Scalar Encoder with Buckets

Names of [Team\_ScalarEncoder](https://github.com/UniversityOfAppliedSciencesFrankfurt/se-cloud-2022-2023/tree/Team_ScalarEncoder) members:

* Aqib Javed
* Haris Abbas Qureshi
* Shahab Uddin
* Saad Jamil

*Abstract*—Scalar Encoder is one of the encoding techniques and is a part of Hierarchical Temporal Memory (HTM). HTM is a machine intelligence technology which is trying to imitate the process and architecture of neocortex. The main purpose for scalar encoder is to encode numeric or floating-point value into an array of bits, where the output has 0’s with an adjacent block of 1’s. The location of the block of 1’s varies continuously depending on the input value.

* *Keywords- HTM, neocortex, numeric, array, block*

getBucketIndices(*inputData*)

Returns an array containing the sub-field bucket indices for each sub-field of the inputData. To get the associated field names for each of the buckets, call getScalarNames().

|  |  |
| --- | --- |
| Parameters: | inputData – The data from the source. This is typically a object with members. |
| Returns: | array of bucket indices |

getBucketInfo(*buckets*)

Returns a list of EncoderResult namedtuples describing the inputs for each sub-field that correspond to the bucket indices passed in ‘buckets’. To get the associated field names for each of the values, call getScalarNames().

|  |  |
| --- | --- |
| Parameters: | buckets – The list of bucket indices, one for each sub-field encoder. These bucket indices for example may have been retrieved from the getBucketIndices() call. |

w – The number of bits that are set to encode a single value - the "width" of the output signal restriction: w must be odd to avoid centering problems.

minval – The minimum value of the input signal.

maxval – The upper bound of the input signal

periodic – If true, then the input value "wraps around" such that minval = maxval For a periodic value, the input must be strictly less than maxval, otherwise maxval is a true upper bound.

There are three mutually exclusive parameters that determine the overall size of of the output. Only one of these should be specifed to the constructor:

n – The number of bits in the output. Must be greater than or equal to w radius – Two inputs separated by more than the radius have non-overlapping representations. Two inputs separated by less than the radius will in general overlap in at least some of their bits. You can think of this as the radius of the input. resolution – Two inputs separated by greater than, or equal to the resolution are guaranteed to have different representations.

Note: radius and resolution are specified w.r.t the input, not output. w is specified w.r.t. the output.

Example: day of week. w = 3 Minval = 1 (Monday) Maxval = 8 (Monday) periodic = true n = 14 [equivalently: radius = 1.5 or resolution = 0.5]

The following values would encode midnight – the start of the day monday (1) -> 11000000000001 tuesday(2) -> 01110000000000 wednesday(3) -> 00011100000000 ... sunday (7) -> 10000000000011

Since the resolution is 12 hours, we can also encode noon, as monday noon -> 11100000000000 monday midnt-> 01110000000000 tuesday noon -> 00111000000000 etc.

It may not be natural to specify "n", especially with non-periodic data. For example, consider encoding an input with a range of 1-10 (inclusive) using an output width of 5. If you specify resolution = 1, this means that inputs of 1 and 2 have different outputs, though they overlap, but 1 and 1.5 might not have different outputs. This leads to a 14-bit representation like this:

1 -> 11111000000000 (14 bits total) 2 -> 01111100000000 ... 10-> 00000000011111 [resolution = 1; n=14; radius = 5]

You could specify resolution = 0.5, which gives 1 -> 11111000... (22 bits total) 1.5 -> 011111..... 2.0 -> 0011111.... [resolution = 0.5; n=22; radius=2.5]

You could specify radius = 1, which gives 1 -> 111110000000.... (50 bits total) 2 -> 000001111100.... 3 -> 000000000011111... ... 10 -> .....000011111 [radius = 1; resolution = 0.2; n=50]

An N/M encoding can also be used to encode a binary value, where we want more than one bit to represent each state. For example, we could have: w = 5, minval = 0, maxval = 1, radius = 1 (which is equivalent to n=10) 0 -> 1111100000 1 -> 0000011111

Implementation details:

range = maxval - minval h = (w-1)/2 (half-width) resolution = radius / w n = w \* range/radius (periodic) n = w \* range/radius + 2 \* h (non-periodic)

Constructor & Destructor Documentation

|  |  |  |  |
| --- | --- | --- | --- |
| def \_\_init\_\_ | ( |  | self, |
|  |  |  | w, |
|  |  |  | minval, |
|  |  |  | maxval, |
|  |  |  | periodic = False, |
|  |  |  | n = 0, |
|  |  |  | radius = DEFAULT\_RADIUS, |
|  |  |  | resolution = DEFAULT\_RESOLUTION, |
|  |  |  | name = None, |
|  |  |  | verbosity = 0, |
|  |  |  | clipInput = False, |
|  |  |  | forced = False |
|  | ) |  |  |

w – number of bits to set in output minval – minimum input value maxval – maximum input value (input is strictly less if periodic == True)

Exactly one of n, radius, resolution must be set. "0" is a special value that means "not set".

n – number of bits in the representation (must be > w) radius – inputs separated by more than, or equal to this distance will have non-overlapping representations resolution – inputs separated by more than, or equal to this distance will have different representations

name – an optional string which will become part of the description

clipInput – if true, non-periodic inputs smaller than minval or greater than maxval will be clipped to minval/maxval

forced – if true, skip some safety checks (for compatibility reasons), default false

See class documentation for more information.

Hierarchical Temporal Memory

Encoder -

Encoder is chosen according to the type of the inputs. There are some encoders available for popular input type:

- Scalar Encoder

- Datetime Encoder

- Boolean Encoder

- Category Encoder

- Geo-Spatial Encoder

In this project we are using - Scalar Encoder

Scalar Encoder is one of the encoding techniques and is a part of Hierarchical Temporal Memory (HTM). HTM is a machine intelligence technology which is trying to imitate the process and architecture of neocortex. The main purpose for scalar encoder is to encode numeric or floating-point value into an array of bits, where the output has 0’s with an adjacent block of 1’s. The location of the block of 1’s varies continuously depending on the input value.

The scalar representation of value (e.g. for categories, this is the internal index used by the encoder). This number is consistent with what is returned by [getScalars()](https://nupic.docs.numenta.org/1.0.3/api/algorithms/encoders.html" \l "nupic.encoders.base.Encoder.getScalars" \o "nupic.encoders.base.Encoder.getScalars). This value is always an int or float, and can be used for numeric comparisons.

HTM consists of 2 different components: Spatial Pooler and Temporal Memory.

1. Spatial Pooler -

Encoder produces output to be fed into Spatial Pooler algorithm. Type of Spatial Pooler (SP) that is used in this example is the multithreaded version that utilize multicore of the machine to run the spatial pooler algorithm.

SpatialPoolerMT spatialPooler = new SpatialPoolerMT(hpa);

patialPooler.Init(memory, UnitTestHelpers.GetMemory());

1. Temporal Memory -

The output of Spatial Pooler (SDR) is used as the input of Temporal Memory.Temporal memory algorithm will then learn the temporal pattern from spatial pattern.

TemporalMemory temporalMemory = new TemporalMemory();

temporalMemory.Init(mem);