Greed is Feed: An Attempt to Find a Fair Selection Scheme Amidst Inequality

Nicholas Pun

Abstract. Suppose m players select from a pool $\mathcal{U}$ of n elements without replacement and with the additional restriction that their selections are limited to subsets $\mathcal{U}_1, \ldots, \mathcal{U}_m \subseteq \mathcal{U}$, respectively. When the U_i's only partially intersect, a prior player's selection may not affect all the players coming after them. Further, by changing the order that the m players draw from the pool, we may increase or decrease the probability of an element being chosen once it reaches a later player. What ordering will yield the most uniform probabilities for every player? In this paper, we answer this for the 2-player case and show that being "greedy" in our ordering may not yield optimal results for a very specific instance of a 10-person case through empirical results.

Contents

1 Motivation

The idea for this problem came from the All Random, All Mid (ARAM) game mode in popular multiplayer online game *League of Legends*[1]. In this game mode, two teams of 5 are randomly given characters (known as champions) to battle against each other and take out the opponent's home base first. There are several rules in play when players randomly select their champions:

- There is a universal pool of 152 champions[2] to select from and selections occur without replacement

[1] https://na.leagueoflegends.com/en-us/

[2] As of October 29th, 2020

- Players don't immediately have access to the entire pool and must purchase additional champions to increase their personal pool. This will result in different players having access to different subsets of the universal pool.
- There are always 14 champions that are *"free-to-play"* or accessible to all 10 players. (This prevents the case where a player runs out of champions to select from their pool)
- Players line-up in some order to make their selections

The last rule makes for an interesting probabilistic concern since a prior player's selection may only affect a fraction of the players after them (since some players may not have access to that champion in their personal pool). This differs from the case where everyone has access to the universal pool and are uniformly affected by a prior player's selection. As such, we might expect that as we change the ordering of the players, their individual probabilities of selecting a particular champion will change as well.

To see this concretely, let's consider the scenario where there are only 2 players. Player A has access to only 2 champions from the universal pool while Player B has access to the entire universal pool. If Player A goes first, regardless of the champion that is selected, Player B will have less champions to choose from. However, if Player B goes first, there is a probability that they select a champion that is inaccessible to Player A. As such, Player A's pool will be unaffected when it comes to their turn to choose. We will delve more into this example in Section 3.1.

2 Problem Definition

In this section, we will present a generalization of the problem described in Section 1 and also refine the questions we want to answer in this paper.

Let $\mathcal{U} = \{u_1, \ldots, u_n\}$ be the universal pool of n elements and $\mathcal{U}_1, \ldots \mathcal{U}_m \subseteq \mathcal{U}$ be the individual pools for Players P_1 through P_m respectively. Let $\sigma : [m] \longrightarrow [m]$ be an ordering of the m players. (So, $P_{\sigma(1)}$ is the player that goes first, $P_{\sigma(2)}$ second and so forth) Suppose the m players pick from $\mathcal{U}$ using the following scheme:

Input : Universal pool $\mathcal{U}$, player pools $\mathcal{U}_1, \ldots, \mathcal{U}$, and player ordering σ
Output: An assignment of $P_1, \ldots, P_m$ to unique elements $u_{P_1}, \ldots, u_{P_m}$

$\mathcal{U}_{\texttt{curr}} \leftarrow \mathcal{U}$
for $i \leftarrow 1$ **to** m **do**
  Select an element u_j (uniformly at random) from $\mathcal{U}_{P_{\sigma(i)}} - \mathcal{U}_{\texttt{curr}}$ and assign it to $P_{\sigma(i)}$
  $\mathcal{U}_{\texttt{curr}} \leftarrow \mathcal{U}_{\texttt{curr}} - \{u_j\}$
end

We've already seen that changing the order of the players has an impact on the individual probability distributions. (We will also show this more carefully in Section 3) Since we're ultimately playing a game, it's natural to want an ordering that maximizes fairness. That is, we want an ordering of players such that the probability for player P_i to select element $u \in \mathcal{U}_i$ is *close to* uniform $\left(\frac{1}{|\mathcal{U}_i|}\right)$. By *close to*, we mean that the following metric should be minimized:

Definition 1 (Hellinger Distance[3])**.** Let $P = (p_1, \ldots, p_k)$, $Q = (q_1, \ldots, q_k)$ be two discrete proba-

[3]See `http://encyclopediaofmath.org/index.php?title=Hellinger_distance&oldid=47206`

bility distributions, their *Hellinger distance* is defined as:

$$H(P,Q) = \frac{1}{\sqrt{2}}\sqrt{\sum_{i=1}^{k}(\sqrt{p_i} - \sqrt{q_i})^2}$$

Remark. $0 \leq H(P,Q) \leq 1$

Unfortunately, the question doesn't seem to yield a simple answer and so, while it will drive the research below, we may not see it answered by the end of paper. Instead, we fall back to a simpler question and end up comparing two candidate orderings: the random ordering and the "greedy" ordering.

In Section 3, we will solve the two-player case and see why a greedy approach is a candidate solution. We will also see why the general case is difficult to handle, leading to the idea of developing empirical evidence, which will be explored in Section 4. Finally, we will leave some possible directions to further this research in Section 5

3 A Mathematical Approach

3.1 Solving the Two-Player Game

Let's restrict ourselves to the case of two players: P_1 and P_2, with individual pools $\mathcal{U}_1$ and $\mathcal{U}_2$. To get rid of some boring cases, for the remainder of this section, we will further assume that $\mathcal{U}_1 \cap \mathcal{U}_2 \neq \emptyset$ and $\mathcal{U}_1 \neq \mathcal{U}_2$. It's easy to see that failure of the first assumption puts us in the case where selections are independent and failure of the second assumption puts us in the case where every owns the universal pool.

With those assumptions out of the way, what remains are two cases:

- $\mathcal{U}_1 \subset \mathcal{U}_2$: $\mathcal{U}_1$ is contained in $\mathcal{U}_2$
- $\mathcal{U}_1 \setminus \mathcal{U}_2 \neq \emptyset,\ |\mathcal{U}_1| < |\mathcal{U}_2|$: Both pools contain elements unique to themselves. Note that if $|\mathcal{U}_1| = |\mathcal{U}_2|$, then the ordering doesn't matter.

Proposition 1. *If $\mathcal{U}_1 \subset \mathcal{U}_2$ then P_2 should go first*

Proof. We show that when P_2 goes first, this induces a uniform distribution for both players. Let $\mathcal{U}_1 = \{a_1, \ldots a_m\}$ and $\mathcal{U}_2 = \mathcal{U}_1 \cup \{b_1, \ldots b_m\}$.

Since P_2 goes first, $\Pr(P_2 \text{ selects } a_i) = \Pr(P_2 \text{ select } b_j) = \frac{1}{n+m}$ for any i, j

Then, for P_1:

$$\begin{aligned}
\Pr(P_1 \text{ selects } a_i) &= \sum_j \Pr(P_1 \text{ selects } a_i | P_2 \text{ selects } b_j) \Pr(P_2 \text{ selects } b_j) \\
&\quad + \Pr(P_1 \text{ selects } a_i | P_2 \text{ selects } a_i) \Pr(P_2 \text{ selects } a_j) \\
&\quad + \sum_{k \neq i} \Pr(P_1 \text{ selects } a_i | P_2 \text{ selects } a_k) \Pr(P_2 \text{ selects } a_k) \\
&= \sum_j \frac{1}{n} \cdot \frac{1}{n+m} + 0 + \sum_{k \neq i} \frac{1}{n-1} \cdot \frac{1}{n+m} \\
&= \frac{m}{n(n+m)} + \frac{n-1}{(n-1)(n+m)} \\
&= \frac{1}{n}
\end{aligned}$$

So, the selection probability is uniform for both players. □

Proposition 2. *If $\mathcal{U}_1 \setminus \mathcal{U}_2 \neq \emptyset$ and $|\mathcal{U}_1| < |\mathcal{U}_2|$, then P_2 should go first.*

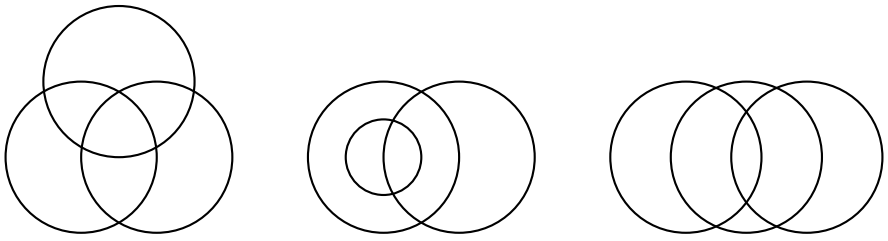
Proof. See Appendix A. The proof follows the same idea as above, but the computation is a bit more involved. □

In either case, we see that P_2 should always go first. Of course, since the assignment of the pools was arbitrary, the actual property that leads to this result is that $|\mathcal{U}_2 \setminus \mathcal{U}_1| > |\mathcal{U}_1 \setminus \mathcal{U}_2|$. In words: after removing the elements within the intersection of $\mathcal{U}_1$ and $\mathcal{U}_2$, $\mathcal{U}_2$ has more elements left over. Then, when P_2 goes first, they will have a higher probability of selecting an element *not* owned by P_1, making it such that the two player's choices are independent of each other. As hinted in Section 2, we will refer to this approach of "picking the player with the most leftover elements to go first" as the "greedy" approach. We state the conjecture that will drive the remainder of the paper:

Conjecture 1. *The greedy approach yields an ordering that minimizes the Hellinger distance between every player's selection distribution to the uniform distribution.*

3.2 Analyzing the Three-Player Game and Beyond

The two-player case is fairly straightforward since we reduced it to two real cases. However, as we move on to the more general case, we will find that the number of cases blow up exponentially. For example, with 3 players, we may see the following complex intersections:



As such, our approach of trying to break down the probabilities using law of total probability quickly becomes unwieldy.

There were attempts to model this as a sort-of stochastic process. However, such a model would require memory of arbitrary length since we can always create arbitrarily far dependencies. For example, we can consider a game where only two players have an element x in their pools, but the ordering places them first and last. This makes it seem difficult to model (most techniques only care about the previous state or a history of finite length), however, we admit here that the author also has limited knowledge in probability theory.

Nevertheless, we work with the intuition that we built when analyzing the two-player case. In particular, we want to get a better handle on the idea that a greedy approach may yield the best ordering. Intuitively speaking, by using a greedy approach, at each step we maximize the probability of making independent selections. As such, it feels like this will bring us towards a solution most close to uniform. In the next section, we will continue down this line of thought by trying to find empirical evidence of our conjecture.

4 An Empirical Approach

For the remainder of this section, we restrict ourselves back to the instance of our problem described in Section 1. The universal pool contains 152 elements and all 10 players share 14 common elements.

4.1 Initial Simulations

First, we summarize the technical details:

- A player is represented by their individual pool, which is a 152-length bit array. A 1 is present if the player owns the element and it is 0 otherwise.
- Our simulator performs the scheme given in Section 2.
- We run the simulator 100000 iterations per instance of randomly generated pools (where one instance is 10 players). Due to limited computational resources, we found that the gain in accuracy in performing more iterations was not worth the increase in time. To obtain the individual probability distributions, we keep track of the elements chosen per player and divide by 100000 (the total number of iterations).

We randomly generated 300 instances of our problem and ran them through our simulator with both a random ordering of players and a greedy ordering. Then, we measured the Hellinger distance of the resulting distributions to the uniform distribution. The aggregate results can be seen in Figure 1.

Interestingly, the results show that the greedy ordering yields distributions that are *ever so slightly* less uniform than when using a random order. This appears to contradict the conjecture we set out to gain evidence for! However, as we noted previously, we only obtained 300 samples out of the exponentially many possible instances of our problem. There's nothing special about this number, we just decided to stop the simulations at a round number after several hours of computation. The simulator scaled poorly with the number of iterations it was performing per instance. For this reason, we were motivated to find some way to speed up the process of simulating our game. This leads to the work done in the next section, where we completely replace the simulator with a neural network.

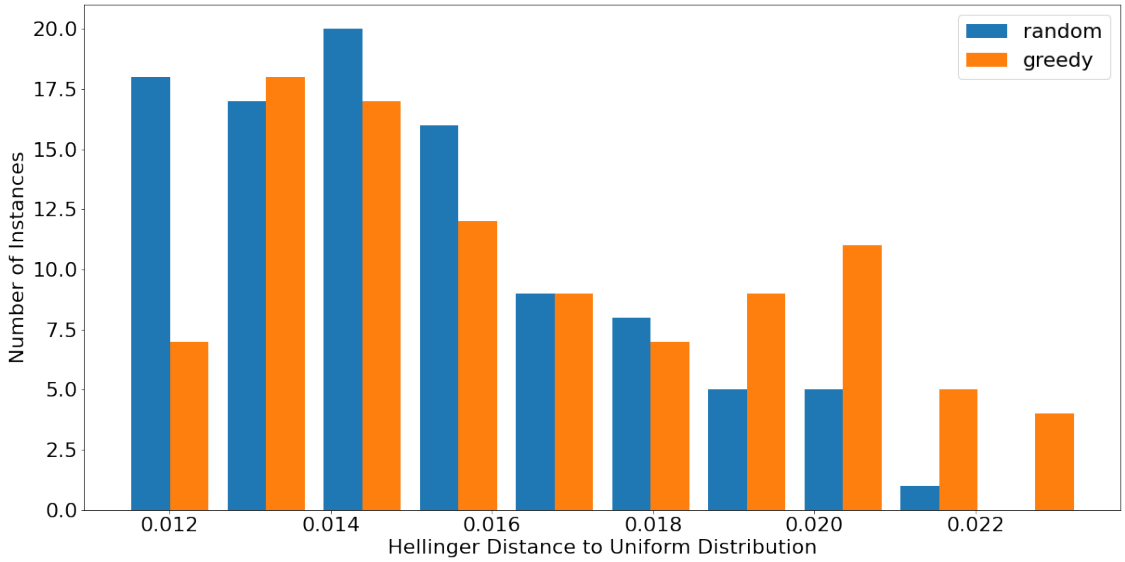


Figure 1: Hellinger distance of simulated distributions to the uniform distribution using a random ordering and greedy ordering.

4.2 Using a Neural Network

We end up using a very simple network (Table 1) to complete our task (Other, more complex networks were experimented on and the results are compiled in Appendix B). We call our network *ARAMNet*, since our motivations came from the game of ARAM.

Layer	Kernel Shape	Output Shape	# Params
LSTM		(10, 152)	186.048k
BatchNorm	(10, 1)	(10, 152)	20
Flatten		(1, 1520)	
Linear	(1520, 1520)	(1, 1520)	2.31192M
Reshape		(10, 152)	
Softmax		(10, 152)	

Table 1: ARAMNet

The most important part we had to design into our network was the ordering of the players. It had to be the case that if we modified the ordering, the output would reflect the change as well. (Unless, of course, our particular instance was order-invariant) This naturally led us to the idea of treating the 10 players as temporal sequence of data instead of just a matrix of pool encodings. More precisely (for those familiar with RNN-related notation), our input data is a sequence of 10 players: $(P^{<1>}, \ldots, P^{<10>})$, which we feed into an initial layer of LSTM cells. The network then outputs a 10-by-152 matrix M where $M_{ij} := \Pr(P_i \text{ selects } j)$ (i.e. The rows are precisely the discrete probability distribution for player P_i), performing exactly what our simulator was doing.

Although we said we would replace the simulator, we use it here one last time to generate 6000

samples for our network to train on. Our goal here is to minimize the Hellinger distance between the network output and simulator output, and we achieve a final training loss of 0.025 and validation loss of 0.035. We train using AdamW with all the recommended settings, except for an increase in the weight decay to 0.25, and a batch size of 16. The learning rate was adjusted using *ReduceLROnPlateau*[4], a built-in Pytorch learning rate scheduler, with a patience of 4 and a threshold of 0.01. The adjusting learning rate produces the bumps we see in Figure 2.

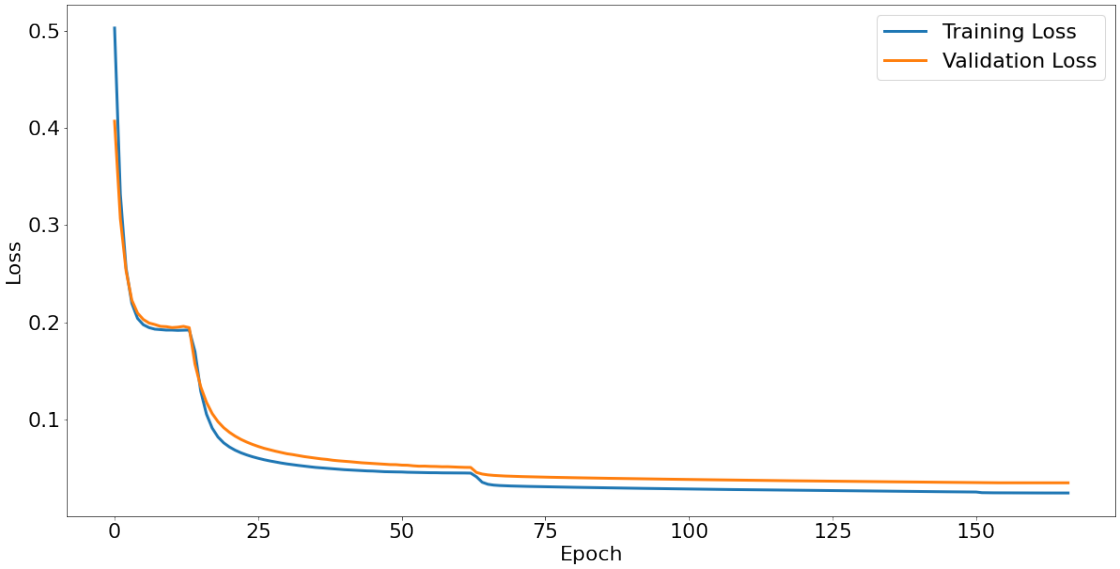


Figure 2: Training and Validation learning curve for ARAMNet

4.3 Revisiting the Simulations

Now we can revisit our simulations with a much more efficient simulation function. This time, we simulate over 1 million randomly generated pools, producing the results in Figure 3. As a side remark, the reader may notice that in Figure 3 the distance to the uniform distribution is higher than in Figure 1. This is expected since ARAMNet has $\sim$0.0325 validation loss.

Figure 3 shows that the greedy ordering produces distributions that have larger distance to the uniform distribution than when using a random ordering. Indeed, the mean Hellinger distance to the uniform distribution is 0.03578 when using a random ordering and 0.03603 when using a greedy ordering. Now, we can be a bit more sure that the phenomena we saw in Section 4.1 was not due to a small sample size. In fact, it may actually be the case that the greedy ordering is not as optimal as we conjectured. This doesn't quite disprove our conjecture since all of this evidence is empirical, but it does suggest that there may be other mechanisms at play within the player orderings.

5 Conclusions and Outlook

We began with the very general question "What is the best ordering for our players?". In completing the proof that the greedy ordering works best when there are two players, we got stuck trying to carry forward our techniques on the cases with more players. As such, we set out to find empirical evidence that perhaps the greedy ordering is optimal for more than two players as well. In doing

[4]See `https://pytorch.org/docs/stable/_modules/torch/optim/lr_scheduler.html#ReduceLROnPlateau`

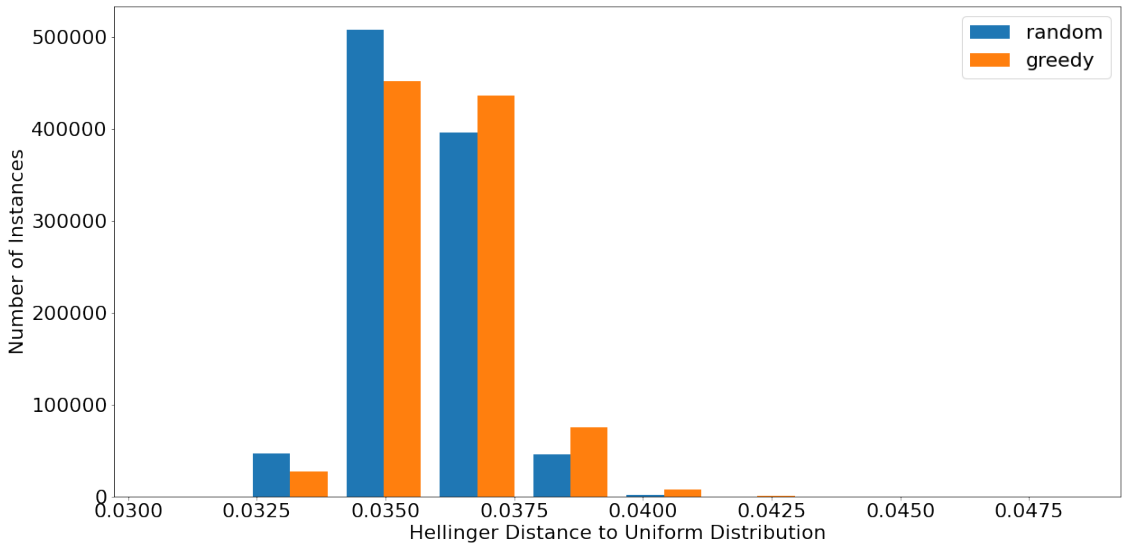


Figure 3: Hellinger distance of distributions produced by ARAMNet to the uniform distribution using random and greedy ordering

so, we find that simulating this game takes an extremely long time, and as a result, created a neural network that would handle the simulations. With that said and done, we find empirical evidence suggesting the opposite of what we set out to show: the greedy ordering is *not* optimal (at least for our restricted case of 152 elements and 10 players). This leaves us with one last follow-up question: if the greedy ordering isn't the best ordering, what other mechanisms are at play that will determine the best ordering?

Overall, we've built some useful tools to perform empirical analysis on this problem. In particular, the network can always be tweaked to accept different instances of our problem. This is also the weakness of our tool as it lacks generality. Indeed, we still lack a good grasp of the mathematics behind our problem and future research should focus on this aspect.

A Proof of Proposition 2

Recall the statement of our proposition: *If* $\mathcal{U}_1 \setminus \mathcal{U}_2 \neq \emptyset$ *and* $|\mathcal{U}_1| < |\mathcal{U}_2|$*, then* P_2 *should go first.*

Let:

- $\mathcal{C} = \mathcal{U}_1 \cap \mathcal{U}_2 = \{c_1, \ldots, c_x\}$. We will index elements of $\mathcal{C}$ with u (that is, $1 \leq u \leq x$ where possible)
- $\mathcal{U}_1 = \{a_1, \ldots, a_y\} \cup \mathcal{C}$. We will index elements of $\mathcal{U}_1$ with v (so $1 \leq v \leq y$)
- $\mathcal{U}_2 = \{b_1, \ldots, b_z\} \cup \mathcal{C}$. We will index elements of $\mathcal{U}_2$ with w (so $1 \leq w \leq z$)

By assumption, $y < z$.

P_2 goes first:

$$\Pr(P_2 \text{ selects } b_w) = \Pr(P_2 \text{ selects } c_u) = \frac{1}{x+z} \quad \forall u, w$$

For P_1, we solve $\Pr(P_1 \text{ selects } a_v)$ and $\Pr(P_1 \text{ selects } c_u)$ separately:

$$\begin{aligned}
\Pr(P_1 \text{ selects } a_v) &= \sum_w \Pr(P_1 \text{ selects } a_v \mid P_2 \text{ selects } b_w) \Pr(P_2 \text{ selects } b_w) \\
&\quad + \sum_u \Pr(P_1 \text{ selects } a_v \mid P_2 \text{ selects } c_u) \Pr(P_2 \text{ selects } c_u) \\
&= \sum_w \frac{1}{(x+y)(x+z)} + \sum_u \frac{1}{(x+y-1)(x+z)} \\
&= \frac{z}{(x+y)(x+z)} + \frac{x}{(x+y-1)(x+z)} \quad \forall v
\end{aligned}$$

$$\begin{aligned}
\Pr(P_1 \text{ selects } c_u) &= \sum_w \Pr(P_1 \text{ selects } c_u \mid P_2 \text{ selects } b_w) \Pr(P_2 \text{ selects } b_w) \\
&\quad + \sum_{k \neq u} \Pr(P_1 \text{ selects } c_u \mid P_2 \text{ selects } c_k) \Pr(P_2 \text{ selects } c_k) \\
&= \sum_w \frac{1}{(x+y)(x+z)} + \sum_{k \neq u} \frac{1}{(x+y-1)(x+z)} \\
&= \frac{z}{(x+y)(x+z)} + \frac{x-1}{(x+y-1)(x+z)} \quad \forall u
\end{aligned}$$

Similarly, when P_1 goes first:

$$\begin{aligned}
\Pr(P_1 \text{ selects } a_v) = \Pr(P_1 \text{ selects } c_u) &= \frac{1}{x+y} \quad \forall v, w \\
\Pr(P_2 \text{ selects } b_w) &= \frac{y}{(x+y)(x+z)} + \frac{x}{(x+y)(x+z-1)} \quad \forall w \\
\Pr(P_2 \text{ selects } c_u) &= \frac{y}{(x+y)(x+z)} + \frac{x-1}{(x+y)(x+z-1)} \quad \forall u
\end{aligned}$$

In both cases, the player that goes first is granted a uniform distribution over their choices. So, only the distribution of the 2nd player diverges from uniform. We show that when P_2 goes first, the

distance between the induced distribution for P_1 is closer to uniform than P_2's induced distribution when P_1 goes first.

The Hellinger distance between P_1's induced distribution (when P_2 goes first) and the uniform distribution is:

$$\sum_v \left(\sqrt{\frac{1}{x+y}} - \sqrt{\frac{z}{(x+y)(x+z)} + \frac{x}{(x+y-1)(x+z)}} \right)^2 + \sum_u \left(\sqrt{\frac{1}{x+y}} - \sqrt{\frac{z}{(x+y)(x+z)} + \frac{x-1}{(x+y-1)(x+z)}} \right)^2$$

A similar monstrosity can be produced for P_2's induced distribution (when P_1 goes first). The proof completes by showing the difference between the two is less than 0 which is a routine exercise of applying the assumption that $y < z$ (the inequality $\frac{1}{(z-1)y} < \frac{1}{z(y-1)}$, which is a direct consequence of $y < z$ will also be handy) repeatedly.

B An Exploration into Various Neural Architectures for the Simulator Replacement

Recall that ARAMNet (Table 1) is a very simple network consisting of a single layer of LSTM cells followed by a single linear layer. As such, most of our exploration will be focussed on various deep versions of this.

We start with two natural deep versions of ARAMNet. Network 2 (Appendix B) extends the number of LSTM layers, while Network 3 (Appendix B) extends the number of Linear layers. Next, we treat ARAMNet as its own module (without the final softmax layer). Stacking ARAMNet modules one after another gives us Network 4 (Appendix B). Finally, we out something really funky. Taking inspiration by the law of total probability, we wanted to help the neural network learn that type of formula. As such Network 5 (Appendix B) does exactly this and perform additions and multiplications on the outputs of ARAMNet modules.

Layer	Kernel Shape	Output Shape	# Params
LSTM (3 layers)		(10, 152)	∼558k
Flatten		(1, 1520)	
Linear	(1520, 1520)	(1, 1520)	∼2M
Reshape		(10, 152)	
Softmax		(10, 152)	

Table 2: Network 2 Architecture (We omit the BatchNorm layer and Dropout layers between the individual LSTM layers)

Layer	Kernel Shape	Output Shape	# Params
LSTM		(10, 152)	∼184k
Flatten		(1, 1520)	
Linear (3 Layers)	(1520, 1520)	(1, 1520)	∼7M
Reshape		(10, 152)	
Softmax		(10, 152)	

Table 3: Network 3 Architecture (We omit the BatchNorm, Dropout and Activation layers between each Linear layer)

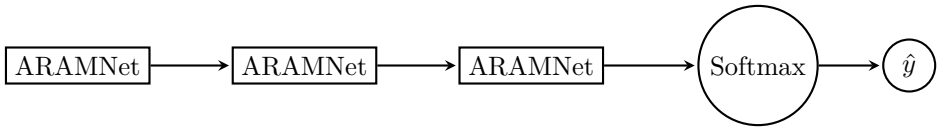


Figure 4: Network 4 Architecture Diagram

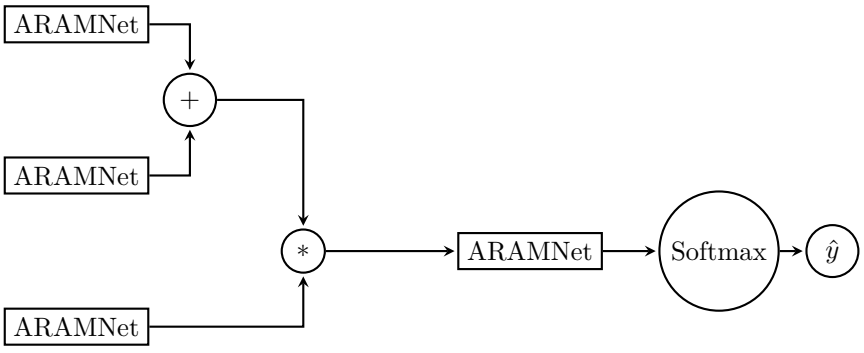


Figure 5: Network 5 Architecture Diagram

We trained the 4 networks on our small validation dataset of 300 randomly generated samples, using the default settings for Adam and a batch size of 32, and plotted their learning curves

alongside ARAMNet in Appendix B. Network 2 and (intriguingly) Network 5 were also considered as candidates to replace the simulator function. In particular, we were very surprised that Network 5 achieved a loss matching ARAMNet. It is, in fact, learning something interesting about the data and not just memorizing it, since the network contains ∼9 million parameters, yet, Network 3 with ∼7M performs *much* worse.

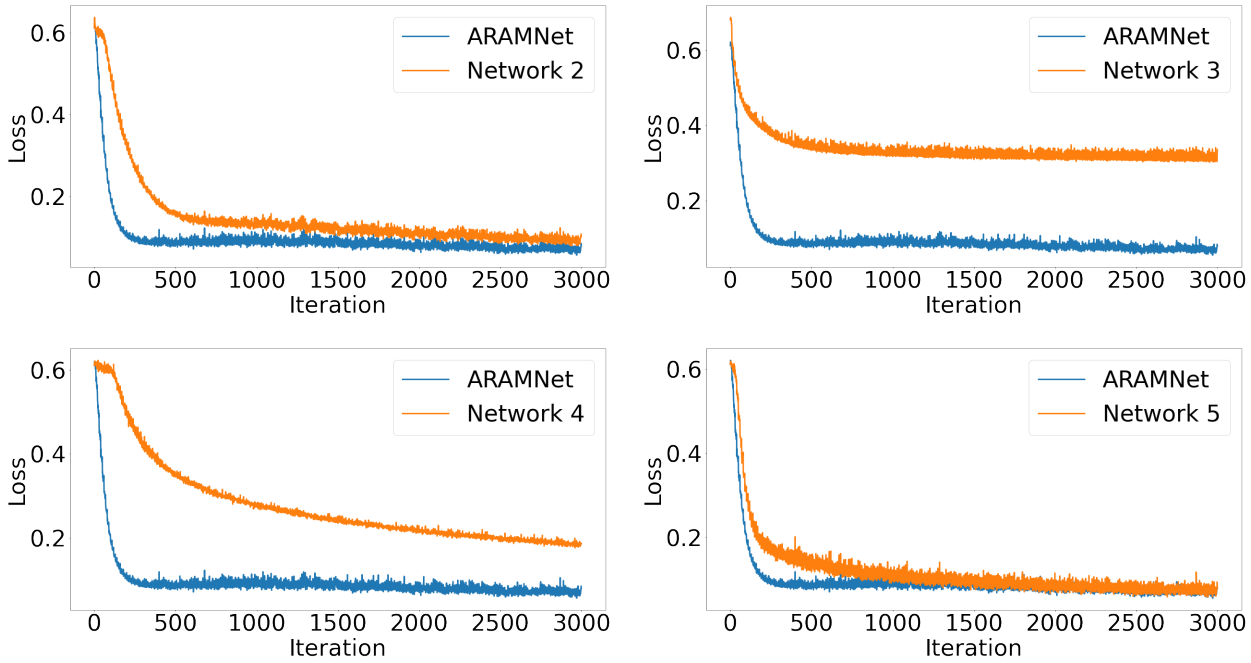


Figure 6: Learning Curve of the 4 alternate networks compared to the learning curve of ARAMNet on the validation dataset

However, these two networks were more difficult to train and we had trouble preventing them from overfitting. Although they could reach training losses matching that of ARAMNet, their validation loss were much too high for them to be useful simulator replacement candidates In Appendix B, we can see just how large the training-to-validation loss gap is, as compared to Figure 2.

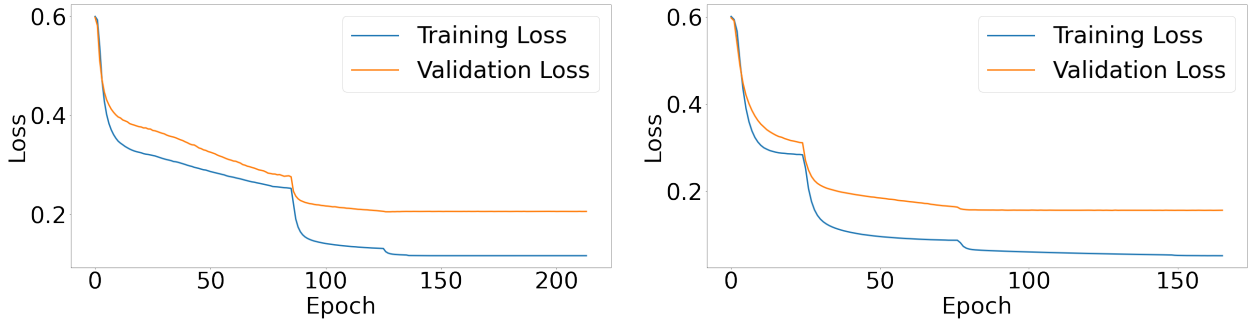


Figure 7: Learning curves for Networks 2 and 5 when we trained using the same optimizer and settings as with ARAMNet.