Sum Communication Game using EGG

Silvia Sapora

1. Introduction

The goal of this report is to set up and analyze a multi-agent communication game between two Neural Networks: a Sender and a Receiver. The Sender is given two input integers ($a, b \in [0, N]$) and the Receiver has to output their sum.

The Sender is only allowed to send the Receiver a single-symbol message. The symbol is chosen from a finite vocabulary $V = \{v_1, ..., v_j\}$ of unique symbols. The size of the vocabulary j can be varied. We will analyze how changes to the vocabulary size affect communication and performance.

We start with the hypothesis that, given this game setting, the optimal result would be a system where the Sender learns to calculate the sum of the two numbers and sends the Receiver a symbol representing their sum. After this, the Receiver just has to map the symbol to its corresponding value.

Despite this, other scenarios where the Sender sends the Receiver a symbol representing the two input numbers and the Receiver sums them are also possible, as well as scenarios where the Sender and Receiver somehow "share" the computational work and the message sent represents something in-between sum and starting inputs.

We will set up our experiments and report to try and answer the following questions:

1. Does the Sender network learn to match each symbol to a sum? Or is the Receiver performing the computation?
2. What does the exchanged symbol represent?
3. Is the system overfitting? Is it simply memorizing the mapping between input numbers and output sum? In other words, does the network generalize?
4. How does pre-training the Sender network (or the Receiver network) change the result of the experiment?
5. How is training and performance affected by different input data?
6. How is training and performance affected by varying vocabulary size?
7. How does performance change if we approach the game as a regression problem?

2. Proposed Method

For the game's implementation, we will use the EGG toolkit. EGG allows developers to implement their own multi-agent games with discrete channel communication. All we have to do is define the following components:

- Input data
- Agents' architecture
- Communication type (one-symbol, fixed-length or variable-length multiple-symbol messages)
- Loss

We will describe our game set-up through the components listed above.

2.1. Input data

To generate input data, we implemented a few custom Pytorch datasets. Each dataset sample will include three elements: the two input numbers a, b and their sum $s := a + b$. k is the number of samples in each dataset and can be customized. In the problem specification, $a, b \in [0, N]$ and $s \in [0, 2N]$.

Skewed Integers Dataset For this first dataset, we want to create as many distinct samples as possible for each N. Unfortunately, including all possible combinations of sums in the dataset will lead to a skewed dataset (the sums distribution is not uniform). In classification problems, skewed datasets might cause the model to overfit to one (or a few) classes. We explain how we deal with this issue in the Loss Subsection 2.4.

To create this dataset, we start by creating the arrays containing the two input numbers. Each array will contain k i.i.d. samples drawn from the same uniform distribution $\mathcal{U}(0, N)$. This will results in two arrays $a = [a_1, ..., a_k]$ and $b = [b_1, ..., b_k]$. We generate the sum array $sums = [s_1, ..., s_k]$ where $s_k := a_k + b_k$. We then remove all duplicates, as to avoid train and testing sharing information[1].

[1] two sample are considered equal if their ordering is the same. So $(5, 2, 7)$ is considered different from $(2, 5, 7)$. This is done so the network can learn commutativity.

As can be seen in Figure 1 (right), the two input numbers are uniformly distributed, while their sum is not.

For this dataset, we create two versions: one where we directly input the two integers in the network, and another one where the two integers are one-hot encoded first.

Uniform Float Dataset For this dataset, we want the sums to be uniformly distributed. To ensure this, we start by drawing k samples from $S \sim \mathcal{U}(0, 2N)$. For each drawn element s_i, we generate the first number a_i by drawing from $\mathcal{U}(\max(0, s_i - N), \min(s_i, N))$ (this is to ensure both a and b are still within $[0, N]$). The second number is simply calculated as follows $b_i := s_i - a_i$. The resulting distribution can be observed in Figure 1 (left). The key point here is that our samples are floating point numbers instead of integers. This allows us to generate a larger dataset with more samples while the sums are still uniformly distributed. To be able to use this dataset in a classification setting, we calculate the output to be the class corresponding to the closest integer (e.g. 3.4 would be 3, 4.9 would be 5). We also filter this dataset to remove duplicates.

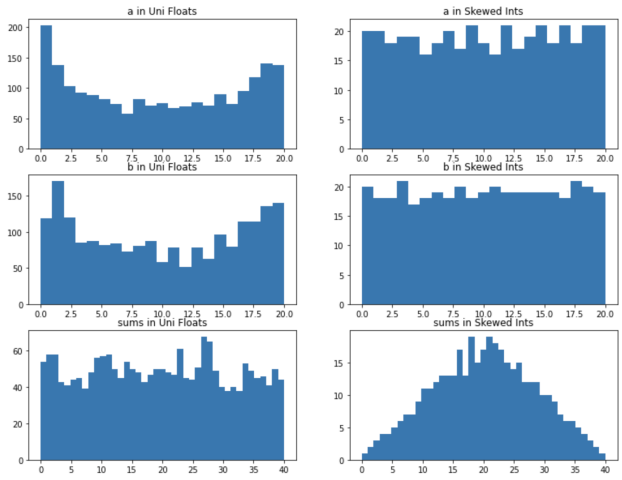

Figure 1. Data distributions of the two datasets (for $N = 20$ and $k = 2000$)

The charts represent the data distributions across the two datasets (Uniform Floats dataset on the left and Skewed Ints dataset on the right). On the two top rows, we show the data distributions of the two input numbers a and b. On the bottom row, we show the data distributions of their sums.

Test/Train split Each dataset is then randomly split into two parts: one to be used during training and one during test, to verify how well the set-up generalizes. The two datasets are split to be disjointed, so we are able to test the system's generalization abilities.

2.2. Networks Architecture

To choose a suitable network architecture, we wanted to make sure the architecture would have enough capacity and complexity to be able to correctly learn the sum function correctly. To do this, we first trained a single network in the sum prediction task. This removed the complexity caused by the communication setting, and ensured we had a sufficiently large network to learn.

After achieving acceptable performance ($> 80\%$ on all Ns) on one architecture, we used it for Sender network. The architecture is the following: two hidden layers of size $2N + 1$ and RELU activation. For the Receiver network we simply used one hidden layer of the same size as the vocabulary and RELU activation (we experimented a bit with this but noticed Receiver architecture wasn't impacting performance much).

This means both architectures vary depending on N. This choice was made to allow networks trained to predict sums on smaller Ns to learn more effectively given fewer examples.

2.3. Communication type

To enable communication, EGG provides Gumbel-Softmax and REINFORCE wrappers. The REINFORCE wrapper is mostly useful when exchanging multiple symbols. Since this was not the case for our set-up, the REINFORCE wrapper was briefly tested out but quickly discarded as its performance was significantly worse than the Gumbel-Softmax one.

2.4. Loss Function

We can set up this problem as either a regression or a classification problem. Depending on which set-up we choose, we will need different loss functions. In the classification case, we pass a weight tensor to rescale the loss of each class. This will allow the model to pay equal attention to examples from minority classes when the dataset presents a skewed class distribution. The weight tensor is calculated as follows:

$$weights := [\frac{k}{n(0)}, ..., \frac{k}{n(N)}]$$

where $n(N)$ indicates how many times a certain class (in this case N) appear in the dataset. The weights vector is then normalized.

Classification For classification, we use Cross Entropy Loss.

$$CrossEntropyLoss = -\sum_{i=0}^{k} y \log \hat{y}$$

Regression For regression, we use MSELoss.

$$MSELoss = \frac{1}{k}\sum_{i=0}^{k}(y_i - \hat{y}_i)^2$$

In both formulas, y represents the correct sum and $\hat{y}$ the output of the Receiver's network.

3. Experiments

In the interest of simplicity and brevity, we will only comment and discuss a subset of our results. Moreover, all results discussed here have been calculated for the best performing hyperparameter combination (the one described in Subsection 3.2). We will report and compare results for different Ns ($N = 5$, $N = 20$ and $N = 50$) as well as comparison of regression vs classification performance, across the different datasets and vocabulary sizes.

To easily compare regression vs classification performance, we report both accuracy (calculated by rounding the regression's prediction to the nearest integer) and absolute error. All performance calculations are reported for both the training set as well as the test set (for the Uniform Float dataset we test on the Skewed Integer dataset).

3.1. Baseline

To begin, it is useful to calculate a baseline for the network's performance. In other words, if the network was randomly guessing, what would its loss be? This gives us a good starting point to understand if the network is learning effectively or not. For classification on uniformly distributed datasets, we will consider an accuracy of $\frac{1}{2N+1}$ as our baseline (where $2N + 1$ is the number of classes we need to predict). For regression, we will consider our baseline to be an absolute error of N.

3.2. Hyperparameter Search

Training was explored using multiple learning rates and batch sizes. We tested the network's performance with `straight-through=True` (so picking the argmax argument of the message during training) and without, as well as different epoch numbers. For each hyperparameter setting, we ran the network 5 times and picked the best result.

We settled on `batch_size = 5`, `learning_rate = 1e-3` and `straight_through = False`. `n_epochs` varies depending on the size of the dataset, spanning $[30, 200]$.

3.3. Datasets

We analyze how different datasets affect performance. Results can be observed in Table 3.3. Excluding a few exceptions, generally the Uniform Floating dataset outperforms the others. We guess this is because it provides the system with more examples to learn from, so it can generalise better.

3.4. N

Generally, it seems like the system's performance gets worse as N increases. This is with the exception of systems trained using the One-Hot encoded dataset, where a larger number of samples generally seemed to aid learning.

3.5. Pretraining

Pretraining the Sender network to sum the two digits and output their sum as a classification problem proved to be extremely successful and very helpful for the network to learn. This was true for both regression and classification tasks. It was particularly useful for larger Ns.

3.6. Vocabulary Size and Symbol Meaning

Many vocabulary sizes were explored. Ranging between N and $N * N$. If the network is pre-trained, generally it will limit itself to using $(N \times 2) + 1$ symbols at most, each representing one or more sums.

When using the Uniform Float and Skewed Ints datasets, generally each symbol will represent a range of sums. So, as an example, symbol number 2 might indicate sums 1, 2 and 3.

In the one-hot encoded dataset, on the other hand, symbols might end up representing sums that might be quite "far" from each other, indicating the system might be trying to communicate some kind of "in-between" state of the sum. Interestingly, the same sum is also often encoded as different symbols, depending on which two input numbers we are using.

With the Uniform Float and Skewed Ints datasets, it seems like the main issue is that the system is not making use of all its available vocabulary, using one symbol to represent many sums in a range. On the other hand, we get almost no overlap, and each sum (on the input side) will always correspond to the same symbol (with the exception of sums on the edge of two ranges). Figures 2, 3 and 4 show symbol usage for $N = 20$ for different datasets.

3.7. Generalization

For all the considered datasets, we split the possible distinct inputs into 80% train and 20% test. The datasets are split to be non-overlapping. This means measuring generalization is straightforward. If performance on the test network is good then it means the networks didn't simply memorize the inputs and outputs.

Pretraining	N	Loss	Dataset	Acc	Abs Error	Acc (train)	Abs Error (train)
Yes	5	MSE	Uni. Floats	65.9	0.3991	62.4	0.4010
		CrossEntropy	Uni. Floats	**89.3**	0.1071	87.1	0.3063
			Skewed Ints	50.0	0.500	71.4	0.2857
			Skewed 1H	12.5	1.1250	42.9	0.7857
	20	MSE	Uni. Floats	24.6	1.1057	24.7	1.0848
		CrossEntropy	Uni. Floats	**76.4**	0.2359	68.9	0.4081
			Skewed Ints	68.5	0.3258	81.2	0.1903
			Skewed 1H	49.4	0.9662	62.2	0.5852
	50	MSE	Uni. Floats	11.5	2.4666	10.9	2.4720
		CrossEntropy	Uni. Floats	49.3	0.5322	44.9	0.6989
			Skewed Ints	45.3	0.7773	44.8	0.8259
			Skewed 1H	**90.7**	0.2873	99.2	0.0362
No	5	MSE	Uni. Floats	**61.6**	0.4055	62.25	0.3977
		CrossEntropy	Uni. Floats	42.9	0.6428	45.4	0.6863
			Skewed Ints	0.0	1.875	21.4	1.9642
			Skewed 1H	0.0	1.500	42.9	0.7142
	20	MSE	Uni. Floats	24.65	1.0805	24.5	1.0849
		CrossEntropy	Uni. Floats	**28.1**	1.1420	23.4	1.3856
			Skewed Ints	26.4	0.9638	27.1	0.9969
			Skewed 1H	15.7	1.7191	21.6	1.6846
	50	MSE	Uni. Floats	9.2	2.9198	9.12	2.9313
		CrossEntropy	Uni. Floats	**16.5**	2.0294	13.2	2.4863
			Skewed Ints	10.7	2.6890	12.6	2.5326
			Skewed 1H	11.8	10.0899	13.2	10.6276

Table 1. Summary of results. Accuracy and absolute error are calculated over the Skewed Integers test dataset for all runs. We cannot train the system using MSE loss and the Skewed Integers dataset as MSE loss cannot take class weights into consideration. Best performance for each N-Pretraining combination is highlighted.

Looking at the results in Table 3.3, we can see the Skewed Integers and One-Hot Encoded datasets tend to cause the network to overfit significantly, particularly when N is smaller and therefore dataset size remains small. The Uniform Floats dataset, on the other hand, consistently shows good generalization properties, even though the test dataset is made up of integers while the system trained on floats.

Moreover, we also test our networks against input numbers outside of the $[0, N]$ range (excluding the ones that accept one-hot encoded values as that wouldn't be possible). Results can be observed in Figures 5, 6 and 7.

4. Conclusions

We can now try to summarize our findings and answer our original questions:

1. The Sender network was generally performing most of the sum computation
2. When the system learns successfully from the available data, the Sender network does match each symbol with its corresponding sum. Alternatively, when trained on the Uniform Floats and Skewed Integers datasets, it would send a symbol representing a range of sums. We didn't really notice any case where the Sender network started passing a symbol representing the two input integers, but this might have been because we didn't make our Receiver network complex enough. When trained on the One-Hot Encoded dataset, we noticed the symbols didn't really correspond to either sums nor the input integers, indicating some kind of in-between encoding was being used.
3. The system tends to overfit in cases where it doesn't have a lot of data to learn from. In other cases, it seems to generalize fairly well
4. Pretraining the Sender network really helped in improving performance
5. Using floats to train the network is an efficient strategy to improve performance on integer datasets
6. The system didn't use more than $2N + 1$ symbols in the vocabulary
7. Regression didn't seem a particularly effective strategy. We hypothesize that this is because of the discrete nature of the message sending process

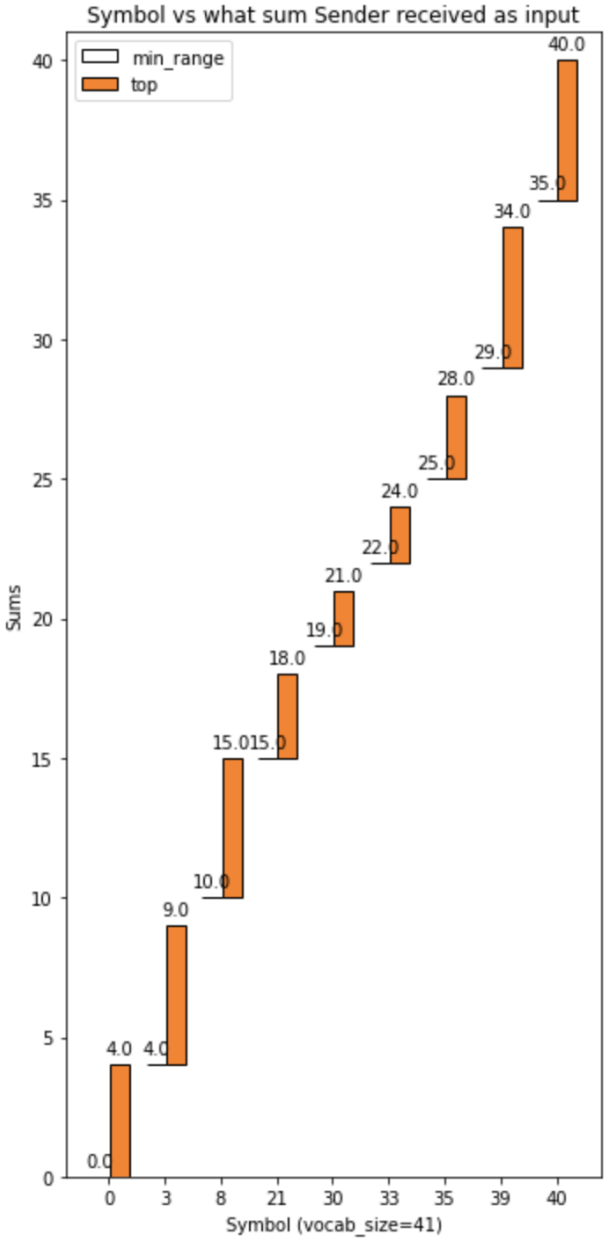

Figure 2. Symbol analysis for model trained using the Uniform Float Dataset

Symbols chosen on the x-axis. Sum of input numbers on the y-axis. $N = 20$, trained using CrossEntropyLoss without pretraining

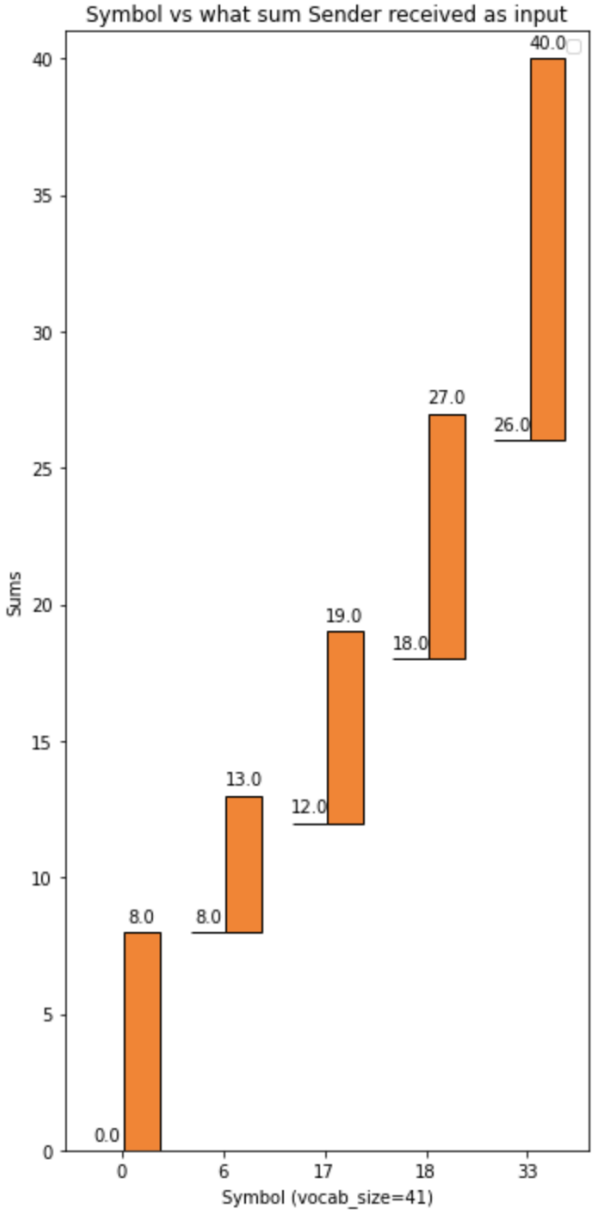

Figure 3. Symbol analysis for model trained using the Skewed Ints Dataset

Symbols chosen on the x-axis. Sum of input numbers on the y-axis. $N = 20$, trained using CrossEntropyLoss without pretraining

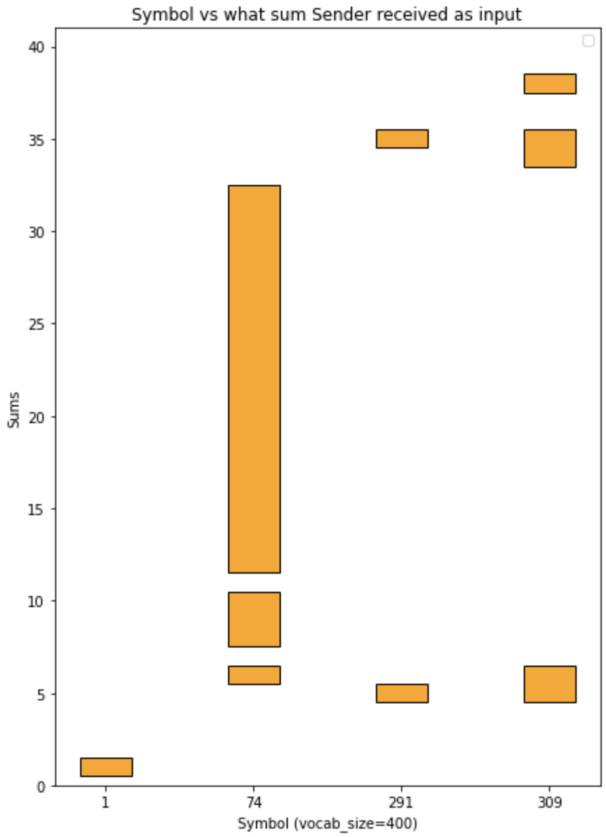

Figure 4. Symbol analysis for model trained using the Skewed One-Hot Encoded Dataset
Symbols chosen on the x-axis. Sum of input numbers on the y-axis. $N = 20$, trained using CrossEntropyLoss without pretraining. The same pattern appears in both train and test datasets.

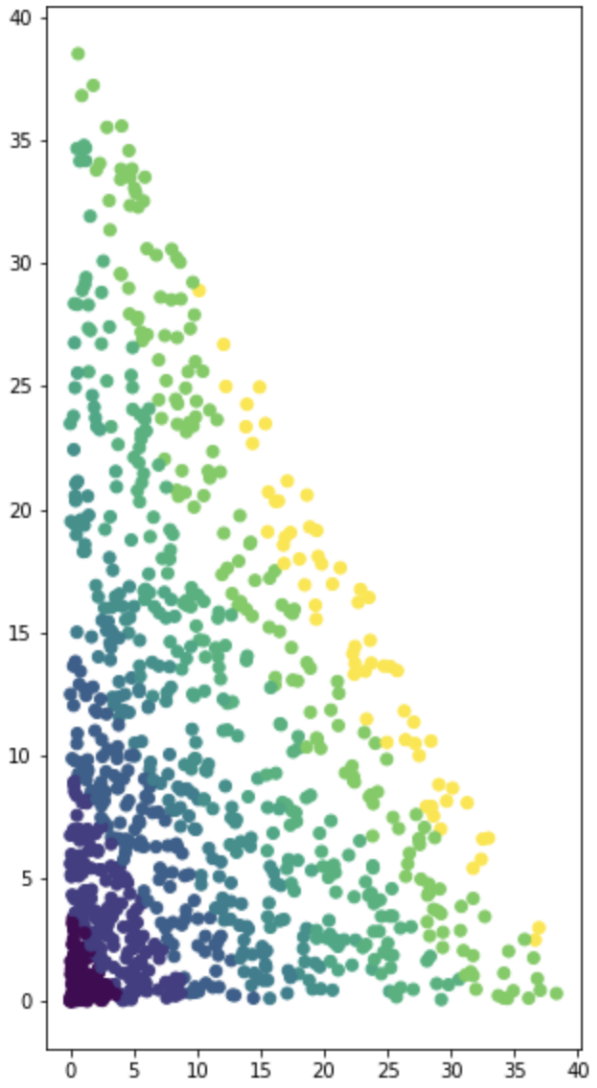

Figure 5. Performance for Uniform Floats Dataset
Each color represents a different output from the Receiver. x-axis is the first value, y-axis is the second value the Sender receives as input. The system generalizes well even to inputs outside of its training dataset $[0, 20]$ (in the graph: $[0, 40]$).

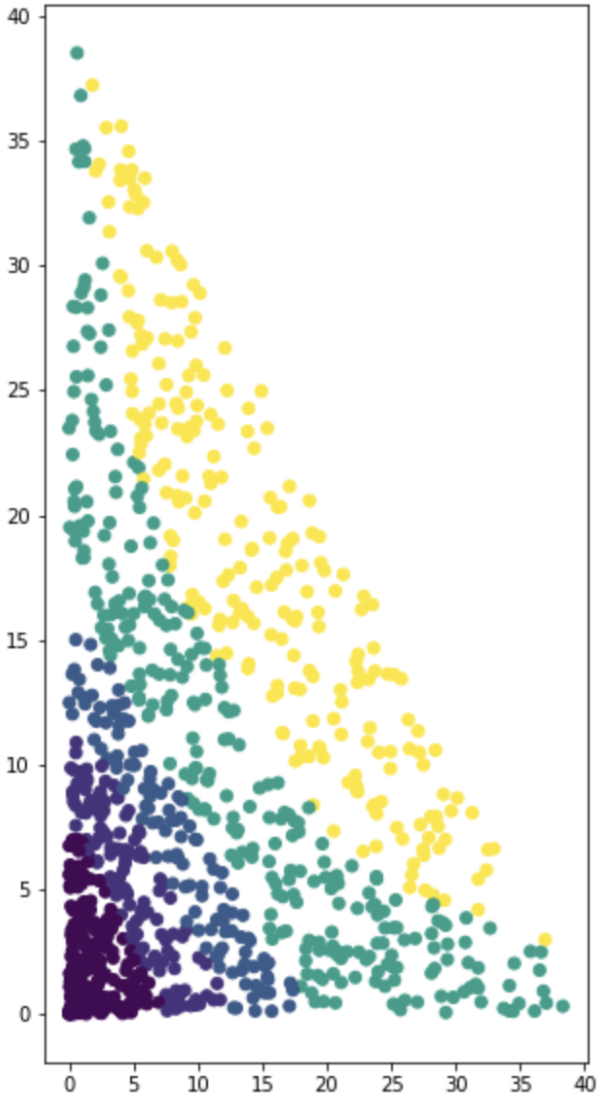

Figure 6. Performance for Skewed Integers Dataset
Each color represents a different output from the Receiver. x-axis is the first value, y-axis is the second value the Sender receives as input. The system generalizes well even to inputs outside of its training dataset $[0, 20]$ (in the graph: $[0, 40]$).

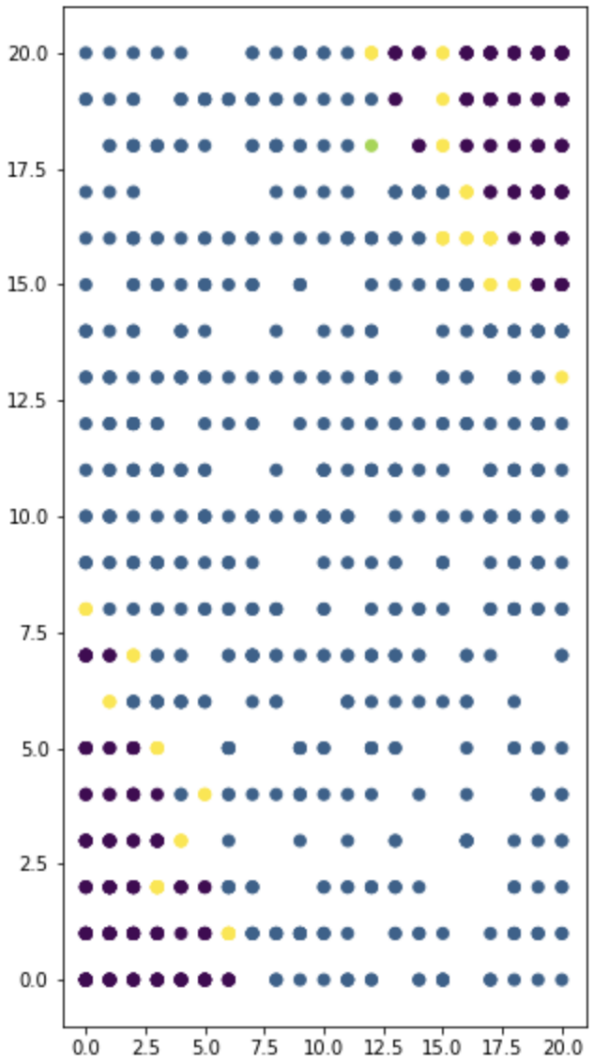

Figure 7. Performance for Skewed One-Hot Encoded Dataset
Each color represents a different output from the Receiver. x-axis is the first value, y-axis is the second value the Sender receives as input.