HIERARCHICAL TEMPORAL MEMORY FOR REAL-TIME ANOMALY DETECTION

by Ihor Bobak,
Lead Software Engineer at EPAM Systems

August 29, 2017

ANOMALY DETECTION

Anomaly = something that deviates from what is standard, normal or expected.
Anomaly Detection = finding patterns in data that do not conform to expected behavior.

Applications:

- Sensor networks
- Network intrusion
- Insurance / credit card fraud
- Healthcare informatics / medical diagnostics
- Industrial damage detection
- Image processing / video surveillance
- Novel topic detection in text mining

Examples:

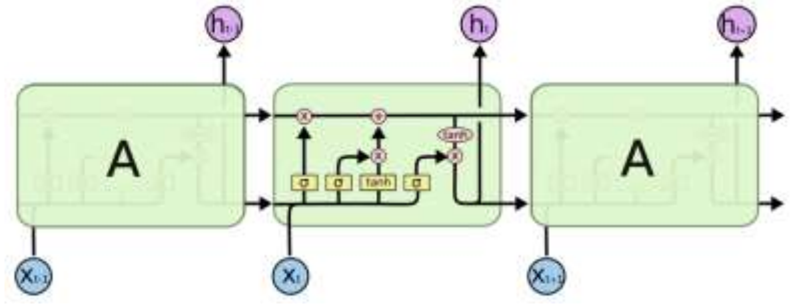
- Anomalous temperature in the refrigerator of a truck may tell us that soon it will be our or order
- Anomalous CPU utilization at a server of a cluster may mean that HDD is failing and needs a replacement.
- Anomalous traffic pattern means that someone hacked the computer (or doing this at the moment)
- Anomalous MRI image or EKG may indicate a health problem.
- Anomalous level of VOCS (volatile organic compounds) may mean that a nearby chemical plant may have a trouble.

RETROSPECTIVE

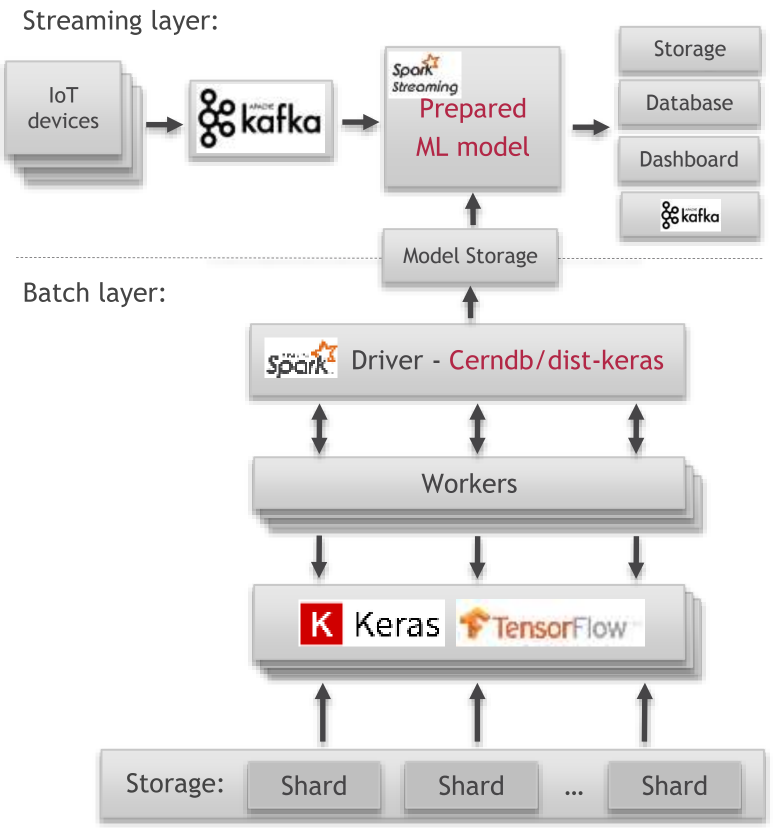
For our POC scalable anomaly detection in time series we looked at paralleling different LSTM models implemented in Keras+Tensorflow using cerndb/keras.

Drawbacks:

1) the data changes in real-time and no gaps are allowed, but the model should be re-trained (= takes time).
2) high computational intensity of LSTM (GPU is needed).

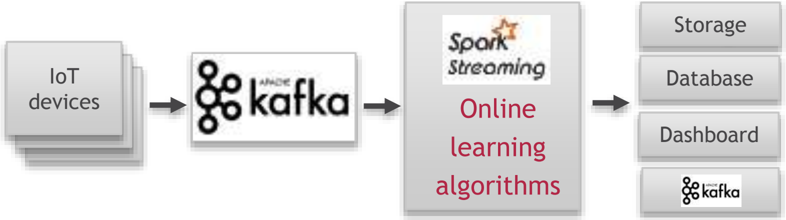


Reference: https://github.com/cerndb/dist-keras



ARCHITECTURE

But then we came to a conclusion that we need this:



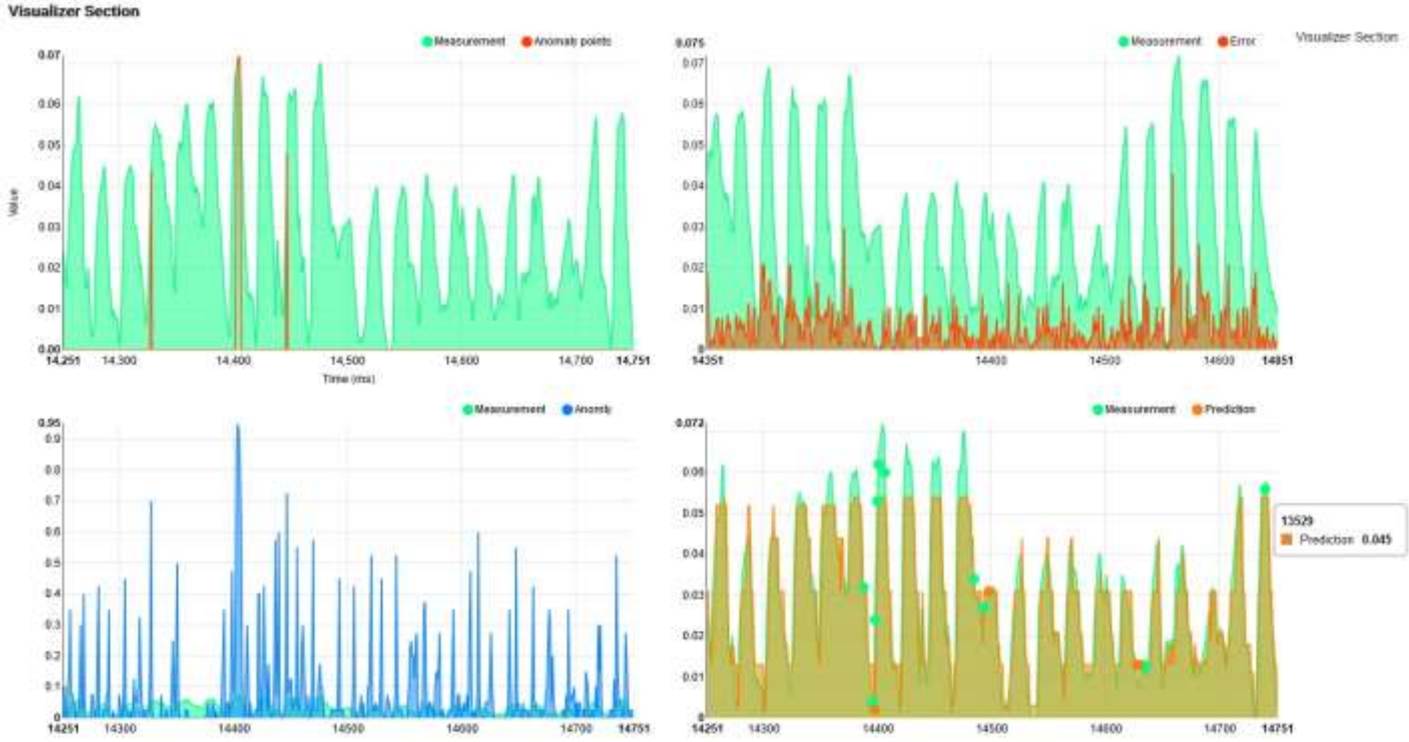
Demands:

- **REAL-TIME:** the streaming data coming in should be immediately predicted or anomalies found
- **Continuous learning:** the model should learn online as the data comes
- **Unsupervised:** no one is going to label the data. NEVER.
- **Not CPU "greedy"**
- **No hyper-parameter tuning** or other human intervention
- **Noise robustness**

THE END RESULT

This is a Zeppelin Notebook which visualizes the enriched data stream and allows to see the anomalies.

As you may notice, HTM network allows to get the high accuracy prediction and find the anomalies in data.



ANOMALY DETECTION TECHNIQUES

Research Area
- Machine Learning
- Statistics
- Information Theory
- Spectral Theory

→ **Anomaly Detection Technique** ←

Problem Characteristics
- Nature of Data
- Labels (yes/no)
- Anomaly Type
- Output

←

Application Domains
- Intrusion Detection
- Fraud Detection
- Damage Detection
- Medical

Machine Learning: neural networks (deep learning, RNN, CNN), SVM, logistic regression, decision trees/ensembles, clustering, etc.

Information Theory: calculating metrics like information gain, entropy, relative entropy, conditional entropy.

Statistical Based: ARIMA, Seasonal Hybrid ESD, estimation of parametric distribution model + apply a test (e.g. calc z-score).

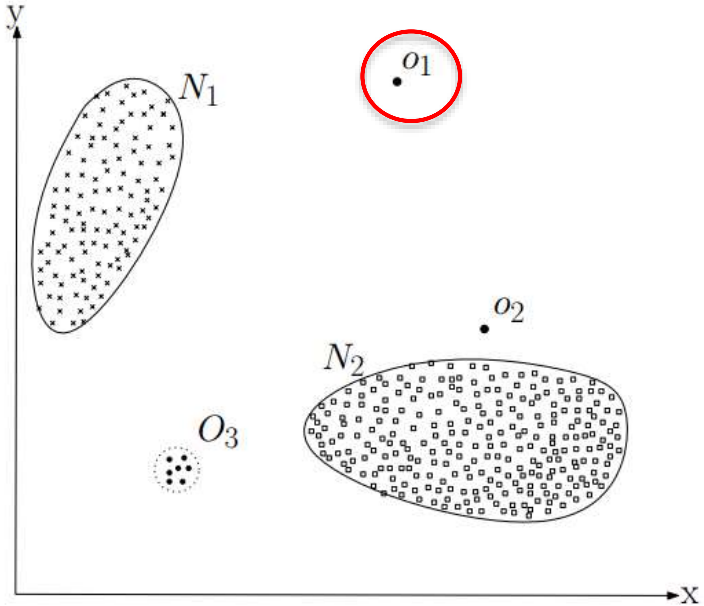
Spectral Theory: PCA with assumption that anomalies and normal data significantly differ.

TYPES OF ANOMALIES

Non-temporal anomalies – when we have some observations happening irregularly.

Example: credit card fraud detection.

Such tasks are usually solved using clustering methods, and these types of anomalies are outside the scope of our presentation.



TYPES OF ANOMALIES

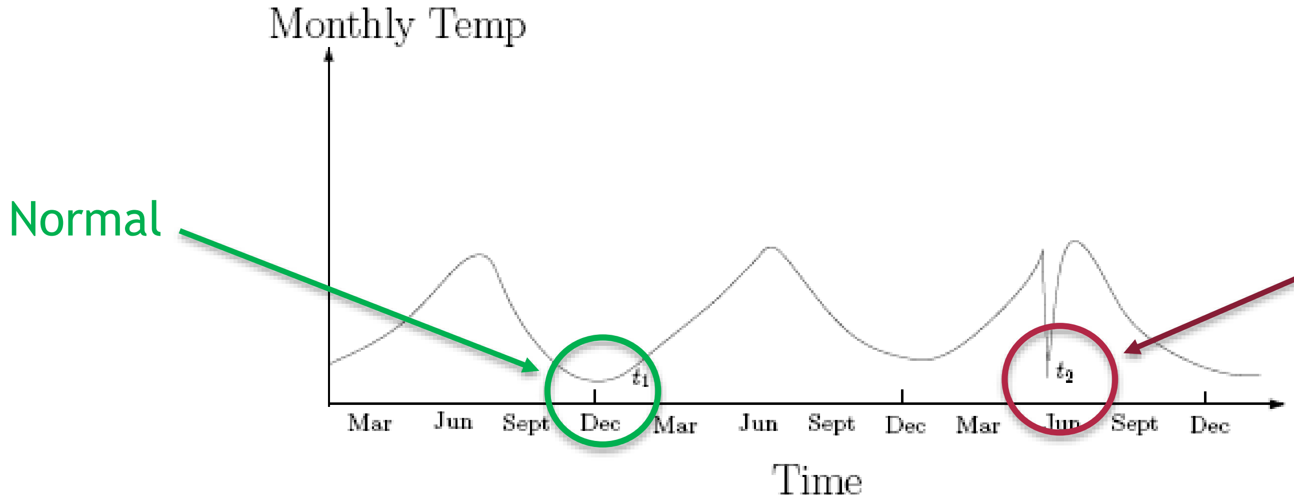
Point anomaly – when an instance of data is very much different from the rest of data.

Example: a spike of CPU activity on a server.

TYPES OF ANOMALIES

Contextual anomaly – when an observation is unusual in a certain context but is NOT unusual in another context.

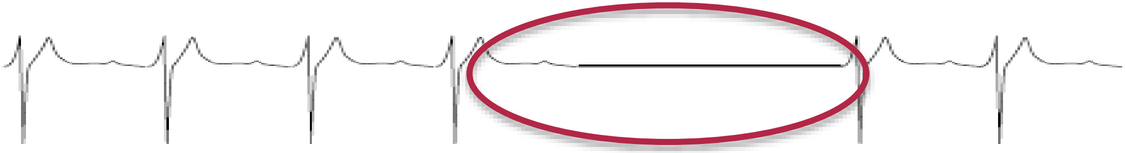
Example: air temperature:



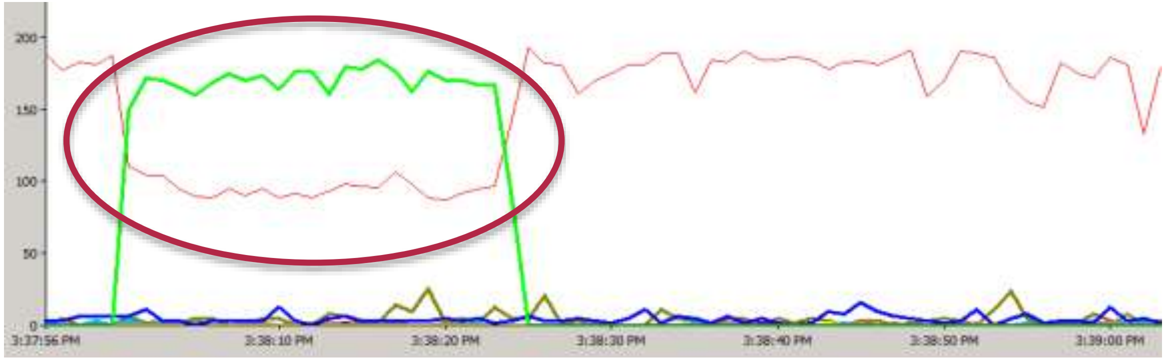
TYPES OF ANOMALIES

Collective anomaly – when a collection of related data instances are anomalous. Requires relationship among data.

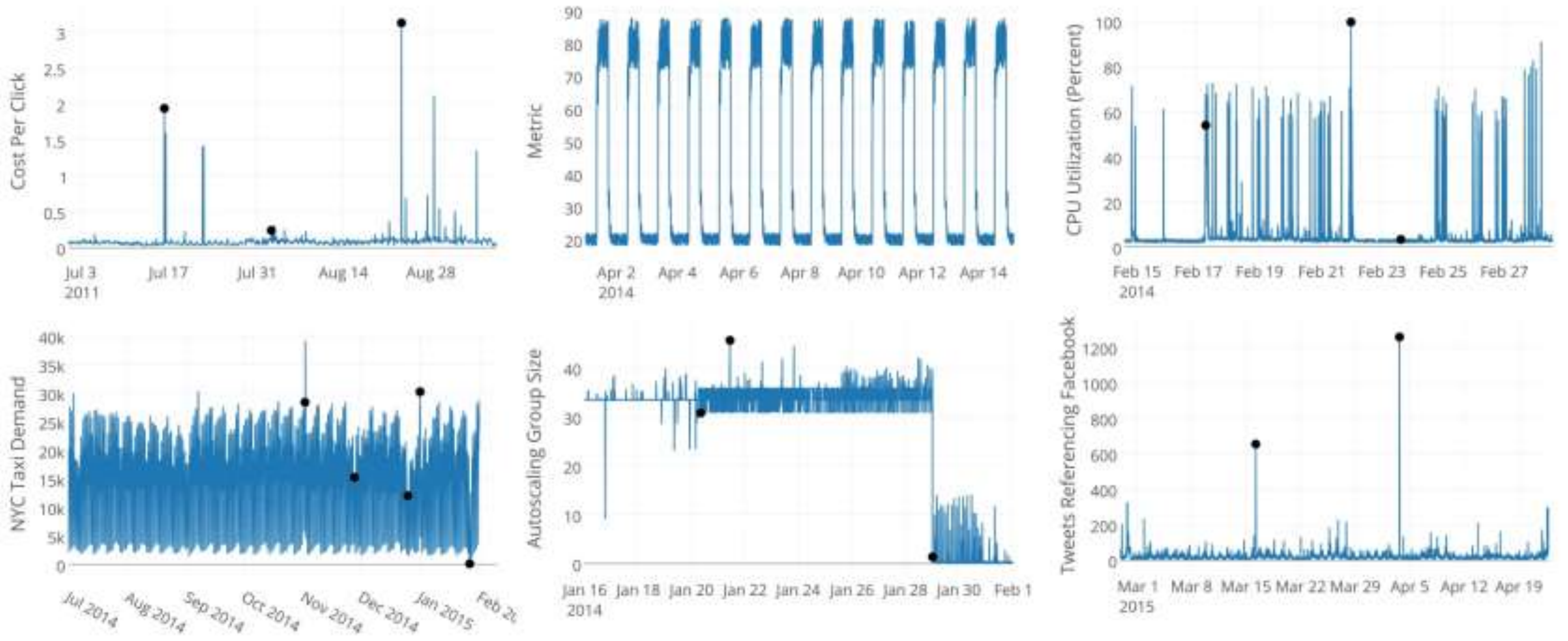
EKG



CPU utilization



ANOMALIES: EXAMPLES

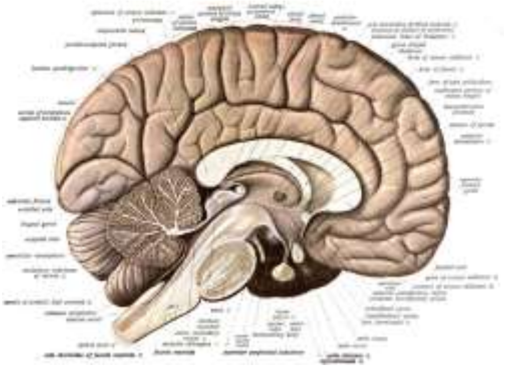


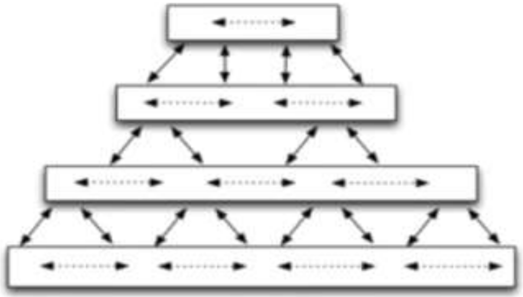
1) click-through prices for online advertisement 2) an artificial stream with some noise but no anomalies 3) AWS Cloudwatch CPU utilization data 4) hourly demand for New York City taxis 5) autoscaling group data for a server cluster 6) a stream of tweet volumes related to FB stock

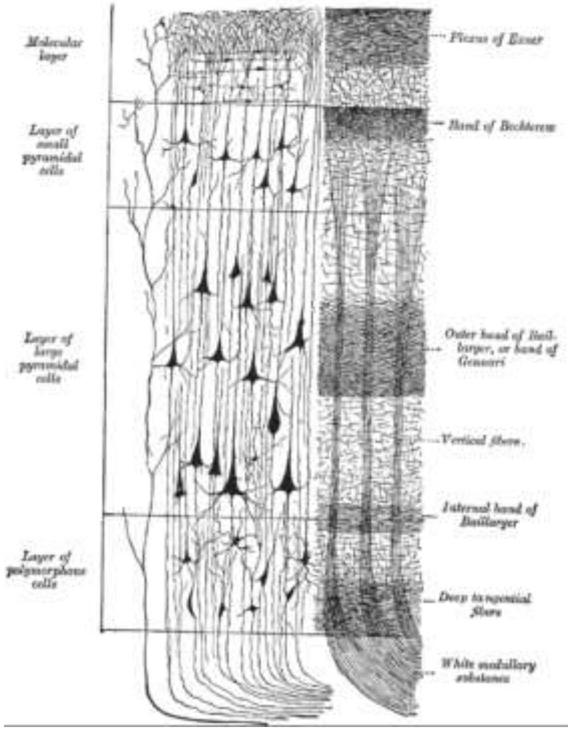
NEOCORTEX

The HTM (Hierarchical Temporal Memory) is based on the concepts of how the neocortex works:

- Neocortex is divided into regions, connected with each other. Regions are logically linked into hierarchical structure.
- Raw sensory data come from eyes/ears/etc. to lower levels of the hierarchy, processed and passed to higher levels. Inputs from all senses are essentially the same.
- At any point of time a neuron can be on or off. About 2% of neurons are “ON”.







WHO IS BEHING THIS?

Following Jeff Hawkings (the creator of Numenta), ANNs appeared almost 50 years ago. Since then neurobiology moved forward a lot, but ANNs did not change at all. Despite the name of neural netwoks, ANNs do not have much to do with real neurons at all. Yes, they evolved into convolutional neural networks, RNN, deep learning, but they are not close to biological models.

Jeffrey Hawkins (June 1, 1957) is the American founder of Palm Computing (where he invented the PalmPilot) and Handspring (where he invented the Treo). He has since turned to work on neuroscience full-time, founded the Redwood Center for Theoretical Neuroscience (formerly the Redwood Neuroscience Institute) in 2002, founded Numenta in 2005 and published *On Intelligence* describing his memory-prediction framework theory of the brain. In 2003 he was elected as a member of the National Academy of Engineering "for the creation of the hand-held computing paradigm and the creation of the first commercially successful example of a hand-held computing device.“

https://en.wikipedia.org/wiki/Jeff_Hawkins

https://news.ycombinator.com/item?id=8544561

HTM

- Encoders take data (numbers, dates, temperature, GPS coordinates, etc.) and convert them into SDR.
- Temporal memory is the algorithm that learns transition of patterns.
- HTM systems learn continuously: as input data changes, the HTM model updates itself.
- HTM builds a predictive model of the world, so every time it receives input, it attempts to predict what is going to happen next.

SDR DEFINITION

HTM systems require data input in the form of Sparse Distributed Representations (SDRs).

SDR is a bit array with semantic meaning.

0 1 1 0 1 1 0 1 = m in ASCII.
0 1 1 1 1 1 0 1 = } in ASCII – has nothing to do with m

ASCII is NOT an SDR because position of 1 in this array means nothing, i.e. no semantic meaning.

SDR SPARCITY

Dense representation

50% on

1	1	0	1	1	0	0	0	1	1	1	1	1	1	1	0
0	1	1	1	1	0	0	1	1	0	0	1	0	0	1	0
0	0	0	0	1	1	1	1	1	0	0	1	0	1	0	1
0	1	1	1	0	1	0	0	0	1	1	1	0	0	0	1
1	1	1	0	1	0	1	0	1	0	0	0	0	0	1	1
0	1	0	0	0	0	1	1	0	0	0	1	1	0	1	1
0	1	0	1	0	1	1	1	1	1	1	1	1	0	0	0
0	1	1	0	1	0	1	1	0	1	1	1	0	0	0	1
0	1	1	1	0	0	1	1	0	1	1	0	1	1	1	1
0	0	1	1	1	1	1	1	1	1	1	1	1	0	0	1
0	0	1	1	0	0	0	0	1	0	1	1	0	1	1	0
0	1	0	1	1	1	0	0	1	0	0	0	0	1	1	1
0	1	0	0	1	0	0	0	1	1	1	1	0	0	1	0
1	0	0	0	0	0	0	1	1	1	1	0	0	1	0	0
0	0	0	0	0	1	0	1	0	0	1	1	0	1	1	0
0	0	1	1	0	1	1	0	1	1	1	0	0	1	1	0

Capacity = 2^256.

That’s good, but this is not a good SDR.

Sparse representation

2% on

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0

Capacity: 174,792,640

SDR CAPACITY

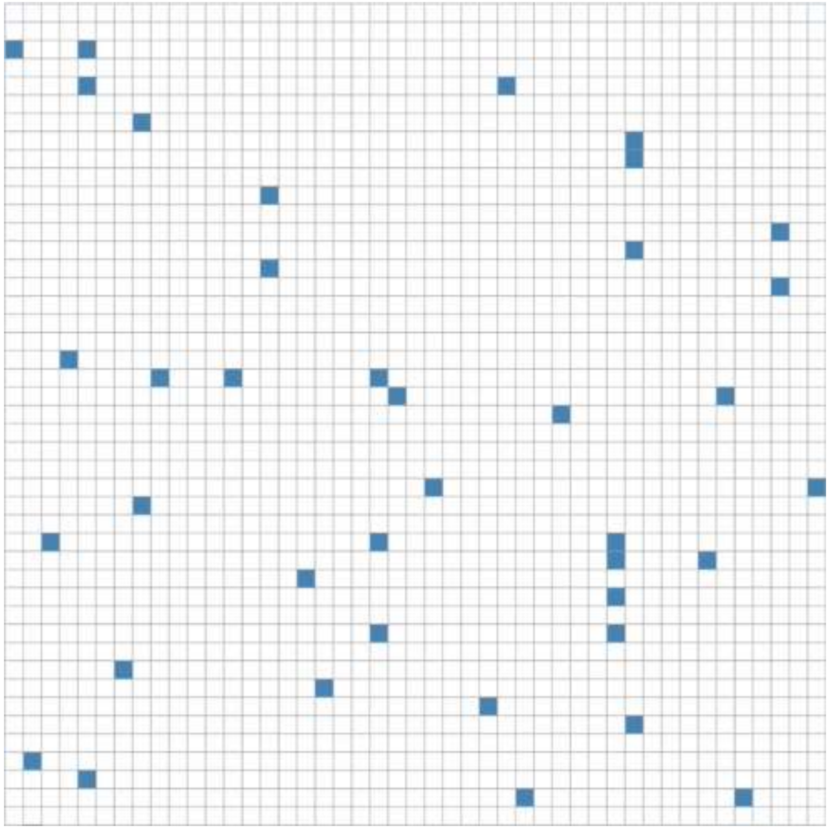
n = 2048 – array length

w = 40 - population

w/n = 0.0195 – sparsity

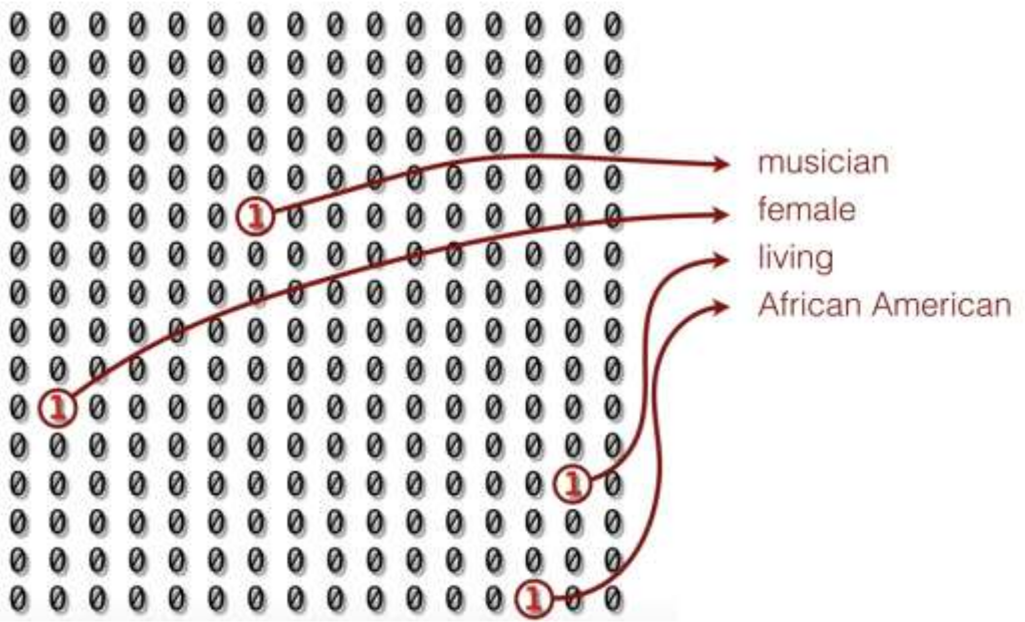
Capacity = $\binom{n}{w} = \frac{n!}{w!(n-w)!}$

= 2371778511645358086693262639700863268089720614584700731712608317643033681970419921664



SDR FEATURE REPRESENTATION

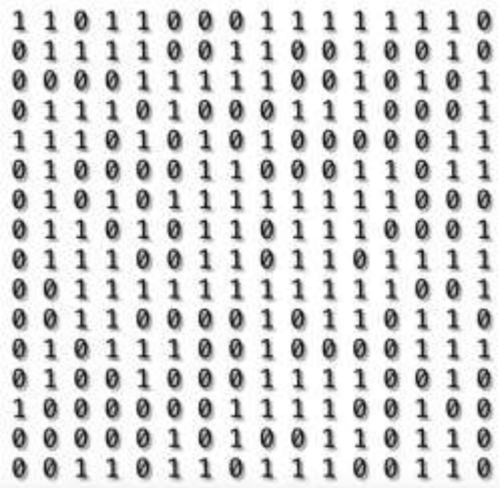
Feature representation:



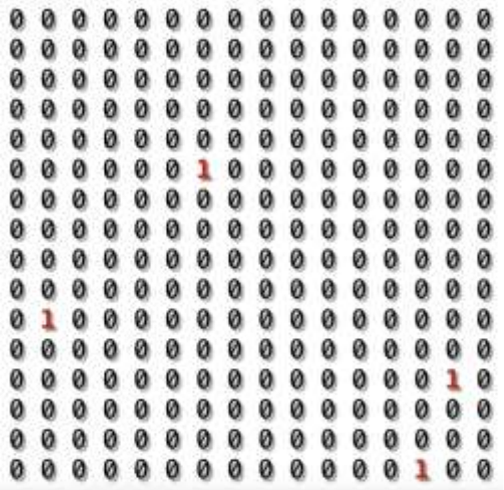
SDR COMPRESSION

Compression

Badly compressed:



Well compressed: store just the indexes (32 bits).



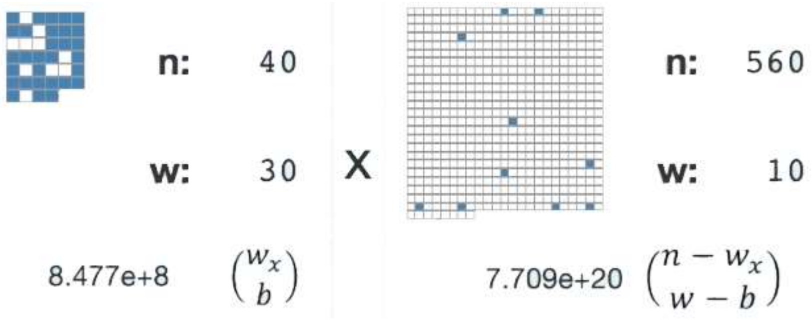
SDR OVERLAPPING

Question: for an SDR x of size [n,w] what is the probability that another SDR will overlap on exactly **b** bits?

Answer: Let $\Omega_x(n,w,b)$ be the overlap set for x. Then $|\Omega_x(n,w,b)| = \binom{w}{b} \times \binom{n-w}{w-b}$

However, the ratio of $|\Omega_x|/Capacity(n,w)$ is usually very small.

E.g. for n=600, w=40
the probability of false positive
(coincidental overlap)
of another SDR with x on 30 bits
is just 1.5e-33



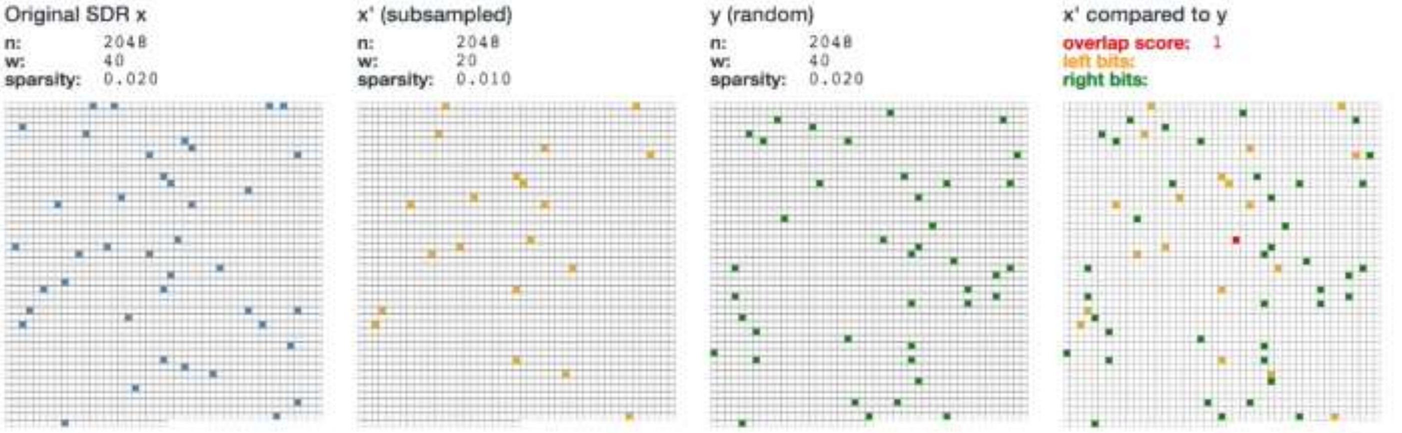
Conclusion: if we get a match, it is a real match, but not some random match.

SDR SUBSAMPLING

Let us imagine that we don't want store all the bits of SDR x, and we take just 50% of bits – name it SDR x'.

The chance that a random SDR will match x' is also too low.

For n=2048, w=40, sample=50% the chance of false positive is just 3.85e-13:



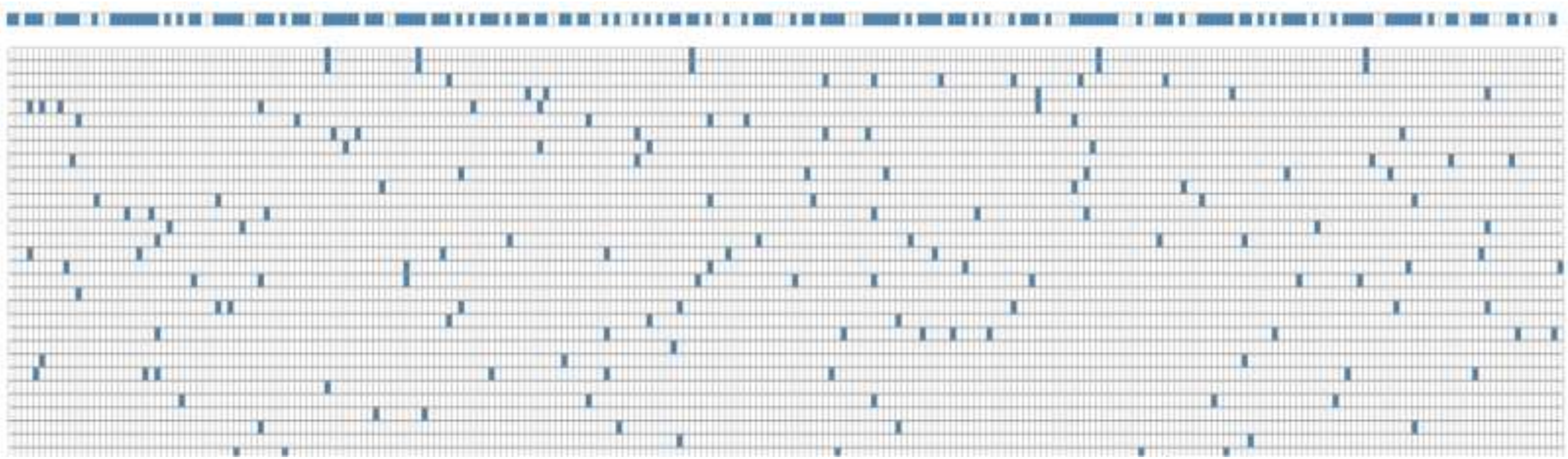
Conclusion: if somehow we loose some of "on" bits, we are still resilient to noise (i.e. we won't confuse the remainder part of SDR with a noise)

SDR UNIONS

SDRs with similar overlaps have similar semantic meanings. Imagine there is a stream of SDRs and we're collecting those SDRs and putting them to different sets (buckets). As each new SDR comes down the stream, we can compare to different sets to see if we've seen it before.

To speed up the process, we can compare it with bucket union-SDR instead of comparing with each SDR in each bucket:

Union:
of
50 SDRs:



For n=2048, w=40, probability that a random SDR will match the union is 7.78e-9.

Conclusion: if we have a match of a new SDR with a union, with high probability it is a real match with one of SDRs which form this union.

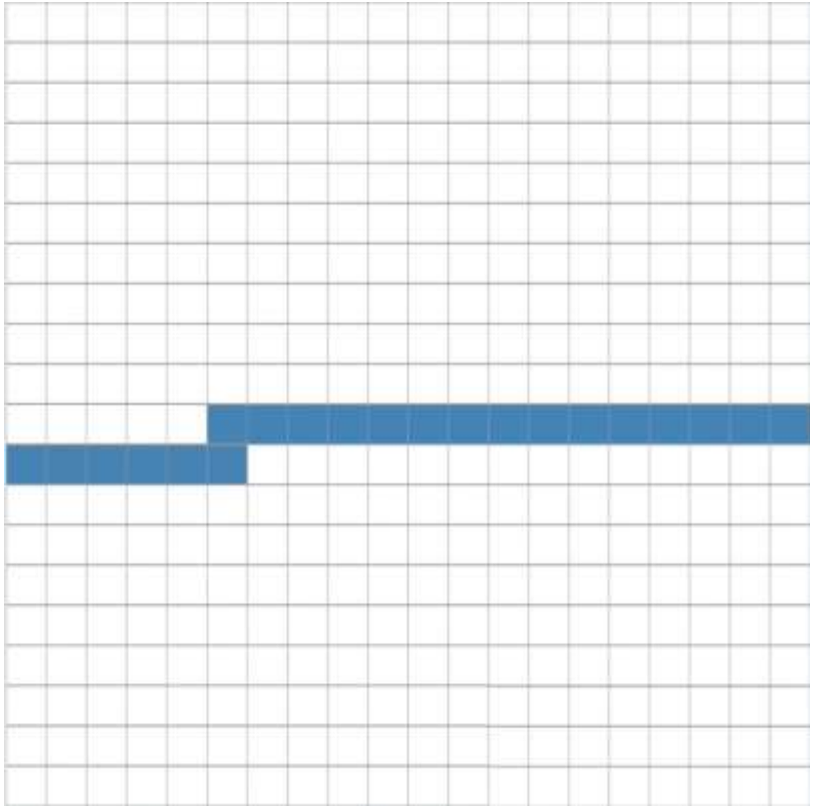
SCALAR ENCODERS

Encoders are used to convert any data to SDR – the input format suitable for HTM.

The simplest encoder is a scalar encoder which transforms a value to an SDR.

Example: for n=400, w=21 there will be 380 buckets. Each bucket represent one or more numbers.

If we encode values from 0 to 100, this SDR (at the picture) corresponds to 54.



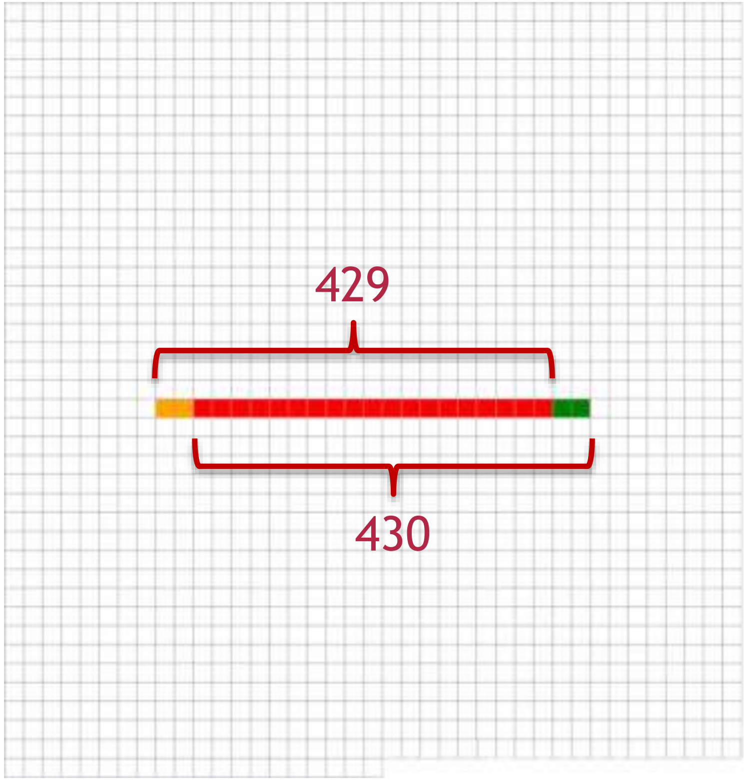
SCALAR ENCODERS

N=1580
w=21
Buckets=1560

Min=0
Max=809

The difference between values 430 and 429 will be in two bits.

They are semantically similar: 429 is close to 430. How close? Depends on w (see next slide).



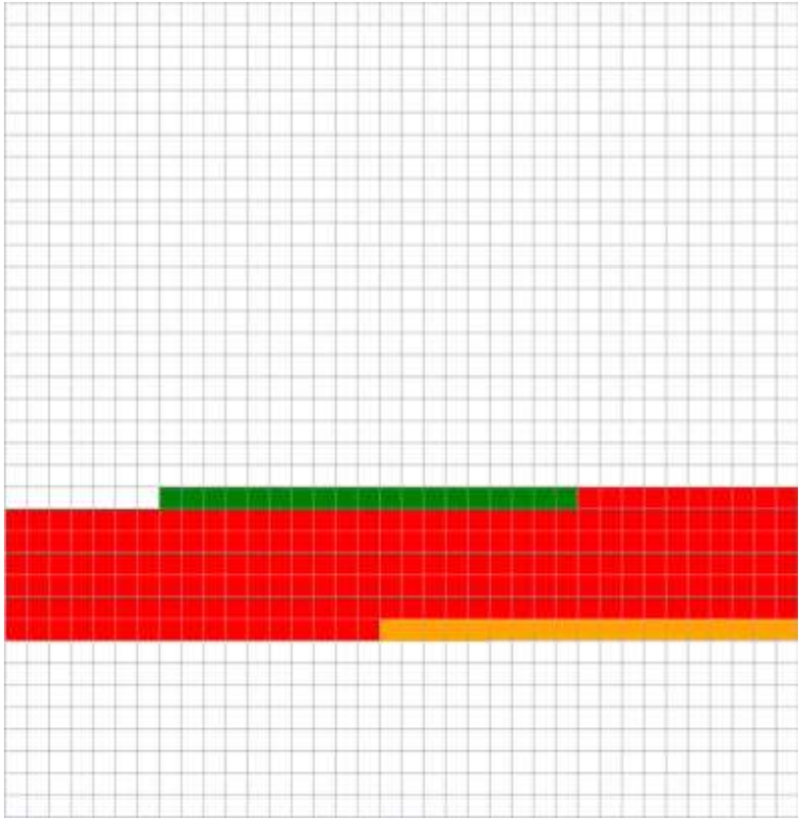
SCALAR ENCODERS

N=1330
w=226
Buckets=1105

Min=42
Max=100

Green+red = 84
Red+yellow = 85

A huge overlap (red bits) means that we consider 84 and 85 to be semantically very similar.



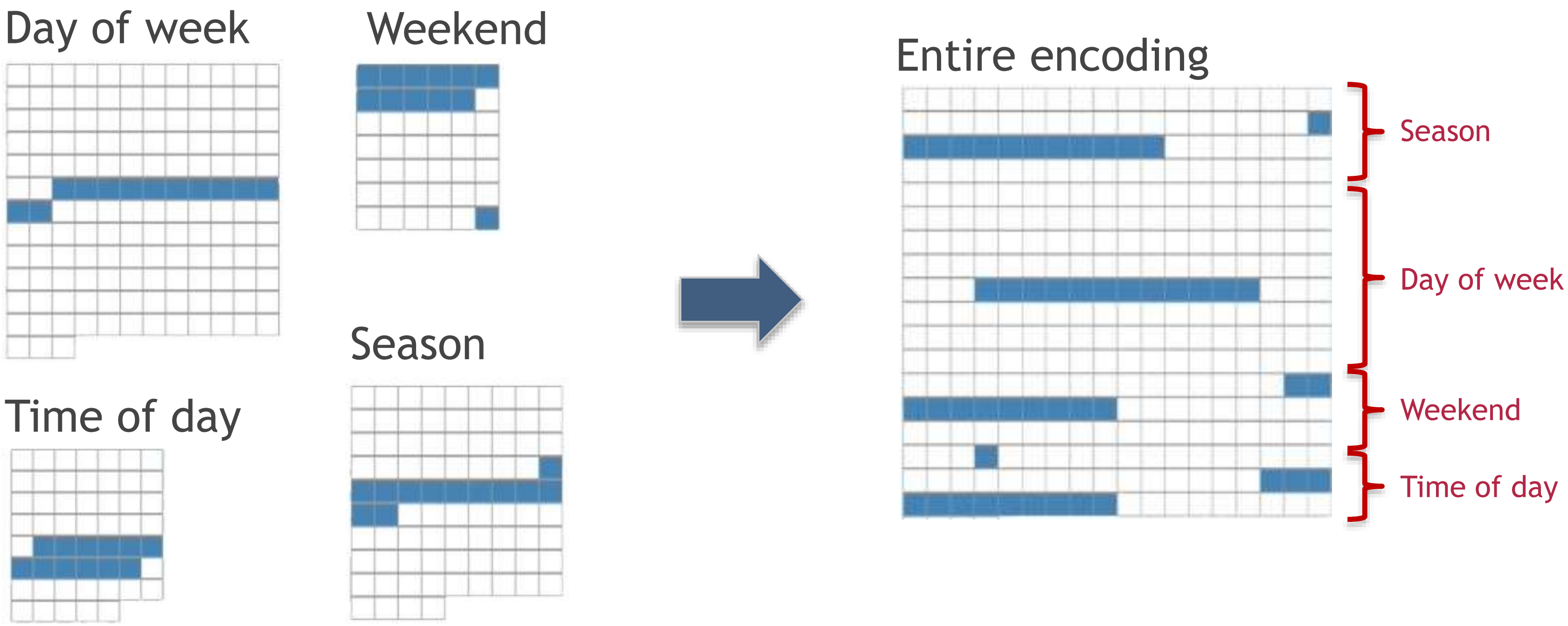
ENCODING PRINCIPLES

There are four principles of SDR encoding:

1) Semantically similar data should have SDRs with overlapping bits.
2) The same input should create the same SDR as output.
   Note: reverse is not true: one SDR can correspond to several inputs.
3) The output should have the same dimensionality for all inputs (n)
4) The output should have similar sparsity for all inputs and have enough one-bits to handle noise and subsampling.

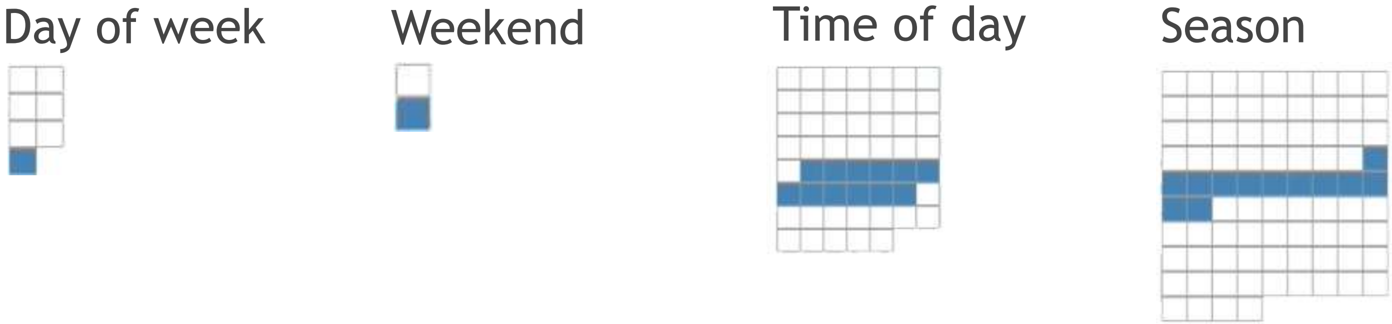
DATE ENCODER

The value “06/06/2016 2:44 PM” will be encoded as:



DATE ENCODER

Question: why won't we use minimum bits for some parts, e.g. for day of week and weekend?

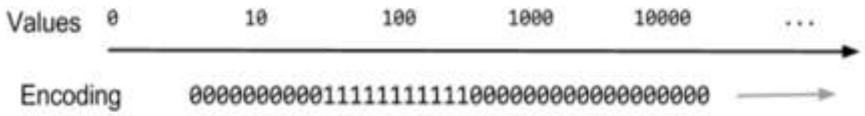


Answer: because we decrease the semantic meaning of day of week and weekend in comparison to other parts.

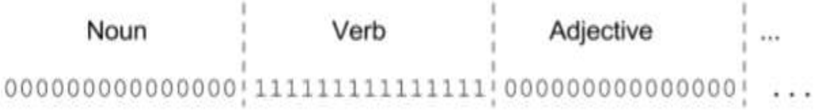
Note: there are cases when NOT all parts are needed.

OTHER ENCODERS

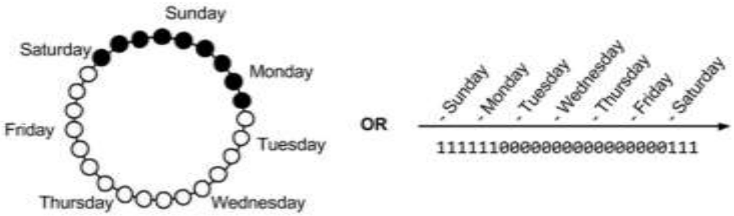
- Numeric log encoder



- Delta encoder is used when ranges are unknown, i.e. when data is constantly increasing. It is also used when surrounding conditions change, that is when the outside temperature changes, and the measurement will change ranges accordingly.
- Category encoding:



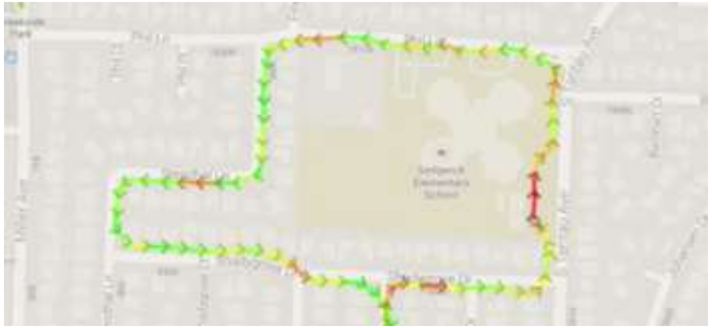
- Cyclic encodings:



GEOSPATIAL ENCODER

Given an input: latitude, longtitude, speed:

1) Find the square in the grid where coordinate falls
2) Draw a box around square with radius proportional to speed.
3) Choose top w squares by hash(x,y)
4) Activate bits that correspond to squares



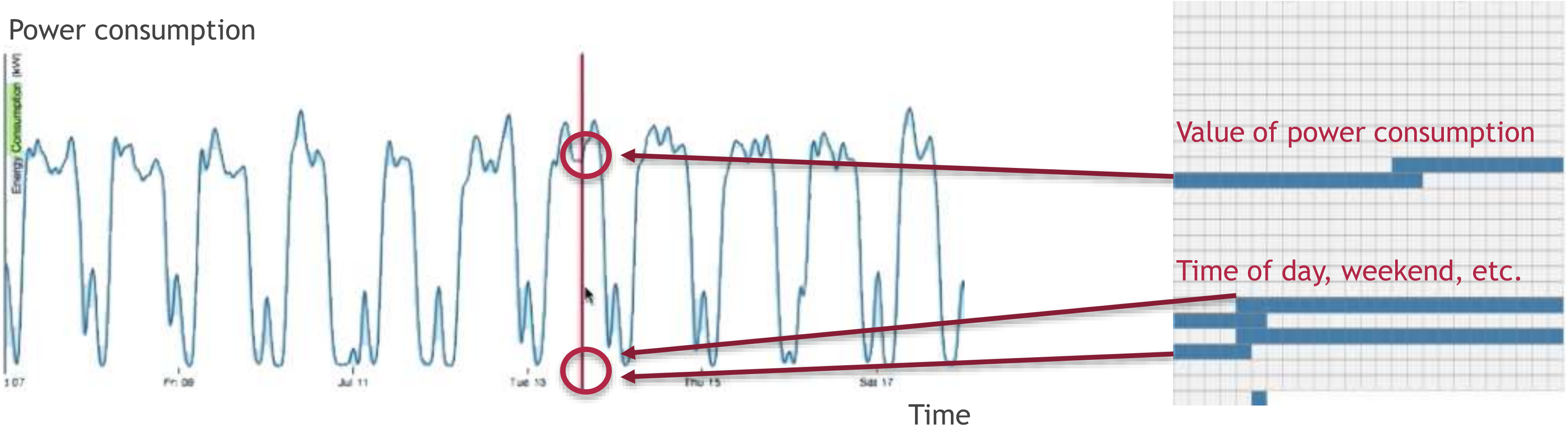
References:

https://www.youtube.com/watch?v=KxxHo-FtKRo
https://www.youtube.com/watch?v=M4dD9wCQLkA
https://github.com/numenta/nupic.geospatial

SPATIAL POOLER

Input Space Capacity = 5.1e+103

Power consumption



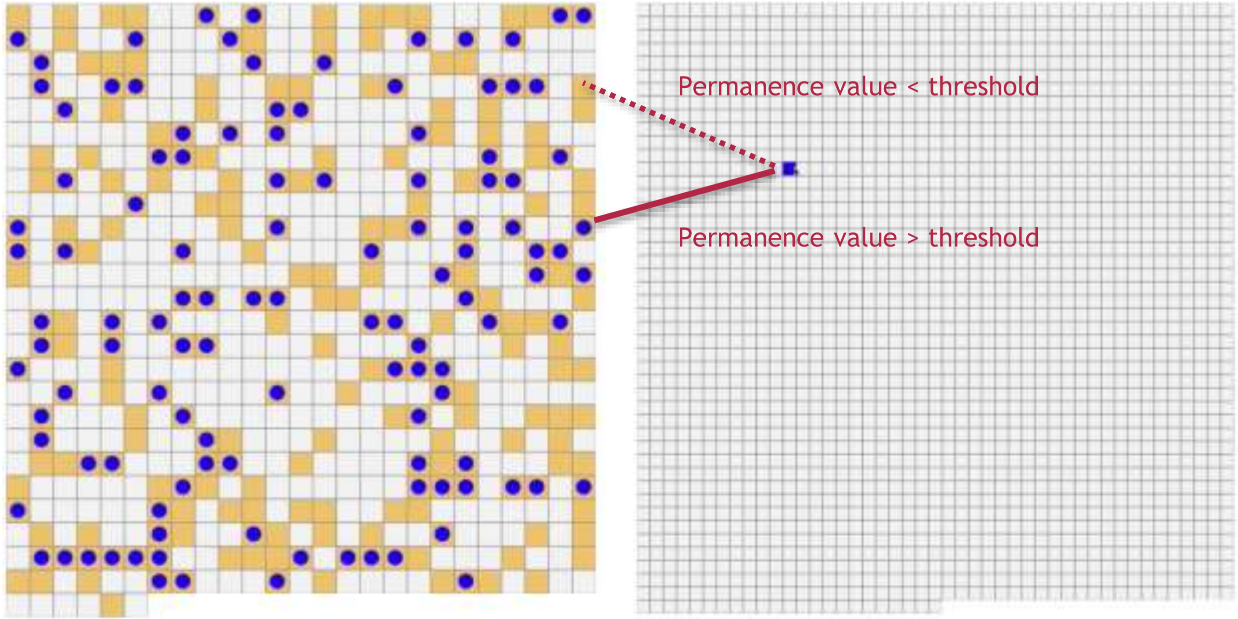
The input space capacity is extremely huge.

Sparial Pooler finds patterns which are relatively similar and will convert them to SDRs of a smaller density.

SPATIAL POOLER

Input space

Spatial Pooler Columns



Note: we are not looking at a specific SDR here!

Yellow squares denote positions, but NOT SDR.

Legend:

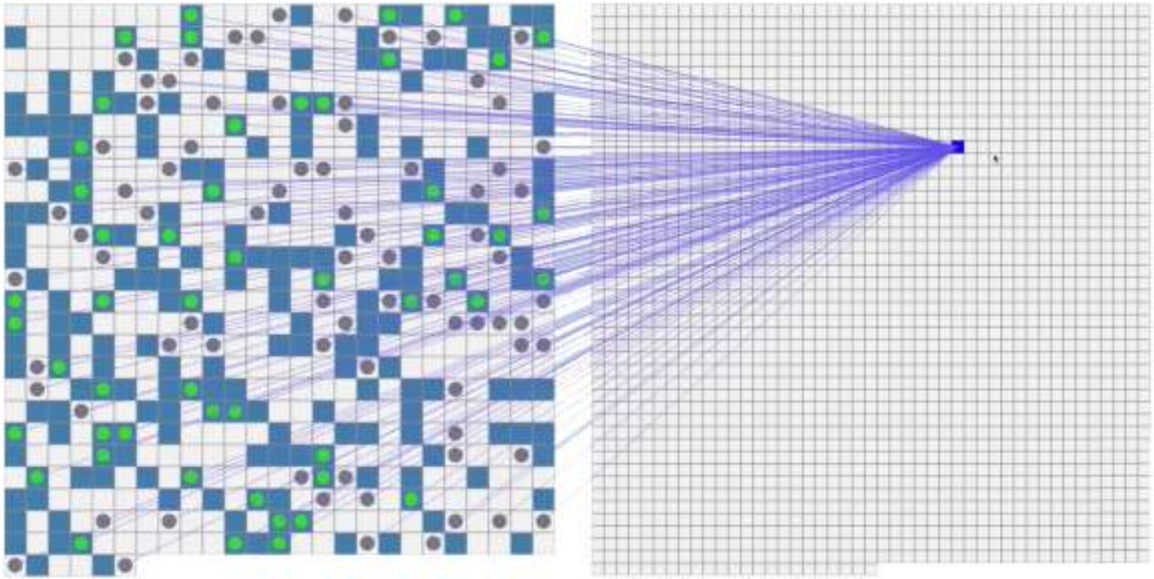
- Yellow squares are potential connections
- Blue dots are real connections whose permanence values are greater than some threshold
- Blue square is a column of the spatial pooler which is connected to all the input bits marked with blue circle.
- **White bits** of the input space are those bits which will never be connected to that cell.

SPATIAL POOLER COLUMN ACTIVATION

For a given input SDR, a column becomes active if its connections overlap with more than some T bits in “on” status (T is some threthold). How to decide what is the value of T – see next slide.

SDR

Spatial Pooler Columns



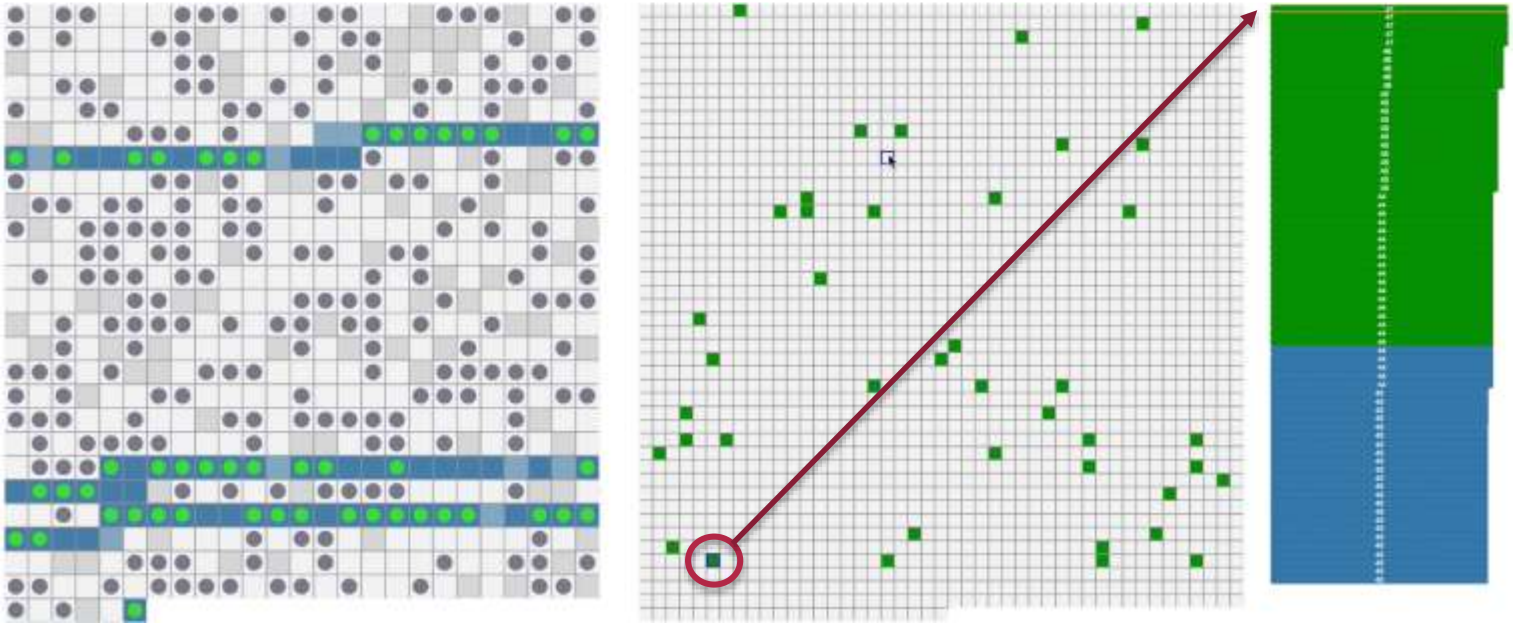
Legend:

- Blue bits (left image) are parts of a specific SDR.
- **Circles** are those bits where the column has connections
- Green circles denote overlap between “on” bits and connections.

Thus, the overlap score = 51

SPATIAL POOLER ACTIVATION

We set those columns to "active" state who are at the top by the number of overlaps of their connections to the input SDR:



This is the rating of columns based on the number of overlapping connections

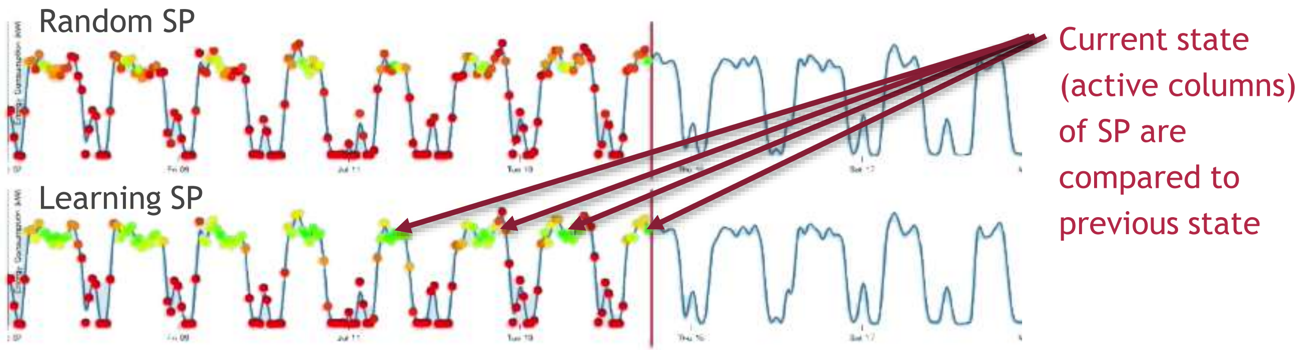
Note: the settings of "40" is set for a parameter "number of active columns per inhibition area".

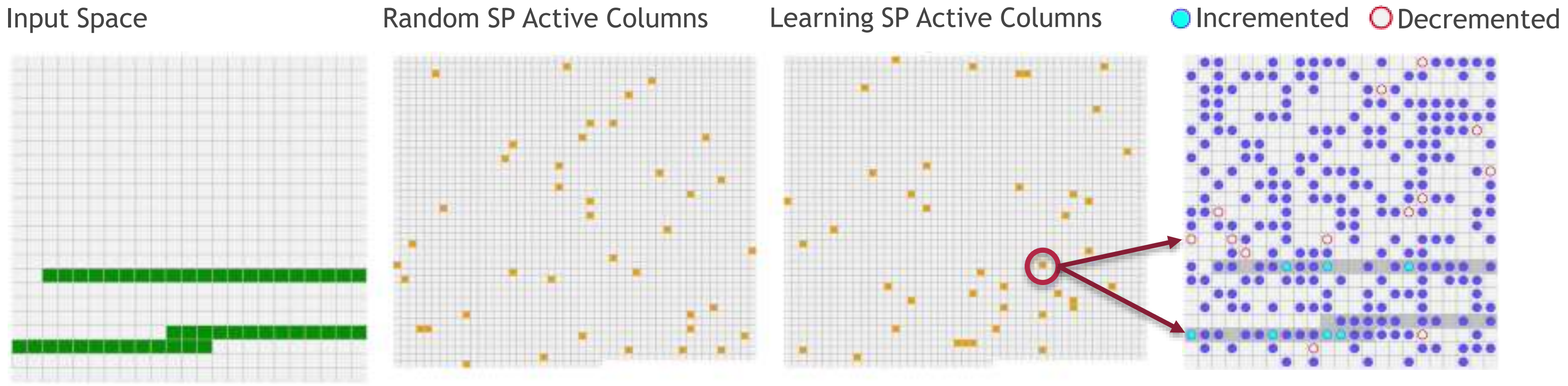
SPATIAL POOLER LEARNING

Learning process is next:

1) Only those columns learn which are activated

2) Permanence values of the connections are incremented or decremented.



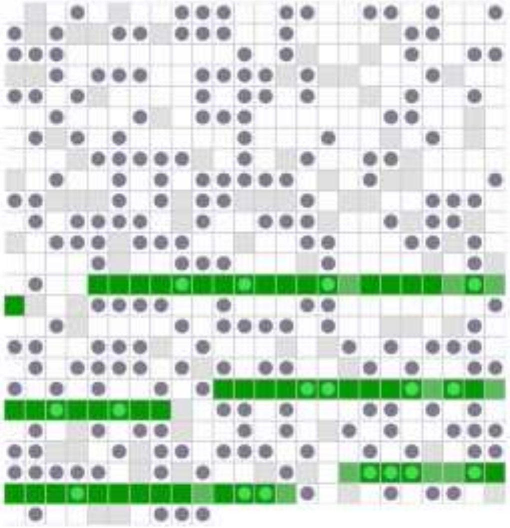


BOOSTING

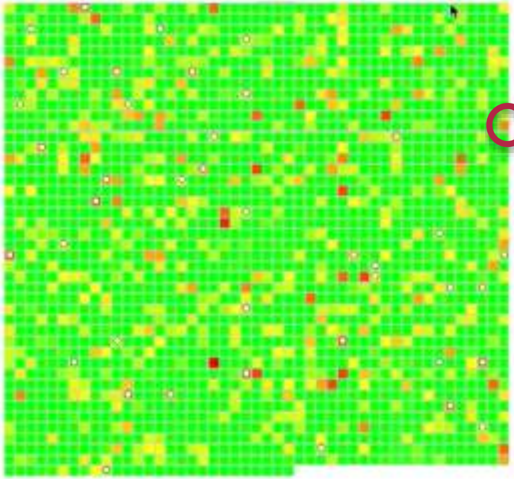
In order for the column to update permanence values, it must be selected as a winning column. Looser columns are inhibited from expressing themselves. Boosting occurs BEFORE inhibition, artificially increasing the overlap score for less active columns and decreasing the overlap for more active.

Overlap score is multiplied by boosting factor if the column rarely activates, e.g. if overlap = 17, boosted overlap will be 17 * 1.2 = 20.4

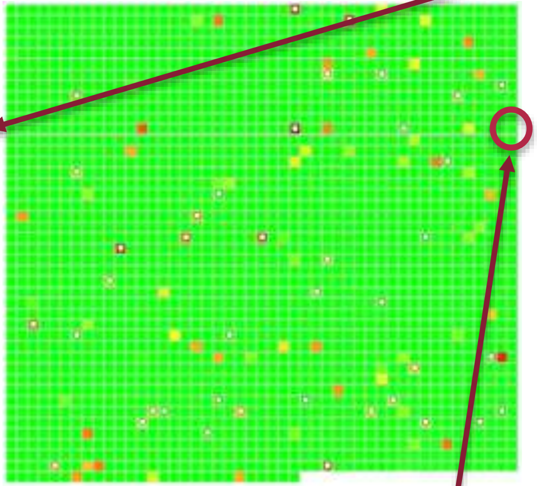
Input SDR



With boosting



Without boosting



Red = the column was active in 100% of cycles, green = point was never active

Without boosting this column was never activated

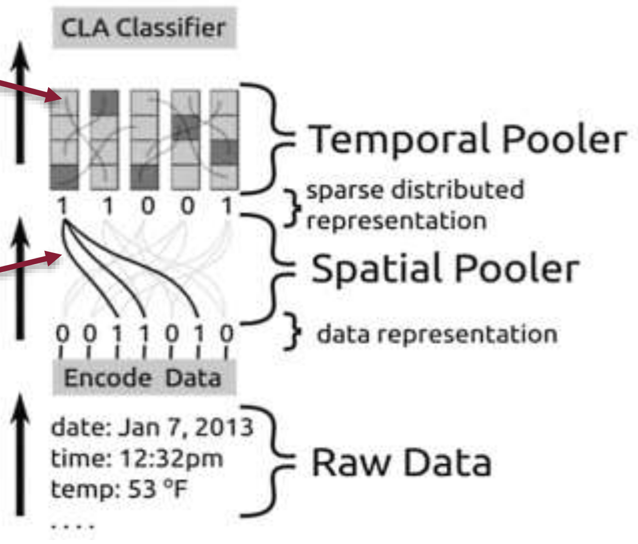
TEMPORAL POOLER

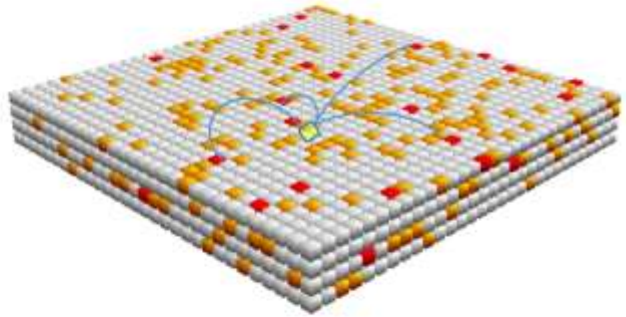
The TP algorithm does two things:

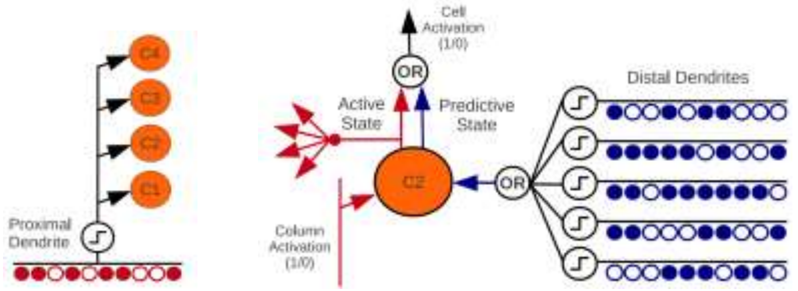
1) Learns the sequences of active columns from Spatial Pooler.
2) It makes predictions about what pattern is coming next based on a temporal context of each input.

Distal connections (between cells of the columns)

Proximal connections (from sensors to columns)

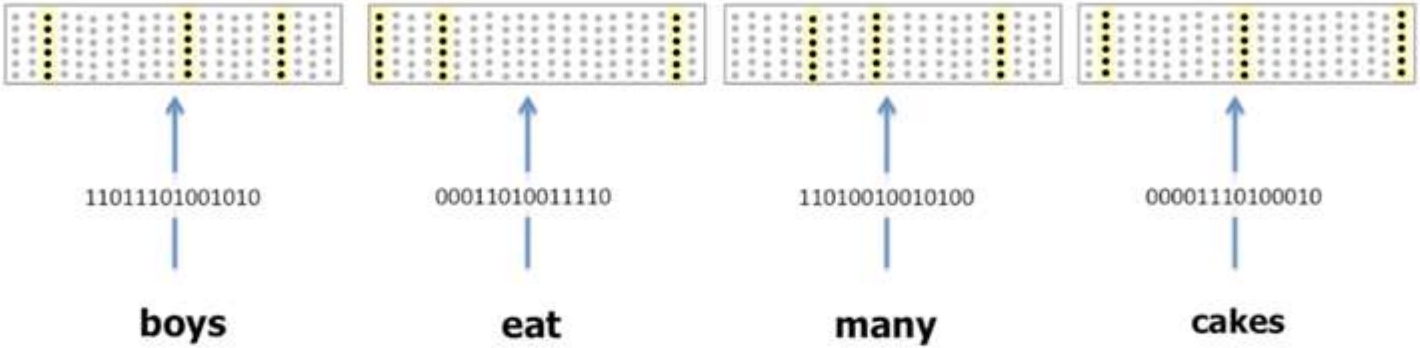




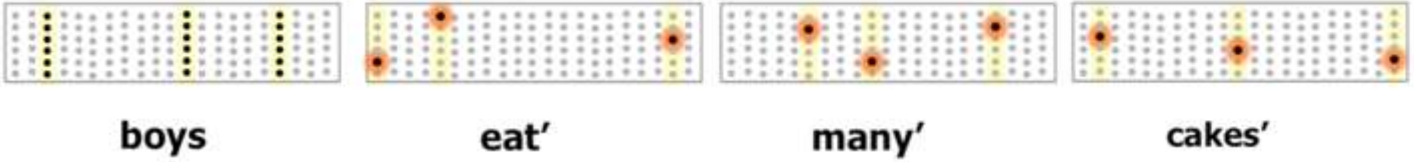


TEMPORAL POOLER EXPLAINED

Active cells before TP (after SP):

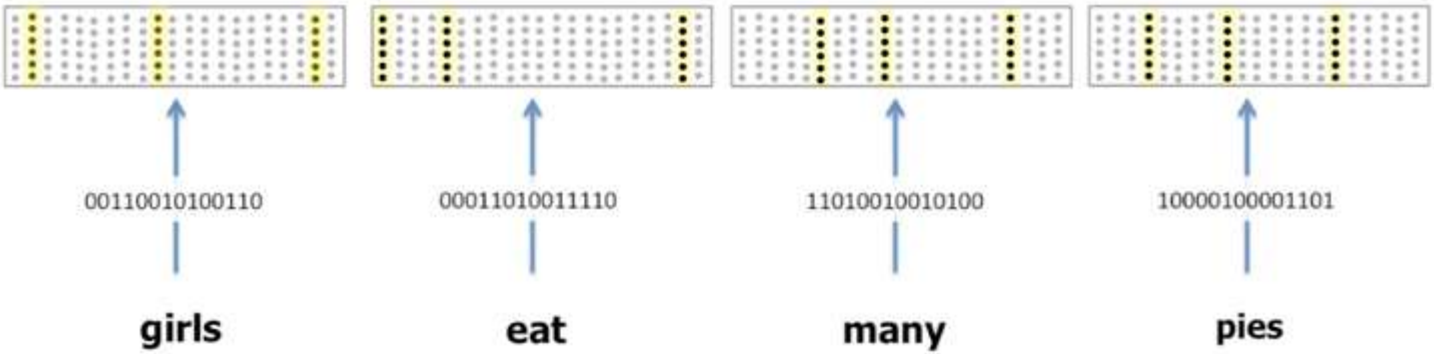


Active cells after TP:

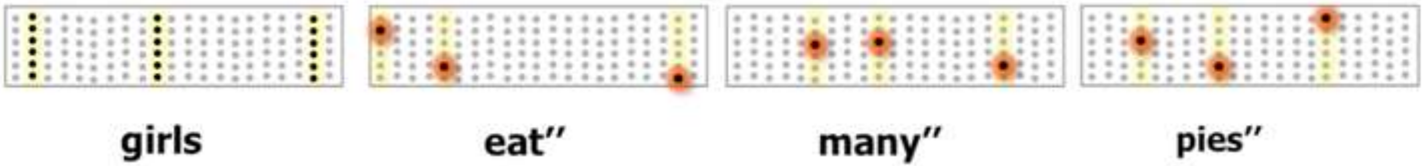


TEMPORAL POOLER EXPLAINED

Active cells before TP (after SP):

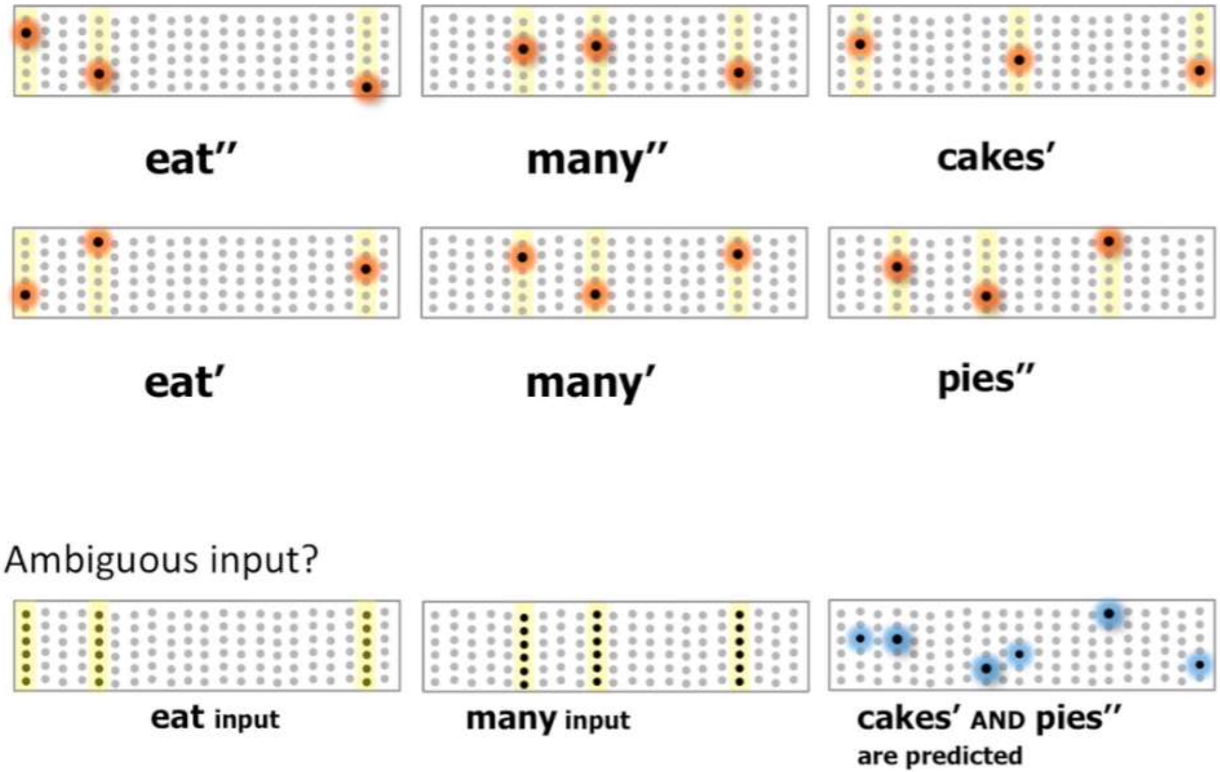


Active cells after TP:



TEMPORAL POOLER EXPLAINED

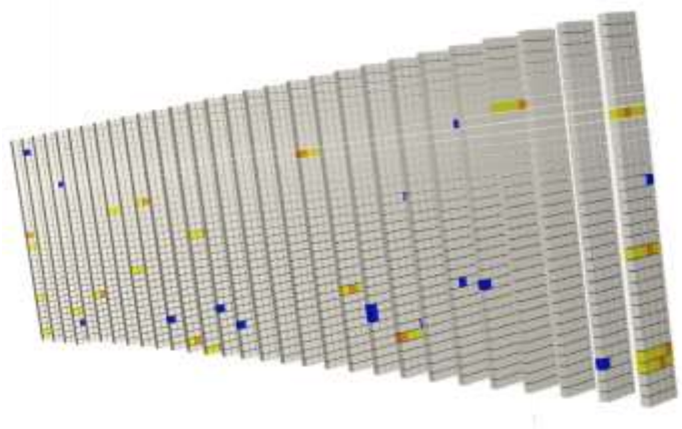
In the case if we get an input with not enough previous context, both cakes and pies will be predicted.



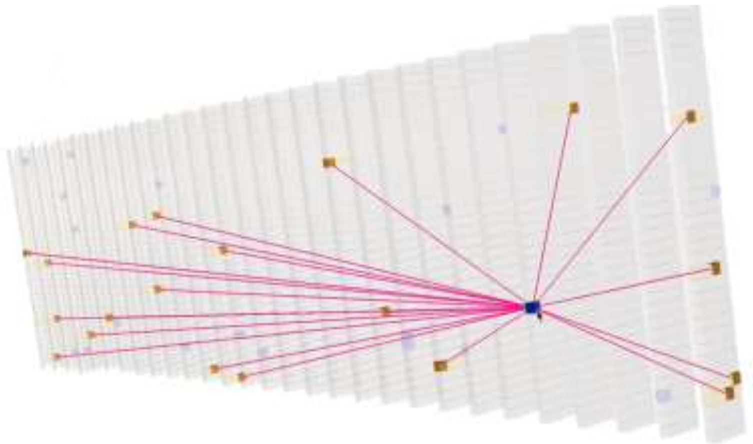
TEMPORAL POOLER

There are two phases (if not to consider learning) of the temporal memory algorithm:

1) To identify which cells within active columns will become active on this time step.
2) Choose a set of cells to put in the predictive state. It means that those steps are predicted to fire on the next time step.



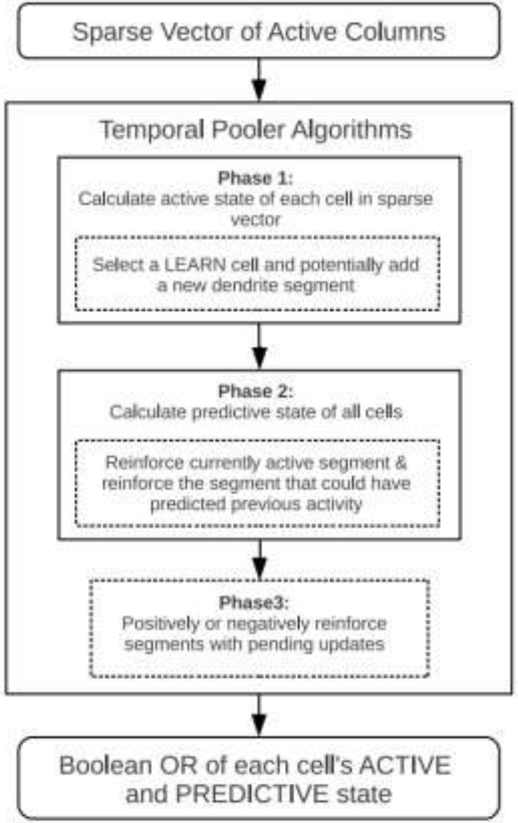
Column is active
Cell is active
Cell is predicted to be active on the next step



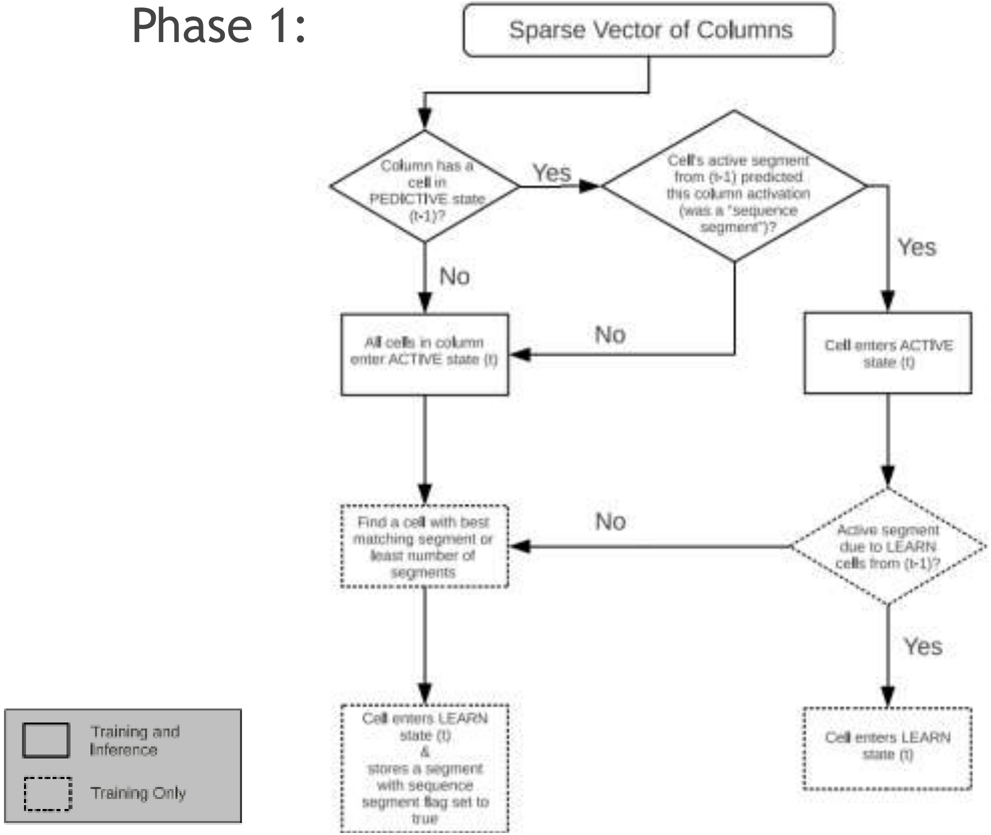
The cell comes into predictive state because it has enough distal connections to other active cells

TEMPORAL POOLER LEARNING

All algorithm:

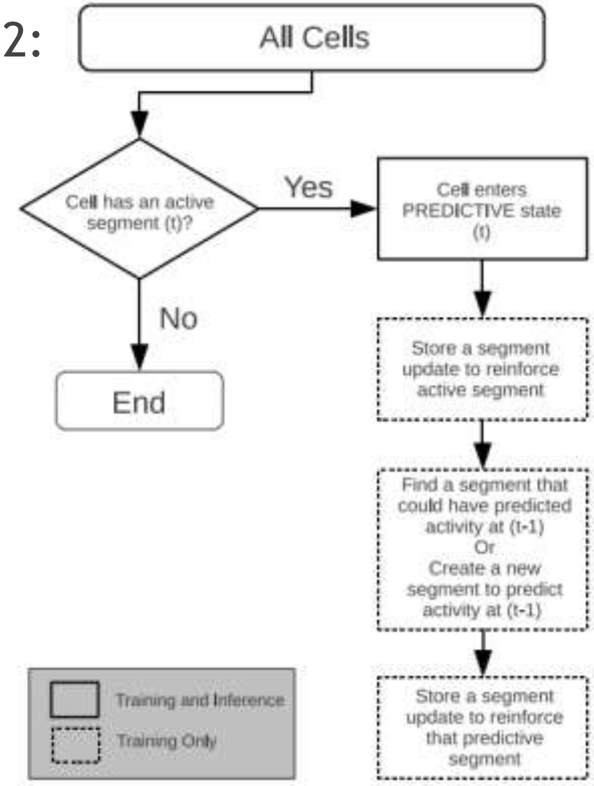


Phase 1:

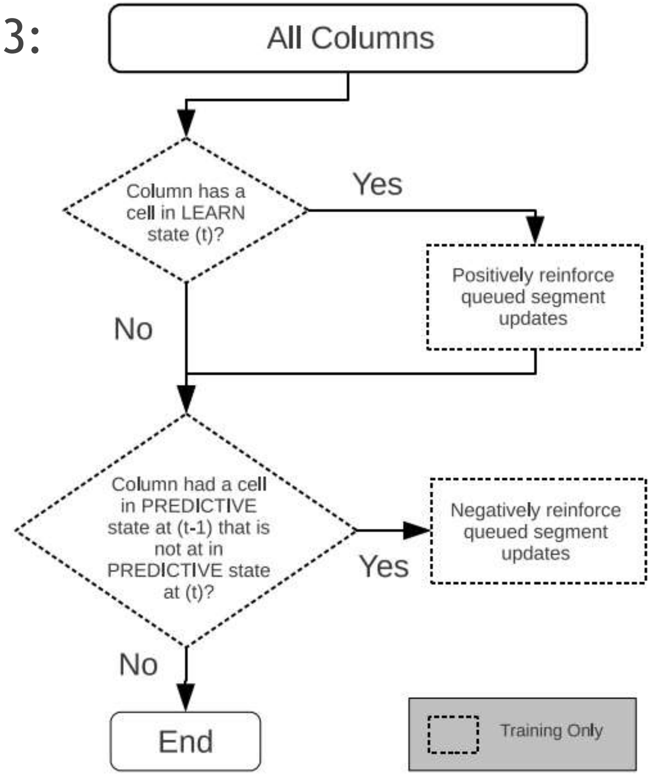


TEMPORAL POOLER LEARNING

Phase 2:



Phase 3:

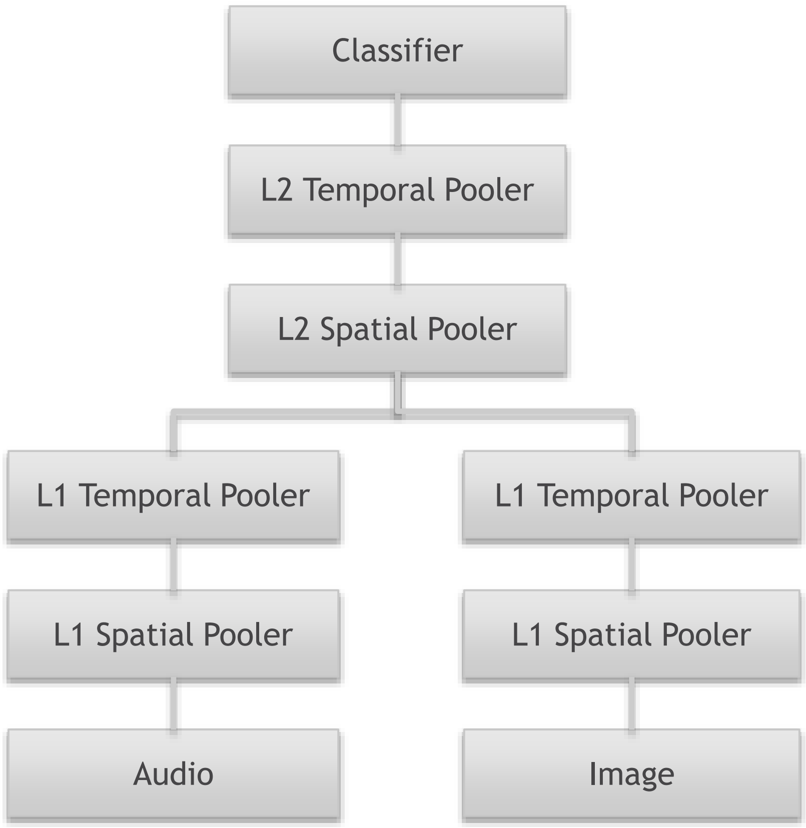


TOPOLOGY

The networks and regions API are intended for arbitrary topologies.

Here is an example of a topology: you may have audio and video encoders which feed bit arrays to SP and TP studying sequences.

Then the outputs of those TPs are fed to the higher level SP that combines inputs.



CLASSIFIER

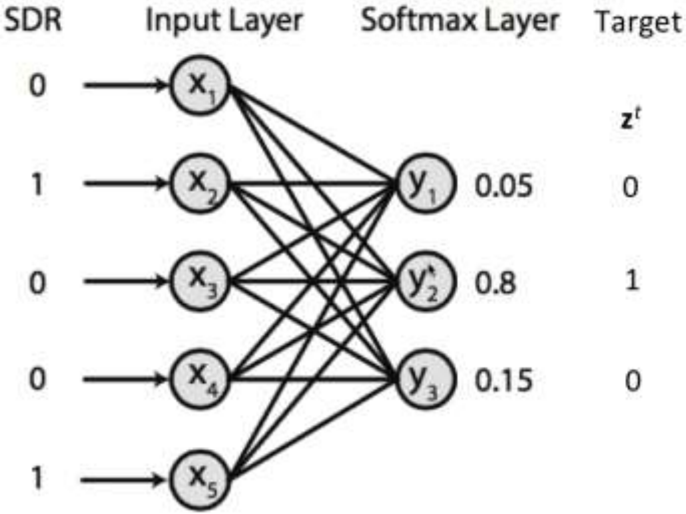
Classifier tries to infer the output from active columns in the upper region in the hierarchy.
There are some kinds of classifiers implemented in NuPIC:

- **KNNClassifier** - maintains a set of template SDRs in memory;
- **CLAClassifier** - heuristic voting algorithm;
- **SDRClassifier** - feedforward neural network that uses maximum likelihood estimation.

Question: What if we need prediction?
Answer: Use classification+bucketing instead, because you're never going to predict values like 0.2344521 – instead we need 0.23 if the measurement precision is 0.01. It is enough to make 100 classes (=buckets), and this is what is done in NuPIC.

SDR CLASSIFIER

Purpose: to learn associations between a given state of the Temporal Memory at time *t* , and the value that is to be fed into the Encoder at time *t+n* (n is the number of steps into the future you want to predict).



$$a_j = \sum_{i=1}^{N} w_{ij} x_i \qquad p(C_k|\mathbf{x},\mathbf{w}) = y_k = \frac{e^{a_k}}{\sum_{i=1}^{K} e^{a_i}}$$

$$L = -\ln \prod_t p(\mathbf{y}^t|\mathbf{x}^t;\mathbf{w}) = -\sum_t \ln p(\mathbf{y}^t|\mathbf{x}^t;\mathbf{w})$$

$$\frac{\partial L}{\partial w_{ij}} = \frac{\partial L}{\partial a_j}\frac{\partial a_j}{\partial w_{ij}} = (y_j - z_j)x_i$$

actual output

target output

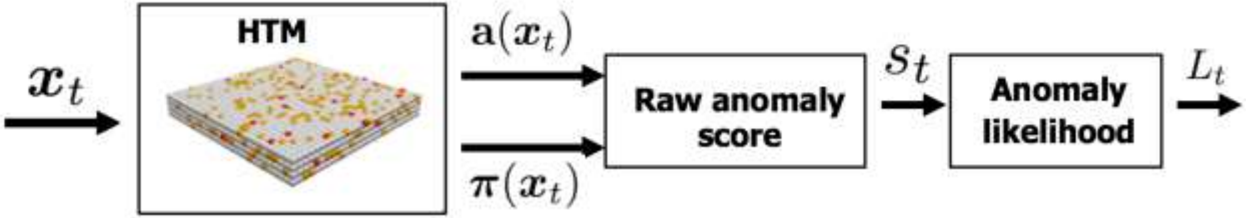
input

$$\Delta w_{ij} = \begin{cases} \alpha(y_j - z_j) & i \in \{i : x_i > 0\} \\ 0 & otherwise \end{cases}$$

Time complexity: O(sNK), s = sparcity

ANOMALY

The HTM model receives continuous stream of inputs: $\ldots, \boldsymbol{x}_{t-2}, \boldsymbol{x}_{t-1}, \boldsymbol{x}_t, \boldsymbol{x}_{t+1}, \boldsymbol{x}_{t+2}, \ldots$
Let $\mathbf{a}(\boldsymbol{x}_t)$ be the SDR got from the last layer of the HTM.
Let $\boldsymbol{\pi}(\boldsymbol{x}_t)$ be the prediction SDR for $\mathbf{a}(\boldsymbol{x}_{t+1})$, so the whole stem looks like this:



Option 1: to compute a raw anomaly score that measures the deviation between the model's predicted input and the actual input:

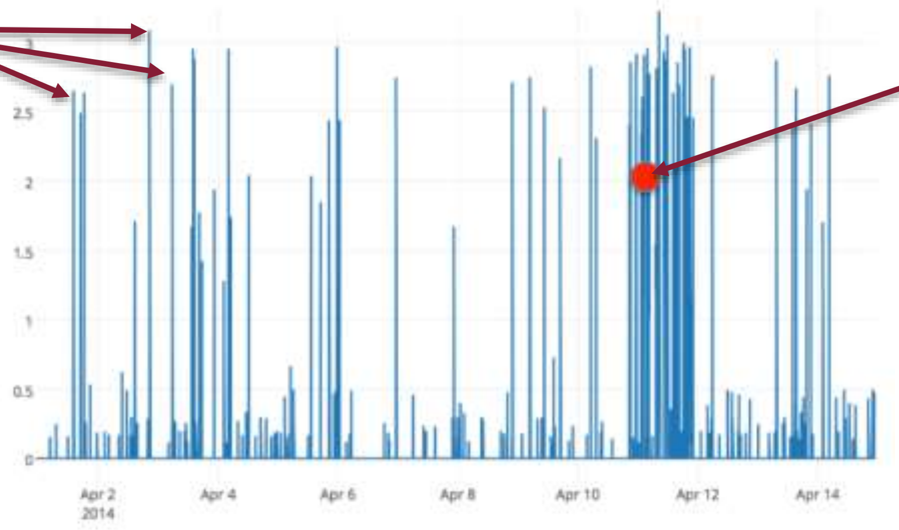
$$s_t = 1 - \frac{\boldsymbol{\pi}(\boldsymbol{x}_{t-1}) \cdot \mathbf{a}(\boldsymbol{x}_t)}{|\mathbf{a}(\boldsymbol{x}_t)|}$$

But this won't work fine in all cases – see the next slide.

ANOMALY

Raw anomaly score works well for predictable scenarios, but in many practical applications the underlying system is inherently noisy and unpredictable.

These peaks are not anomalies: it is normal for a load balancer to have occasional jumps. So, with raw anomaly score we'll get lots of false positives.



Latency (in seconds) of a load balancer on a production website.

Sustained increase in the frequency of high latency requests is unusual.

If we used raw anomaly score

$$s_t = 1 - \frac{\boldsymbol{\pi}(\boldsymbol{x}_{t-1}) \cdot \mathbf{a}(\boldsymbol{x}_t)}{|\mathbf{a}(\boldsymbol{x}_t)|}$$

It would produce us a lot of false positives.

ANOMALY LIKELIHOOD

Option 2: calculate the anomaly likelihood - a metric defining how anomalous the current state is based on the prediction history of the HTM model:

- Maintain a window of the last W raw anomaly scores
- Model the distribution as a **rolling normal distribution** with sample mean and variance continuously updated as follows:

$$\mu_t = \frac{\sum_{i=0}^{i=W-1} s_{t-i}}{W} \qquad \sigma_t^2 = \frac{\sum_{i=0}^{i=W-1} (s_{t-i} - \mu_t)^2}{W-1}$$

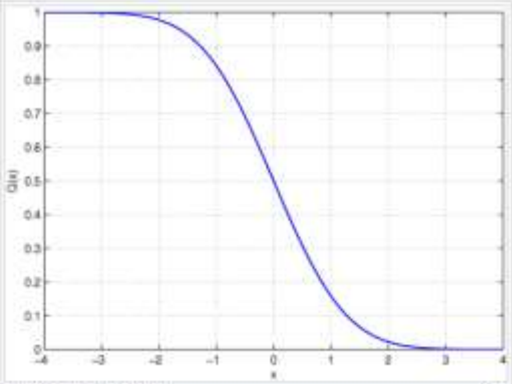
Then the recent short term average of raw scores is computed,

and the Q function is applied. Anomaly likelihood: $L_t = 1 - Q\left(\frac{\tilde{\mu}_t - \mu_t}{\sigma_t}\right)$

where $\tilde{\mu}_t = \frac{\sum_{i=0}^{i=W'-1} s_{t-i}}{W'}$ W' is a window for a short term

moving average, where $W' \ll W$

Anomaly detected is when $L_t \geq 1 - \epsilon$



A plot of the Q-function

IMPLEMENTATIONS

There are two major implementations of HTM:

- **NuPIC** – Numenta Platform for Intelligent Computing,
  https://github.com/numenta/nupic - Python
  https://github.com/numenta/nupic.core - C++ core
- **HTM.java -** an official community driven port of NuPIC
  https://github.com/numenta/htm.java

Following the readme,
HTM.Java Receives new TemporalMemory - HTM.Java now fully in sync!! (10/13/2016)

However, by our observation, the HTM.java is missing the easiest API which is present in its Python equivalent.

API

According to docs http://nupic.docs.numenta.org/stable/index.html

- Online Prediction Framework (OPF)
  - Model Parameters
  - Create an OPF Model
  - Feed the Model Data
  - Extract the results
- Network API
  - Network Parameters
  - Create a Network
  - Add a Sensor Region
  - Add a Data Source to the Sensor Region
  - Add an Encoder to the Sensor Region
  - Add a Spatial Pooler Region
  - Add a Temporal Memory Region
  - Add a Classifier Region
  - Link all Regions
  - Set the Predicted Field Index
  - Enable Learning and Inference
  - Run the Network
  - Getting Predictions
- Algorithms API
  - Encoding Data
  - Spatial Pooling
  - Temporal Memory
  - Getting Predictions

OPF - the easiest API allowing to write a config file and instantiate the HTM. Missing in HTM.java

Network API – gives more flexibility, e.g. you can make your own AnomalyLikelihood class and pass it to the constructor.

Algorithms API. Most probably it was done just to allow us to play around with low level features and to make experiments.

SPARK IMPLEMENTATION

Class HTMNetwork - encapsulates inside the HTM Network.

Used inside the Spark streaming application as the state for a separate IoT device's stream.

```java
public class HTMNetwork implements Serializable {
    private static final long serialVersionUID = 1L;

    private static final String STR_DT = "DT";
    private static final String STR_MEASUREMENT = "Measurement";
    private static final String STR_DT_FORMAT = "YY-MM-dd HH:mm";

    private String id = null;
    private Network network = null;
    private ResultState resultState = new ResultState();

    public HTMNetwork() {...}

    public HTMNetwork(String id) {
        this.id = id;

        final Parameters parameters = getNetworkParams();
        final MultiEncoder encoder = MultiEncoder.builder().name("MultiEncoder").build();
        MultiEncoderAssembler.assemble(encoder, getFieldEncodingMap());

        this.network = Network.create(id, parameters)
                .add(Network.createRegion( name: "Region 1")
                        .add(Network.createLayer( name: "Layer 2/3", parameters)
                                .alterParameter(Parameters.KEY.AUTO_CLASSIFY, Boolean.TRUE)
                                .add(Anomaly.create())
                                .add(new TemporalMemory())
                                .add(new SpatialPooler())
                                .add(encoder)
                        ));
    }
```

SPARK IMPLEMENTATION

Class AnomalyDetector is the Spark streaming application which does the following:

- Gets the stream of IoT devices from Kafka
- Enriches the records by using HTM to find the anomaly score
- Writes this data back to Kafka.

The key point here is mapWithState transformation – the stateful map which uses one HTM instance for every device in order to feed the data from this device sequentially (as it comes in through time).

```java
private static JavaStreamingContext createStreamingContext(String appName, String checkpointDir, Durat
    SparkConf sparkConf = new SparkConf().
            setAppName(appName)
            .setMaster("local[1]") //TODO: Adjust parameters for standard run
            //.set("spark.streaming.backpressure.enabled", "true")  // investigate if this will influ
            .set("spark.streaming.kafka.maxRatePerPartition", "90") // this is per second, so we have
            .set(SPARK_KRYO_REGISTRATOR_REQUIRED_CONFIG, "true")
            .set(SPARK_INTERNAL_SERIALIZER_CONFIG, KryoSerializer.class.getName())
            .set(SPARK_KRYO_REGISTRATOR_CONFIG, SparkKryoHTMRegistrator.class.getName());

    JavaStreamingContext jssc = new JavaStreamingContext(sparkConf, batchDuration);
    jssc.checkpoint(checkpointDir);

    JavaInputDStream<ConsumerRecord<String, MonitoringRecord>> kafkaDStream =
            KafkaUtils.createDirectStream(jssc, LocationStrategies.PreferConsistent(),
                    KafkaHelper.createConsumerStrategy(rawTopicName));
    JavaPairDStream<String, MonitoringRecord> pairedDStream =
            kafkaDStream.mapToPair((ConsumerRecord<String, MonitoringRecord> kafkaRecord) ->
                    new Tuple2<>(kafkaRecord.key(), kafkaRecord.value()));
    JavaDStream<MonitoringRecord> statedDStream = pairedDStream
            .mapWithState(StateSpec.function(mappingFunc)).persist(StorageLevel.MEMORY_ONLY_SER());
    statedDStream.foreachRDD((JavaRDD<MonitoringRecord> rdd) ->
            rdd.foreachPartition((Iterator<MonitoringRecord> iterator) -> {
        try (KafkaProducer<String, MonitoringRecord> producer = KafkaHelper.createProducer()) {
            while (iterator.hasNext()) {
                MonitoringRecord record = iterator.next();
                ProducerRecord<String, MonitoringRecord> kafkaRecord = new ProducerRecord<>(
                        enrichedTopicName, getKey(record), record);
                producer.send(kafkaRecord);
            }
        }
    }));
    statedDStream.print();
    return jssc;
}
```

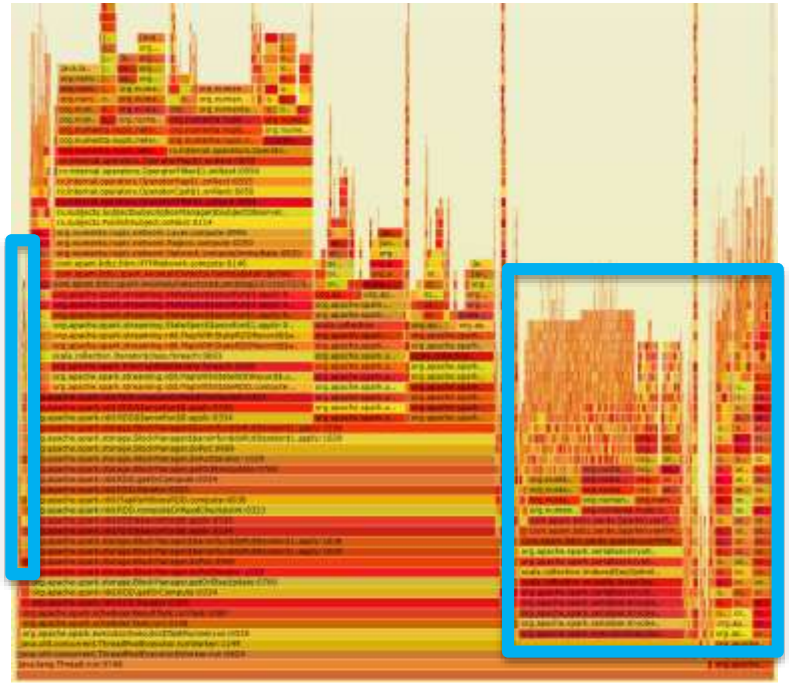
SPARK IMPLEMENTATION

The stateful mapping function gets the state for a particular device (it creates it if it doesn't exist). It gets the Date, Time and Measurement fields from the record, passes them to the HTM and gets the inference. Then it enriches the record with this information and returns it to the new stream.

```
private static Function3<String, Optional<MonitoringRecord>, State<HTMNetwork>, MonitoringRecord> mappingFunc =
        (deviceID, recordOpt, state) -> {
            // case 0: timeout
            if (!recordOpt.isPresent())
                return null;

            // either new or existing device
            if (!state.exists())
                state.update(new HTMNetwork(deviceID));
            HTMNetwork htmNetwork = state.get();
            String stateDeviceID = htmNetwork.getId();
            if (!stateDeviceID.equals(deviceID))
                throw new Exception("Wrong behaviour of Spark: stream key is $deviceID%s, state key is $stateDeviceID%s");
            MonitoringRecord record = recordOpt.get();

            // get the value of DT and Measurement and pass it to the HTM
            Map<String, Object> m = new HashMap<>();
            m.put("DT", DateTime.parse( record.getDateGMT() + " " +
                    record.getTimeGMT(), DateTimeFormat.forPattern( "YY-MM-dd HH:mm")));
            m.put("Measurement", Double.parseDouble(record.getSampleMeasurement()));
            ResultState rs = htmNetwork.compute(m);
            record.setPrediction(rs.getPrediction());
            record.setError(rs.getError());
            record.setAnomaly(rs.getAnomaly());
            record.setPredictionNext(rs.getPredictionNext());

            return record;
        };
```

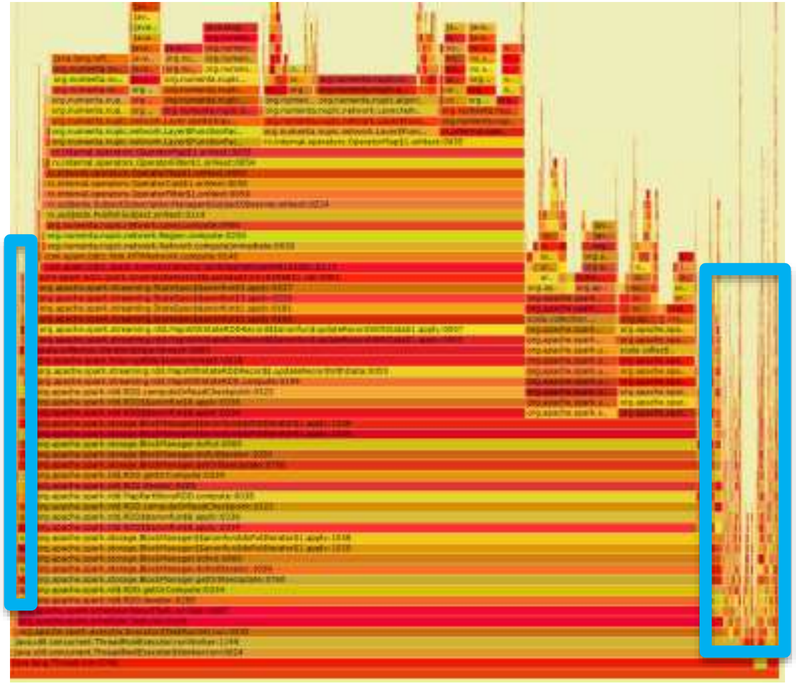
HTM.JAVA: SERIALIZATION IMPROVEMENTS

HTM.java's fast serialization



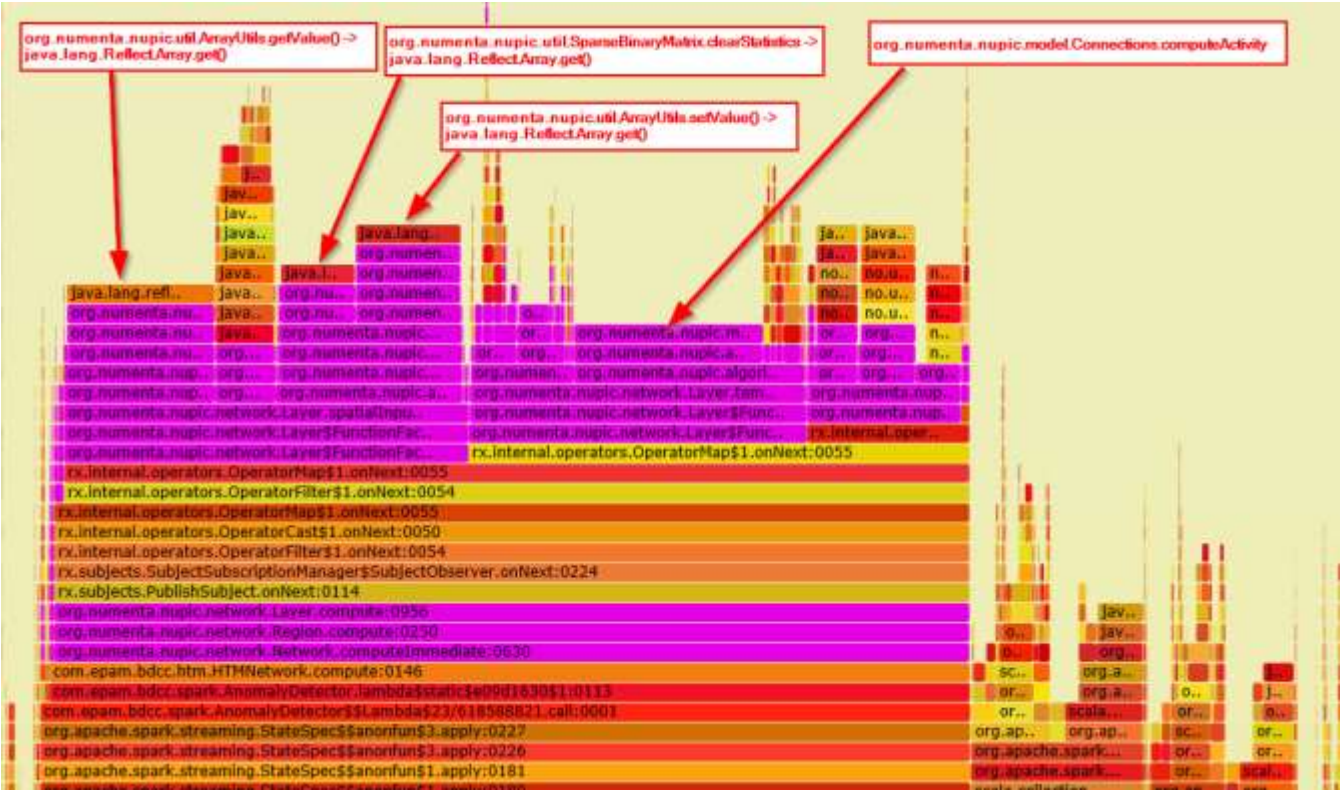
https://github.com/RuedigerMoeller/fast-serialization

Kryo Serialization



HTM.JAVA: WORK WITH ARRAYS

There is still a room for improvement: quick changes with static array manipulation gave 10% increase of performance.



HTM.JAVA VS NUPIC PERFORMANCE

HTM.Java: **198 records/second**

NuPIC (Python wrapper on top of C++ code): **840 records/second**

Words of David Ray (HTM.Java lead):

my guess is that that goal (of 10000 records/sec) is considerably outside of the performance band of the HTM Algorithm (as a non-hardware solution). No promises but I'll do the best I can.

Reference: https://discourse.numenta.org/t/performance-optimization-of-htm-java/2652/14

HARDWARE ACCELERATION

Currently the first steps are done: https://discourse.numenta.org/t/htm-opencl/1708/11
Jonathan Mackenzie - OpenCL: https://github.com/JonnoFTW/htm-cl
Henry Mao - Tensorflow: https://github.com/calclavia/htm-tensorflow - only SP implementation exists.

Parallelisation is available at the level of (following Jonathan Mackenzie **):**

Spatial Pooler

Since each column has many synapses, we can easily parallelise column level operations:

- *Calculating overlap: each column is a single work group, overlap boosting is also done here*
- *Updating permanenses after the set of active columns is decided*
- *Updating boost factors*

Temporal memory

- *Columns can be processed in parallel during inference and learning*

CLA Classifier

Since every time step we request a prediction for requires a new set of table, each step can be done in parallel

- *Updating the moving average for each corresponding on-bit of the input can also be parallelised*

SDR Classifier

- *I'm not familiar with this classifier, but I understand it is intended to replace the CLA classifier*

HARDWARE ACCELERATION

Sheenam Jayaswal & Ziyang Liu made implementation on Adepteva Epiphany https://en.wikipedia.org/wiki/Adapteva
http://meseec.ce.rit.edu/756-projects/fall2014/1-2.pdf

Authors' conclusions:

The HTM algorithm was implemented sequentially as well as in parallel with 16 available cores.

• For the small training set, row-based mapping was found to be most effective.

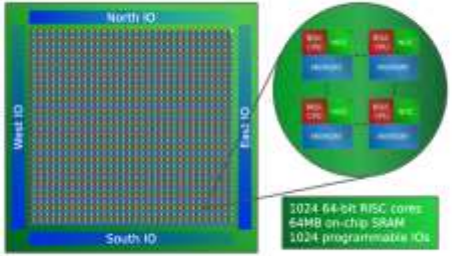
• For the larger training set, all mapping methods were nearly identical.

• Future work can include construction of a multi-layered HTM network.

• Testing complex RGB images can help verify correctness.

• Parallelization at the level of dendrites or synapses could be evaluated.

• Implementation of HTM on FPGAs and GPUs could be done



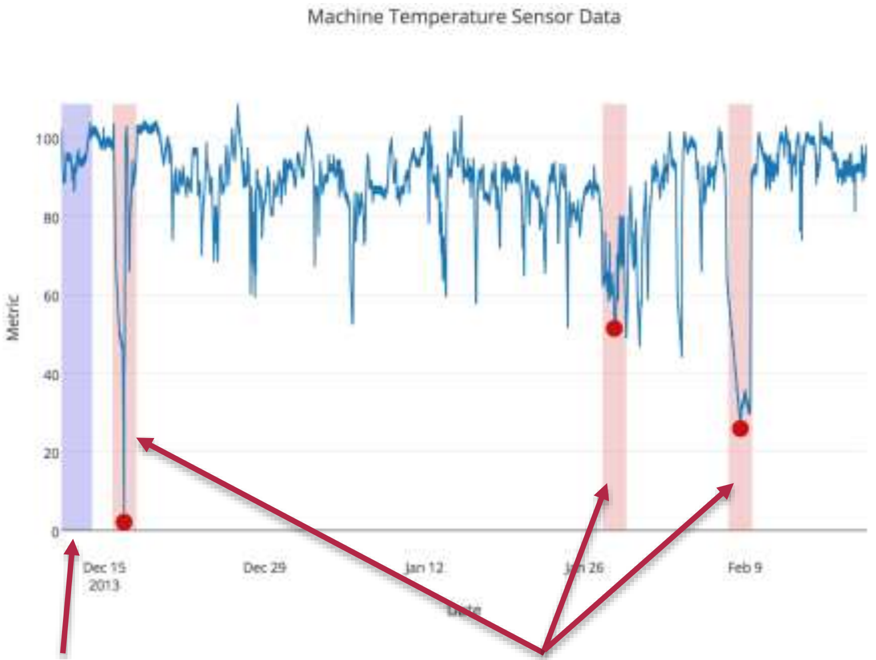
Epiphany V:
1024 64-bit RISC processors
64-bit memory architecture
64-bit and 32-bit IEEE floating point support
64 MB of distributed on-chip SRAM

Parallella Board:
#1 in energy efficiency @ 5W
• 16-core Epiphany RISC SOC
• Zynq SOC (FPGA + ARM A9)
• Gigabit Ethernet
• 1GB SDRAM
• Micro-SD storage

Mapping Method	Max clock cycles	Speedup	Efficiency
Sequential implementation	141,349,367	1.0000	1.0000
Block-Based	19,319,505	7.3164	0.4573
Column-Based	25,949,537	5.4471	0.3404
Row-Based	10,175,309	13.8914	0.8682

NUMENTA ANOMALY BENCHMARK (NAB)

- NAB: a rigorous benchmark for anomaly detection in streaming applications
- Real-world benchmark data set
  - 58 labeled data streams
    (47 real-world, 11 artificial streams)
  - Total of 365,551 data points
- Scoring mechanism
  - Rewards early detection
  - Different “application profiles”
- Open resource
  - AGPL repository contains data, source code, and documentation
  - http://github.com/numenta/NAB

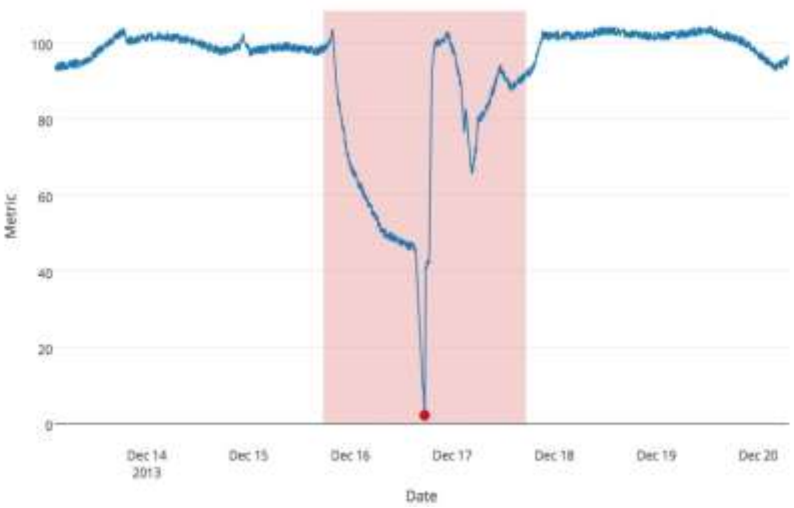


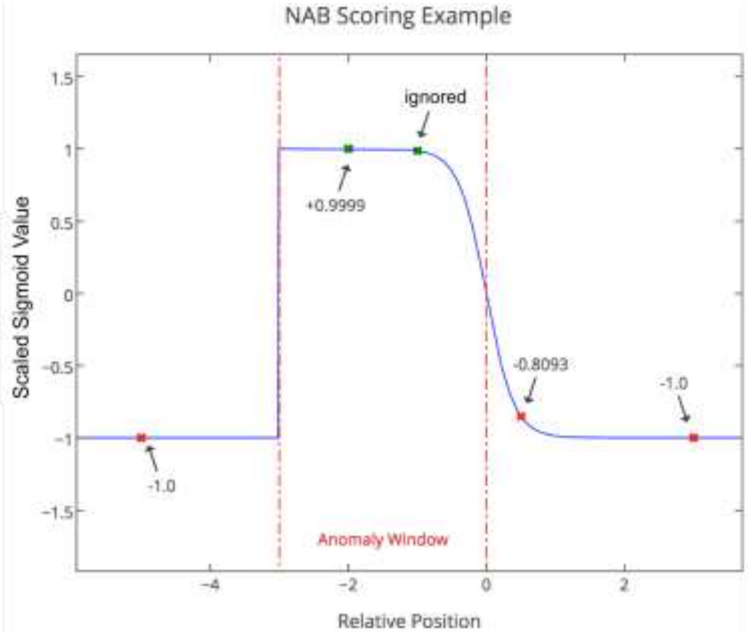
Probationary period: the detector is allowed to learn the data patterns without being tested.

Anomaly windows: detectors are allowed to detect anomaly within windows to get “+” score.

NAB: SCORING CRITERIA

NAB scoring function gives higher score to earlier detections in window:





See details at: https://arxiv.org/abs/1510.03336

NAB: SOME DETAILS

The perfect detector:

- Detects anomalies as soon as possible
- Provides detections in real time
- Triggers no false alarms
- Requires no parameter tuning
- Automatically adapts to changing statistics

Scoring methods in traditional benchmarks are insufficient:

- Precision/recall does not incorporate importance of early detection
- Artificial separation into training and test sets does not handle continuous learning
- Batch data files allow look ahead and multiple passes through the data

Application profiles:

- Standard
- favor low false positives
- favor low false negatives.

Profiles assign different weightings based on the tradeoff between false positives and false negatives. E.g. EKG data on a cardiac patient favors False Positives, IT/DevOps professionals hate False Positives.

NAB emulates practical real-time scenarios:

- Look ahead not allowed for algorithms. Detections must be made on the fly.
- No separation between training and test files. Invoke model, start streaming, and go.
- No batch parameter tuning. Must be fully automated with single set of parameters across data streams. Any further parameter tuning must be done on the fly.

NAB: RESULTS

When	Sun, Jul 24, 2016 – Fri, Jul 29, 2016
Where	IEEE WCCI 2016 (World Congress on Computational Intelligence) Vancouver, British Columbia Canada
Web	Event Website
Topic	Numenta Anomaly Benchmark Competition for Real-time Anomaly Detection

Detector	Standard Profile	Reward Low FP	Reward Low FN
Perfect	100	100	100
HTM AL	70.1	63.1	74.3
CAD OSE+	69.9	67.0	73.2
nab-comportex+	64.6	58.8	69.6
KNN-CAD+	58.0	43.4	64.8
Relative Entropy	54.6	47.6	58.8
HTM PE	53.6	34.2	61.9
Twitter ADVec	47.1	33.6	53.5
Etsy Skyline	35.7	27.1	44.5
Sliding Threshold	30.7	12.1	38.3
Bayesian Changepoint	17.7	3.2	32.2
EXPoSE	16.4	3.2	26.9
Random	11	1.2	19.5
Null	0	0	0

Detector	Latency (ms)	Spatial Anomaly	Temporal Anomaly	Concept Drift	Non Parametric	NAB Score
HTM	11.3	✔	✔	✔	✔	70.1
Relative Entropy	0.05	✔	✔	✔	✔	54.6
Twitter ADVec	3.0	✔	✔	✔	✗	47.1
Etsy Skyline	414.2	✔	✗	✗	✗	35.7
Sliding Threshold	0.4	✔	✗	✗	✗	30.7
Bayesian Changepoint	3.5	✔	✗	✔	✗	17.7
EXPoSE	2.6	✔	✔	✔	✔	16.4

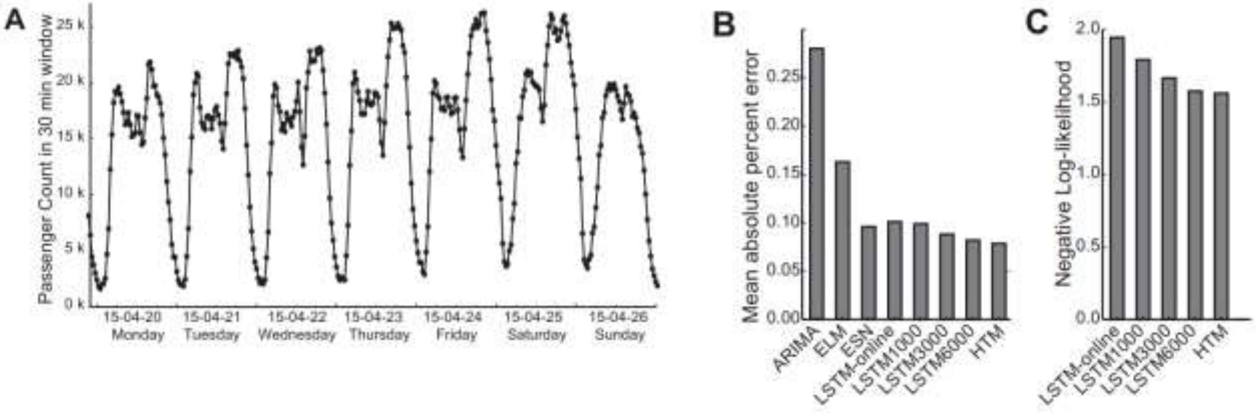
References:
https://numenta.com/events/2016/07/24/numenta-anomaly-benchmark-competition-at-ieee-wcci-2016/
http://www.sciencedirect.com/science/article/pii/S0925231217309864?via%3Dihub

COMPARISON WITH ARIMA, LSTM, ELM, ESN

https://arxiv.org/ftp/arxiv/papers/1512/1512.05463.pdf

The authors compared HTM with other sequence learning algorithms in the prediction task, including statistical methods:

- ARIMA - autoregressive integrated moving average
- ELM - feedforward neural networks: online sequential extreme learning machine
- LSTM - recurrent neural networks: long short-term memory
- ESN - echo-state networks



Prediction of the New York City taxi passenger data.
A. Example portion of taxi passenger data (aggregated at 30 min intervals). The data has rich temporal patterns at both daily and weekly time scales.
B-C. Prediction error of different sequence prediction algorithms using two metrics: mean absolute percentage error (B), and negative loglikelihood (C).

CONTACTS

Ihor Bobak
E-mail: ibobak@gmail.com
Skype: ibobak