context in situated natural language learning. In *Proceedings of CoNLL*, pages 104–111.

S. Gelly, Y. Wang, R. Munos, and O. Teytaud. 2006. Modification of UCT with patterns in Monte-Carlo Go. Technical Report 6062, INRIA.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, 19(2):313–330.

Raymond J. Mooney. 2008a. Learning language from its perceptual context. In *Proceedings of ECML/PKDD*.

Raymond J. Mooney. 2008b. Learning to connect language and perception. In *Proceedings of AAAI*, pages 1598–1601.

James Timothy Oates. 2001. *Grounding knowledge in sensors: Unsupervised learning for language and planning*. Ph.D. thesis, University of Massachusetts Amherst.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. 1986. Learning representations by back-propagating errors. *Nature*, 323:533–536.

J. Schäfer. 2008. The UCT algorithm applied to games with imperfect information. Diploma Thesis. Otto-von-Guericke-Universität Magdeburg.

B. Sheppard. 2002. World-championship-caliber Scrabble. *Artificial Intelligence*, 134(1-2):241–275.

D. Silver, R. Sutton, and M. Müller. 2008. Sample-based learning and search with permanent and transient memories. In *25th International Conference on Machine Learning*, pages 968–975.

D. Silver. 2009. *Reinforcement Learning and Simulation-Based Search in the Game of Go*. Ph.D. thesis, University of Alberta.

Jeffrey Mark Siskind. 2001. Grounding the lexical semantics of verbs in visual perception using force dynamics and event logic. *Journal of Artificial Intelligence Research*, 15:31–90.

N. Sturtevant. 2008. An analysis of UCT in multi-player games. In *6th International Conference on Computers and Games*, pages 37–49.

Richard S. Sutton and Andrew G. Barto. 1998. *Reinforcement Learning: An Introduction*. The MIT Press.

G. Tesauro and G. Galperin. 1996. On-line policy improvement using Monte-Carlo search. In *Advances in Neural Information Processing 9*, pages 1068–1074.

Adam Vogel and Daniel Jurafsky. 2010. Learning to follow navigational directions. In *Proceedings of the ACL*, pages 806–814.

Chen Yu and Dana H. Ballard. 2004. On the integration of grounding language and learning objects. In *Proceedings of AAAI*, pages 488–493.