

ALiEN: the Sum Game

Timothee Mickus

August 2021

1 Introduction

Emergent communication studies generally focus on games: setups involving agents that must interact through communication in order to solve a task. Our focus here is on the “sum game”. The first agent (the “sender”) sees two integers a and b between 0 and N included and produces a message. The second agent (the “receiver”) is shown this message and must output the sum $a + b$.

Multiple approaches to implement this game can be devised. Two seem to us particularly relevant. The first consists in casting this game as a categorization problem: the receiver must select one of $2N+1$ potential targets, given the sender’s message. The second consists in considering the numerical aspect of the game: the receiver must output some scalar value corresponding to the sum $a + b$. While the former approach suggests to tackle this game using classification tools, e.g., using a cross-entropy loss; the latter suggests to employ scalar regression techniques, e.g., using a mean-squared error loss. The angle we take here consists in comparing these two approaches, and see whether they impact the emergent communication protocol between the two agents.

2 Experimental design

2.1 Model implementation

The two types of models we will compare are designed to be as similar as possible. They are composed of two agents each: one sender, which uses an input embedding representing the pair $\langle a, b \rangle$ to prime a RNN and produce a message, and one receiver, which reads the sender’s message using a RNN and derives a prediction from the final hidden state.

The sole difference between the two types of model is how the final prediction is derived, and the loss function used to train the models. In the categorization approach, the final hidden state is linearly transformed into a distribution over all $2N+1$ possible answers; the entire model is then trained to minimize cross entropy. In the regression approach, the

final hidden state is transformed into a scalar value s mapped onto the interval $[0 ; 2N]$ using a scaled sigmoid function $2N \cdot \sigma(s)$; the entire model is then trained to minimize the squared error.

2.2 Data

Throughout our experiments, we use $N = 63$; i.e., all integers $\langle a, b \rangle$ are comprised between 0 and 63 included and their sum s is between 0 and 126. This corresponds to $64^2 = 4096$ possible pairs $\langle a, b \rangle$. We randomly split all possible pairs $\langle a, b \rangle$ in three groups. 80% (3276 pairs) are used for training models, 10% (410 pairs) are used for selecting hyperparameters, and 10% (410 pairs) are used to further test the model’s ability to generalize. All models are trained and tested and the same random splits, to ensure our experiments are comparable.

2.3 Hyperparameter space

The hyperparameters that we consider can be divided in four categories.

Learning mechanism:

- (i) what learning rate to use
- (ii) whether gradient is propagated with Gumbel Softmax or REINFORCE
- (iii) what temperature to apply (if using a Gumbel Softmax)
- (iv) what entropy coefficient to apply to the sender (if using REINFORCE)

Data handling:

- (v) whether to represent inputs with one-hots or binary expansions¹
- (vi) how many examples to present per batch
- (vii) whether to average the loss over all examples in a batch

¹In the latter case, embedding components correspond to the 0-padded binary expansion of the integer. This limits the number of parameter weights used to represent items and encodes the structure of the input space more explicitly.

Model architecture:

- (viii) whether to use a RNN, GRU or LSTM for generating & decoding messages
- (ix) what dimensionality the symbol embeddings should have
- (x) what dimensionality the hidden representations should have

Symbolic channel:

- (xi) how many distinct symbols to use
- (xii) whether or not to use a curriculum learning mechanism²
- (xiii) how often the maximum length should be updated during the curriculum
- (xiv) how many epochs the curriculum learning should span

Note that these hyperparameters do not include maximum message length, which we set to 8 in all experiments, for simplicity. Also note that there is an interaction between hyperparameters. For instance, (iii) and (iv) are never in effect jointly, and depend both on the value for (ii); likewise, if (vi) is set to 1, then (vii) has no effect; lastly, (xiii) and (xiv) depend on (xii) being set.

2.4 Selecting hyperparameters

Hyperparameter choices are likely to impact regression-based game differently from categorization-based games. We therefore require a principled way of selecting hyperparameters. Given the large search space we consider (see Subsection 2.3), a grid search would be inefficient. Instead, we opt for Bayesian Optimization. This technique consists in iteratively selecting hyperparameters that are the most likely to improve some objective value over the current optimum. The probability of yielding an improvement given a configuration of hyperparameters is updated after each iteration using Bayes’ theorem.

We perform such 100 iterations in total. In order to construct a prior distribution to bootstrap the Bayesian Optimization process, we spend the first 10

²I.e., instead of using the maximum default length for messages, we set the maximum length to 1 before training begins and update it every so often, until the original maximum default length is reached. Exploring the full action space is therefore easier for the sender. As messages cannot be too complex, the receiver must therefore adopt a policy that covers the most general cases. Whenever the maximum length is updated, the agents can build on their simpler, partly correct, previous language to reach higher accuracy.

Hyperparameter	Regression	Categorization
Learning rate	0.0015	0.00025
Mechanism	GS	GS
Temperature	10	$1.26 \cdot 10^{-5}$
Entropy coefficient	(1.0)	(0.7)
Input format	Binary	Binary
Batch size	2	256
Average over batch	No	Yes
Cell architecture	GRU	GRU
Embeddings dim.	16	1024
Hidden dim.	16	1024
Vocabulary size	13	28
Curriculum	No	No
C. updates	(5)	(3)
C. duration	(28)	(31)

Table 1: Hyperparameters for best models

iterations testing purely random samples of hyperparameter configurations. We use accuracy on a development set as the objective value to optimize: for each configuration of hyperparameters sampled, we train 3 different models for up to 100 epochs, compute the maximum accuracy they reach on the development set and return the average.³

This methodology is however computationally costly. In order to limit the time required by each experiment, we implement two mechanisms. First, we stop early any run that has not seen any improvement in accuracy over 10 epochs. Second, we keep track and save the model that performs best on development data at any point during the experiment, and will use only this optimal model for experiments down the line. These two mechanisms may impact results: in principle, configurations could yield higher performances if we used a longer patience or no early-stopping; likewise, we may end up selecting for manual analysis a model that happens to apply well to the development set.

3 Results & Analysis

We begin by focusing on the hyperparameters corresponding to the best models, shown in Table 1. Hyperparameters that have no effect are displayed between parentheses. Some parallels between regression and categorization models exist: both use a GRU-based architecture, with binary expansion vectors to embed the input pairs, trained with a Gumbel-Softmax and no message length curriculum.

³This average is motivated by the fact that different runs with the same hyperparameter configuration can in practice bring about very different results.

	Train	Dev	Test
MSE	0.260	0.259	0.269
Acc.	0.712	0.741	0.734

(a) Regression-based game

	Train	Dev	Test
XENT	2.718	2.751	2.723
Acc.	0.191	0.195	0.163

(b) Categorization-based game

Table 2: Performances of best models

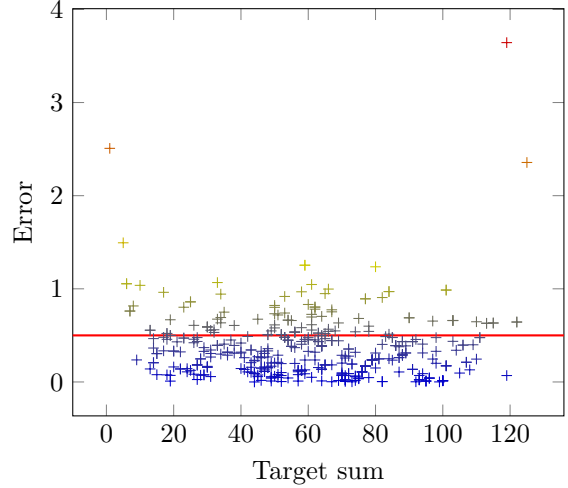
The remaining hyperparameters are however in stark contrast. The regression model employs a lightweight architecture, with comparatively few neurons, whereas the categorization model uses the largest available size. While the regression model benefits from few examples per batch, the categorization model uses large batches and averages the loss over all items. The initial temperature of the Gumbel-Softmax are 6 orders of magnitude apart. This disparity shows how crucial the hyperparameter selection process is.

Automatic metrics for the most accurate model of each type are displayed in Table 2. Both models yield accuracy scores much higher than random baselines. For reference, given that we use $N = 63$, a most frequent baseline corresponds to an accuracy of 0.016, whereas a uniform random guess yields an accuracy of 0.008.⁴ The regression-based model yields a much higher accuracy than the categorization-based model. The former scores in the 71 ~ 74% range, while the latter only reaches 16 ~ 20%. This high margin of improvement strongly suggests that a numerical regression approach is more suitable in order to achieve a high degree of communication on the sum game.

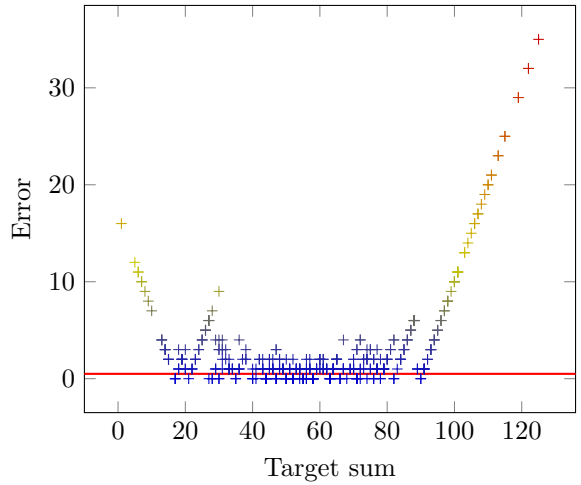
Figure 1 displays the margin of error between the prediction of the receiver and the target sum $a + b$. The red line on each sub-figure delineates predictions counted as correct from those counted as erroneous. Comparing the two subfigures reveals that the regression-based model’s errors are of a lesser magnitude (< 4) than the categorization-based model (< 40). Both models tend to err more for extreme target sums: this stems from the fact that for targets near 0 or $2N$, there are fewer pairs $\langle a, b \rangle$ that sum to these values, hence there are fewer datapoints to train the models on these values.

Another point to take note of is that the scatter-

⁴We refer the reader to Appendix A for the description and computation of these baselines.



(a) Regression-based game



(b) Categorization-based game

Figure 1: Prediction error on the development set

plot of categorization-based exhibits linear trends. These are due to the model predicting the same value for a number of distinct contiguous targets. For instance, all target sums below 17 will correspond to a prediction of 17, whereas all target sums above 90 will correspond to a prediction of 90. This behavior gives rise to errors of greater and greater magnitude for values near the extrema 0 and $2N$, and thus translates as the two linear trends on either side of the scatter-plot. This behavior is however not limited to extreme values; instead, it seems to apply to a number of intervals of target sums throughout the development set. As a concrete example, most of the pairs whose sum is between 65 and 61 are predicted to sum to 63.

We now focus on analyzing the contents of the mes-

	Train	Dev	Test
$\rho_{\bar{e}}$	0.140	0.152	0.131
$\rho_{\langle a, b \rangle}$	0.459	0.487	0.454
ρ_{a+b}	0.722	0.722	0.704

(a) Regression-based game

	Train	Dev	Test
$\rho_{\bar{e}}$	0.204	0.210	0.192
$\rho_{\langle a, b \rangle}$	0.545	0.573	0.538
ρ_{a+b}	0.846	0.862	0.831

(b) Categorization-based game

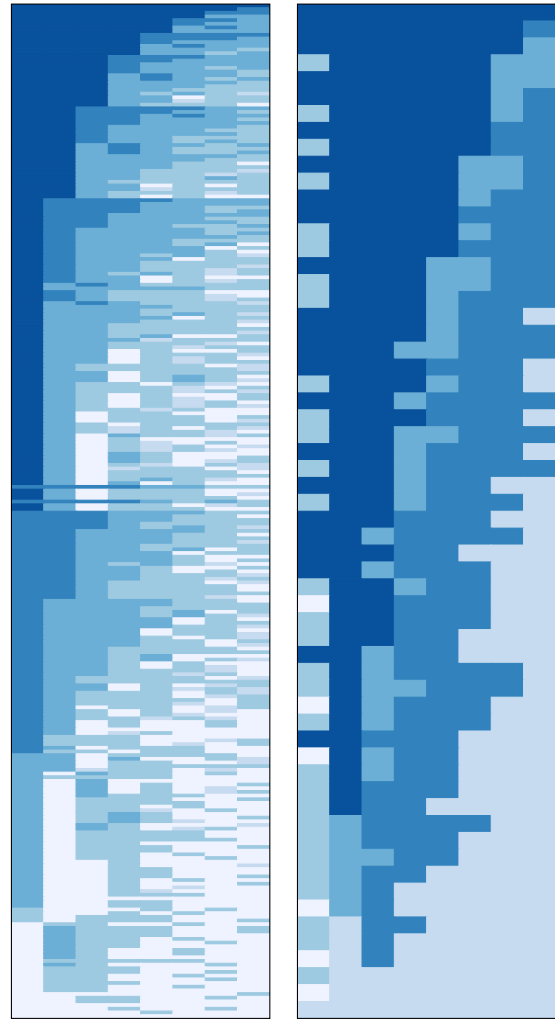
Table 3: Topographic similarity scores

sages exchanged by the agents in both models. Table 3 displays the results from different topographic similarity measures. All use an edit distance between messages as the form metric and an Euclidean distance for the meaning metric. For $\rho_{\bar{e}}$, we represent meanings as the corresponding input pair embeddings; for $\rho_{\langle a, b \rangle}$, as the point $\langle a, b \rangle$ in the 2D plane; and for ρ_{a+b} , as the 1D point on the real line at the sum $a + b$. Each of these three correlations probe a different aspect. With $\rho_{\bar{e}}$, we study whether the sender interprets its input or simply passes along all the information to the receiver; with $\rho_{\langle a, b \rangle}$, whether the message encodes the input pair (i.e., whether the agents agree on letting the receiver compute the sum $a + b$); and with ρ_{a+b} , whether the message encodes the sum (i.e., whether the agents agree on letting the sender compute $a + b$).

Results for these correlation metrics suggest three important elements. First, senders of both models tend to interpret the input, and not simply pass along the information to the receiver. Second, messages correlate very highly with the sum: this suggests that messages mostly focus on forwarding the sum. This should also indirectly explain the correlation observed when using the input pair as a meaning: operands are likely to be correlated with the corresponding result.⁵ Third, topographic similarity scores for the categorization-based model are higher than for the regression-based game, despite its lesser accuracy.

From these topographic similarity scores, one could posit that the failure point in the categorization model is the receiver, not the sender whose messages seem to encode the necessary information. This would intuitively make sense: we require of the sender

⁵Distance between two points $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$ correlates with the absolute difference of their components' sums $|(x_1 + y_1) - (x_2 + y_2)|$ ($\rho = 0.6$ for our entire dataset).



(a) Regression-based (b) Categorization-based

Figure 2: Color-coded overview of all messages on development set for best models of both types

here not only to interpret the meaning of the message (viz., the scalar value of the sum $a + b$) but also to memorize how to express this meaning (viz., which unit of the output layer should be activated). In contrast, in the regression model, the sum $a + b$ is both the meaning encoded in the message and what the model has to output. Hence the task ought to be simpler for a sender trained on regression.

This analysis, however, does not hold if we have a look at the actual messages produced by the sender in the categorization-based game. Figure 2 displays the messages produced by our two best models on the development set. To produce this visualization, we first order all messages from highest to lowest predicted sum, then drop duplicates, and finally assign colors to symbols according to the order in which they are

encountered. Each row of the colored matrices therefore correspond to a distinct message; darker cells indicate symbols used in messages describing items predicting to have a high sum whereas lighter cells correspond to symbols from messages of sums predicted to be lower.

To begin with our manual analysis, we can point at the fact that both models always produces messages of the maximum length of 8, but a limited number of symbols: 5 for the categorization-based model, and 6 for the regression-based model. Most importantly, the categorization-based model produces much 60 distinct messages, whereas the regression-based message produces 276: given that there are 108 distinct targets in the development set, the sender in the categorization-based approach cannot be said to encode all the necessary information. Why then did we observe this high correlation of messages edit distance with absolute difference of targets?

Looking at the messages from the development set for the categorization-based model, as shown in Sub-figure 2b, we observe that all pairs predicted to sum at 90 are described with the same message “7 7 7 7 7 7 7 7”. No pairs are predicted to yield a sum between 89 and 83; pairs predicted to sum at 82 all receive one of the two following messages: “7 7 7 7 7 7 7 12” or “7 7 7 7 7 7 7 11” (the latter occurs only twice). The next cluster of pairs all predicted to sum to the same value (78) are described with one of the following: “7 7 7 7 7 7 11 11” (occurs 15 times), “7 7 7 7 7 7 12 11” (4 times), “26 7 7 7 7 7 12 12” (once).

In short, messages for a given cluster of prediction are very similar to one another; the most common messages for two subsequent clusters of prediction, such as 90 and 82, or 82 and 78, differ by one or two symbols. If we were to provide a naive characterization the general trend, messages generally transition from higher to lower prediction clusters by ‘pushing’ new symbols at the end of a message. In all, throughout the development set, categorization-based messages clearly describe clusters of prediction; subsequent clusters tend to be described by highly similar messages with low edit distance. This also explains why we observed a high correlation of messages with the target sum: edit distance does reflect the ordering of prediction cluster, which in turn tend to consistently group together intervals of targets.

The messages produced by the regression-based game, shown in Sub-figure 2a, appear to be more complex. The model is more accurate, hence predictions clusters tend to be much smaller and often correspond to examples that do sum to the same value. Moreover we observe many synonyms. For

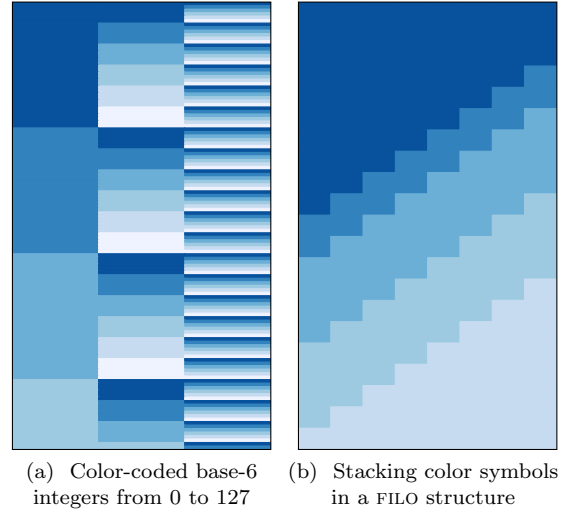


Figure 3: Visualizations of synthetic regular patterns

instance, all three instances of a prediction of 98 are described with different messages, all separated by an edit distance of 1 or 2: “6 6 6 5 1 1 4 1”, “6 6 6 5 1 1 1 4” and “6 6 6 1 5 1 1 4”.

At first glance, a ‘pushing’ trend similar to what we observed for messages from the categorization-based model seems to be in place. We do see that the highest target sums of 125 and 122, as well as one of the instances of 119, all receive the same message “6 6 6 6 6 6 6 6”, which yields a prediction of 122. The second instance of 119 is described with “6 6 6 6 6 6 6 5”, and corresponds to a prediction of 119. The following highest target sums, 115, are described and correctly understood with “6 6 6 6 6 6 5 5”. The next group of targets, at 113, correspond to the message “6 6 6 6 6 6 1 1”. As target values decrease, the number of “6” symbols at the beginning of the message decreases as well, replaced by “5” and “1”, then “4” and other symbols.

This ‘pushing’ is trend is however much less straightforward than previously: aside from cases of swapping, “5” symbols are strictly absent from items predicted to sum between 69 and 58, but are almost systematically attested for items with predicted sums above 40 and below 122. If we consider the full set from top to bottom, we can make two interesting observations. First, we find repeating patterns, but smaller in scale and offset by one index. Second, symbols tend to switch much more frequently at the end of the messages. Rather than ‘pushing’ symbols, this would suggest some crude form of counting. Figure 3 provides a synthetic visualization of what would entail these two trends: Subfigure 3a shows a regular base-six encoding of integers from 0 to 127 and Sub-

figure 3b a stacking operation in a FILO datastructure. The regression-based model’s messages depart from the regular encoding in two ways. First, they are much less regular; second, symbols appearing in messages predicting low values tend to not appear in messages predicting high values and vice-versa.⁶

In short, we find that the higher accuracy of the regression model is reflected in the greater complexity of its communication protocol. Despite this distinction, there are obvious parallels between the two models’ protocols. Both encode the sum rather than the paired input integers; both minimize the distance between messages for items predicted to be almost equal; both use a handful of symbols.

4 Conclusions

We have implemented and compared two approaches for the sum game: one based on a classification setup, and one based on regression. We designed our experiment to make the models as comparable as possible by using the same hyperparameter space and selection process for both approaches. The regression approach was found to be much more accurate. Notwithstanding, we observe similarities in the languages that emerge from the two models.

While we have implemented a number of mechanisms ranging from message length curriculum to structured input representations, there are other techniques we could apply to improve our results. For instance, from a purely technical point of view, we could initialize the RNN and embeddings of both agents with the same weights: this would make the hidden states of both agents highly similar in early stages of the training. A more conceptual solution could lie in switching the agents’ roles randomly: by optimizing for both sides of the task, they could converge on a communication protocol more efficiently.

A Random baselines

Given a pair of integers $\langle a, b \rangle$, both drawn uniformly from $U(0, N)$ we search what is the probability that their sum $a+b$ equals some value y . More formally, we denote $Y = a+b$; $a, b \sim U(0, N)$, and we characterize $P(Y = y)$ by counting the number of ordered pairs $\langle a, b \rangle$ that sum to y to express this probability.

⁶An interesting direction suggested by this analysis would be to look at whether there is a gradient that would range over the stacking pattern of a FILO datastructure, the behavior of the categorization-based model, that of the regression-based model, and the pattern of digits in numbers.

We note that the value y is bounded in $0 \leq y \leq 2N$. Either one of the two following cases must be true: either $y \leq N$, or $N < y \leq 2N$. If $y \leq N$, then we have one trivial solution, where $a = y$ and $b = 0$. All other solutions will have the form $a' + b' = y$, with $a' = a - k$ and $b' = b + k$. As a cannot be negative, we thus get $y+1$ different solutions. If $N < y \leq 2N$, then we have the trivial solution $a = N$ and $b = y - N$. All other solutions will have the form $a' + b' = y$, with $a' = a - k$ and $b' = b + k$. As b cannot be greater than N , we thus get $N - (y - N) + 1 = 2N - y + 1$ solutions. Rearranging both cases, and by independence of a and b , we thus get:

$$P(Y = y) = \frac{1}{(N+1)^2} \left(\min(y, 2N - y) + 1 \right)$$

From this, we can get a most frequent baseline by simply considering

$$\max_{y^*} P(Y = y^*)$$

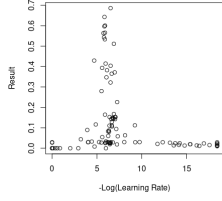
This optimum is found at $y^* = N$. Producing this value regardless of input would yield an accuracy of $\frac{1}{N+1}$. A less strict random baseline would consist in selecting a value uniformly at random between 0 and $2N$; i.e. $\hat{Y} = U(0, 2N)$. By independence and marginalization, we see that:

$$\begin{aligned} P(Y = \hat{Y}) &= \sum_{y \in \Omega(Y)} P(Y = y) P(\hat{Y} = y) \\ &= \frac{1}{2N+1} \sum_{y \in \Omega(Y)} P(Y = y) \\ &= \frac{1}{2N+1} \end{aligned}$$

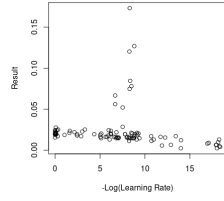
These baselines hold under the assumption that a and b are drawn uniformly. In particular, this entails that splitting the data into train and test partitions will impact these baselines. In practice, we found that models could reach an accuracy of 0.03 on the development set with a purely random strategy, i.e., roughly twice what our calculations suggest.

B Effects of hyperparameters

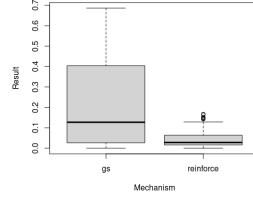
Figure 4 below displays visualizations of the accuracy on development reached by models during the hyperparameter selection process. Since configurations are not sampled independently from one another, we provide no statistical overview of whether differences they suggest are significant. The main takeaway is that the regression process seems more sensitive to hyperparameters selected. Surprisingly, vocabulary size has little effect, which we link to the few symbols actually used in the best models’ messages.



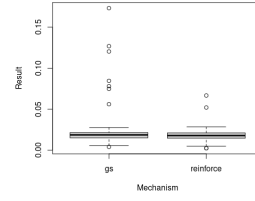
(a) LR, Regression



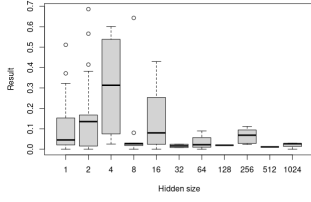
(b) LR, Categorization



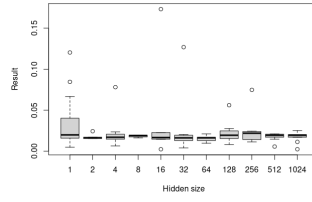
(c) Mechanism, Regression



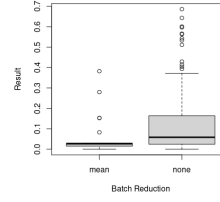
(d) Mechanism, Categorization



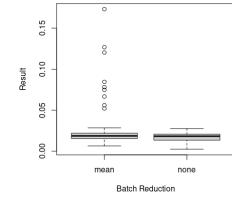
(e) Batch, Regression



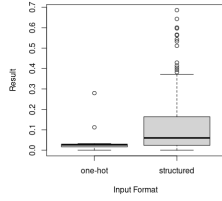
(f) Batch, Categorization



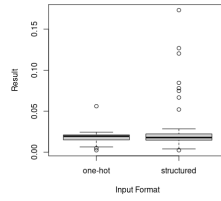
(g) Loss aggregation, Regression



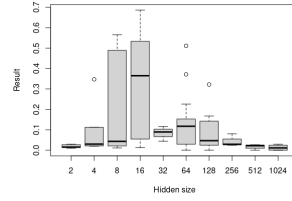
(h) Loss aggregation, Categorization



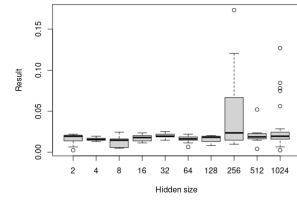
(i) Input, Regression



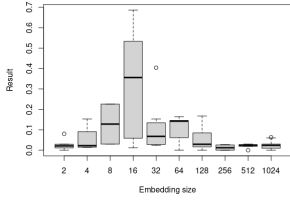
(j) Input, Categorization



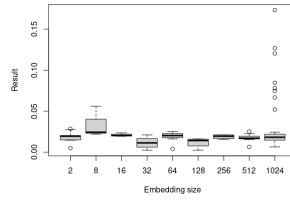
(k) Hidden size, Regression



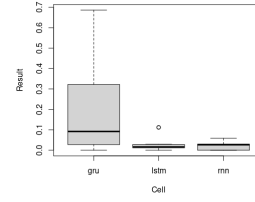
(l) Hidden size, Categorization



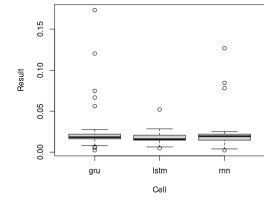
(m) Embedding size, Regression



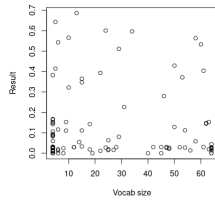
(n) Embedding size, Categorization



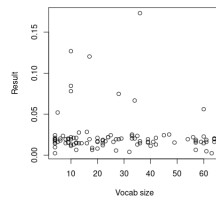
(o) Cell type, Regression



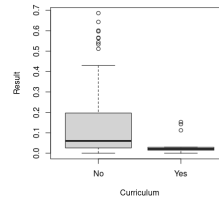
(p) Cell type, Categorization



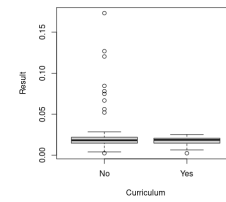
(q) Vocab size, Regression



(r) Vocab size, Categorization



(s) Curriculum, Regression



(t) Curriculum, Categorization

Figure 4: Overviews of hyperparameters sampled during selection process