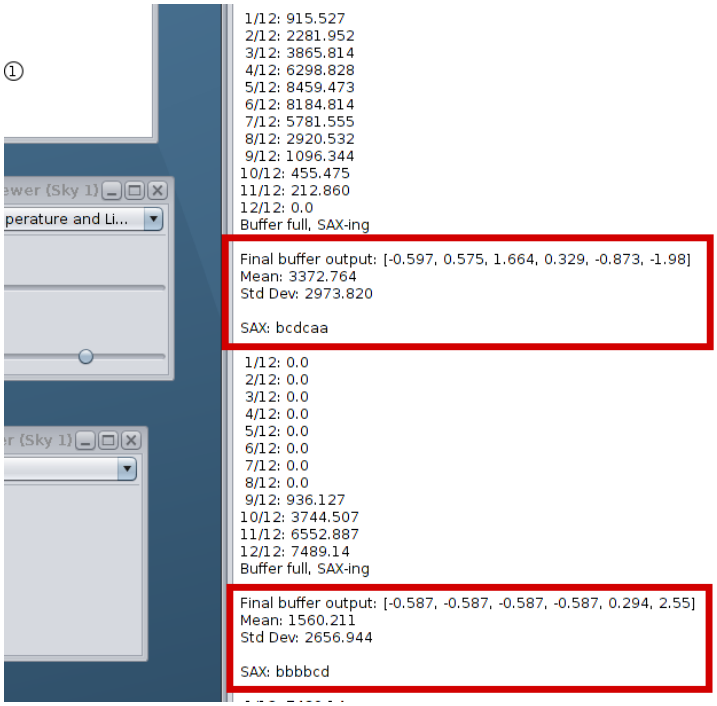

Figure 1: Demonstration of SAX aggregation with window size of 2 and alphabet of length 4

1 Description

Symbolic Aggregation Approximation (SAX) was implemented as an in-network data processing technique, compressing the representation while allowing further processing on this symbolic string. Figure 1 shows two rounds of SAX output following data collection, a window size of 2 was used and an alphabet of length 4, i.e the characters `a` through `d` inclusive.

The use of string representation allows further processing and analysis techniques to be used such as string pattern matching, Euclidean distance and hashing operations.

It is also an opportunity to reduce the required memory footprint. 12 C `floats` total 48 bytes of data, this can be reduced by a factor of 4 using `chars` instead, a window size of 2 further halves the number of output samples and lowers the required memory to just 6 bytes.

2 Specification

SAX is implemented in two stages, that of transforming the time-series into Piecewise Aggregate Approximation (PAA) representation and then representing this numeric series with a symbolic alphabet.

2.1 PAA

PAA is an effective method for reducing the dimensionality of a time-series by focusing on the trends and patterns of the data as opposed to individual values. It is a lossy operation that can be used to strike a balance between frequent periodic sampling in order to keep the system responsive while reducing the storage and processing requirements for such a large data stream. This process is completed in two steps, Z-normalisation and aggregation.

Z-Normalisation The standard deviation and mean of the data series were first calculated using previously written functionality to calculate these values for arbitrary arrays of numbers. This normalisation process takes a series of data and transforms it such that the output series has a mean of 0 and a standard deviation of 1. This changes the context of the values from being measured in lux to being a measure of a samples distance from the mean, 0, in standard deviations. This allows (somewhat) direct comparison of different time-series.

Aggregation Following Z-normalisation, the size of the series is reduced by applying a windowing function. This takes subsequent equally-sized groups of samples and reduces the group to the mean of those values, reducing the length of the series by a scale factor equal to the size of the group.

Following Z-normalisation and aggregation, the original time series has been reduced to a given length of samples with a mean of 0 and standard deviation of 1.

2.2 SAX

SAX is an extension to the PAA representation that uses an alphabet of symbols instead of numeric values. Following Z-normalisation as part of the PAA process, a time-series of data will follow a Gaussian distribution profile. Each value describes how many standard deviations it is away from the mean of the series (how far away from the central Gaussian peak it is), an approximation of the value could be found by dividing the area of the Gaussian profile into segments and referring to each by a character. Each data value can now be described by a segment identifier. These segments should not be of equal width, however - values are likely to be closer to the mean, referring to these by a single character would be unproductive. Instead the Gaussian profile is divided into segments corresponding to equal probabilities or areas under the curve.

These segments are realised using breakpoints, the standard deviations that describe the edges of each segment. By comparing each datum to subsequent breakpoints the segment that the value lies within can be identified and the corresponding character retrieved for representation.

3 Implementation

The SAX functionality was added as an alternative buffer rotating mechanism over the original 12-to-1/4-to-1/12-to-12 aggregation system. This rotation mechanism lies between receiving the full data buffer on the processing thread and passing it to the `handleFinalBuffer(buffer)` function for display.

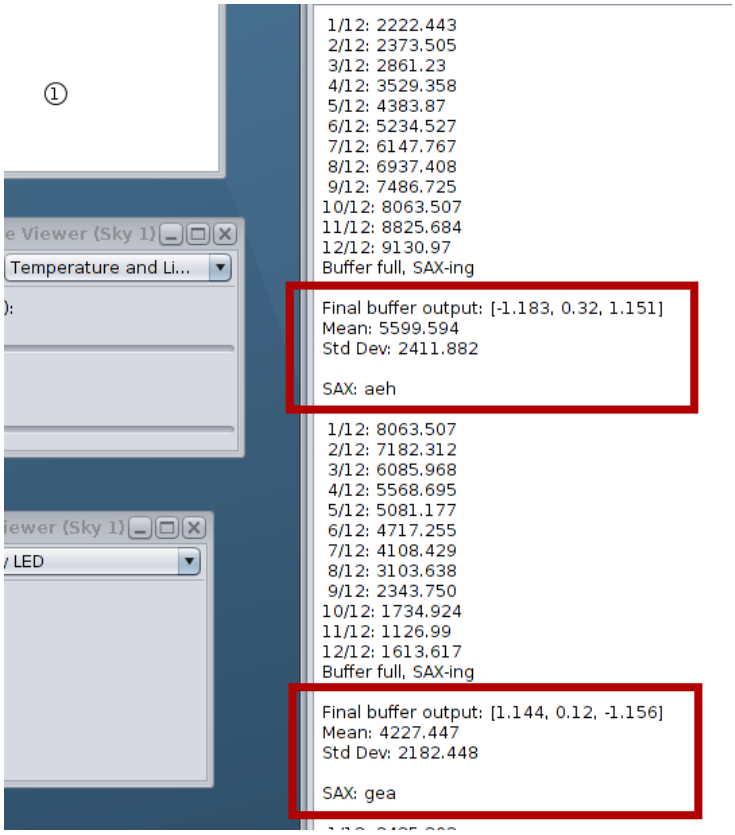

Figure 2: Demonstration of SAX aggregation with window size of 4 and alphabet of length 8

The length of the output buffer is calculated using the full data buffer's length and the group size with which it is divided. This size is used to allocate a new buffer to store the PAA representation of the data.

From here the input buffer is Z-normalised using the `normaliseBuffer(buffer)` function from the `sax.h` header. This function iterates over each value in the buffer, subtracts the buffer's mean and then divides by the standard deviation (the mean and standard deviation are stored as members of the buffer prior to passing to the function).

Following this, the buffer is aggregated using the same 4-to-1 aggregation function `aggregateBuffer(bufferIn, bufferOut, groupSize)` used previously. This functionality was used as the group size is variable and the same required windowing and average function is used, as such it could be reused with the desired aggregation level. Figure 2 shows an output using a window size of 4 instead of figure 1's width of 2. The output from this function represents the PAA form of the initial data series.

The `handleFinalBuffer(buffer)` function takes a `Buffer` struct as input which is defined as being a collection of `floats`. In order to maintain this structure and compatibility with the non-SAX aggregation, the buffer is passed to this function in PAA form without SAX conversion to a string. In order to complete the system, the buffer must be *stringified* within this final method following a pre-processor check that SAX is being used.

SAX symbolic representation is completed using the `stringifyBuffer(buffer)` function of the `sax.h` header. This function allocates a string of suitable size before iterating over each value of the buffer and calling `valueToSAXChar(inputValue)` to retrieve the corresponding `char`. As the breakpoints are a constant for a given number of segments and would require computation, the values for the breakpoints are defined by the pre-processor based on the number of segments defined by the `SAX_BREAKPOINTS` macro.

For each value, the breakpoints are iterated over. Specific cases are defined for the beginning and end of the breakpoints as these are one-sided inequalities. For the rest, the value is compared to two neighbouring breakpoints. A true condition for any of these checks indicates that the correct segment for the value has been identified. The same return value, `SAX_CHAR_START + i`, is used in every case. `SAX_CHAR_START` is a macro used to define the first character of the alphabet being used for SAX representation (likely either `'a'` or `'A'`), `i` is the iteration variable for the loop, it is used as an offset from the alphabet start and is evaluated to a `char` for return.