Cloud Implementation of project:

Noise Robustness Investigation of Hierarchical Temporal Memory's Spatial Pooler

Master Information Technology
Cloud Computing

Sang Nguyen
1185021
phuocsangnguyen97@gmail.com

Duy Nguyen
1067783
ngthanhduy7@gmail.com

***Abstract*—This report provide information about the cloud implementation of the a previously finished project, which is named "Noise Robustness Investigation of Hierarchical Temporal Memory's Spatial Pooler". Specifically, the original project includes source code and result of the experiment, which is used to demonstrate the noise robustness of the Hierarchical Temporal Memory's Spatial Pooler against Additive Gaussian Noise. In this report, the detail about the cloud extension of the mentioned project is discussed. Generally, cloud-related functions will be added to the old code as well as the deployment of the experiment will be migrated to a cloud environment.**

Keywords—Hierarchical Temporal Memory, Spatial Pooler, noise robustness, additive Gaussian noise, cloud migration

I. Introduction

This project is built up from a previous project, which investigates the noise robustness of the Hierarchical Temporal Memory (HTM)'s Spatial Pooler.

About the previous project: this project is implemented based on theory of HTM, which in turn is derived from the "Thousand Brains Theory of Intelligence" introduced by Jeff Hawkins [1]. The theory is aimed to understand and to explain how human's brain works and, from that, HTM is the framework to computationally mimic the ability of the brain from a functional point of view. The basic structure of a HTM system contains 3 main blocks: Encoder, Spatial Pooler (SP) and Temporal Memory (TM). The main function of the Encoder is to translate different types of perception inputs into a final unified representation, which could be understood by the SP. The Spatial Pooler then processes and picks up important semantic meanings of the translated inputs in order to ensure a robust representation of these inputs by means of Sparse Distributed Representation. Finally, the Temporal Memory learns and tries to predict the pattern of the sequential inputs [2]. In this previous project, we only focused on Spatial Pooler's noise robustness against Additive Gaussian Noise (AGN). Specifically, we implemented the source code to configure the SP with common parameters, train the SP with noise-free input and then compare the outputs of the SP with respect to noise-free and noisy inputs.

In this project, the cloud implementation is carried out based on the afore-mentioned project. Firstly, the old project will be migrated to a newly created project, the dependencies for old and new code will be recovered and added. Secondly, new cloud-related functions will be added, for example: function to initiate resource on the cloud, function to retrieve and upload files to the cloud, function to trigger remotely the experiment on-demand,… Finally, the whole project will be packaged as a container image, which is uploaded to a public repository, and be deployed to the cloud.

The goal of this project is to migrate the previous old project, which runs locally, to a cloud environment. This helps to improve performance as well as accessibility of the experiment.

This report includes description about the implementation of the cloud migration as well as some test result demonstrating functionality of the experiment. In this experiment, the cloud service provider is Microsoft Azure and the container platform is Docker.

II. Specification of the original project

The following information is quoted directly from our previous project [3].

A. Overview

All experiments in this paper are conducted with .NET® implementation of the Scalar Encoder and the Spatial Pooler in [4].

In this experiment, the tested HTM module includes two components: Scalar Encoder and Spatial Pooler.

The function of the Scalar Encoder is auxiliary: to encode scalar values into proper bit arrays. The function of Spatial Pooler is the focus of this paper: noise robustness. When the Spatial Pooler is said to "be trained" or to "be used to infer a data set", it means that the data sets are first encoded by the Scalar Encoder (with the specified settings in this paper) and then fed to the Spatial Pooler.

The experiment includes 2 phases:

- Training: The Spatial Pooler is trained with a simple noise-free data set.
- Testing: The trained Spatial Pooler is used to infer noisy versions of the noise-free data set.

The outputs of the Spatial Pooler from these two phases are then compared. From that, the influence of Gaussian noise on the Spatial Pooler output could be quantitatively observed.

B. *Scalar Encoder's Parameter*

Table 1. Scalar Encoder's Settings

Parameter	Value
W	65
N	465
MinVal	-20.0
MaxVal	20.0
Periodic	false
ClipInput	true
Offset	108

(All other parameters are set to default values as in [4])

According to data in Table 2 and the Scalar Encoder's implementation in [4], it could be calculated that the *resolution* of the Scalar Encoder in this paper is 0.1.

C. *Spatial Pooler's Parameter*

Table 2. Spatial Pooler's parameters

Parameter	Value
inputDimensions	445
comlumnsDimension	2048
potentialRadius	-1
potentialPct	1
globalInhibition	true
numActiveColumnsPerInhArea	0.02*2048 (2%)
stimulusThreshold	0.5
synPermInactiveDec	0.008
synPermActiveInc	0.01
synPermConnected	0.1
dutyCyclePeriod	100
maxBoost	10

(All other parameters are set to default values as in [4])

The number of learning iteration is 20.

D. *Generating noise-free data set*

The following periodic function is used to generate noise-free input data set:

$$f(x) = 10 \cdot \cos(0.01\pi \cdot x) \cdot \cos(0.05\pi \cdot x) \quad (1)$$

The above function will be sampled at integer-values of x from 0 to 199 to generate 200 samples. These values are saved in CSV format and is used as input value.

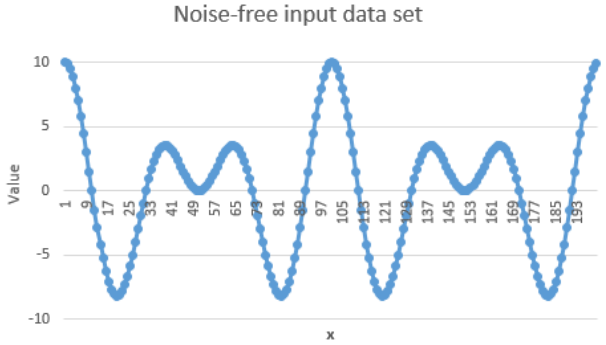


Figure 1. Noise-free data set

E. *Generating noisy data sets*

1) *Noise type*

Gaussian noise is utilized in this paper. The random variable, which is used to create noise in the experiment, follows Gaussian distribution.

Consequently, the random variable can be characterized by *mean* and *standard deviation. Mean* is the value that the random variable would most likely obtain. *Standard deviation* specifies the range around *mean*, within which the random variable would obtain the value with probability 68%. Roughly speaking, *standard deviation* shows how much the random variable fluctuates around the *mean* value.

For example, if a random variable follows Gaussian distribution with *mean* 0 and *standard deviation* 1, this random variable would most likely obtain value 0 and it would obtain values in the interval from -1 to 1 for 68% of the times.

2) *Generating noisy data sets from noise-free data set*

To generate the above mentioned type of random variable, the following function from Microsoft (MS) Excel® is used:

NORM.INV(RAND(), m, s)

This function helps to generate a random variable with *mean m* and *standard deviation s*. This random variable is added to every sample in the noise-free data set. The resulting data set is the noisy version of the noise-free data set.

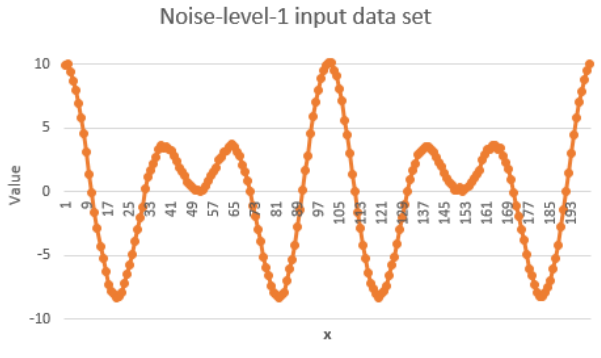
Different values of *s* are used to create different level of noisy data set.

In this paper, noise level is defined to be the ratio between standard deviation s and input resolution.

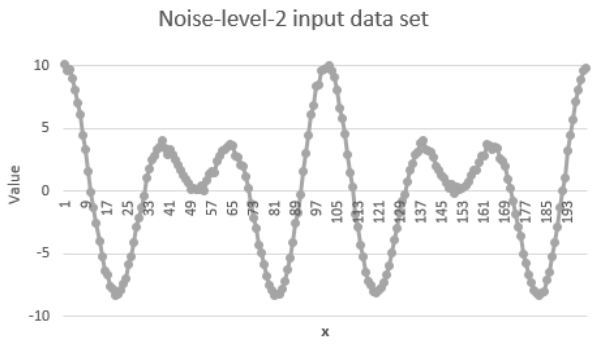
The reason why it is necessary to define the *noise level* this way is that this value can capture the relative amount of value fluctuation in the noisy input data. Systems with finer solution tend to be more susceptible to small noise.

In this paper's experiment, there are in total 10 noisy input data sets. Their *noise level* vary from 1 to 10.

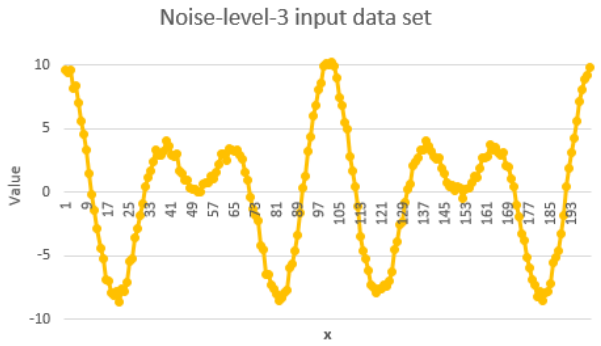
Figure 2 shows some noisy input data sets with additive Gaussian noise variable, whose mean is 0 and standard deviation is 0.1, 0.2 and 0.3 respectively. Different standard deviations *s* result in different levels of noise. The *noise level* of the data set in Figure 2 is 1, 2 and 3.



a)



b)



c)

Figure 2. Noisy input data sets. Only 3 out of 10 utilized noisy sets are shown. All 10 input data sets are available at [3].

F. *Comparison*

A measure of similarity, which is somewhat similar to "Hamming distance", is utilized in this experiment. This measuring function is already implemented in "NeoCortexApi.Akka" solution in [seCloud]:

```
public static double GetHammingDistance(int[] originArray, int[] comparingArray, bool countNoneZerosOnly = false)
```

Code snippet 1. Function to calculate "Hamming distance" from [4]

Specifically, this function takes two arrays as inputs and returns percentage of identical elements between these two arrays as output. In this paper, this percentage measure is named "Hamming distance" (however, it should not be confused with the mathematical Hamming distance). In our experiments, the arrays to be compared are bit arrays and only active bits are taken into account (`countNoneZerosOnly = true`).

III. Cloud Implementation

A. *Overview*

The operation of the cloud-migrated project could be summarized as followed:

- The user will be provided a queue. When the user wants to run the experiment, he will need to send an experiment request message to this queue in the specified format.
- The queue will be constantly checked every 3 seconds for new request. When the request arrives, the queue will trigger the execution of the experiment. Only one experiment instance can run at a time. Subsequent request will be processed sequentially in arrived-time order.
- When the experiment is executed. The program will first investigated the necessary input files, which are specified by user in the request message, and download these inputs from the cloud.
- Next the experiment is carried out with the downloaded input files.
- The output files will be uploaded to a specified cloud. The detail about the current experiment instance will also be uploaded to the cloud.
- The program will go back to the initial state to process the next request or wait for the new request message.

B. *Application's general settings*

The settings for most important parameters of the project's program is contained in the file "appsettings.json".

```
{
  "Logging": {
    "IncludeScopes": false,
    "LogLevel": {
      "Default": "Debug",
      "System": "Information",
      "Microsoft": "Information"
    }
  },

  "MyConfig": {
    "GroupId": "prof",
    "StorageConnectionString" : "*** hidden-due-to-security-reason ***",
    "TrainingContainer": "vngroup-training-files",
    "ResultContainer": "vngroup-result-files",
    "ResultTable": "vngroupresultstable",
    "Queue": "vngroup-trigger-queue"
  }
}
```

The "Logging" section is used for logging and debugging.

The "MyConfig" section is used to manage project-wide parameters:

- "GroupId" : value assigned by the instructor of this course.
- "StorageConnectionString" : key value to authenticate the program with the cloud service account.
- "TrainingContainer" : name of the cloud storage containing training and input files for the experiment.
- "ResultContainer" : name of the cloud storage containing outputs from the experiment.
- "ResultTable" : name of the table storage, to which the detail about every experiment instance are summarized and uploaded.

- "Queue" : name of the triggering queue. User will send message in the specified format to this queue in order to run the experiment remotely.

C. Experiment request message

The experiment request message has the following format:

```
{
"experimentId" : "123456789",
"trainingFileRobustness" : "sinusoidal.csv",
"inputFileRobustness" : "Noisy_N-0-1_sinusoidal.csv",
"outputFileRobustness" : "output_robustness_Noisy_N-0-
1_sinusoidal.csv",
"trainingFileSpecificity" : "sequence.csv",
"inputFileSpecificity" : "noisy_sequence.csv",
"outputFileSpecificity" : "output_specificity.csv",
"name" : "Sang Nguyen Duy Nguyen",
"description" : "debugging"
}
```

Message format explanation:

- "experimentId" : identifier of experiment and should be assigned uniquely to each run of the experiment.
- "trainingFileRobustness", "trainingFileSpecificity" : name of the blob, which will be downloaded and used as the training file for the 1st and 2nd experiment respectively.
- "inputFileRobustness", "inputFileSpecificity" : name of the blob, which will be downloaded and used as the testing file for the 1st and 2nd experiment respectively.
- "outputFileRobustness", "outputFileSpecificity" : name of the blob, which will be uploaded as the output of the 1st and 2nd experiment respectively.
- "name" : Name of the person who requested this experiment.
- "description" : Description of this experiment request

D. List of input files

- Robustness test's training file: "sinusoidal.csv"
- Specificity test's training file: "sequence.csv"
- Robustness test's testing file:
    1. "Noisy_N-0-0p1_sinusoidal.csv"
    2. "Noisy_N-0-0p2_sinusoidal.csv"
    3. "Noisy_N-0-0p3_sinusoidal.csv"
    4. "Noisy_N-0-0p4_sinusoidal.csv"
    5. "Noisy_N-0-0p5_sinusoidal.csv"
    6. "Noisy_N-0-0p6_sinusoidal.csv"
    7. "Noisy_N-0-0p7_sinusoidal.csv"
    8. "Noisy_N-0-0p8_sinusoidal.csv"
    9. "Noisy_N-0-0p9_sinusoidal.csv"
    10. "Noisy_N-0-1_sinusoidal.csv"
    11. "Noisy_N-0-1p5_sinusoidal.csv"
    12. "Noisy_N-0-2_sinusoidal.csv"
    13. "Noisy_N-0-2p5_sinusoidal.csv"
    14. "Noisy_N-0-3_sinusoidal.csv"
    15. "Noisy_N-0-3p5_sinusoidal.csv"
    16. "Noisy_N-0-4_sinusoidal.csv"
- Specificity test's testing file: "noisy_sequence.csv"

E. Experiment result's Table Entity:

The result entity contains the following fields:

1. **PartitionKey** : required keyword for a table entity. This takes the value of "GroupId" key in "appsettings.json".
2. **RowKey** : required keyword for a table entity. This takes the value of "ExperimentId" key from the Azure Queue experiment request message.
3. **TimeStamp** : property of Azure Table entity.
4. **Description** : description of the experiment, provided by the person, who requested to run the experiment. This takes the value of "description" key from the Azure Queue experiment request message.
5. **DurationSecRobustness, DurationSecSpecificity** : duration of each experiment. Normally, the robustness test takes 8-10 mins and the specificity test takes about 10 mins to finish.
6. **EndTimeUtc** : the time, at which the experiment is finished.
7. **ExperimentId** : identifier of a specific run of the experiment. This is the same as RowKey. This takes the value of "ExperimentId" key from the Azure Queue experiment request message.
8. **Name** : name of the person, who requested this run of the experiment. This takes the value of "name" key from the Azure Queue experiment request message.
9. **StartTimeUtc** : the time, at which the experiment is started.
10. **inputFileUriRobustness, inputFileUriSpecificity** : URI of the Azure Blob Storage blob, which is used as the testing file for the experiment. This is the URI of the blob, whose name is specified at "inputFile" key from the Azure Queue experiment request message and whose Azure Blob Storage container is specified at "TrainingContainer" key in "appsettings.json".
11. **outputFileUriRobustness, outputFileUriSpecificity** : URI of the Azure Blob Storage blob, which is uploaded as the output file of the experiment. This is the URI of the blob, whose name is specified at "outputFile" key from the Azure Queue experiment request message and whose Azure Blob Storage container is specified at "ResultContainer" key in "appsettings.json".

12. **trainingFileUriRobustness, trainingFileUriSpecificity** : URI of the Azure Blob Storage blob, which is used as the training file for the experiment. This is the URI of the blob, whose name is specified at "trainingFile" key from the Azure Queue experiment request message and whose Azure Blob Storage container is specified at "TrainingContainer" key in "appsettings.json".

IV. Running cloud-based experiment

The project is packaged into a Docker image, which is uploaded to a public Docker image repository and could be found at [5]. User could request the experiment directly with a running instance of the project's image or could clone and run the Docker image by himself before sending a request.

V. Conclusion

The migration of the old project to the new cloud environment helps to reduce the burden on the local hardware, improve the performance as well as time effieciency of the experiment, and increase the accessibility of the experiment.

References

[1] J. Hawkins and S. Blakeslee, *On intelligence*. New York: Times Books/Henry Holt, 2008.

[2] J. Hawkins and C. Maver, "HTM Overview Chapter," in *Biological and Machine Intelligence*, Release 0.4., Numenta, 2016.

[3] S. Nguyen, D. Nguyen, "se-cloud-2019-2020" 2020. [Online]. Available: https://github.com/UniversityOfAppliedSciencesFrankfurt/se-cloud-2019-2020/tree/VNGroup/Source/MyProject/MyProject

[4] D. Dobric, "NeoCortexApi," 2019. [Online]. Available: https://github.com/ddobric/neocortexapi/.

[5] S. Nguyen, "sangnguyenfrauas / mycloudproject," Docker Hub. [Online]. Available: https://hub.docker.com/repository/docker/sangnguyenfrauas/mycloudproject. [Accessed: 22-Oct-2020].