CARNEGIE MELLON UNIVERSITY

School of Architecture

College of Fine Arts

Thesis

Submitted in Partial Fulfillment of the requirements for the degree of

Master of Science in Computational Design

TITLE:

Design Games and Machine Learning

A Method for the Application of Machine Learning in Design Environments

AUTHOR:

Michael Jeffers

ACCEPTED BY ADVISORY COMMITTEE:

Professor Ramesh Krishnamurti	Principal Advisor	DATE
Professor Daniel Cardoso Llach	Advisor	DATE
Professor Joshua Bard	Advisor	DATE

Acknowledgements

My sincere gratitude goes to Advisors Ramesh Krishnamurti, Joshua Bard and Pedro Veloso in helping shape the formulation, technical and philosophical hurdles of this thesis.

The Carnegie Mellon School of Architecture for providing opportunities like the Robotics Fellowship.

Jordan Parsons for being a phenomenal and supportive friend and collaborator.

Alex Fischer for sharing your experience, expertise, and guidance.

To all of my friends, colleagues, and students: it has been an honor and privilege.

Abstract

Machine learning techniques have been applied to an increasing variety of problems, of increasing complexity that were once considered infeasible for artificial intelligence solutions of any kind (Sliver et al. 2016). Design problems are part of a category of problems that are considered “wicked” (Rittell et al. 1973), such that success, and the metrics to determine success, are either undefined or mutable. We propose a solution to this by creating a shared, abstract, environment that is computable but also provides enough user-freedom to exercise a design problem through exploring solutions. We use a game-like environment to model not only complex simulations of systems, but also allow both human and AI actors to “play” in a shared space to exploring design alternatives. To address the openness of the design problem, we close the problem for the AI by providing user-controlled weights on game performance metrics. Thus allowing the AI system to optimize in favorable ways with respect to the bespoke demands and goals of the designer.

Introduction

The primary objective of the thesis is to demonstrate if there exists a viable application of machine learning techniques to the design process. The theoretical discussion will attempt to make the case this applies to any similar problem, agnostic to specific disciplines. The case study will examine the specific efficacy of an example implementation of such a system in an applied setting through a software prototype.

There are technical and theoretical constraints to both the theory and the application. Identifying particular techniques, and in what paradigms they exist within machine learning, is critical to understanding the bounds of the AI system in how would be implemented.

Classifying design itself as a type of problem is essential to understanding the feasibility of any process to addressing a design problem. The characteristics of such problems: the problem formulation is incomplete and dynamic, understanding the problem only through exploring solutions, solutions are innumerable, conflicting parameters or constraints to satisfy (Rittel et al. 1973), are well discussed in literature. This presents a unique challenge for AI, notably that AI is unable solve a problem if we can not even definitively say what the criteria is.

If the solutions to a design problem are potentially innumerable, then we need to filter the solution-space and explore by example rather than by an exhaustive mapping(Silver et al. 2016).

Machine learning algorithms need to train on data. If input data can not be classified, or scored, the algorithm has nothing with which to optimize. Design presents two issues here. The criteria to evaluate solutions is either unknown, or ambiguous, as a property of being “wicked”(Rittell et al 1973). To address the first issue, we provide a way to make this known. The designer must have a way to explicitly provide this criteria in order to for the AI to evaluate game states. Second, to adapt to changing evaluations of solutions, the AI system must be

able to learn continuously over new cases. While a learning system can fit a particular model of provided data, if the evaluative criteria of how a state is scored changes, that data no longer truly represents the current world.

We achieve this in the prototype with a game interface that lets the player weight in-game metrics that are used to produce a score. In essence, the player is determining the policy that is applied to score the game states. The AI system is implemented as Q-Learner multilayer perceptrons that employ incremental learning to continuously train over new observations. Not only to provide more cases, but to adapt to changing score-weighting on the part of the player. The player may freely decide to change the criteria applied to determine the score, thus the learning system must be able to observe this change to re-fit and model it.

The design game itself models an Urban Planning/Design problem as a city-building simulation game. This follows the language from the existing genre of City-building games, most deriving origins from the canonical SimCity (Wright & Bremer 1989). It models the problem by providing the necessary tools to manipulate only certain attributes of the City, specifically zoning. The City itself grows and decays based on its own internal rule systems and logics that attempt to model real-world behaviours and patterns, forming a simulation.

This type of game lends itself to a discrete event simulation, where the evaluative criteria of the AI is also discrete with time. Though this nature is not implicit to how other design games can or should be constructed to represent other specific design problems in a way conducive to machine learning systems. Other disciplines may benefit from entirely different representations and may also be better solved by alternative implementations of an AI system. The thesis only is directed to outline what applies in principle to all cases, and demonstrates this with a specific case through a software prototype.

Motivation

Machine Learning is being applied to an increasing variety of problems. Observing the phenomena it is clear the scope of what is being automated is expanded and breaks through traditional barriers that were once to be out of the scope of automation, or exceed human performance in those areas (Silver et al. 2016).

The question then is: what in Design can/will be automated in a similar way? We already have tools that automate menial tasks, or physical operations. There exist many algorithms that can filter or produce results but do not have a meaningful interaction with the designer. These tools are being increasing integrated into design practices, but are often in a subservient role or do not address higher level issues in the design process. What if the designer and AI interacted in the same space as equals? Is the learning of the AI or the designer around the design problem enhanced? Let us imagine a system that interacted in the same way, and same space, as the designer. But at the same time provide a way for the designer to alter the behaviour of the agent, and tell it what to prioritize. What does this achieve?

Theory

To demonstrate the plausibility of using learning systems in design environments for any such design problem, we need to establish the logics that led to this conclusion as well as determine how the problem and AI were modeled. First, we will define some terms that may deviate slightly from definitions in other disciplines. Because we are crossing disciplines with some terminology, these definitions can be ambiguous to different readers. For other technical definitions, see the Appendix.

Definitions

Game

A *game* is defined as a rule-based, constrained, abstract environment that models a *problem* where *players* interact with this environment in a knowable way. *Games* have a set of criteria that can evaluate game-states for satisfaction of various win-conditions. This set can be empty. In other words, *games* can have ends, win/lose conditions, or not. Some of these may produce comparable values from game-states. This value can be a score.

Player/User/Designer

Player or *user* is the actor interacting with the *game*. *Designer* is the actor in the context of a *design problem*. Assume rational actions on the part of these actors and assume some genuine intent to use the *game* to learn and produce favorable results to the user or solve problems. These three may at times be synonymous.

Problem and Solution

A *problem* is a set of constraints, or criteria, that requires some level of resolution or can be "satisficed" (Simon 1947). This state will be known as a *solution*. A *problem's* set of *solutions* (*solution* space) can be of any magnitude. The countability of the quantity of *solutions*, whether they are enumerable (determinate), or innumerable (indeterminate), is dependent on the *problem* type. *Problem* constraints can be absolutes or merely "satisfizable" criteria where there is an accepted threshold for satisfaction, or where any range of values is accepted. The *problem* may provide criteria that can evaluate a *solution* for fitness (how well criteria is satisfied), therefore allowing the possibility of *solutions* to be comparable. In the case where *solutions* are comparable, there are "better" or "worse" *solutions* but they are all members of the *solution* space. While a provided *solution* might "solve" the *problem*, this is not necessarily the final.

Design

Design is an activity of the "discovery and elaboration of alternatives" (Simon 1972), meaning solutions have to be found and often exist in a space that is unbounded, and therefore

not fully enumerable (see *Design Problem*). *Design* can take place through a number of disciplines or activities.

Design Problem

Following the definition of *problem*, a *design problem* is a *problem* is solved through the "discovery and elaboration of alternatives" (Simon 1972). *Design problems* are also a subset of Wicked Problems (Rittell et al. 1973). Following that definition a few properties of note:

- The *Solution-space* is not enumerable (Rittell et al, 164).
- *Solutions* are "good or bad" but not "true or false" (Rittell et al, 163).
- There is no immediate test to evaluate a *Solution* (Rittell et al, 163).

Recall our definition of *problem*: where there may be criteria that can evaluate *solutions* for fitness to the *problem's* specification. *Design problems* do not have this as part of their formulation. The criteria of evaluation may be partially or entirely absent. When a *designer* evaluates the "good"-ness of "bad"-ness of a *solution*, they apply criteria that is not explicitly part of the *problem*. Or following the "Wicked" property, the formulation of the *problem* itself is incomplete (Rittell et al, 161).

Furthermore the set of *solutions* to a *design problem* is innumerable as a property of this definition.

Open/Closed/Closable

Open is everything that is not *closed*.

Closed describes *problems* and g*ames* following the closed-world assumption definition. A *game* has axioms in the form of rules; a *problem* has constraints. An *open problem* is therefore "wicked", because there are unknown constraints. A *closed problem* can unambiguously evaluate a *solution* for truth, whereas an *open problem* can not evaluate truth (or fitness) in all cases. The same follows for *games*.

Closable describes an *open problem* or *game* that can be provided truths that could not otherwise be derived, producing a new closed-world. All unknown conditions must be addressed to provide a definitive truth value. *Solutions* to a *closable problem* therefore can be definitively evaluated.

Continuous Learning System

A *continuous learning system* is a learning system that can actively train over new cases presented to it, for example, from direct observations, Strictly speaking, a *continuous learning system* is any learning system that is not Offline Learning (see appendix). A *continuous learning system* therefore can train on data that is not yet available or not yet determined. Essentially it is a learning system that is constantly training on new data, regardless the source.

Argument Outline

Axioms

- AI can solve *closed games*, where the parameters are computable.
- *Open games* are *games* with undefined win-conditions.
- *Games* model *problems*.
- *Design problems* are o*pen*.

Statements

- An *open game* can model an *open problem*.
- An *open game* can model a *design problem*.
- Defining win-functions can *close* a *game*.
- A *design problem* can be modelled as a *closable game*.

Conclusion

∴ An *continuous learning system* can solve a *design problem* modeled as a *closable game* where the parameters are computable.

Discussion

Axioms - Soundness

AI can solve Closed Games, where the parameters are computable.

In order for an AI system to evaluate a *solution*, it must be able to compare results or compute differences. For some algorithms this is a deviation from an ideal in the form of error. Or other algorithms compare solutions and compute the difference and favor attributes of the higher scoring one. Regardless the technique, values must be computable and differences exist in a continuous metric space that can be explored.

Evidence for AI solving *games* is numerous, but two examples DeepBlue (classical AI) (Campbell et al. 2002) and AlphaGo (Deep Learning) (Silver et al. 2016) are canonical examples of such built and tested solutions. Not only can *solve* the *game*, but have exceeded human performance in both contexts respectively.

Open Games are games with undefined win-conditions(s).

Following the definition of *game*, *games* can have any quantity of win conditions. If all states of a *game* can be known to be win or lose (true or false), this is a closed-world assumption. If a *game* however has a state that can not be evaluated for win (unknown), then it

is open-world, and therefore the *game* is open because it is not *closed*. The same follows for the score function, and scores of game-states.

Games model Problems

Following the definition of *game* and *problem*, both have strong feature similarity. Both have constraints, criteria, and have a set of evaluative functions to determine if provided states are a *solution* or not. Both may have an empty set of evaluative functions. *Games* therefore represent attributes of *problems* in another medium through which it can be explored.

Design Problems are Open.

Design problems are a subset of "Wicked" *problems*. Recall the attributes of "Wicked" *problems*, namely: there is no immediate test of a *solution*. Thus a provided *solution* might be unknown state, therefore the *design problem* occupies an open-world.

Statements

An Open Game can model an Open Problem.

If *games* model *problems*, *games* can be *open*, and problems can be *open*, then *open games* can model *open problems*.

An Open Game can model a Design Problem.

Design problems are a subset of *open problems*, and therefore inherit the property above.

Defining win-functions can Close a Game.

Should sufficient definition of win conditions be provided so that all game-states can be definitively evaluated, then the *game* will be *closed*. It is possible that this alone may not close all open games. An incomplete win-criteria would still result in games-states that are unknown, and therefore remain *open*. A complete definition of win-conditions for all possible game-states will *close* the *game*.

A Design Problem can be modelled as a Closable Game.

If *Design Problems* can be modelled by *Open Games*, and *Open Games*, if provided sufficient criteria, can be *Closed*, then *Design Problems* can be modelled by *Closable Games*.

Conclusion

An Continuous Learning System can solve a Design Problem modeled as a Closable Game where the parameters are computable.

We have a *Game* that models a *Design Problem*. This *Game* has an interface where some criteria can be dynamically manipulated or added to the *Game* system to define or redefine the evaluation of a given *Game* state. Should this definition hold true for all Game-states and all *Game-states* can be evaluated, all *Game-states* exist in an explorable metric-space. If this space is computable and static, an AI can explore this space and optimize

in it. If this space is dynamic over time, only a *continuous learning system* can account and adapt to the change to refit the new world. Because a *Closable Game* is modelled as a *Game* with variable evaluative criteria (subject to change by the Player) the AI system must have some adaptive property. Any non-Offline method, allows for a learning system to training on current observations.

Should the definition of a win-condition, or *Game-state* evaluation criteria, change while maintaining a *Closed* evaluation of all *Game-states*, a *continuous learning system* can self-correct for newly detected error in fitting.

Implications and Applications

There are a few points the conclusion and argument itself bring up that need to be separately addressed as well as discuss how this then translate into a strategy to engineer not only a Game but AI to achieve this in some demonstrable way. The specifics of the Game and AI internals will be expanded on in their own respective chapters. Here we will discuss how the theory is applied in this context.

Some of these points are:

- AI with dynamic metric-space.
- Design as a Game.
- How to achieve Closability.

AI with dynamic metric-space.

The problem for AI is that the set of solutions need to be evaluable, unambiguously, at some discrete point in time where the world is fixed. If when computing the difference of members in this space differs arbitrarily in the same world, then any AI system will fail to converge on any optimal, which would be impossible to determine. Even if by luck it was found, it could arbitrarily become sub-optimal later.

A dynamic metric-space is dynamic over discrete-time, but not dynamic within any singular instance of this time. This is a different notion of time from that of the game's simulation time steps. This time refers to state-changes of the Closed-world itself. Following the *Closable* definition, when criteria is provided or changed through an external actor, an Open-world or an existing Closed-world becomes a new, different, Closed-world. Each new world exists on different instances in a sequence of time. They are sequential because one is used to produce the next.

A dynamic metric-space is created by the Closable problem. Because new worlds are being created where criteria or evaluation of solutions can change, the fitness of a learned model in one world fails in the next.

Thus to tackle this issue, a learning system must continuously observe the world extracting training data from it, so as to implicitly capture any changes in how game-states are scored.

Design as Game

Disadvantages

Incomplete Representation of Problem

This is an issue with all forms of representation and is an innate problem with Design Problems in particular. Abstraction, and therefore loss of information, is inevitable. The question then becomes how much and to what degree is this information lost before the Game becomes useless as a Design tool.

However we accept information loss and transformation with all forms of representation. Some devices present issues of resolution that actually produce additional information through its documentation.

In the case-study example, we see a simulation modeling issues of urban and city conditions very crudely modelled. While humans and human decision making is not represented in a plausible way, the aggregate behaviour and rulesets create a plausible simulation of city in spite of this. The question is, does the Game represent the relevant issues for the purposes of producing a plausible design solution? If this question is properly addressed, the innate incompleteness of the representation will not compromise its utility as a design tool.

Advantages

Games as a recontextualization of Problem

We gain insights into a problem through alternative representations of it. All forms of representation require an understanding of the material to synthesize and translate the matter to a new medium. Even if the translation process itself provides no insight, seeing the problem represented in a new context itself can reveal hidden attributes that may have not existed in other forms.

Feature similarity

Games and Design both can be described with particular attributes that are held in common. Competition, evaluation, rules, constraints, actor(s), object(s) (targets of action that change state with actions) parameters, and conflicts all are elements found in both. It follows then that a Game could model an analogous Design problem through representing these features and mechanics in the Game. This affords us a strategy of translating a design problem to a game.

Design as “serious play”

Play in design is an existing methodology for eliciting design alternatives which comes from the notion of “serious play”(Roos et al. 1999) used in a variety of contexts. It is noted that removing the formality of the problem and recontextualizing or gamifying the problem allows for more lateral exploration.

Closability: Closing games with user-controlled weighting

Disadvantages:

Player introspection is flawed

Assuming an honest Player, even feedback from the Player here is innately flawed. Implicit biases will always exist, therefore introspection can never be objective.

However through exploring the effects of the applied metric-weighting by the user, they may observe results that move against their assumptions of what those values would lead to. This is part of solving any design problem, questioning designer's assumptions about the solution space.

Player intent is unknowable

Not assuming an honest Player, the Player may game the Game system. Manipulating the weights to change the outcome, but not using the weights as an honest mechanism to document intent or goals.

This can be advantageous as well. It may be the case that to maximize the learning behaviour of the AI system to achieve some ultimate goal might require frequent manipulation of the weighting system. Having the weights always reflect the end-intent might not benefit the AI in the long run. Local minima, and idiosyncrasies over the course of the learning maturation of the ANN's in a dynamic environment may require tuning of these weights to guide learning through the course of development to ultimately achieve a better result.

Player may value attributes not represented as metric in Game.

This is where the boundary exists for this system, and all systems like this that attempt to represent a Design problem. All environments of representation, Games, drawing, 3D-modelers, all have innate strengths and weaknesses in what information they provide and are able to manipulate. In a computable space, we can not extract meaningful metrics on attributes of the Problem that can not be represented in the same medium. In the case of values that can't be computed or represented computationally, these soft values might only be able to be approximated through some interpretive step, or decomposed into properties that can be computationally represented.

Not all Open Problems, or all parts of Open Problems may be Closable

It is reasonable to consider some metrics for evaluation, or all metrics, may not be computable values given some type of Problem. If we can represent the Problem, then the parts of the *Problem* fully translated to a computable space are what can be computationally explored, and *Closable*. Arbitrary factors are still *Closable* if they can be represented as a computable value. When a value can not be expressed as a computable value, this metric can not be represented and therefore can not be used in the evaluative cycle of the system.

This is also acceptable. All forms of representation suffer similar problems of being incomplete forms to explore the problem. The important part here is that, if a *Problem* can have any part represented computationally, then we can demonstrate a system that can explore a solution space in a *Closable* environment.

Advantages

Can extract subset of Open Problem, and Close it.

It is true not all Open problems can be completely Closed, as mentioned in the disadvantages of this approach. However, if we can even demonstrate a system that does represent a significant component of a larger Open Problem, and produce a Closed space for it to interact with User and AI, then we achieve some amount of automation in an otherwise complex and “wicked” environment, not typically conducive to such tools.

Is self-documenting

Because all of the solutions exist in this space of exploration and feedback, all in the same environment, continuously evaluated against Player-controlled metrics for such evaluation, we can see the design process in a contiguous metric space. The possibility to have these implementations begin to document a more complete mapping of the solution space could be quite illuminating for the designer. This is a practical performance tradeoff, as the computation and documentation of all of this is resource intensive, but the ability to fluidly navigate a tree of explored design solutions has great utility to the design process. For process-oriented design interfaces, tracing history too is advantageous.

With this, the results can be correlated with the user-weighting, and can provide the necessary insight to the designers as to if their decision making, values, and assumptions about the problem are valid if the results do not support their assumptions.

Can automate parts of the Design process

This is what we propose as the speculative step after what this system can demonstrate: a learning system in a *Closable-Game* space can find solutions and reliably improve these solutions. The work is far from doing this, reliably, even for the problem set provided. But even if on the average case with or without Player interactions, if the system can improve the score and build a plausible solution this is an important step. Because, even if imperfect, the solution can potentially cover a lot of solution-space and with some degree of significance over a brute-force random population of alternatives. This decreases the evaluative workload for the user over generative design methods where the designer is the sole determinant of the success of the algorithm and the designer is responsible for adjusting the algorithms parameters to produce more favorable results. In this case the designer still sets an agenda, but the evaluative work and parameter adjustment is automated by the learning system.

Case Study

The Game

Description

The game is modelled as a City Planning Game, similar in genre to SimCity and other themed city-builders. There is the notion of time and dynamic interaction and growth of elements in the city, people, and buildings. These systems can not be directly controlled by the player but influenced by a smaller subset of actions. The player's actions in this game are limited to Zoning. Zoning entails painting a tile with a zone property which allows for certain types of buildings to grow on that tile. This indirect control of the world-state only lets the user promote or inhibit certain conditions, but can not force growth of buildings on zones that are not suitable. This action intends to parallel that of the human actor, designer, in the real world. This level of manipulation of the game-state and metrics that are eventually scored creates a level of difficulty for the User and AI alike. Actions do not have direct or known consequences, and actions are context-dependent; these issues must be learned by both actors.

Intent

The intent of this Game is to provide a design tool, one that combines the process of exploration and evaluation into one interface, and additionally, has a learning system interacting with the same Game to assist or offer alternatives to achieve the design goals of the user.

The representation of the urban-planning problem is its own challenge. This simulation is by no means a full model of urban systems. However, simulations can never fully represent a complete reality. To use a simulation effectively, one must experiment and compare simulation against simulation, not against reality. This tool, like any other, is a best-guess as to what should happen based on a given design agenda with the provided data. Just as a drawing is an intent to achieve something, the reality will always differ.

The integration of an AI, specifically one that learns directly from actions in the Game, was an attempt to model how a naive user might approach the same problem. While the effects of both a human and AI user learning and playing in the same space are hard to objectively extract conclusions from, the hope was that the learning of both could be enhanced by this interaction.

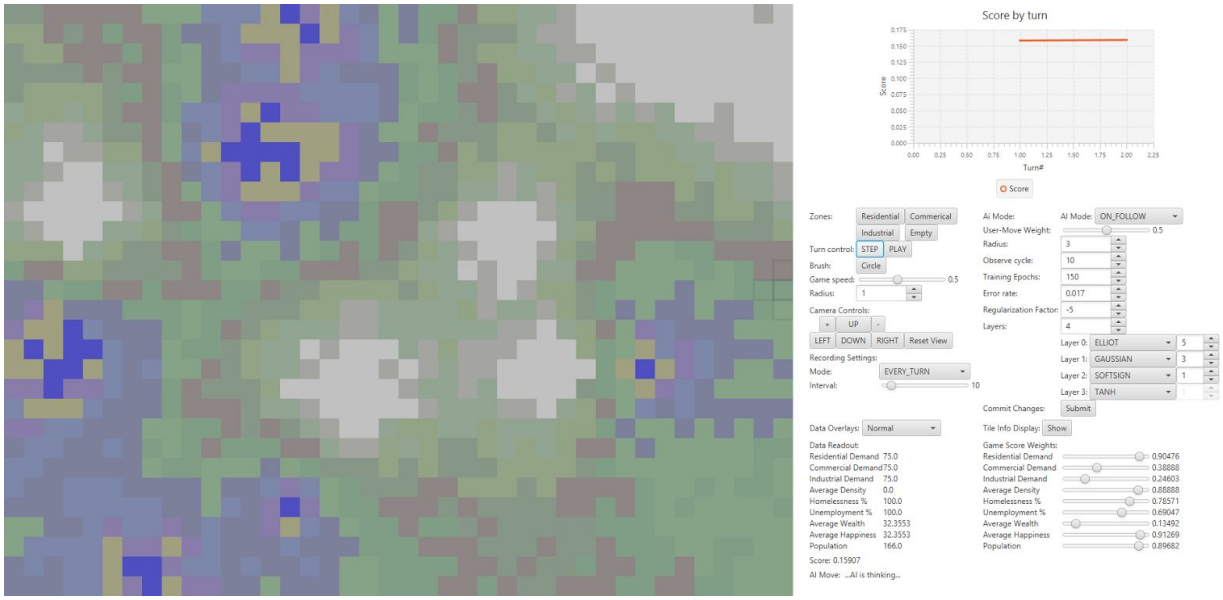
Use Case

The Designer would use this Game for a number of purposes, but primarily as a learning and experimental tool. This walk-through demonstrate the core functionality of the Game:

Walkthrough

The Designer loads a map of landcover data of a relevant area of inquiry, and initializes the Game. After the Game World is loaded, the Designer can adjust the weights of all of the City-Data parameters to indicate what the priorities are, or should be, for the Designer and the AI to work with and evaluate against. These can be changed at anytime. Then the Designer may choose to alter the configuration of the AI system's hyperparameters and model-specific parameters. At this point once the AI and score-weights are configured, the Designer can then choose to participate in the Game through a number of modes. The Designer can play the Game without AI intervention, with AI "hints", or with the AI making moves in the Game too. Along with each of these modes the Designer may configure the AI to interpret Player-moves as training cases with some weighting of success or importance.

The Game itself can then be played and turns take place either through time or through moves made in the World. Throughout the Game, each turn generates and records the City-data and scores as computed with the Designer's weighting taken into account. This way the Designer and AI can see how the score may rise and fall with moves or the consequence of moves they make.



The initial state with randomly generated world.

Object Model

World

The `World` object serves as the `Model` implementation. `World` is composed of a collection of `Tiles` and a `City` which describe the map and the internal attributes that are not seen. Actions initially pass through this object which further cascades the action, or other computation to its delegates. The `World` object controls the life-cycle of the `Tile` array and `City` object.

City

The `City` serves largely as a collection of `Persons`, governs the life-cycle of `Persons`, and packages relevant metrics of city properties which might be transmitted to the `View` or `AI` system. The `City` object can be queried for the `CityData` object which is used to produce the score when used with a `WeightVector` from the `View`.

Tile

`Tiles` model the land/terrain of the game-map, which has various properties determined by its type: material value, base landvalue. Some terrains are not able to be occupied by zones. `TileTypes` also have maximum allowable densities also related to this notion of terrain. The other value attributes also affect the growth of different `ZoneTypes` on those tiles.

Zone

`Zones` are the object that Player's can directly manipulate, which determine what occurs on a `Tile` in the `World`. `Zones` have 4 types: `Residential`, `Commercial`, `Industry`, and `Empty`(the default, and allows for erasing other zones). `Zones` contain the `Building` object, and control its lifecycle. `Zones` and their types compute a growth-value, which is determined each turn based on initial `Tile` conditions, and dynamically produced values, like pollution or augmented landvalues as the result of other favorable zones developing nearby.

Building

`Buildings` are entities that have a `Density`, and occupants. This object has two primary abstract subtypes: `PlaceOfWork` or `Home`. An occupant in a `PlaceOfWork` is employed by that building, whereas occupying a `Home` gives that person a residence. These are important properties to maximize (should the player value person-related metrics). `Buildings` modify the `Wealth` and `Happiness` of their occupants.

Person

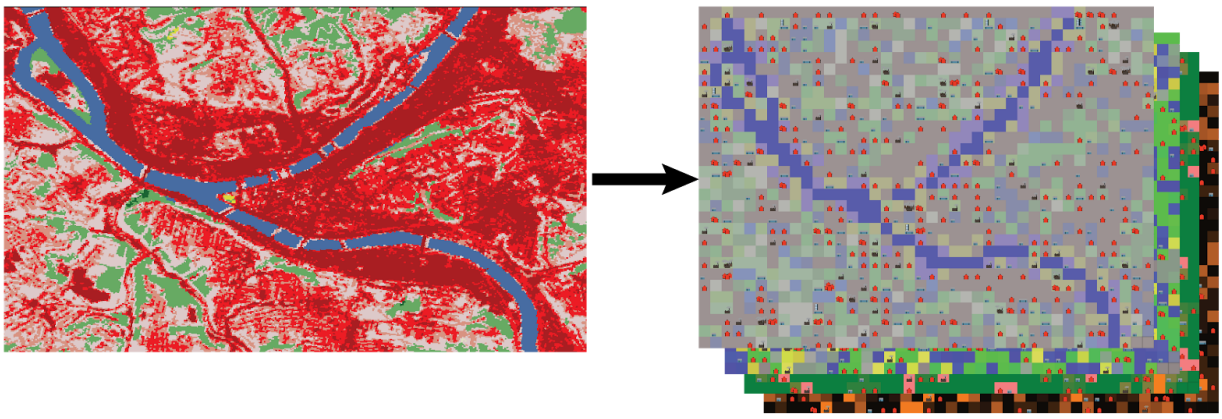
`Persons` are objects that are controlled by the `City`, but can be associated with at most two `Building` objects in the `City`. This is a tricky design issue because the `Person` is now coupled to two larger classes, and could be simultaneously referenced in three separate collections. Enforcing that the `City` is the primary object of a `Person` object's life-cycle, it follows the `City` is the ultimate determinate of when and how `Person` objects are removed, and propagate removals from other `Building` collections to ensure the `Person` is fully dereferenced.

`Persons` contain notions of `Happiness`, `Wealth`, and track its state and maintains pointers to `Buildings` - for employment and housing. `Persons` also have age which strictly increases with each turn.

Rules and Behaviours

World

The `World` initialization has a couple methods method for producing simple notions of terrain for the game-map. The first initialization method can interpret Geotiff images from NLCD 2011 LandCover classifications. Beginning with sampling the image at the same resolution as the world `Tile` array, the pixel at that location is correlated to the nearest matching classification to the NLCD 2011 LandCover legend. This then determines a fixed set of `TileTypes` that the `Tile` could be. Next, if the user chooses, the `Tiles` that are shown as having existing development can be initially zoned and grown to reflect the land-cover data. This allows users to load an existing city from land-cover data, either with the `City` or with just the terrain types. This process is subject to error, reading an image, incorrect format, path, or other user supplied parameters. So as a default behaviour the world can also be generated as gradient of terrains between randomly generate mountain peaks and waterbodies.



Land cover data on the left and the World representation in the Game

Tiles

After initialization, most behaviour is delegated, or the `Tile` largely serves as a collection of attributes and pointer to its constituent `Zone`. The type of the `Tile` is used when computing the growth-value of different `ZoneTypes` on such a `Tile`. The `TileType` contains

the intrinsic parameters of the Tile, like material value, base land value, and density parameters. The growth of Industrial zones are heavily affected by the Tile's material value, whereas Residential zones are affected by land values.

Additionally there are accumulated values on top of the intrinsic parameters. The Tile can become polluted, or accumulate or lose land value over time. Every Zone, and the Buildings constructed on them have effects that are applied to their Tile and neighboring Tiles. These attributes are proportional to the Density of the source Building, and diminish over the range of their application. If a large Factory moved in next door to Residential Buildings, the pollution will quickly diminish the land values of neighboring tiles, and the residents will move out.

Zones

Zones compute and maintain a growth-value. This growth-value determines the Density level, and therefore effective occupancy of the buildings on this tile. A Zone will not advance to the next stage until it can afford to grow based on this value. The growth value is calculated as a result of the Zonetype and the base and accumulated tile properties (like pollution and land values). Different Zonetypes' growth-value are affected differently by different tile properties. Residential suffers greatly from pollution on its Tile. Commercial is more tolerant but still suffers from slowed growth when exposed to pollution levels.

Buildings

Building behaviour is dictated by its hosting Zone which controls its lifecycle. The computation of the Density change as the result of a change to growth-value on the Zone is delegated to the Building of the Zone. Buildings do manipulate the attributes of its cccupants. The amount they increase these factors is based on distance between PlaceOfWork and Home, as well as the Density (productivity or growth-state) of the Building.

Citizens

Persons at every turn in the game have a few simple behaviours: they attempt to find a job vacancy if they are unemployed, they attempt to find a home if they are homeless. If they have a home, their happiness increases. If they are employed, their wealth increases. If these two factors are satisfied, they have a chance of spawning additional Persons to the City. When their age exceeds a given amount, they are removed from the city.

Software Architecture and Concurrency

Architecture

This game utilizes a Model-View-Controller architecture with an additional AI module each of which are on separate threads and are linked by observer patterns as a way to send messages to work-queues on each other's threads. With this pattern everything is communicating with or to the Controller, which will also pass messages, or execute other communications to other modules based on data it is notified to. The controller also contains the main game loop, turn-cycles which is messaged to the Model and AI modules.

The AI module is much different in implementation than the View, but the coupling to the controller is very similar.

Design issues

Responsibility for manipulating the Person object's properties given its 2-way association with Buildings and the City was important to ensure was handled carefully. Either object could propagate an event that causes the Person to be removed from itself, or both the City and Building. Because Zones can be routinely wiped out, making sure that Persons that once occupied a now null Building are not still indicated as occupying that building. Second, Persons may leave a City through death or discontent. This behaviour is controlled and evaluated by the City. Thus propagating the removal of a Person through Buildings they once occupied must be ensured. Otherwise the Buildings will remain occupied by Persons not under the control of the City. This is the danger and tradeoff of having this 2-way coupling and behaviours. However we can reason about this behaviour as it relates to the domain-model.

Architecture erosion was imperative to limit to fight coupling between modules that were not the Controller. As variety of functionality in various modules were developed, hooks to pass data or query another module for data began to bloat the interface. Implementing an observer design pattern for passing data between the modules early on helped avoid this issue. It was found to be advantageous to create custom struct-like objects to be generalized data packets that had subtypes that determined their purpose or ultimate destinations. This allows the Controller to pass messages by checking high-level types of the data and then routing it to the intended recipient.

This decoupling and event-based notification ensured that an action from the View would not execute a world update action on the View thread. This helped keep the GUI alive during model updates or AI training cycles. While the time-steps of each turn were discrete and locking enforced a procedure for each turn with the Model and AI, it still allowed for asynchronicity of AI and Player actions as they arrive on the Controller's notifications.

Concurrency model

Each module, Model, View, Controller, and AI are on their own threads. The GUI utilizes the JavaFX gui-framework, which requires its own special thread singleton that it controls for rendering purposes. However, if there are asynchronous instructions to be passed to it, there are Runnable tasks that can be added from other modules so that the JavaFX view thread can execute them when it schedules the task. The Controller is the coupling node between AI, Model, and View, and as such observes all three, and can pass messages to each of their queues to be processed on their own control threads.

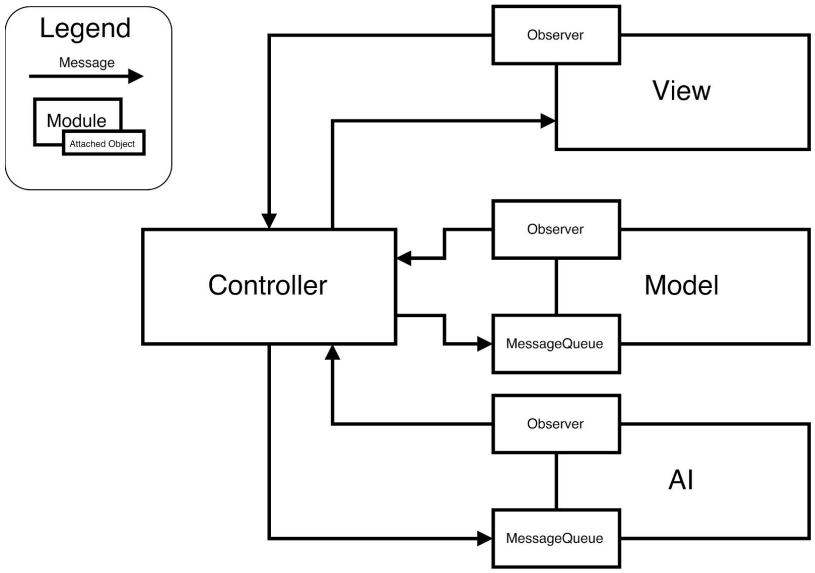
The Model and AI since they are parsing or modifying a large grid or array of Tile values, this becomes an embarrassingly parallel cellular automata problem to some degree. To take advantage of concurrent actions over an array of tiles, we subdivide the problem and fork off these tasks to works that can scale based on the size of the world. The Model executes updates each turn, on each tile, which update their constituent values. However this is not a perfect cellular automata problem as Tiles may have Buildings that emit effects to their neighboring Tiles as well. This creates race conditions of concurrent non-atomic modifications,

even to primitive values. Simple synchronization is enforced over these statements, which sacrifice some of the performance gains of the mass parallelization for correctness. The performance of this is still greater much than the sequential method and can handle larger world sizes without quadratically increasing in computation-time.

Meanwhile, the AI can be training on its own thread, and the View running undisturbed. This opens the possibility that the game and Player can act faster before the observed world-state according to the AI can be acted on. If a player action or AI action (in a non-paused Controller state) notifies the Controller's observer after the turn has passed, it is possible that the action can occur on a now old-state. Even Players' observation-act time is non-zero. This is accepted behaviour with all games with accelerated time-based turns, and a Paused turn-mode is provided to those that would like to act more deliberately per turn. This concurrency model, using work-queues to process these messages, ensures that the model will apply the actions of the AI and/or Player in sequence of when the messages were received.

AI interface and module design

The design of the interface of the AI module, as a high-level boundary between all of the neural network internals, but the game logic was difficult to decouple. This is particularly challenging because there needs to be a way that the Model information becomes an input vector, and an Action can be formulated from an output vector. The specific logic and interpretation of Model data to normalized floating point numbers was easy to separate, but the formulation of an Action requires knowledge of Zoning, and world positions. As a way to follow the initial model of the AI being "another player" it makes sense to have the AI emit Action objects (as `ViewData`) to the Controller, just like how the Player does via the GUI.



The characterization of the AI system as "another player" makes sense to characterize its output, but its input is notably different as well as its interaction with the Controller. Given alternative implementations, this could be furthered decoupled.

With the strategy to create multiple subnetworks, the opportunity to create an inheritance model for the basic network internals and configurations, and a pseudo-composite pattern for the networks. A complete composite pattern does not work in this case because the nature and behaviour of the sub-nets were not the same and had specific purposes. This specificity is an unfortunate tradeoff to how the AI system was internally designed. However trading off cleaner design of a more generically applicable implementation gave us increased performance as the networks were specifically oriented to the particular sub-problems. However wherever possible, common functionality or parameters were abstracted to increase reusability and maintainability which proved of great utility when implementing the user-configured set of hyperparameters for the neural networks. `AiConfigData` was the data-packed that was used to cascade hyperparameter changes to the initialization of all the constituent neural networks.

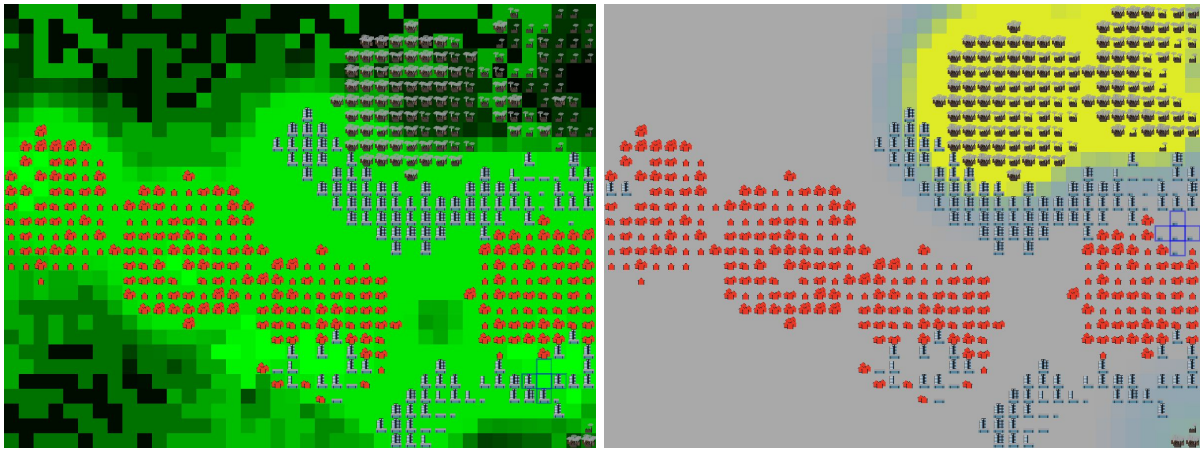
A number of interfaces characterize the behaviours of the various MLPs used in the AI module. These all cascade down through the highest level object that governs the life-cycle of all the other sub-networks, the Merge Network. With this design, we also have the opportunity to cascade hyperparameter changes relatively easily by isolating all of the neural network and training data objects to an abstract class for all of the networks regardless their specific purpose. All subnets inherit from the abstract network class, extending and implementing more specific interfaces, like the Mapper behaviours. All of which is described in the AI System chapter.

Data Layers

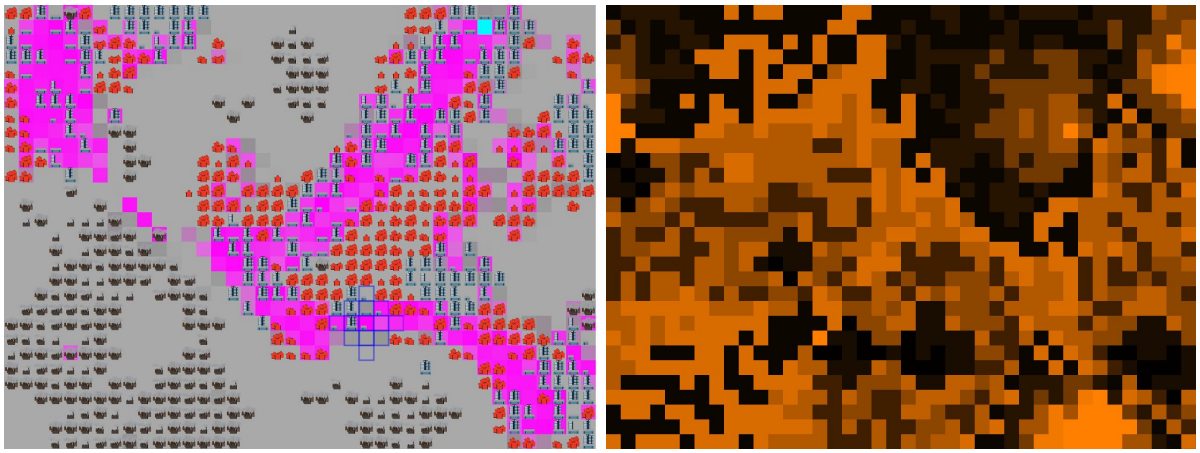
A lot of information is present in each Tile that is imperative for the player to be able to understand. To provide a more graphical and easily interpreted report of this information, data-layers were implemented.

These data-layers provide switchable view-modes to see the World and all of the Tiles colored according to some attribute including: density-level, land value, pollution, material value, and AI-Policy map.

The AI-Policy map is a view of the World as the AI-system “sees” it. This view runs a query through the AI module, given the current World, and the current Zone selection. The user can select different ZoneTypes in the control panel, and see where the AI system scores Tile locations higher to place such a ZoneType (at that particular time). The highest scoring tile is shown in cyan, and the rest form a gradient from pink to dark gray.



Land-value shown Left; Pollution shown Right.



Policy-map Left; Material-value Right.

AI Modes

We discussed the Game as its own standalone tool for solving design problems, just as any other software or media serve as ways to represent and solve the problem, and for the designer to learn, react and iterate through those tools. In an attempt to cast this as a multifunctional tool to adapt to the various needs and desires of the user, several modes are implemented to allow various interactions between the user and the AI, all at the discretion of the user.

On - The AI system is actively learning, and acting in the game space.

Assist - The AI system is actively learning, and will compute best-moves to make in each observe-act cycle. However the move is not committed to the Model, but only highlights the tiles in the suggested Zoning color, and reports to the interface the text-based specification of the move. The user is free to follow this suggestion, or ignore it.

Off - Of course the user may want to play the game without any interaction or performance reduction with an active AI system computing on each turn.

Follow - Follow mode is a modifier to the On and Assist modes where the AI will learn directly from User-moves (and its own). The difference here is that the User can score their own

moves arbitrarily and independently of the Game scoring logic and user-determined weighting. Essentially this provides a more direct way to influence the behaviour of the AI system.

AI system

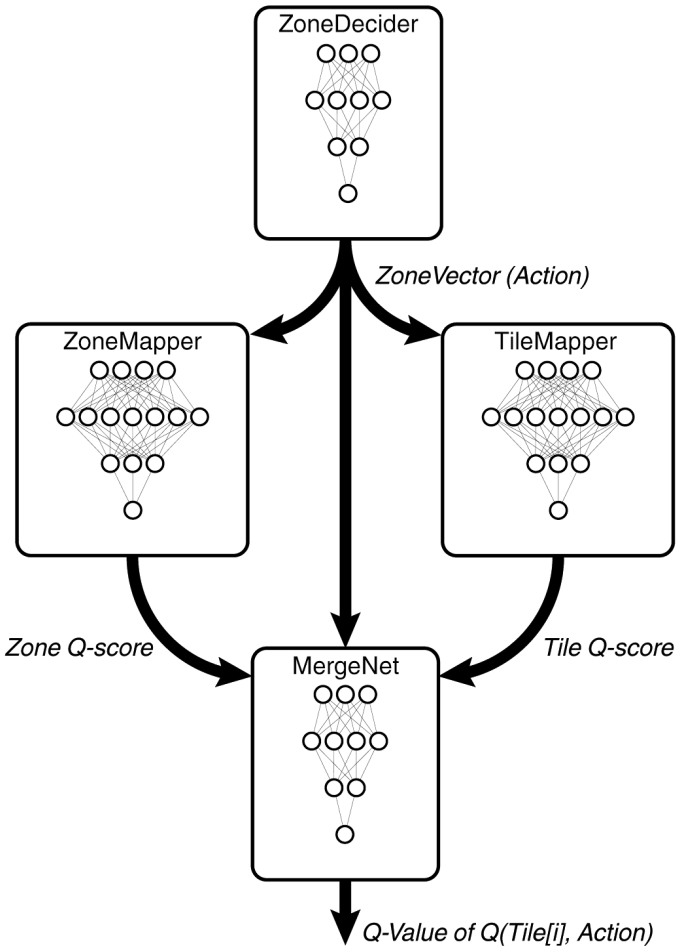
AI System Description

The AI System is composed of several Q-Learner MLP's each structured around a particular subproblem within the game logic. This is all wrapped inside a larger interface discussed in the Architecture. As each turn cycles, the state and recent action is passed to this interface where either the AI system uses it as a case, stores the world-state, or ignores it all and simply takes note of the passage of the turn.

Subnetworks

The purpose of subdividing the neural networks was to divide the magnitude of complexity of from a singular network, to provide a simpler representation of the world-state to each network, and have a focused purpose to the Q-Value's effect on the ultimate action to be taken. Subdividing the problem also radically reduces the search space and complexity, making the possibility of convergence plausible in the scope of a turn.

To guide the subdivision, we supposed a process a designer's decision making might take. First, one decides with what zone they should add to the city, based on city data metrics they observe on the feedback panel. Then they might select a location for the new zoning, looking at local tile data and current zoning conditions as they are. This is certainly not the only way to approach the problem, and specifically allocating networks to each of these in a particular order and purpose forces this to be the applied method. However a singular deep generic network would have the complexity of all possible actions to consider



and score, convergence and learning therefore suffer greatly in performance.

This subdivision is able to allow each Q-learner to attribute a Q-value to each set of possible actions for its scope. The `ZoneDecider` picks of 4 possible actions, which one is best, determining the `ZoneType` for the ultimate action. The Mapper-type MLPs walk the entire world array of Tiles, observing their attributes, and neighboring tile attributes, as the input layer, and scoring that Tile with a Q-value based on how it would affect the global score if the `ZoneType` selected by the `ZoneDecider` were to be placed in that location. The Merge Network takes the Mappers array of data, and further filters their output to come to a singular value for each Tile. The Tile with the highest value becomes the target of the action.

ZoneDecider

The `ZoneDecider` interprets a state vector produced by `CityData`. This is the same `CityData` that is weighted and scored by the user. Essentially the goal is to look at the state of the city (not any one, or collection of, Tiles) and determine the best `ZoneType` of the next action to add to the city, regardless of its placement.

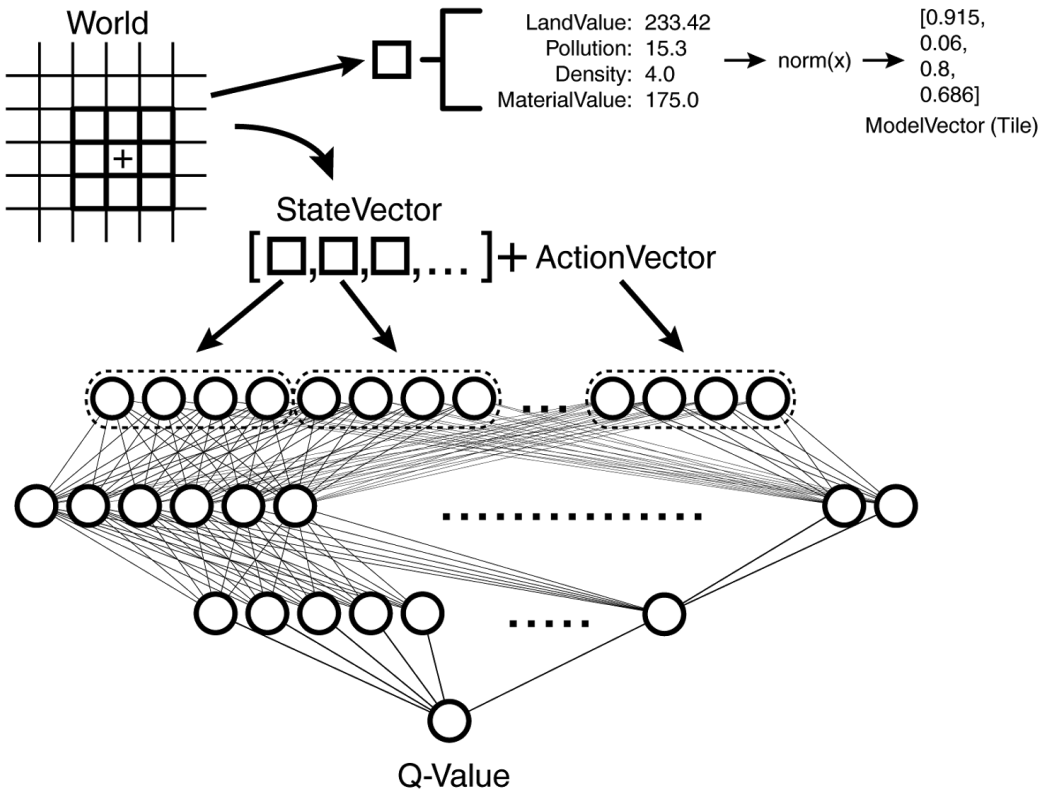
The action-space given a single state representation is only the length of the vector of possible `ZoneTypes` to choose from. In the game's current implementation, this is 4 (`Residential, Commercial, Industrial, Empty`). Q-values are produced for the given state with the 4 possible action choices, the highest output (greatest increase in score as a result of such an action) is chosen.

This choice is then propagated to other `Mapper` networks in the AI system as the assumed `ZoneType` of the action to "apply" to each tile in the world-state. The Mapper-type MLP's run the action against every tile input and the tile with the highest Q-score becomes the target of the action.

ZoneMapper

The `ZoneMapper` is the first of two Mapper-type neural nets. These are Q-learner MLP's that apply an action, to every sub-state (or tile) in the World. Each Tile is Q-scored provided the action of the `ZoneType` from `ZoneDecider`. The only thing that differs is the input layer vector, and how it is derived.

The `ZoneMapper` represents each tile as a Zone-Vector a vector of 4 binary values, where only one is "on" or 1, determined by the `ZoneType`. This is the same vector-representation of the action, as a zone-vector.



TileMapper

The `TileMapper` is the same Mapper type as the `ZoneMapper` with a different input vector, and derivation of input values.

A `Tile` is represented by a few key attributes. These attributes include: achieved `Density`, land value, material value, and pollution.

Merge network

This MLP merges values of each `Mapper` for each `Tile` value. This network does the least work and computation, but can drastically change placement strategies for the application of a move on the world-state. It too is a `Mapper`, and produces a map of Q-values. The input-layer is simply the Q-score of the tile from the `TileMapper`, the Q-score from the `ZoneMapper`, and the action-vector. From this the output Q-score may balance the influence of one `Mapper` type for another, depending on the `Zone` chosen by `ZoneDecider`, or may gradually change policy over the course of the game, shifting to favor current `Zone` distributions over `Tile`-attributes.

Hyperparameters & AI Parameters

There are a number of customizable hyperparameters that can be set upon initialization of any ANN. Notably the number of neurons on each layer, number of layers, activation

functions of those layers, target error rate, and max training cycles (in cases where error did not converge to meet error target).

However with this particular application of this AI system in this particular game environment, there are a few other considerations and parameters. These are context-specific parameters. While they are used to alter the actual network sizes, or how the AI interact with the game specifically, they are parameters that have meaning to neural networks outside of this context.

Game AI Parameters

Observe Radius

This would determine the manhattan-neighborhood of tiles to “observe” in the input layers of the Mapper subnetworks. This parameter would scale the size of the input layer of the Mapper networks, quadratically. The radius could provide opportunities for the Mapper networks to make more informed observations about local surroundings and provide that many more opportunities to learn higher level patterns. However the tradeoff is performance. Because it scales the input layer, the layers beneath need to scale to avoid significant information loss. The balance of this is arbitrary and could be machine-dependent, or designer’s time-dependent.

Observe Cycle

This parameter determines the number of turns to pass between Actions and Observations of score-change. This is critical in a sequential simulation environment. Because when an action is applied in this game environment, the resultant effects occur and continue to occur over a span of turns, the true reward-signal that could be derived may not show in a score-change for some time. This could also change over the course of the development of a City. Zoning an empty tile may quickly grow, but rezoning a polluted tile could take longer for the pollution to dissipate and land values to recover and then grow normally. Again, this is purely an issue of time. The computation-time of training on the observation is the same regardless how many turns one waits. However the observations that can be deduced from the observation is highly dependent on how long you wait to capture that change.

User-Move Weight

This relates specifically to the Follow modes. Should the aI system be thus configured, user’s moves automatically get pushed to the AI interface with that score (regardless of what the move is and what the score may actually be). This lets the user influence the training of the system to try to encourage a particular behaviour. However, because this is just another training case, highly scored or otherwise, it is not a user-mimicry feature. Though it is possible with enough reinforcement that behaviour may replicate patterns the user is intending to create.

Hyperparameters

Layers

Another simple parameter that trades off performance with the capacity for modeling more complex functions or behaviours. It is also shown that a single layer perceptron can also not model anything that is nonlinearly separable (Minsky & Papert 1988).

Neuron Densities

This parameter adjusts the number of neurons on a given layer as a factor of the Input Layer size. This is derived from general rules of thumb, which are highly debated. Generally speaking the layer following the input should increase in size, especially if it is known the problem is not linearly separable. There are some proposals about how to achieve this if the network is to precisely model some input data (Elisseeff, & Paugam-Moisy 1997). However this same approach acknowledges overfitting training data is not ideal for a network applied to cases outside of this set.

Activation Functions

These functions can radically determine the rate of learning and saturation of nodes. There are also a number of follies, possibilities and tradeoffs. Most important to note is the domain of the input and output. The steepness of the curve can determine how quickly nodes activate, but could also easily saturate if the curves are very steep. If curves are steep, the ability to discriminate between values over a range becomes more difficult. Additionally, the computational complexity of the derivative calculation can drastically affect performance on training or backpropagation.

Learning Rate

This parameter is automatically calculated and regulated by the Encog (Heaton 2015) library's special propagation algorithm. Though it is worth mentioning this hyperparameter as it is central to weight-adjustment calculations during propagation. Essentially it is a number between 0 and 1 (often very small) that multiplies the computed difference of the current weight against what the propagated error needs to fully correct. If the weights are shifted to be exactly what the error indicates for that round of propagation, then it may skip over a weight distribution that better fits the problem globally across the network. This naturally slows the process but forces training to explore the parameter space as to not skip over a more optimal distribution.

Error Rate

Error is a target parameter for the training of the networks to achieve in a fixed number of cycles. Because the training data, and learned cases themselves, might mutually conflict especially problems like the urban planning problem. This allows the training to halt if the training can not converge to a precise model of the data. This needs to be avoided to maintain generality and avoid overfitting. However too much error could produce an even more useless weight matrix in the network as the tuning of the network does not closely model anything. It is

also possible that a given network can not model the data provided and properly converge on it. However if that is the case, then the problem is in too few neurons or layers to represent the problem.

Regularization Factor

As a way to mitigate overfitting, a smoothing factor can be introduced to reduce the change in the weight adjustment over a training cycle. Essentially this parameter is subtracted from the learned-change in the weight (which is already modified by a learning rate). Because the learning-rate multiplies the difference in actual-weight and target-weight (according to error calculated through backpropagation). This applies a constant which reduces this magnitude evenly across all weight adjustments. This slows the learning process and minimization error, but mitigates overfitting.

Max Training Epochs

As a practical concern, there are a number of cases where the configuration of the neural network may not converge under the target error rate. This could be due to a poor MLP architecture (too sparse or shallow), bad activation function combinations, too tight of an error rate, etc. To provide an escape for the AI system, if training fails to converge after some number of cycles, it will cease. However, if this is too low a network that could converge is not given the chance and the network is too loosely fit.

Discussion/Challenges

The AI system needs to be able to interpret a representation of the World data, and compute some result to determine an Action to be taken. The engineering behind this was an exercise in understanding how to represent values and transform them between domains. However, this more importantly was something that fed back through the development of the Game Scoring algorithm and weighting system, as this was all coupled.

Given the generality and precedent for the application of ANNs, this option was appealing. However, there are practical concerns about the size and shape of a giant or deep NN architecture. As the problem is complex on its face, one could either ignore a great deal of data to simplify the input, or sparsely structure the neural network, which would not have the proper order to model the complexities of the problem or tackle finer-grained behaviors. It is possible that a giant, deep, monolithic neural network to compute the whole problem could: A) never converge; B) not be enough space-time to converge.

Reinforcement learning also presents some problems. Q-learners are “exploration insensitive” (Kaelbling et al. 1996), they take the optimal path first and always. Because this is a time-series set of actions on the same space, if a move was found to be optimal, and the city did not change significantly, the next turn the same move would prove to be just as optimal. To prevent stagnation, or action-cycles, a brief action history was created. This forces a naive heuristic for exploration. If the same Action is found to be optimal in its short-term memory, it is forced to explore an alternatives. There are still many considerations for how to tune the

behaviour of the system in very-late game scenarios. One could imagine a point at which abstention is the optimal move. However, there is no definitive way to determine if the Game is at a complete “end-state” as any slight imbalance in parameters may cause the city to decay slowly over time, and abstention will no longer be optimal, and there is no known period of turns to wait in order to detect this change.

This brings up another issue with respect to time. Because the Q-learner system requires computing a reward, but the score changes and trends over many turns as the result of some Action, it is ambiguous to determine what the true reward signal should be for the previous state-action pair that was applied to the world-state. To address this the Observe Cycle parameter was introduced to allow this parameter to be changed or adjusted to balance the tradeoff of delaying feedback for potentially more informative Q-value error calculation.

This quickly becomes another learning and optimization problem. There are a number of Projects that are exploring the automatization of optimizing, not only hyperparameters of a learning algorithm, but dynamically choosing from a set of machine learning algorithms to optimize results (Thorton et al. 2013). While the discussion of the tradeoffs of different strategies and the particular tuning of some hyperparameters is of some interest, it is well within the horizon that these issues will reach a point of resolution and automation in the near future.

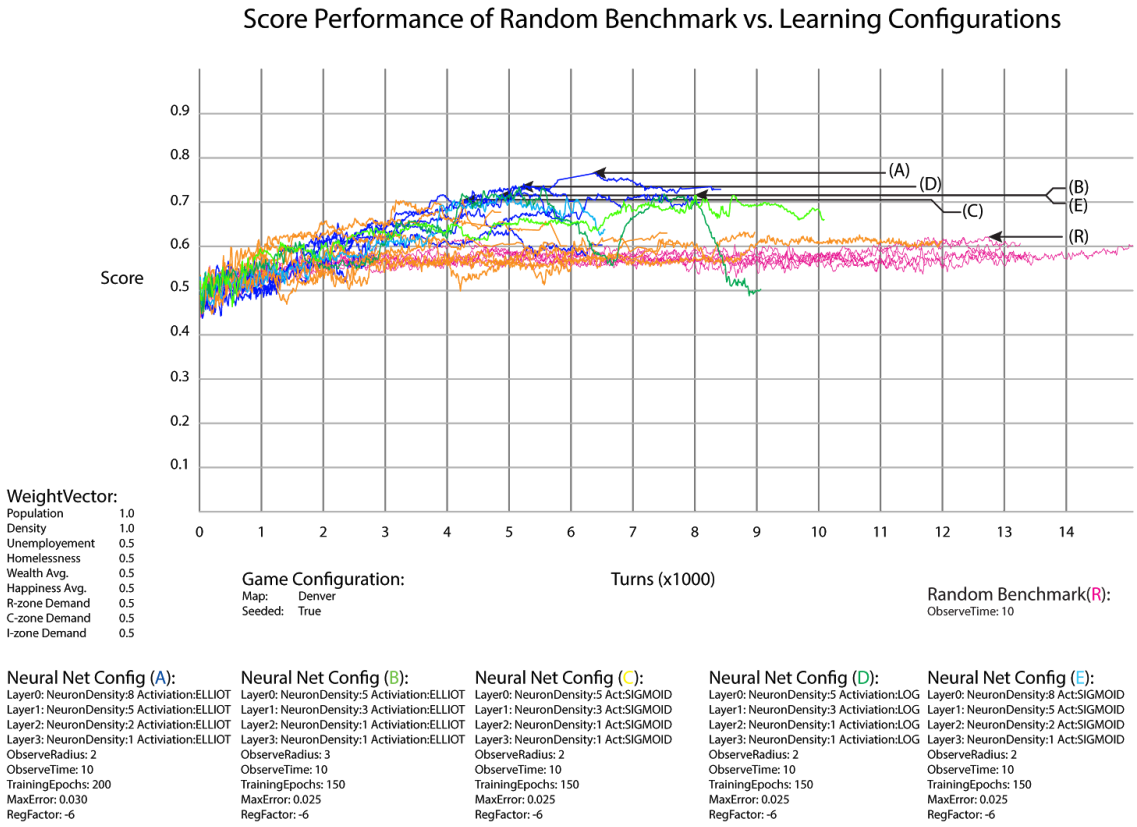
Thus providing an interface to allow these to change dynamically at runtime then sets the stage for these parameters to change through an automated system.

Conclusion

Findings

Score optimization of Learning AI vs Random Benchmark

It is noted that with most “early game” states, any move is usually going to produce an increase in score. Testing against random, turn for turn, will always favor random in a pure score-based observation in the early-game. However, once the network begins to “mature”, acquiring a health set of observations and adjusting weights to reliably act on learned patterns, the learning system begins to outperform the Random Benchmark. Random can quickly fill a map with zoned tiles, but the subsequent actions that occur can be self-defeating, erasing previous progress arbitrarily. Thus the score seems to hit a ceiling and oscillates. With the learning system, growth is slow and deliberate.



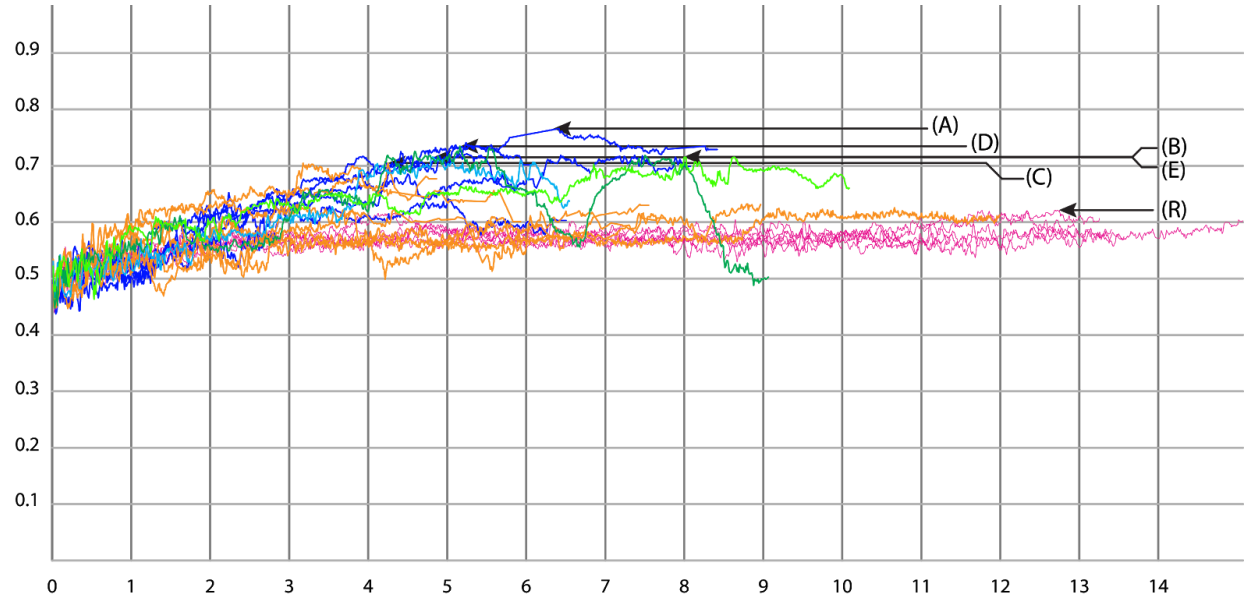
Learning rate and configurations

Configurations of the neural networks largely follows what is found in literature. If the network is too sparse, it is too simple and often can not model complex relationships. This usually results in an inability to converge under error tolerances. Activation functions too can greatly affect the learning rate if properly configured. Users do have the option of mixing types, and sometimes to useful effect. However some can drastically shift the domain of the output, causing other activation functions to “saturate”. This also provides a way to slightly tune performance as some activation functions may have a complex derivative for computing error through backpropagation.

Besides the layers and neuron densities, the target error rate and max training epochs help calibrate performance and overfitting issues. By allowing a loose error rate, and/or higher regularization factor, we can reduce overfitting and excessive training epochs trying to fit under tight error tolerances.

Most importantly, if a learning system is to act in a space that has changing evaluations over a solution-space, it follows that the learning system itself might need to be restructured to better be able to fit the new metric space that it is exploring.

In practice, there are more configurations (with the number of options and parameters provided) that resulted in terrible performance, than it did in novel idiosyncratic arrangements that out performed conventional strategies. The ability to rapidly customize configurations of these parameters largely leads to useless networks that often can not converge under typical error rates. Most reliable success was found using conventional selections of network density and depth, along with typical sigmoidal activation functions. However, it is impossible to rule out that some problem might be better modelled by some other arrangement of these parameters that have not yet been discovered. Providing the flexibility to adjust (and in the future optimize) these parameters is the only way to achieve that.



Qualitative observations

The ambition of course with the software artifact was to see if there was some observable high-level behaviour that was able to take into account intrinsic game rules and behaviours and how they affected the play environment. Just as a player learns to play SimCity, by trying something to achieve a goal, so to the AI system (following a reinforcement model) attempts the same thing. Specific advantages of certain Tiles or pollution tolerance of Commercial versus Residential are all programmed to create a slight asymmetrical advantage that could be observed in a City that takes advantage of these properties. However the complexity of these interactions make the observation of these behaviours particularly difficult to distinguish vs. a random or lucky guess.

The Policy-map Data layer is an attempt to provide an insight into how the AI system sees the world. The Policy map lets the user see for a given Zone, how the AI system scores each tile, and for a given AI move, how it scored the map. Using this as a way to more objectively make observations about the learning progress made on a high level is an important tool, and does shed some evidence on this progress.

Early game moves tend to value TileMapper outputs more heavily. Industry Zoning is applied to Tiles with high material values, and Residential on Tiles with high land values. Also, clustering of types becomes common. However the relationships between clusters is something that takes longer to "learn" as the negative effects of Industry near Residential occur over a number of turns.

Contribution

Feasibility and Role of AI in Design Environments

Tools that automate sub-processes are ubiquitous. The implementation of some tools often require a lot of manual tuning on the part of the Designer. For example, a Genetic-Algorithm solver still needs a way to classify the outputs and score them to determine fitness. This follows the generative design methodology where the Designer is the evaluation-function and backpropagation (adjusts the parameters of the algorithm used). The question this thesis asks, can machine learning push this boundary?

The answer arrived at is: yes, but not without its own challenges. We see that learning algorithms are designed specifically to be universal approximators by adjusting to fit a certain model, and not through prescriptive procedure or search with heuristics/metaheuristics, but by adjusting internal parameters to slowly approximate and fit observable data. However the problem still exists: how is the output evaluated? In a Closed-world this is solved, but in design this is an Open question that can only answered, and changed, by the Designer.

This sets up the same Closing Problem we see with the formulation of the Closable Game to allow the Designer to provide the necessary input to classify outputs. If we then provide the interface to allow the Designer to seamlessly manipulate this classification, weighting of scores, then the cycle of output-evaluate-adjust-rerun is closed into one singular application. This allows for the rapid elaboration of not only design alternatives, but design policies of scoring those design alternatives.

The Framework

Despite the software implemented only being a prototype of a design game with an AI module, the demonstrated value from this comes in the form of a framework that handles the coupling between a game system and an AI. Framework design is particularly problematic as every decision on what is and is not provided in the interface can drastically restrict functionality or the space of possible implementations that can utilize the framework. How much responsibility belongs to the framework and how much remains client-side while still trying to ensure that both the game and AI have necessary data is a challenge.

What we gain from the framework however is the functionality of being able to reliably couple a learning system to a game, regardless the implementation of either, and the interface to allow dynamic alteration of scoring algorithms at runtime. This means open games could be

implemented modelling design problems, another client may deliver a learning system, and the play of the game can be scored and altered by the player to suit their needs and to further control the behaviour of the AI in the game.

Essentially the framework solves the *closability* problem for the AI system for any provided *Open Game* on the game client. Thus any Design Game could be implemented by domain experts that suit their needs and adapt to their process. The same holds for the AI system client-side implementations. The combinations and uses within any discipline explode from this point. The utility of such a framework could be the next step for practical machine learning applications in design practice.

Reflections

We have established the feasibility of how, and through what qualities, AI can be employed in a design environment. We did this by first classifying what about design problems in particular make them unappetizing to apply AI solutions to. Literature reveals that design problems are ill-defined, and have countless solutions, all of which may only partially satisfy the problem's criteria, and evaluating the fitness of a solution is ambiguous or otherwise undefined (Rittell et al. 1973). Given this problem, producing an environment that could represent the problem became an exploration in gamification, through which we borrowed heavily from precedent in existing game typologies (Wright & Bremmer 1989). Following contemporary research on machine learning as applied to games (Silver et al. 2016) we found relevant paradigms and strategies in machine learning techniques that fit well with our game model and learning objectives. We were able to tackle the "wickedness" of the design problem by providing a novel addition to the sandbox-game genre: a method for explicit self-scoring. This allows the uncertainty about game-states to be known and scorable, by the policies dictated by the designer. Which in turn, the AI system could optimize within, learning from direct observation of the player's and its own actions in the game.

To some extent designers do this with their computational tools or other ML strategies, but through other means of explicitly setting the agenda and then running the system with the evaluative criteria set at compile-time. Our solution allows for the dynamic alteration of the evaluative step of the output from this learning system, at runtime, anytime. Naturally the tradeoff is the ability for the learning system to fit the changing model of the world, and how it is evaluated. The advantage is a seamless interaction of designer and AI, and the ability to rapidly iterate not only over solutions but over policies that propagate those solutions as being more favorable through new policies.

The demonstration through the software artifact has the unfortunate effect of creating a focalizer through which to see the future application of these ideas. However given the nature of the claims in the theoretical discussion, the evidence is in playing the game. In terms of the actual performance of the AI system, and solutions through autonomous or cooperative play are difficult to evaluate quantitatively. We establish it can improve on pure brute-force random, but this is a very minimal baseline. Future work may yet prove effective against more rigorous benchmarks and alternative algorithms.

Appendix

Terminology

AI - Artificial Intelligence

Artificial Intelligence(AI) refers to classical and learning techniques (defined under Machine Learning) following the definition: “The branch of computer science that is concerned with the automation of intelligent behavior" (Luger and Stubblefield, 1993). The usage of AI in this document is exclusively about weak or narrow AI.

ML - Machine Learning

Let Machine Learning (ML) be defined as a system of algorithms that inductively self-alter parameters to maximize performance over some task given some set of experiences. (Mitchell, 2006). Machine Learning is a strict subset of Artificial Intelligence (AI).

Metric Space

Metric Space is a set where all members’s distances are defined. In this context we will talk about this as computing differences in game-states, scores, actions, or any other value. AI, or any other form of computation, requires a defined metric space (where the distance function is often a Mahalanobis distance) to compute and optimize for some evaluable criteria, and can only do so if that space can be computable values in some contiguous range.

Online Learning

Online learning refers to how training data is processed relative to a learning system or algorithm. Online techniques are generally memory-less, but there are a number of techniques.

Incremental Learning

“Incremental learning is a machine learning paradigm where the learning process takes place whenever new example(s) emerge and adjusts what has been learned according to the new example(s). The most prominent difference of incremental learning from traditional machine learning is that it does not assume the availability of a sufficient training set before the learning process, but the training examples appear over time.”(Geng & Smith-Miles 2015)

ANN - Artificial Neural Networks

Artificial Neural Networks are composed of layers filled with neurons. Each neuron is connected to each of the neurons of the next layer. Each neural connection is also Weighted. This Weight is the parameter that is changed through training/”learning”. Each neuron takes each input signal, modified by the weight, sums them together, and computes the output using an Activation Function. This function is simply a means to push incoming signals up or down by

some factor to determine how much incoming signal will cause this neuron to “activate” and pass the output signal on (and by what intensity).

MLP - Multilayer Perceptron

A MLP is the most common type of ANN architecture and follows the typical rules of layers being fully connected to the next layer in the network, but not internally, or cyclically like in recurrent networks, or partially connected like convolutional networks. A MLP is said to be a “Universal function approximator” as it takes any number of parameters (input layer) and can be trained to produce the outputs. The training is most commonly a form of Back-Propagation, where the ideal outcome is differenced against the actual output, this difference is the error and this error propagates back through the network, adjusted the weights of each connection by some factor to reduce the error. In typical Supervised Learning back propagation this is performed with a large dataset, and the network fits the input-output pairs within some degree of error. Eliminating error is problematic because this will lead to overfitting. If the training data does not exemplify all possible real-world cases (which for our problem, is certainly true) then the MLP will misclassify the real data as it is too strictly formed to the training data. This error in a way becomes a form of tolerance.

Learning Paradigms, Reinforcement Learning

There are different Learning Paradigms in Machine Learning. Supervised Learning requires ideal pairings of input and target output data. These are assumed-true or canonical cases. Unsupervised Learning acts very differently as these algorithms look at data in aggregate and produce higher level observations on the data, without any instruction of the significance or parameters of separation. Reinforcement Learning is between these, and refers generally to a learning system that is reducing error or maximizing reward. (Sutton & Barto 2012) With Reinforcement Learning, a goal is known but the cases are unknown until the system can play.

Q-Learner

Q-Learning is a Reinforcement Learning strategy that takes an input State and Action pair, and produces a guess (Q score) as to what the outcome will be (the output signal, score, or change in score, etc.). The Q-function could be a table lookup, like in a simple path-finder problem, where the distance to the goal is known, or computable immediately. In cases where there does not exist a function or table of all possible state-action pairs, the Q-function can be modeled using a ANN, specifically a Multi-layer Perceptron (MLP).

CWA - Closed-world assumption

Closed-world assumption (CWA) is defined as a model where anything that is not known to be true or can be proven true from existing axioms is assumed to be false. This sets the notion that if something is unknown it is false.

OWA - Open-world assumption

Open-world assumption(OWA) describes the case where if something not known to be true or provably true, it is unknown, not false. Thus unknown remains unknown until proven otherwise unlike CWA.

Bibliography

Bottou, Léon. "Online learning and stochastic approximations." *On-line learning in neural networks* 17.9 (1998): 142.

Campbell, Murray, A. Joseph Hoane, and Feng-hsiung Hsu. "Deep blue." *Artificial intelligence* 134.1-2 (2002): 57-83.

Du, Ke-Lin, and M. N. S. Swamy. *Neural networks and statistical learning*. Springer Science & Business Media, 2013.

Elisseeff, André, and Hélène Paugam-Moisy. "Size of multilayer networks for exact learning: analytic approach." *Advances in Neural Information Processing Systems*. 1997.

Feurer, Matthias, et al. "Efficient and robust automated machine learning." *Advances in Neural Information Processing Systems*. 2015.

Geng, Xin, and Kate Smith-Miles. "Incremental learning." *Encyclopedia of biometrics* (2015): 912-917.

Giraud-Carrier, Christophe. "A note on the utility of incremental learning." *Ai Communications* 13.4 (2000): 215-223.

Grand, Stephen, and Dave Cliff. "Creatures: Entertainment software agents with artificial life." *Autonomous Agents and Multi-Agent Systems* 1.1 (1998): 39-57.

Heaton, Jeff. “Encog: Library of Interchangeable Machine Learning Models for Java and C#.” *Journal of Machine Learning Research* 16 (2015): 1243-47.

Hornik, Kurt. "Approximation capabilities of multilayer feedforward networks." *Neural networks* 4.2 (1991): 251-257.

Kaelbling, Leslie Pack, Michael L. Littman, and Andrew W. Moore. "Reinforcement learning: A survey." *Journal of artificial intelligence research* 4 (1996): 237-285.

Krishnamurti, R. "The MOLE picture book: on a logic for design." *Design Computing* 1.3 (1986): 171-178.

Luger, George F, and William A Stubblefield. *Artificial Intelligence*. Redwood City, CA: The Benjamin/Cummings Publishing Company, 1993. Print.

Minsky, Marvin, and Seymour Papert. *Perceptrons: An Introduction to Computational*. MIT press, 1988.

Mitchell, Tom Michael. *The discipline of machine learning*. Vol. 9. Carnegie Mellon University, School of Computer Science, Machine Learning Department, 2006.

Rittel, Horst WJ, and Melvin M. Webber. "Dilemmas in a general theory of planning." *Policy sciences* 4.2 (1973): 155-169.

Roos, Johan, and Bart Victor. "Towards a new model of strategy-making as serious play." *European Management Journal* 17.4 (1999): 348-355.

Shenouda, Emad AM Andrews. "A quantitative comparison of different MLP activation functions in classification." *International Symposium on Neural Networks*. Springer Berlin Heidelberg, 2006.

Silver, David, et al. "Mastering the game of Go with deep neural networks and tree search." *Nature* 529.7587 (2016): 484-489.

Simon, Herbert A. "Rational choice and the structure of the environment." *Psychological review* 63.2 (1956): 129.

Simon, Herbert A. "Theories of bounded rationality." *Decision and organization* 1.1 (1972): 161-176.

Simon, Herbert Alexander. "Administrative behavior; a study of decision-making processes in administrative organization-3." (1976).

Sutton, Richard S., and Andrew G. Barto. *Reinforcement learning: An introduction*. Vol. 1. No. 1. Cambridge: MIT press, 1998.

Thornton, Chris, et al. "Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms." *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2013.

Watkins, Christopher JCH, and Peter Dayan. "Q-learning." *Machine learning* 8.3-4 (1992): 279-292.

Wright, Will, and Michael Bremer. *SimCity*. Orinda, CA: Maxis Software, 1989. Computer software.