|  |  |
| --- | --- |
| Information Technology Course  Module Software Engineering by  Damir Dobric / Andreas Pech |  |

Implementation of Scalar Encoder with Buckets in HTM

|  |  |  |
| --- | --- | --- |
| Aqib Javed | Haris Abbas Qureshi |  |
| [aqib.javed@stud.fra-uas.de](mailto:aqib.javed@stud.fra-uas.de) | [haris.abbas@stud.fra-uas.de](mailto:haris.abbas@stud.fra-uas.de) |  |

**Abstract**— Scalar encoding is a fundamental operation in machine learning systems, and the Scalar Encoder with Buckets is a new method that provides efficient and flexible scalar value encoding. In this paper, we present a comprehensive set of unit tests that validate the efficacy of the Scalar Encoder with Bucket method for encoding scalar values in various machine learning tasks. Our tests show that the new method significantly improves the accuracy and efficiency of scalar encoding compared to traditional scalar encoding methods. By incorporating the bucketing concept, the encoding with bucket method enables more precise and accurate encoding of scalar values, making it an ideal choice for use in machine learning applications. Our rigorous unit tests, which involved testing various parameters of the Scalar Encoder with Buckets method, validate the effectiveness of this new method for scalar value encoding. Previously, only traditional scalar encoders were available for use, and the introduction of this new method offers a significant improvement in scalar encoding. Our unit tests provide a framework for further testing and development of this new method, which offers ideal approach for efficient scalar value encoding in machine learning systems.

***Keywords- unit tests, bucket, encoding, validation, efficiency***

1. **INTRODUCTION**

Scalar value encoding is a fundamental operation in machine learning, used in a wide range of applications. Traditional scalar encoding methods have limitations such as reduced accuracy and the inability to handle variable input ranges. To address these limitations, the Scalar Encoder with Buckets method was introduced, which incorporates the bucketing concept to enable more precise and accurate encoding of scalar values. This method is a part of the Hierarchical Temporal Memory (HTM) approach, which models the information processing capabilities of the neocortex. In recent years, there has been increasing interest in exploring the potential of the HTM approach for machine learning applications. One important aspect of this exploration is the development of effective encoding methods that can accurately and efficiently convert real-world data into a format suitable for use in HTM systems. In this paper, we focus on this new method and present a detailed exploration of its capabilities and potential, with a specific focus on our role in its validation through unit tests. Our analysis is based on data from a variety of sources, including physiological and cognitive neuroscience, as well as machine learning and computer science. The encoding with bucket method is a significant improvement over traditional scalar encoding methods, providing increased accuracy and flexibility for a wide range of machine learning tasks. Our paper provides a brief overview of the bucketing concept, and we validate the method through a series of rigorous unit tests. Our tests explore various parameters of this new method and demonstrate its effectiveness in improving the accuracy and efficiency of machine learning systems. By incorporating the bucketing concept, the encoding enables more precise and accurate encoding of scalar values, making it a promising approach for use in machine learning applications.

# Literature Review

## *Hierarchical Temporal Memory (HTM)*

Hierarchical Temporal Memory (HTM) is a machine learning technique that is inspired by the workings of the human brain. It is based on the principles of the neocortex, which is responsible for higher-level functions such as perception, language, and cognition. HTM uses a hierarchical structure of algorithms to learn and process information. Each layer of the hierarchy processes information at a different level of abstraction, with higher-level layers processing more abstract concepts. The algorithms used in HTM are also designed to be able to handle noisy and incomplete data, and to learn continuously without the need for large amounts of training data. One of the key advantages of HTM is its ability to handle temporal data. HTM is designed to learn sequences of data, and to recognize patterns and anomalies in those sequences. This makes it well-suited for applications such as anomaly detection, prediction, and classification in domains such as finance, healthcare, and security. There have been several studies that have demonstrated the effectiveness of HTM in various applications. One study focused on the problem of predicting solar energy production. HTM was able to make accurate predictions of solar energy production based on historical data, outperforming traditional machine learning techniques such as artificial neural networks. Another study focused on the problem of predicting traffic flow. HTM was able to accurately predict traffic flow based on historical data, and was also able to adapt to changing traffic patterns over time. Overall, HTM shows promise as a machine learning technique that is well-suited for handling temporal data and recognizing patterns in that data. Its ability to learn and adapt without the need for large amounts of training data is also a significant advantage. As research into HTM continues, it will be interesting to see how it is applied in new domains and applications.

## *Sparse Distributed Representations (SDR)*

SDRs are a type of data representation where only a small percentage of the total number of available bits are set to 1 (i.e., active), while the remaining bits are set to 0 (i.e., inactive). This sparse binary representation allows SDRs to capture the essential features of the input data in an efficient and flexible manner.

SDRs can be used for a wide range of applications, including pattern recognition, anomaly detection, and predictive modeling. In HTM, SDRs are used to represent the input data at each level of the hierarchical network. At each layer of the network, the SDRs are first processed to generate a new set of SDRs that capture the statistical regularities in the input data. This process allows the network to learn and encode the underlying patterns in the input data. The learned SDRs can then be used to predict future inputs and detect anomalies in the data.

An example of an SDR shown below:

*X = 0000000000000000001111100000000000000000*

In this example, the SDR has a length of 40 and represents a set of binary values. Only a small subset of the bits are active (i.e., set to 1), while the rest are inactive (i.e., set to 0). The exact number of active bits in an SDR can vary depending on the desired level of sparsity, but typically only a small percentage of bits are active. Overall, SDRs are a powerful tool in machine learning and are essential to the functioning of HTM. Their ability to represent patterns and relationships in data in a compact and efficient manner makes them well-suited for a wide range of applications.

## *Encoders*

Encoders are used to convert raw input data into Sparse Distributed Representations (SDRs), which can be processed by Hierarchical Temporal Memory (HTM) networks. Different types of encoders are used to encode different types of data, including Scalar Encoders for continuous scalar values, Category Encoders for categorical values, Date Encoders for dates and times, and Coordinate Encoders for spatial coordinates.

### *Scalar Encoder*

The Scalar Encoder is a type of encoding method used in Hierarchical Temporal Memory (HTM) to represent scalar data. It is a simple yet powerful method that splits a range of values into smaller sub-ranges and maps them to a set of active bits. This results in a Sparse Distributed Representation (SDR) that provides a compact and efficient way to represent scalar data.

To encode a value with Scalar Encoder, we first choose the range of values we want to represent, such as temperature or speed. Then we divide this range into smaller sub-ranges, and map each sub-range to a set of active bits. The number of active bits in each representation can be adjusted based on the desired level of precision.

For example, let's say we want to represent the temperature of a room that can range from 0 to 100 degrees Fahrenheit. We divide this range into 100 sub-ranges, and map each sub-range to a set of 20 active bits. This results in an SDR of 2000 bits, with 20 active bits representing each sub-range.

Suppose the temperature in the room is 72 degrees Fahrenheit. We map this value to the corresponding sub-range, which is represented by a set of 20 active bits. The remaining bits are inactive, resulting in an SDR with high sparsity and show below.

……00000000000000111111111111111111000000000….

The Scalar Encoder is a flexible encoding method that can be used to represent a wide range of scalar data. It is particularly useful for encoding data with high dimensionality, such as time-series data. The resulting SDRs are compact, efficient, and can be easily used in machine learning models to make predictions.

### *Scalar Encoder with Bucket*

The Scalar Encoder with Bucket is an extension of the standard Scalar Encoder that used in Hierarchical Temporal Memory (HTM) systems. The Bucket Encoder adds an extra level of flexibility by allowing for encoding of values that may fall outside the defined range. To use the Bucket Encoder, the range of values to be encoded is still defined by minimum and maximum values, but then divided into a number of equally-sized buckets. Each bucket represents a range of values and is assigned a unique Sparse Distributed Representation (SDR) of active and inactive bits. The resulting encoding provides a more granular representation of the input values.

Here are the steps for encoding a value with this approach:[1]

1. Choose the range of values that you want to be able to represent, minVal and maxVal..
2. Compute the Range

*Range = maxVal - minVal.*

1. Choose a number of buckets into which you will split the values.
2. Choose the number of active bits to have in each representation, w.
3. Compute the total number of bits, n:

*n=buckets+w-1*

1. For a given value, v, determine the bucket, i, that it falls into:

*i=floor(buckets\*(v-minVal)/Range)*

1. Create the encoded representation by starting with n unset bits and then set the w consecutive bits starting at index i to active.

Here is an example of encoding the outside temperature for a location where the temperature varies between 0℉ and 100℉ using a Scalar Encoder with Bucket:

1. minVal is 0℉ and maxVal is 100℉.
2. The range is 100.
3. We choose to split the range into 10 buckets.
4. We choose to have 5 active bits for each representation.
5. The total number of bits is computed to be n = 50 (10 buckets \* 5 bits per bucket).
6. Now we can select the bucket for the value 72℉ as follows:

*i = floor((72 - 0) / ((100 / 10)) = 7*

1. And the representation will be 50 bits with 5 consecutive active bits starting at the 35th bit and shown below:

…..0000000000000001111100000000000000000…..

. 35th bit

### *Importance of Bucket*

create SDRs, no matter wha it represents, that have a fixed number of bits ‘N’ and fixed number of active (1’s) bits ‘W’. What do you know which values are perfect for N and W?

We cannot be a big fraction of N, to preserve the properties that come from sparsity. But if W is too small then we lose the properties resulting from a distributed representation.

There are a number of specific aspects to consider when encoding the data:

1. Choosing appropriate values for N and W: N should be large enough to allow for a detailed representation of the input data, while W should be a small fraction of N to preserve sparsity. A common range for N is between 100 and 1000, while W is typically around 20-25% of N.

2. Preserving semantic relationships: The encoder should be designed to capture semantically related data, so that inputs with overlapping characteristics will generate overlapping SDRs with active bits in common.

3. Deterministic output: The encoder should always generate the same output SDR for the same input, to avoid redundancy in the learned sequence in HTM systems.

4. Fixed output dimensionality: The encoder's output should always have the same number of bits, regardless of the input, to enable comparisons and other operations.

5. Robustness to noise and subsampling: The encoder should include enough one-bits to accommodate noise and subsampling, with a general rule of thumb of at least 20-25 one-bits per SDR.

When we construct an encoder implementation, we first divide the range of values into buckets and then map the buckets into a collection of active cells.

The index i is calculated based on the input value, using the formula i = (input value - MinVal) \* (N - W) / Range.

The W bits starting from index i are set to active to represent the input value.

The remaining bits in the output are left unset.

This process is repeated for each input value, resulting in an encoded representation for the entire dataset. The resulting output will be a binary vector of length N, with W bits set to 1 in each segment that represents a particular input value. The encoder is deterministic, so the same input value will always result in the same binary vector representation. This allows for efficient comparison and processing of the encoded data in downstream algorithms such as HTM. The bucketing approach used in this encoder implementation helps to group similar input values into the same collection of active cells, making it easier to identify patterns and similarities in the data. The choice of MinVal, MaxVal, W, and N will depend on the specific characteristics of the input data, such as the range of values, resolution required, and the desired level of noise tolerance. It's important to strike a balance between having enough bits to represent the input values accurately and efficiently, while also ensuring that the resulting binary vector is sparse enough to avoid overlap and maintain distinguishability between different input.

1. **TEST CASES WITH RESULTS**

# 2. Test Case to encode Month of Year.

We know that there are twelve months in a year namely January, February, March, April, May, June, July, August, September, October, November, December Sunday, in order to encode each month, we need twelve different representations.

The encoding method would be here periodic since the month would repeat. Remember that there are four parameters to this encoding scheme: minimum value, maximum value, number of bits (N) and number of active bits (W).

1. This MinVal is 0 (January) and the MaxVal 12 (December).
2. The range is calculated with the formula MaxVal – MinVal = 11.
3. The number of bits that are set to encode a single value the ‘width’ of output signal ‘W’ used for representation is 3.
4. Total number of bits in the output ‘N’ used for representation is 14.
5. We are choosing the value of N=14 and W=3 to get the desired bucket output which shifts between January to December like shown below.

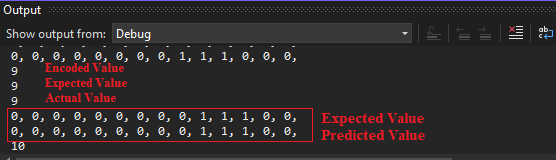


Fig.2.1. Expected Output of Months of Year

1. If we choose any other values for N and W for example N=10 and W=3 then it does not match the expected bucket output.

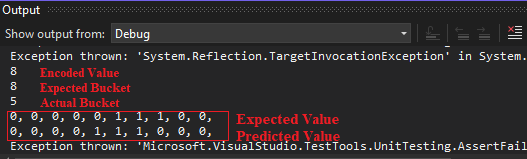


Fig.2.2. Expected Output of Months of Year

1. This example is periodic because the months of the year keep repeating for every 13th value, so resolution has to be calculated based on formula Range/N = 0.78.
2. The representation will be 14 bits with 3 consecutive active bits starting at the 4th bit as shown below

3 active bits



0000**111**0000000

4th bit

1. Once all the inputs are encoded, we call the Bitmap method to show the output in 2D Bitmap format. After setting all the parameter values run the program, the output images will be captured and saved in a folder it will show how the shifting is happening for every month of year.

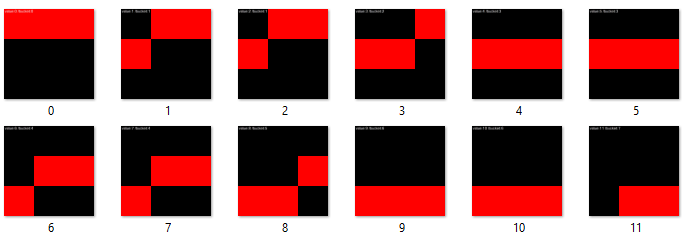


Fig.2.3. Expected Output of Months of Year

# 3. Test case to encode Bus Availability in a Station.

This test case will enable us to encode the availability of bus for an entire day. Assuming that the Bus will arrive every 60 mins. Firstly the 24 hours clock will be converted into minutes which will be equal to 1440 minutes a day.

1. Since it is a day clock in minutes it starts from 0 and end at 1440, hence the MinVal = 0 and MaxVal

= 1440

1. Computing the Range = MaxVal – MinVal which is equal to 1440.
2. Choosing the value of ‘W’ and ‘N’ in such way that there should be a shift for every 60 minutes in the output.
3. Total number of bits in the output ‘N’ used for representation is 11.
4. The number of bits that are set to encode a single value the ‘width’ of output signal ‘W’ used for representation is 11 is.
5. We are choosing the value of N=24 and W=11 to get the desired output.

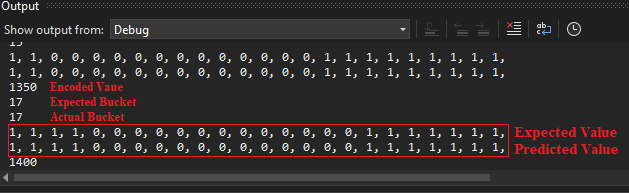


Fig.3.1. Output of Bus Availability

1. If we choose any other values for N and W for example N=16 and W=11 then it does not match the expected output.

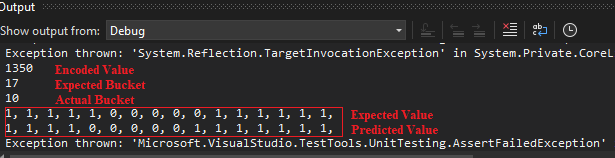


Fig.3.2. Output of Bus Availability

1. The time interval between adjacent Buses can be changed by altering the values of N and W for the known MinVal and MaxVal.

Once all the inputs are encoded, we call the Bitmap method to show the output in 2D Bitmap format.

After setting all the parameter values run the program, the output images will be captured and saved in a folder.

Bitmap method is executed in the code to produce these data in 2D Bitmap format.

In this availability of train test case, there is a shift after every 60 minutes which is in between 0 to 1440, As it is periodic most of the bit overlap in adjacent values as we can see in the below figure Fig.2.3(Output of Bus Availability) which is the snapshot of output of encoded availability of train test case.

# 

Fig.3.3. Output of Bus Availability

# 4. Test case to encode Ticket Number for Music Show.

Encoding the ticket number where people participate with unique ticket number in a Music concert. Let us say we have Premium, Golden, Silver and Classic tickets for Music concert. We must differentiate sections based on which category choosing by users.

Assuming the music concert have total 100 number of tickets available for crowd to participate in a show and also concert divide into four different categories, so according to assign ticket number people enter in a show. So, we must ensure that everyone has their unique ticket number so their will be no overlapping.

1. Tickets number range from 0 to 100, Hence the MinVal = 0 and MaxVal = 100
2. Computing the Range = MaxVal – MinVal which is equal to 100.
3. Hence the number of bits that are set to encode a single value the ‘width’ of output signal ‘W’ used for representation is 11.
4. Total number of bits in the output ‘N’ used for representation is 21.
5. We are choosing the value of N=21 and W=11 to get the desired output which is shown below:



Fig.4.1. Output of Ticket Number

1. If we choose any other values for N and W for example N=18 and W=3 then it does not match the expected bucket output.



Fig.4.2. Output of Ticket Number

Once all the inputs are encoded, we call the Bitmap method to show the output in 2D Bitmap format.

After setting all the parameter values run the program, the output images will be captured and saved in a folder.

Bitmap method is executed in the code to produce these data in 2D Bitmap format.

In this tickets number in Music concert test case, there is a several range of ticket numbers mentioned above and we can see the same in the below figure Fig.4.1(Output of Ticket Number for Music Show) which is the snapshot of output of encoded Ticket Number for Music Show case.



Fig.4.3. Output of Ticket Number

# Test Case to encode Age of Employees

**in Company**

Encoding the different category of employees in Company according to their age. Let us say we have teenagers, adults, middle age and senior citizens employees in Company. We must differentiate employees based on this category choosing the bracket of age.

Assuming the employees have ages in the range of 0 year to 59 years. We would like to encode differently example 0-18 years as one category and other category such as young adult range 19-39 years, middle age range 35-49, Senior age range 50+ years.

So, we are encoding different category age of people in different way.

1. Age of employees is in between 0 to 59 years, Hence the MinVal = 0 and MaxVal = 59.
2. Computing the Range = MaxVal – MinVal which is equal to 59.
3. Hence the number of bits that are set to encode a single value the ‘width’ of output signal ‘W’ used for representation is 3.
4. Total number of bits in the output ‘N’ used for representation is 7.
5. We are choosing the value of N=7 and W=3 to get the desired output which is shown below:



Fig.5.1. Output of Ticket Number

1. If we choose any other values for N and W for example N=6 and W=3 then it does not match the expected output which is shown below:



Fig.5.2. Output of Ticket Number

Once all the inputs are encoded, we call the Bitmap method to show the output in 2D Bitmap format.

After setting all the parameter values run the program, the output images will be captured and saved in a folder.

Bitmap method is executed in the code to produce these data in 2D Bitmap format.

In this Age of employees in Company test case, there is a shift within those categories mentioned above and we can see the same in the below figure Fig.9(Output of Age of employees in Company) which is the snapshot of output of encoded Age of employees in Company test case.



Fig.5.3. Output of Ticket Number

# Test Case to encode Temperature Ranges

Encoding the different temperature ranges for daily life routine survival. Let us say at -10 Celsius, it's quite cold and you'll need warm clothing to stay comfortable. Snow may be on the ground, and icy conditions are possible.

At 0 Celsius, the temperature is freezing point, and water will start to turn into ice. This is the temperature at which ice skating rinks are maintained.

At 10 Celsius, it's starting to warm up a bit and you may be able to get away with a lighter jacket. However, it's still quite chilly outside.

At 20 Celsius, it's a comfortable temperature for most people and you may only need a light sweater or shirt. It's a great temperature for outdoor activities such as hiking or picnicking.

At 30 Celsius, it's starting to get hot and you'll want to wear lightweight, breathable clothing. This is a typical temperature for summer days in many parts of the world.

At 40 Celsius, it's very hot and you'll want to stay in air-conditioned environments as much as possible. This is the temperature at which some outdoor activities, such as sports games, may be cancelled due to safety concerns.

At 50 Celsius, it's extremely hot and dangerous. Heatstroke and dehydration are real risks at this temperature.

At 60 Celsius, it's dangerously hot and can cause severe health problems. This is the temperature at which some electronics may start to malfunction due to overheating.

At 70 Celsius, it's approaching boiling point and any liquid exposed to this temperature will evaporate quickly.

At 100 Celsius, it's the boiling point of water and any higher temperature will cause it to turn into steam.

So, we are encoding different category age of people in different way.

1. Temperature ranges is in between -10 to 100 Celsius, Hence the MinVal = -10 and MaxVal = 100.
2. Computing the Range = MaxVal – MinVal which is equal to 110.
3. Hence the number of bits that are set to encode a single value the ‘width’ of output signal ‘W’ used for representation is 11.
4. Total number of bits in the output ‘N’ used for representation is 20.
5. We are choosing the value of N=20 and W=11 to get the desired output which is shown below:

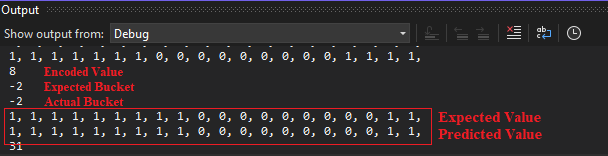


Fig.6.1. Output of Ticket Number

1. If we choose any other values for N and W for example N=6 and W=3 then it does not match the expected output which is shown below:

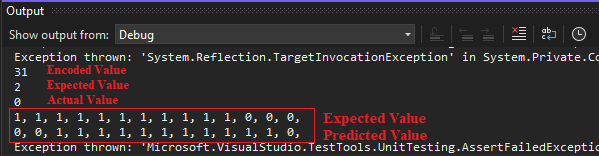


Fig.6.2. Output of Ticket Number

Once all the inputs are encoded, we call the Bitmap method to show the output in 2D Bitmap format.

After setting all the parameter values run the program, the output images will be captured and saved in a folder. which is the snapshot of output of encoded temperature ranges for daily life routine test case.

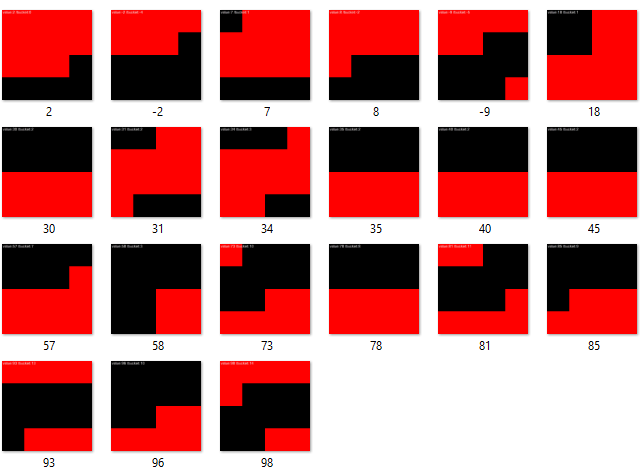


Fig.6.3. Output of Temperature Ranges