Presented at IEEE Conference on Games, August 2021

Interpretable Utility-based Models Applied to the FightingICE Platform

Tianyu Chen, Florian Richoux, Javier M Torres, Katsumi Inoue

https://hal.science/hal-03276322

Introduction

- **Utility functions** are one of the classic ways to model NPC behaviour

- But they can be a pain to make and tune

- This paper lays out a way to learn **utility functions** for NPC from data collected from human-played games using an **Interpretable Convolution Network**

- These **utility functions** will mimic the behaviour displayed in human-played games

Some Definitions

Utility Functions

- For each possible NPC action there is one function that can take some portion of the game state and returns a value
- The NPC will take the action which returns the highest value
- Complex behaviour usually cannot be summed up by one or two independent actions but is instead expressed by complex action interlocks and relations
- Making designing utility functions by hand very tedious

Interpretable Convolution Network (ICN)

- Neural network with a fixed architecture, with a specific purpose for each layer
- Nodes apply a elementary operation from a set of elementary operations defined for the layer
- The weight between nodes is binary
- These traits make the network interpretable, this will be discussed with an example later

So why do they use a ICN?

We could use a normal neural network to make utility functions but we wouldn’t be able to extract them from the network. To use a utility functions we would have to run the neural network in a feed forward manner.

But because ICN is **interpretable** we can extract the function through the network’s meaning. Not only is it more performant, we can modify and tweak the function manually.

Method

Like most machine learning the method involves:

1. Design the network, for some problem
2. Train it, using data on the problem solution

In this paper, the case of generating an NPC enemy AI in FightingICE is used as a proof of method.

Design

Utility Function Decomposition

$$u(\vec{x}) = coef \times combine_{i=1}^{k}\Big(transform_i(\vec{x})\Big) \quad (1)$$

where $\vec{x}$ is a game state, $coef$ is a real value, $combine_{i=1}^{k}$ is a combination of k elements (for instance, the sum $\sum_{i=1}^{k}$) and $transform_i$ is a transformation operation extracting and transforming specific data from a given game state.

$$u_1(\vec{x}) = 0.5\Big(exp\_diff\_HP(\vec{x}) + log\_diff\_speed(\vec{x})\Big)$$

$$u_2(\vec{x}) = 2.Mean\Big(can\_hit(\vec{x}), logistic\_distance(\vec{x})\Big)$$

$$\left(x_1 - \frac{1}{1+e^{-(x_1-0.5)}} + x_2 + \frac{e^{x_2}}{2} - x_3 + \frac{e^{x_3}}{2} - \frac{1}{1+e^{-(x_3-0.5)}} + \frac{1-2*x_5}{2}\right)$$

Figure 3: Most frequently learned utility function for the move forward action in the aggressive behavior

Input	*Meaning*
x_1	Hit points diff. between the player and the opponent
x_2	Distance between the player and the opponent
x_3	Speed difference between the player and the opponent
x_4	Is the opponent attacking?
x_5	Is the opponent in range of attack?
x_6	Is the player at a border of the stage?

ICN Architecture

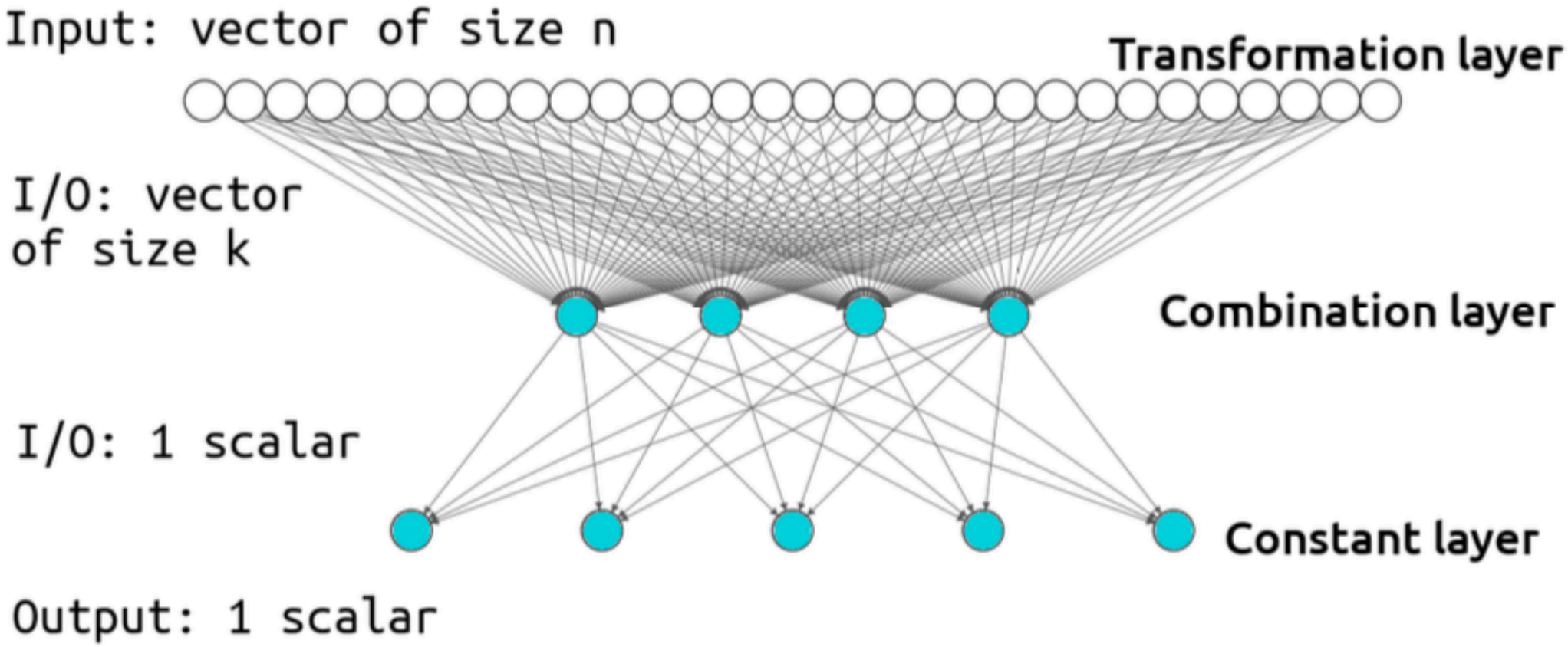

Figure 2: Our ICN model to represent utility functions in FightingICE.

Layer by Layer

- **Transformation layer** has neurons (in this case 30) which take the game state vector and performs some elementary option on it
- This can be linear, exponential, logarithmic or logistic
- For example the exponential of the difference in hp
- **Combination layer** has 4 neurons that will aggregate the results of the transformation nodes connected to it via the nodes function
- In this case the sum, the mean, the maximum and the minimum
- **Constant layer** has 5 neurons that multiply the results of the combination layer by some real coefficient
- In this case: 0.25, 0.5, 1, 2, 5

Properties

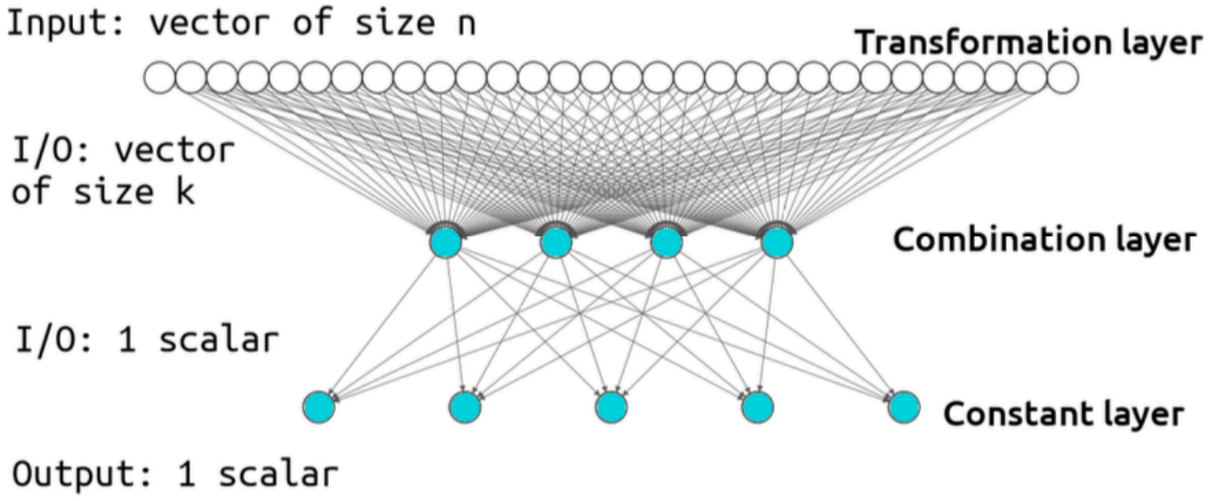

Figure 2: Our ICN model to represent utility functions in FightingICE.

- Connections between nodes are binary, each operation is either chosen or not chosen
- The blue nodes are mutually exclusive
- These two properties make the network interpretable
- With real-valued weights as for regular neural networks, we would end up with utility functions expressed by a combination of all (non-mutually exclusive) elementary operations with different real coefficients
- This setup is very extendable for different purposes

Training

Input Data

- To prove that this network can mimic target behaviour they produce 3 datasets of human play: aggressive, defensive and hybrid

- These included about 20 games for a total of only 30 minutes of play

We defined our three target behaviors as follows:

- **Aggressive**: punch the opponent if it is in reach, otherwise move forward the opponent.
- **Defensive**: block if the opponent is attacking us, otherwise try to move backward to flee.
- **Hybrid**: apply the aggressive behavior first. If our hit points is at least 50 points below our opponent's hit points, then switch to the defensive behavior until the end of the game.

Genetic Algorithm

- Supervised learning from the dataset of human play
- It is labelled as FightingICE will give the game state and the action taken
- The elementary operations aren’t all differentiable so we have to use the genetic algorithm
- Since each function is dependant on each other, loss is defined by the action rank

Evaluation of Method

- Tested through a 100 5-fold cross validation for each dataset

- One model took about 7 minutes to learn on a Intel MacBook

- Random guess accuracy would be 25%

Table I: Number of action entries of our different datasets

	Aggressive	Defensive	Hybrid
move forward	3,456	–	2,302
punch	4,907	–	3,256
move backward	–	4,586	755
block	–	5,053	2,725
Total	8,363	9,639	9,011

Table II: Statistics on the accuracy of our 100 models for each dataset

	Aggressive	Defensive	Hybrid
Best model	79.7%	85.2%	69.3%
Worst model	76.7%	82.4%	57.1%
Median	78.6%	83.7%	64.5%
Mean	78.5%	83.7%	65.1%
Standard deviation	0.004	0.005	0.027

Limitations

- Dataset was noisy as since the same game situation, or very similar ones, can be labeled with different actions

- Relatively simple behaviour

Improvements and Applications

- Data augmentation or taking a larger scope of data composing game states but then penalising models extracting too many data from these game states: forcing learned utility function to only consider the most significant values leading to utility functions more focused to essential data could help with the dataset noise

- Splitting transformation layer into two layers for data extraction and then transforming would make the ICN clearer

- This method could be used in combination with other types of traditional decision modelling to simulate even more complex behaviour in a controlled and customised manner

Novelty of Method

- Using ICN for making function is not new, ICNs were originally proposed to model and learn error function in Constraint Programming

- They only need to learn one ICN at a time, but learning utility functions require learning several functions with the same inputs which is novel

- Learning utility functions via reinforcement learning is not new, it was used to learn cooperative NPC AI

- But being interpretable is novel as in previous research the network was a black box and would have to be retrained if the game changed

- The main originality of the method is to produce interpretable utility functions that can be easily understood, modified at will and implemented within the game logic

Conclusion

Thank you for listening!