Winter 2023 COEN 6331 Assignment Report (IV)

Jun Huang, Student ID: 40168167
Electrical and Computer Engineering, Concordia University
Email: jun.huang@concordia.ca

Abstract

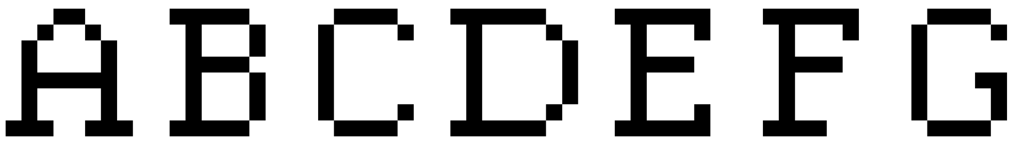
This report presents the experiments and results of building a type one Adaptive Resonance Theory (ART) network to recall 20 given patterns from the corrupted inputs. We simplify the network architecture by trimming down the unused signals and dataflows to reflect our ART network practically. We perform two sets of experiments to explore how the ρ value affects the network's performance on different levels of corrupted samples. The result shows that the lower ρ values limit the memorization capacity and cause unstable performance, while higher ρ values result in a more stable and good performance.

Index Terms

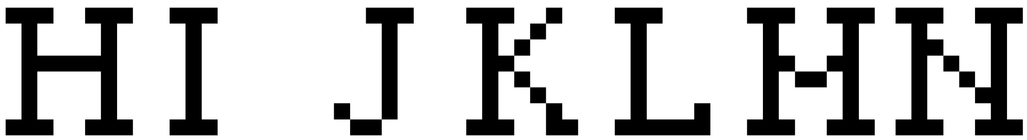
Adaptive Resonance Theory, ART I

I. Problem Description

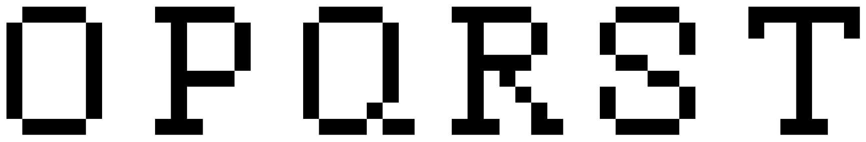
In this report, our goal is to build an type one Adaptive Resonance Theory (ART I), which is for the binary data, to memorize 8×8 alphabet patterns from A to T. The patterns are shown in Fig. 1. We explore the varying of the performance of ART I due to different network settings. Regarding the stucture of ART I network, the hypeparameter of it is only the ρ value, which is the vigilance value in the orienting subsystem.



(a) Pattern 'A' to 'G'



(b) Pattern 'H' to 'N'



(c) Pattern 'O' to 'T'

Fig. 1: Target patterns from 'A' to 'T'

II. Adaptive Resonance Theory I

In this report, we present the type one Adaptive Resonance Theory (ART I) architecture based on the definition presented by the book "Neural networks: algorithms, applications, and programming techniques" [1]. What is different is that we trim down the unused variables and dataflows from the original architecture and present the architecture in a way that could reflect its actual process. Further more, some of the notations are renamed in order to convey the architecture simply and precisely under the scope of this assignment. On the other hand, we also discuss a vital implementation, which is not well discussed in the book, of ***how the network extends itself*** when it searches through all the learned patterns, which are different from the current input pattern.

A. Architecture Overview

As Fig. 2 is shown, we denote the following notations. The input I vector with the length of M is the pattern which should be learned. In the F_1 layer, we split it into two parts. The F_1 A takes three inputs from the original input I, the gain control signal G, and vector V, which is the output from F_2 layer that passes through the top-down LTM traces. It then generates the activities vector X_1 based on th $2/3$ rule. The F_1 B takes the activities vector and apply the binary step function $s(\cdot)$, which can be decribed with Eq. 1, to obtain the output vector S of the F_1 layer.

$$s(x_{1i}) = \begin{cases} 1 & \text{if } x_{1i} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

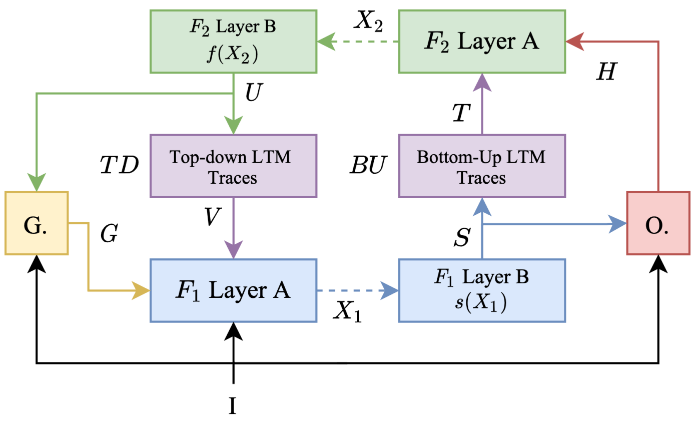


Fig. 2: Simplified architure of ART I network. Blue boxs indicate the F_1 layer. Green boxs indicate the F_2 layer. Puple boxes indicate the long-term memory (LTM) traces. Red box indicates the orienting subsystem. Yellow box indicates the gain control signal.

Output S then passes through the bottom-up LTM traces and applies weight matrix BU to obtain the input vector T for the F_2 layer. It is also conveyed to the orienting subsystem to determine if we should inhibit the certain units in the F_2 from joining the competition. For the F_2 layer, we also split it into two parts. Due to the properties of the $f(\cdot)$ function, where we apply the competitive learning as is shown in Eq. 2, we generate the X_2 vector in an easy way. The F_2 A generates the activities vector X_2 by simply applying the inhibition vector H on the input vector T. We denote this process as function 'inhibit(T,H)'. We will discuss the details of the two activities vectors X_1 and X_2 in the next section.

$$u_j = f(x_{2j}) = \begin{cases} 1 & j = \text{argmax}(\text{inhibit}(T,H)) \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

After obtaining the output vector U of the F_2 layer, the output is conveyed to both the gain control and the top-down LTM traces. The gain control generates the signal so that if U is active and I is inactive, the signal G becomes 1; otherwise, 0. On the other hand, we applies weight matrix TD on the output U to obtain input vector V for F_1 layer.

B. Algorithm Implementation

The activities vector X_1 is determine by the Eq. 3, where A_1, B_1, C_1, and D_1 are constants and the $\dot{x}_{1i}$ is the derivative taken with respect to time. Those constants are initialized under some conditions to guarantee the $2/3$ rule runs correctly. In this report, we omit the mathematical details of these conditions but present them directly.

$$\dot{x}_{1i} = -x_{1i} + (1 - A_1 x_{1i})(I_i + D_1 V_i + B_1 G) - (B_1 + C_1 x_{1i}) \tag{3}$$

Suppose we train a network that takes input vectors with the length of M and N number of units in the F_2 layer. At the beginning of the network we initialize the second activities vector X_2 to all zero.

Since there is no input from I, it is a zero vector and signal G is also 0. This gives us the initial state of the X_1 vector as Eq. 4 when the network reach the equilibrium point, namely, the $\dot{x}_{1i} = 0$.

$$x(0)_{1i} = \frac{-B_1}{1 + C_1} \tag{4}$$

We initialize the two weight matrixes BU and TD with Eq. 5 and Eq. 6, where L is also a constants. Note that BU should be a $N \times M$ matrix so that we would have a input vector $T = BU \times S$ for F_2 layer which length is N. Likewise, TD should be a $M \times N$ matrix.

$$BU(0)_{j,i} = \frac{L}{L - 1 + M} \tag{5}$$

$$TD(0)_{i,j} > \frac{B_1 - 1}{D_1} \tag{6}$$

Next, the network takes an input vector I and apply Eq. 7 to generate the activities X_1, where at this moment, $G = 1$ and V is inactive. It then further generates S, passes it through the BU weights, and generates the vector T.

$$x(1)_{1i} = \frac{I_i}{1 + A_1(I_i + B_1) + C_1} \tag{7}$$

The network refers to the inhibition list H to determine which element in T could join the competition in the F_2 B. After the competition, in which the network applies the $f(X_2)$ function, an output vector U is generated, and its length (or magnitude) should be just one if there is a winner. The activation of F_2 layer is similar to F_1 layer as is shown in Eq.8, where $g(\cdot)$ is the output of other units of the F_2 layer.

$$\dot{x}_{2j} = -x_{2j} + (1 - A_2 x_{2j})(D_2 T_j + g(x_{2j})) - (B_2 + C_2 x_{2j}) \sum_{k \neq j} g(x_{2k}) \tag{8}$$

According to the book, the activities from other units are suppressed to zero, and all the other constants are meant to enhance the activities. Hence the above equantion can be simply to Eq.9, where α and β are the join effect of other terms. For that, the result of a unit's activity of F_2 layer depend on T monotonously. Thus, we can simply the process of $U = f(X_2)$ to Eq. 2.

$$\dot{x}_{2j} = -x_{2j} + (\alpha T_k + \beta) \tag{9}$$

Once the valid vector $U(t)$ occurs, signal G is inactive, and the next activation $X(t+1)_1$ on F_1 A should take the input vector $V(t)$ into account. This results in Eq. 10, where t indicates the last time frame.

$$x(t+1)_{1i} = \frac{I_i + D_1 V(t)_i - B_1}{1 + A_1(I_i + D_1 V(t)_i) + C_1} \tag{10}$$

Then, a new X_1^* and S^* are generated. The network further sends the S^* to the orienting subsystem to determine if the network learns the pattern well and keeps the consistency between the top-down and bottom-up memory. This process is done by comparing if the $|S|/|I| >= \rho$, where ρ is the vigilance parameter between 0 to 1. If the memories' magnitude mismatch $|S|/|I|$ is lower than the ρ, this indicates that the network did not memorize a similar pattern before. It then modifies the inhibition list by adding the winner unit of F_2 and starts the next round of the searching process. If $|S|/|I|$ is greater than the ρ, the network stops the searching and updates both weight matrixes to memorize the new input pattern.

Once the memorization is done, the network reset the value of the two activities vecotrs and the inhibition list to their initial state.

C. *Extension of F_2 Layer*

The book does not discuss what the network would do when none of the existing units matches the input pattern. In this report, we solve this by extending the top-down weight TD with the new input pattern I on its last column as a new unit of F_2 layer. And then, we add a new row at the end of the bottom-up weight BU and initialize this row with Eq. 5. After that, we inhibit all previous units but the new unit and run the network from the beginning. This could finish the extension and adjust the weight of the new pattern directly.

III. Experiments and Analysis

To evaluate the ART I network, we set up two sets of experiments. The first experiment set tests the network with varying ρ values (0.3 to 0.95) for four corruption levels (10%, 20%, 30%, and 40%) of inputs. The second experiment set tests the network with varying corruption (10% to 40%) levels on the inputs for three typical ρ values (0.33, 0.51, and 0.9) we observed in the first set of experiments. For different ρ values, the network memorize different number of valid patterns as Table. I.

TABLE I: The pattern memorization capacity (N) of the ART I network with different range of ρ values

ρ	0.3 ~0.38	0.39 ~0.48	0.4 ~0.58	0.6	0.61 ~0.74	0.76	0.77 ~0.82	0.83 ~0.84	0.85 ~0.88	0.89 ~0.91	0.92 ~0.93	0.94 ~0.95
N	2	3	4	5	6	7	8	9	11	12	13	15

A. *Varying ρ Values*

For the first experiment set, we set up four experiments. Each experiment is conducted with 20,000 corrupted input samples containing 1,000 samples for each pattern. Moreover, the ρ value is varied from 0.3 to 0.95 with a 0.01 interval.

Fig. 3 shows the experiment results on 10% corruption input samples. We can observe that near $\rho = 0.33$, even though the accuracy is almost 100%, the other scores drop significantly to under 70%. The low performance on precision, recall, and F1 score indicates that the accuracy score needs to be more accurate, and the network makes lots of mistakes while classifying the true patterns. A performance of around 65% precision indicates that when classifying particular patterns as their correct pattern, only 65% of them are labelled correctly. A performance of around 65% recall indicates that when classifying particular patterns, only 65% of them are labelled correctly. Near $\rho = 0.51$, all scores drop under 70% to 80%. Near $\rho = 0.9$, all scores remain stable and consistent around 90%. Fig. 3 shows the experiment results on 20% corruption input samples. Its performance curve is similar to 10% corruption but lower than the average of the 10% corruption experiments.

By observing Fig. 5 and Fig. 6, which are the experiments on 30% and 40% corruption input samples, we can tell that around $\rho = 0.31$ to $\rho = 0.38$, the misleading gap between accuracy and other scores is even larger.

B. *Varying Corruption Level*

For the second experiment set, we set up three experiments. We then pick up three ρ values from the previous experiment set, namely, 0.33, 0.51, and 0.9, for further experiments on how they perform on different levels of corrupted samples. Each experiment is conducted with 20,000 corrupted input samples containing 1,000 samples for each pattern. Moreover, the corruption level is varied from 0.1 to 0.4 with a 0.01 interval.

For the experiment with $\rho = 0.33$ value, as Fig. 7 is shown, the network performs misleading accuracy scores where the samples have around 13% and 36% corruption levels. However, around 20% corruption level, the performance is stable and not misleading as consistent as the experiment we did for Fig. 3. For the experiment with $\rho = 0.51$ value, as Fig. 8 is shown, the network performs stably but with lower scores. The network performs stably and with good scores for the experiment with $\rho = 0.9$ value, as Fig. 9 is shown.

Combined with Table. I, this experiment series indicates that when the network memorizes fewer patterns, the network will still be confused by certain levels of corrupted samples. While high ρ values enable the high capacity of the network's memory, the network performs well and is stable without any confusion.

IV. Conclusion

In conclusion, we build the ART I networks to memorize the given patterns and further explore their performance with different ρ values for difference levels of corrupted samples. We observe that with low ρ values, the capacity of the network's memory is small, and this cause the unstable classifying ability and confused by the corrupted samples. However, higher ρ values allow the network to memorize more patterns and can distinguish different patterns from the corrupted samples accurately.

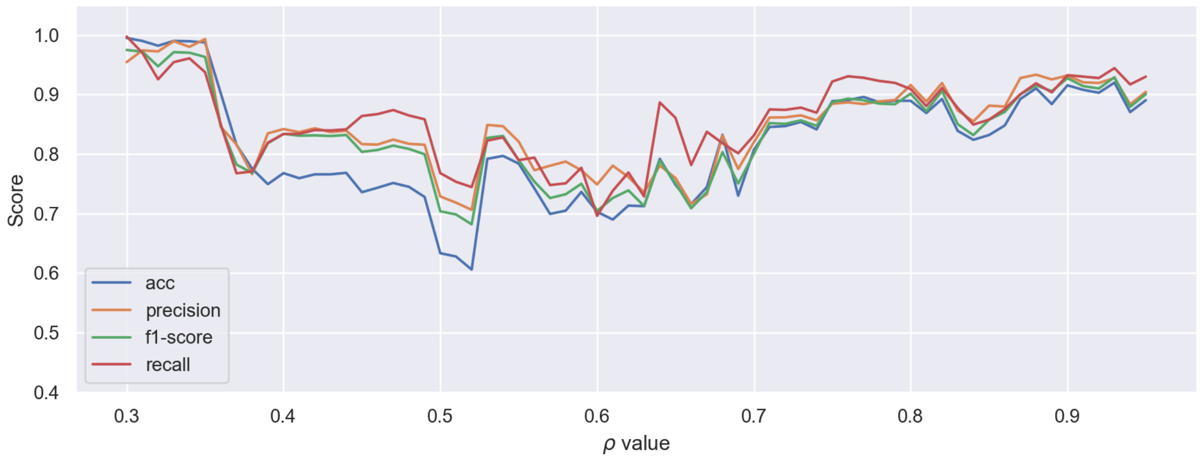


Fig. 3: Performance with different ρ value from 0.3 to 0.9 and fixed 10% corruption level input samples.

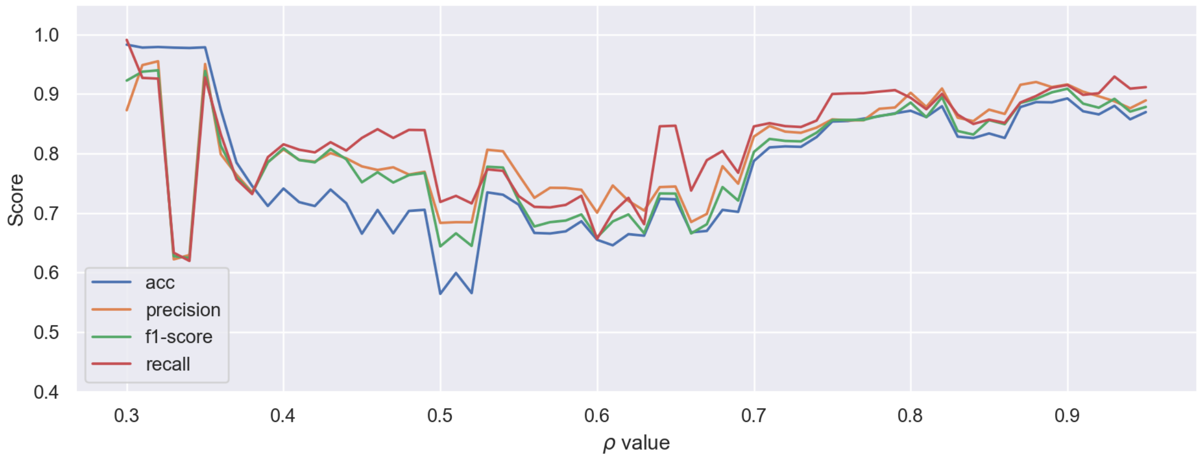


Fig. 4: Performance with different ρ value from 0.3 to 0.9 and fixed 20% corruption level input samples.

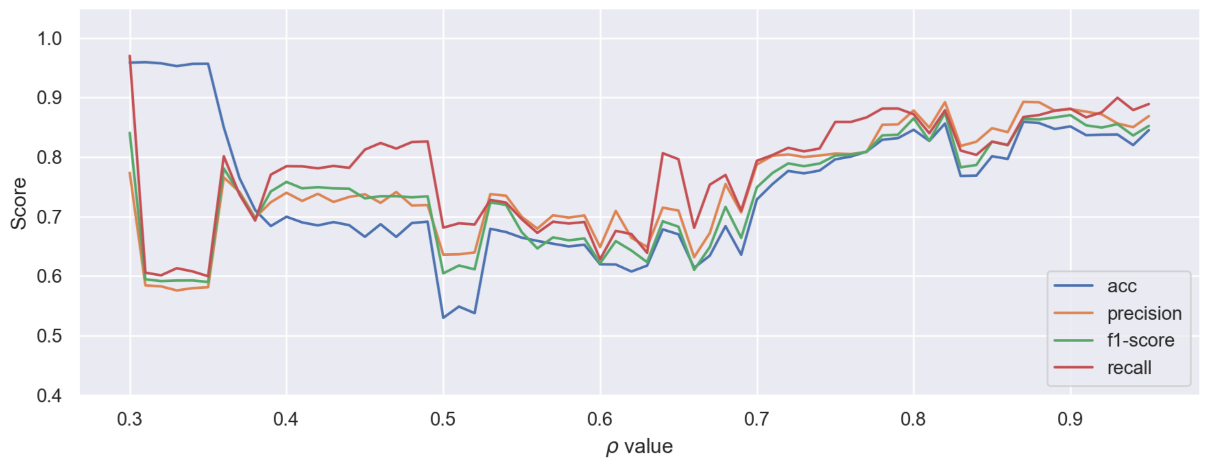


Fig. 5: Performance with different ρ value from 0.3 to 0.9 and fixed 30% corruption level input samples.

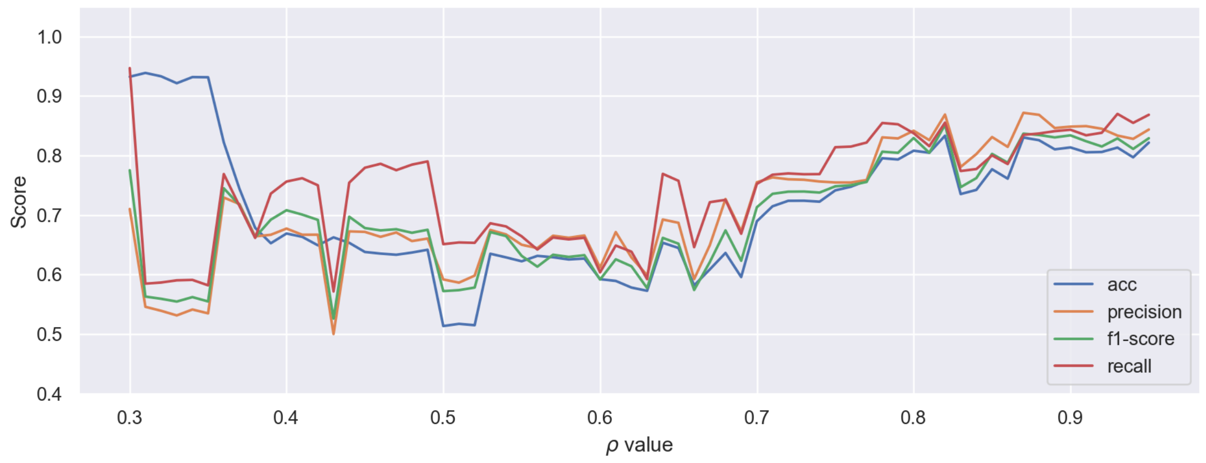


Fig. 6: Performance with different ρ value from 0.3 to 0.9 and fixed 40% corruption level input samples.

REFERENCES

[1] J. A. Freeman and D. M. Skapura, *Neural networks: algorithms, applications, and programming techniques*. Addison Wesley Longman Publishing Co., Inc., 1991.

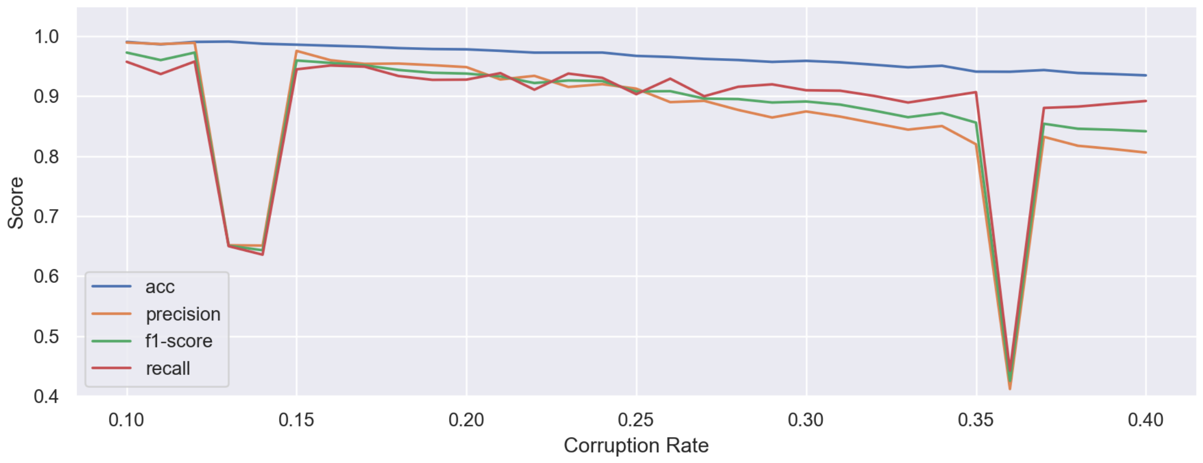


Fig. 7: Performance with fixed 0.33 ρ value and different corruption level input samples from 10% to 40%.

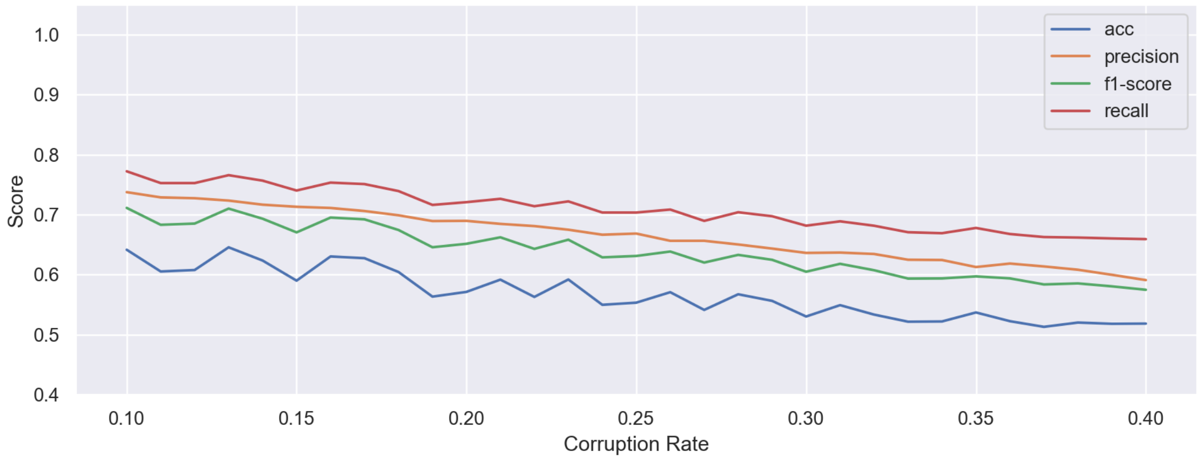


Fig. 8: Performance with fixed 0.51 ρ value and different corruption level input samples from 10% to 40%.

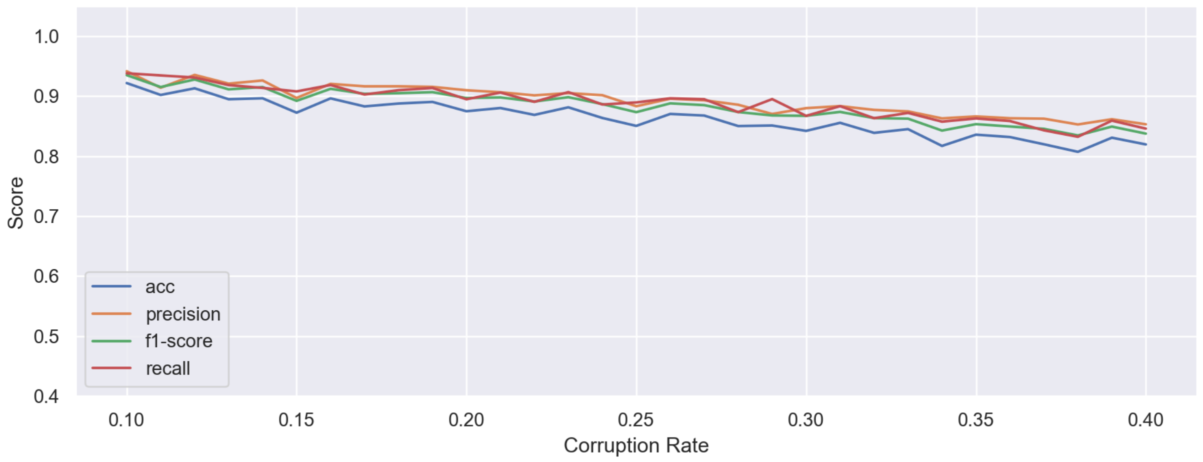


Fig. 9: Performance with fixed 0.9 ρ value and different corruption level input samples from 10% to 40%.