Fast weight programmers implementation

Evan M Drake

22/10/2023

Contents

1 Introduction

Meta-learning is a powerful tool in the presents of high training costs. Techniques like supervised-learning algorithms have training cost as a function of training data size. This is a consequence the sequential design pattern common in building these supervised-learning models [2], as seen below.

- Build a model
- Train the model
- Use the model

Consequently there are two ways to optimize this training cost problem: shorten the training phase, build a faster model. Meta-learning algorithms take advantage of the first method by utilizing few-shot learning [1], or getting rid of the training phase completely [3]. One such learning model is Schmidhuber's *Self referential weighted matrix*(SRWM), it does not require a training phase [1]. This machine can learn and adapt during experimentation. Here I will utilize an SRWM to play the game of American checkers[2]. The goal is to prove proposition 1 and all consequential propositions.

2 Objective

Is it worth adding a diagram to this section? Or is the point clear in simple text?

Proposition 1 *If a SRWM and a weighted matrix are to play the same game of checkers with no training phase the SRWM will play more efficiently.*

In proving proposition 1 we will have proven that for the game of American checkers a SRWM implemented as we do can play more efficiently than a standard weighted decision matrix. We will prove this by constructing both a weighted decision matrix and an SRWM and allowing them to play the same game of checkers. The outcome will be a proof by construction with the assumption that all outlier games will be ignored.
In particular we aim to set ground work for the proof that an SRWM can be used as a drop in replacment for any weighted decision matrix.

[1] SRWMs do not require a training phase to begin learning, but can still benefit from them

[2] American checkers or *English draughts* is explained in section 4

3 Literature Review

3.1 Adaptation Based Meta-learning

In adaptation based algorithms the underlying data structure (usually a network or set of networks) is never changed on a large scale but rather incrementally augmented until it fits the current task. Techniques using this approach tend to experience similar problems such as catastrophic memory loss[3], and change scale[4].

Schmidhuber [3] introduced the idea of a self modifying network that utilizes introspection to discover optimization probabilities. The idea is that given some weighted matrix $M(V, E)$[5], we can relate the change in weights for all E temporally. At a high level, this temporal analysis will expose a sort of gradient decent optimization problem [1]. The introspective phase is conducted with the instruction of special input units that outline a manner in which to analyze the network and output units that outline the modification of the matrix once analysis is complete. Input and output units are designed to describe the matrix by way of time-varying activations [6]. These activations serve to represent the connection weights and the weight modifications respectively for the special input and output units. **Irie et al.** [1] shows a more fine grained approach to Schmidhubers simple self modification, applying it to modern neural network training techniques and providing different implementation strategies. Irie et al. show the possibility that in certain situations self referential weighted matrices can not only adapt their learning methodologies but can also adapt the way they adapt their learning methodologies (meta-meta-learning). Their self-referential weighted matrix can be described as such:

$$y_t, k_t, q_t, \beta_t = \mathbf{W}_{t-1}\phi(x_t) \tag{1}$$

$$\hat{v}_t = \mathbf{W}_{t-1}\phi(k_t) \tag{2}$$

$$v_t = \mathbf{W}_{t-1}\phi(q_t) \tag{3}$$

$$\mathbf{W}_t = \mathbf{W}_{t-1} + \sigma(\beta_t)(v_t - \hat{v}_t) \otimes \phi(k_t) \tag{4}$$

[3]Memory loss is when the learning algorithm is allowed to modify to much information and a net information loss occurs.

[4]I define change scale as the problem of finding modification granularity

[5]Note that it is easier to think of M as a network

[6]Provided a constant stream of problems the paths taken can vary on each iteration

"where the $\otimes$ denotes the outer product, σ is the sigmoid function, and ϕ is an element-wise activation function whose output elements are all positive and sum up to one (e.g. softmax)" [1].

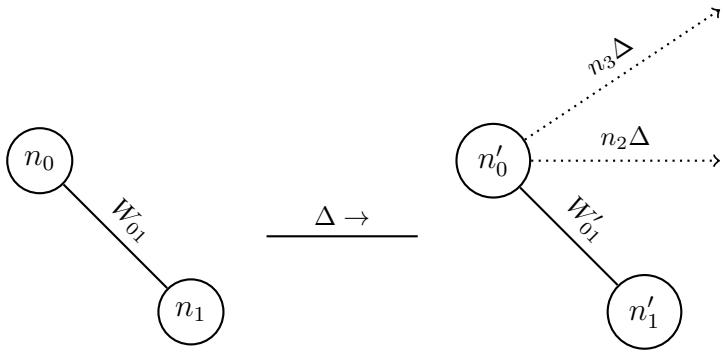

Figure 1: Δ describes how the graph will adapt over time. Each time step is associated with some problem in the constant problem stream, as the problem is solved the graph will adapt in a consistent way. The consistant way is some element (or combination there of) of the set of adaptation techniques, including but not limited too: { rewiring, attribute modification, node modification}. It is worth noting that degree preservation is not (usually) a goal of algorithms in this category.

4 Methods

4.1 American checkers rules

- played on an 8x8 American checkerboard
- 12 pieces per side
- pieces can move forward unless capturing
- to capture a piece one must cross another piece (of opposite loyalty) diagonally forward
- pieces that reach the opposite side promote to *kings*
- *kings* can move (as well as capture) forward or backward

4.2 Axiomatic strategy

I devise to construct a machine out of axiomatic modules, as a functional design. I have defined a board and the structure of the game, building several mapping functions that will progress the game, augment weights, and perform analysis. These mappings are non-bijective as any board of pieces can yield many temporally subsequent boards.

```
mapMatAux :: Int -> Int -> Matrix a -> (Int -> Int -> Matrix a
    ↪ -> b) -> [b] -> [[b]] -> Matrix b
mapMatAux r c mb f l wm
  | r < dem && c < dem  = mapMatAux r (c+1) mb f (f (r+1) (c+1)
    ↪ mb:l) wm
  | r < dem && c >= dem = mapMatAux (r+1) 0 mb f [] (l:wm)
  | otherwise = fromLists wm where
  dem = nrows mb

-- | 'mapMat is a map transform over any 'Matrix' a that exposes
    ↪  the row,column # to f
-- where the function exposes (row -> column -> 'Matrix' a -> b)
-- 'mapMat' will currently only take square matrices
mapMat :: Matrix a -> (Int -> Int -> Matrix a -> b) -> Matrix b
mapMat mb f = mapMatAux 0 0 mb f [] []
```

Figure 2: Haskell map function for making adjustments over any preset checkerboard

Utilizing mappings like the one in Figure 2 I can traverse the board in the same way for all algorithms. The mapping strategy also allows for easy-to-read code as functions can be built around the mapping. Weights can be adjusted with mappings however, the analysis step will require a reduction function with a similar signature.

4.3 Experiment parameters

The experiment is conducted as a comparison between two machines, a standard weighted decision matrix and an SRWM. The comparison is conducted on metrics pulled from games played. The machines will not store data between games but rather will run each game as a few-shot sequence. This way of conducting the experiment will allow for the most targeted testing for direct replacement of weighted decision matrices in standard trials.

5 Results

Do not worry about this section for draft 1

6 Conclusion

Do not worry about this section for draft 1

7 Future Work

This section is not fully thought out yet, so I will be adding to this before the final draft. This work is intended to lead to a detailed study into the likelihood of SRWMs being able to serve as a plugin replacement for weighted decision matrices. I do not intend to make any proof of this point however, my intention for this research is to prompt the question "are they better at other things too?". A good place for such research to start is to extend this experiment to a more complex game (perhaps Chess or Go). Another option is to expand proposition 1. The current experiment utilizes only few-shot learning. A possible expansion is to include learning phases and attempt the same test to see if we observe the same results.

References

[1] Kazuki Irie et al. *A Modern Self-Referential Weight Matrix That Learns to Modify Itself*. June 17, 2022. DOI: 10.48550/arXiv.2202.05780. arXiv: 2202.05780[cs]. URL: http://arxiv.org/abs/2202.05780 (visited on 10/23/2023).

[2] Vladimir Nasteski. "An overview of the supervised machine learning methods". In: *HORIZONS.B* 4 (Dec. 15, 2017), pp. 51–62. DOI: 10.20544/HORIZONS.B.04.1.17.P05.

[3] J. Schmidhuber. "A 'Self-Referential' Weight Matrix". In: *ICANN '93*. Ed. by Stan Gielen and Bert Kappen. London: Springer London, 1993, pp. 446–450. ISBN: 978-3-540-19839-0 978-1-4471-2063-6. DOI: 10.1007/978-1-4471-2063-6_107. URL: https://mediatum.ub.tum.de/doc/814784/document.pdf (visited on 09/18/2023).

Index

A Fine Grain Code Review