Anomaly detection using deep learning to measure quality of Large Datasets

-Sridhar Alla , Syed Nasar

Strata
DATA CONFERENCE

Speakers

- Sridhar Alla is Co-founder & CTO at BlueWhale, a AI+BigData solution provider.
- Author of Beginning Anomaly Detection Using Python Deep Learning – Apress 2019

- Syed Nasar, Solutions Architect at Cloudera.
- Cloudera ML Specialization Team
- Founder of Nashville Artificial Intelligence Society.

@techmazed

Agenda

- Detecting Anomalies in Datasets and ETL processes
  - Thinking Data Quality
  - Types of anomalies
  - Challenges in Anomaly Detection
  - Demo - A simple anomaly detection
- Machine Learning Models
- Deep Learning Models

Detecting Anomalies for Data Quality

Anomalies in Datasets and ETL processes

Thinking Data Quality

Data quality issues in big data space

Six Vs:

- **Volume**
- **Velocity**
- **Variety**
- **Veracity**
- **Variability**
- **Value**

DQAF include

- **Completeness**
- **Validity**
- **Timeliness**
- **Consistency**
- **Integrity**

Types of anomalies

Anomalies can be classified into following categories

- Point Anomalies
  - If one object can be observed against other objects as anomaly, it is a point anomaly. This is the simplest anomaly category and a lot of researches include them.
- Contextual Anomalies
  - If object is anomalous in some defined context. Only in this case it is contextual anomaly **also known as conditional anomaly*
- Collective Anomalies
  - If some linked objects can be observed against other objects as anomaly. Individual object can’t be anomalous in this case, only collection of objects.

Data quality issues

A few examples of data quality issues

- Dataset level
  - Format, delimiters, metadata
- Column level
  - Cardinality
- Record level
  - Missing, duplicates, extreme, range, business rule, derived columns
- Message level
  - Metadata, Malformed, Validating source
- ETL Level
  - Process failure, transformation failures

Representing Data Quality

How to define and capture data quality aspects

Types of metrics

- The ratio of data to errors
- Number of empty values
- Data transformation error rates
- Amounts of dark data

Common methods for evaluating data quality

Data quality related algorithms

- **Dixon's Test** - assumes normal distrib.
- **Rosner's Test** - assumes normal distrib.; records > 25
- **Grubb's Test** - univariate dataset ; normal distrib.
- **Regression analysis**
- **Autoregressive integrated moving average (ARIMA) -** time series; streams
- **Various statistical control charting -** individuals, moving range, moving average, exponentially weighted moving average, etc.
- **Various non-parametric approaches -** kernel function based, histogram based

Challenges

Why is it a difficult problem

- Avoid alert black-holes
  - Option one is to generate alerts, and option two is to record events for later analysis but don’t alert on them.
  - Anomalies are not always rare.
  - A naive approach to alerting on anomalies is almost certain to cause a lot of noise.
- Getting labels is a challenge
- Minimizing false positives
- No general purpose approach
- Flavors vary from heuristics, statistics to Machine Learning to Deep Learning

DEMO

A simple approach to Anomaly Detection
with
Cloudera Data Science Workbench (CDSW)

TDDA

Test Driven Data Analysis

- TDDA is a methodology and software for improving quality for improving robustness of analytic processes
- One can repurpose TDDA's data verification capabilities as an anomaly detection framework

Ref: https://tdda.readthedocs.io/en/tdda-1.0.23/

Dataset in the example

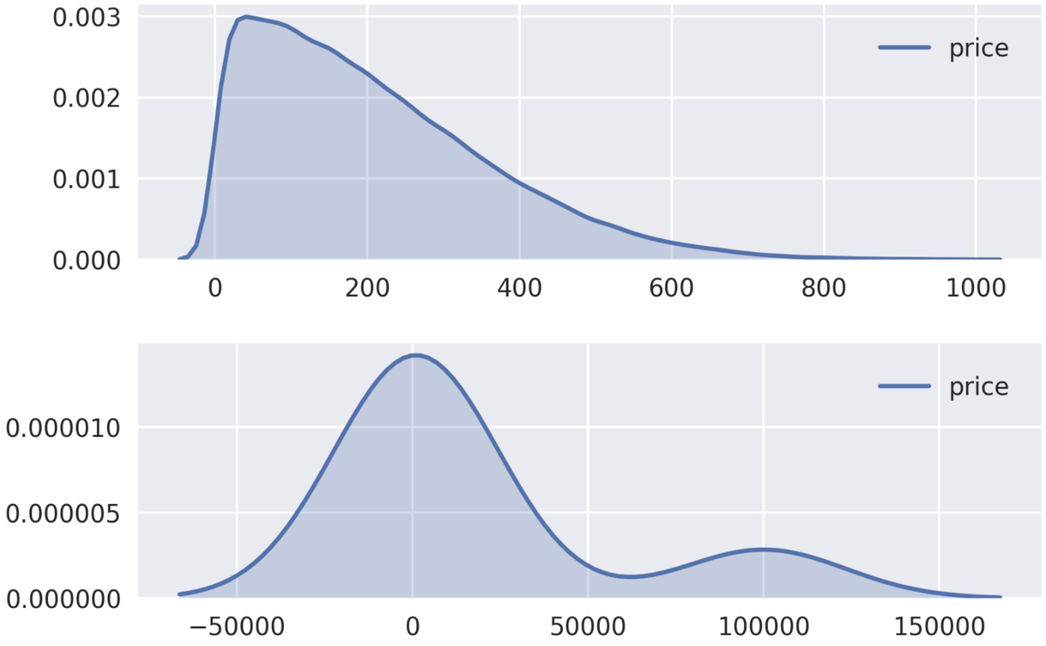
TDDA toy dataset

- Price field
  - Range of values accepted?
  - Datatype?
  - Permitted null values?
  - Duplicates allowed?
- Category field
  - Accepted categories?
  - Format of the category?

	id	category	price
0	710316821	QT	150.39
1	516025643	AA	346.69
2	414345845	QT	205.83
3	590179892	CB	55.61
4	117687080	QT	142.03

Characteristics of the Price attribute

- Left skewness
- Large tail
- Non-normally distributed



TDDA rules manual & automated rules

```
  "fields": {
      "id": {
          "type": "int",
          "max_nulls": 0,
          "no_duplicates": true
      },
      "category": {
          "type": "string",
          "max_nulls": 0,
          "allowed_values": ["AA", "CB", "QT", "KA", "TB", "TQ"]
      },
      "price": {
          "type": "real",
          "min": 0.0,
          "max": 1000.0,
          "max_nulls": 0
      }
```

Handcrafted Rules

```
    "fields": {
        "id": {
            "type": "int",
            "min": 100000546,
            "max": 899995057,
            "sign": "positive",
            "max_nulls": 0,
            "no_duplicates": true
        },
        "category": {
            "type": "string",
            "min_length": 2,
            "max_length": 2,
            "max_nulls": 0,
            "allowed_values": [
                "AA",
                "CB",
                "KA",
                "QT",
                "TB"
            ],
            "rex": [
                "^[A-Z]{2}$"
            ]
        },
        "price": {
            "type": "real",
            "min": 0.0,
            "max": 985.18,
            "sign": "non-negative",
            "max_nulls": 0
        }
    }
```

Discovered Rules

Detected items upon applying manual & automated rules

Output of Handcrafted Rules

	id	category	price	id_nodups_ok	category_values_ok	price_max_ok	price_nonnull_ok	n_failures
Index								
3100	102829374	AA	65.24	False	True	True	True	1
8669	720313295	TB	1004.72	True	True	False	True	1
14110	384044032	QT	478.65	False	True	True	True	1
15322	602948968	TB	209.31	False	True	True	True	1
19941	105983384	AA	8.95	False	True	True	True	1

	id	category	price	id_nodups_ok	category_min_length_ok	category_values_ok	category_rex_ok	price_max_ok	price_nonnull_ok	n_failures
Index										
40	113791348	TQ	318.63	True	True	False	True	True	True	1
3100	102829374	AA	65.24	False	True	True	True	True	True	1
8669	720313295	TB	1004.72	True	True	True	True	False	True	1
14110	384044032	QT	478.65	False	True	True	True	True	True	1
15322	602948968	TB	209.31	False	True	True	True	True	True	1

Output of Discovered Rules

Machine Learning and Deep Learning techniques

Anomaly Detection

- D -Given Data Set consisting of points d1,d2…..dn
- N- Normal Data points= d1,d4,……dn-1
- A -Abnormal/Anomaly Data points = d2,d3….dn
- How do we find A from D given N (labeled or unlabeled)

Machine Learning Algorithms

- **Isolation Forest** – an unsupervised learning algorithm that is good for high-dimensional data. After randomly selecting a feature, it picks a random value between the selected feature's maximum and minimum values to 'isolate' observations.

- **One-Class Support Vector Machine** – Depending on the implementation, it is an unsupervised or semi-supervised learning algorithm that is also suited for high-dimensional data. Can be applied to novelty detection, the classification of new data as outlier or normal after training on normal data only.

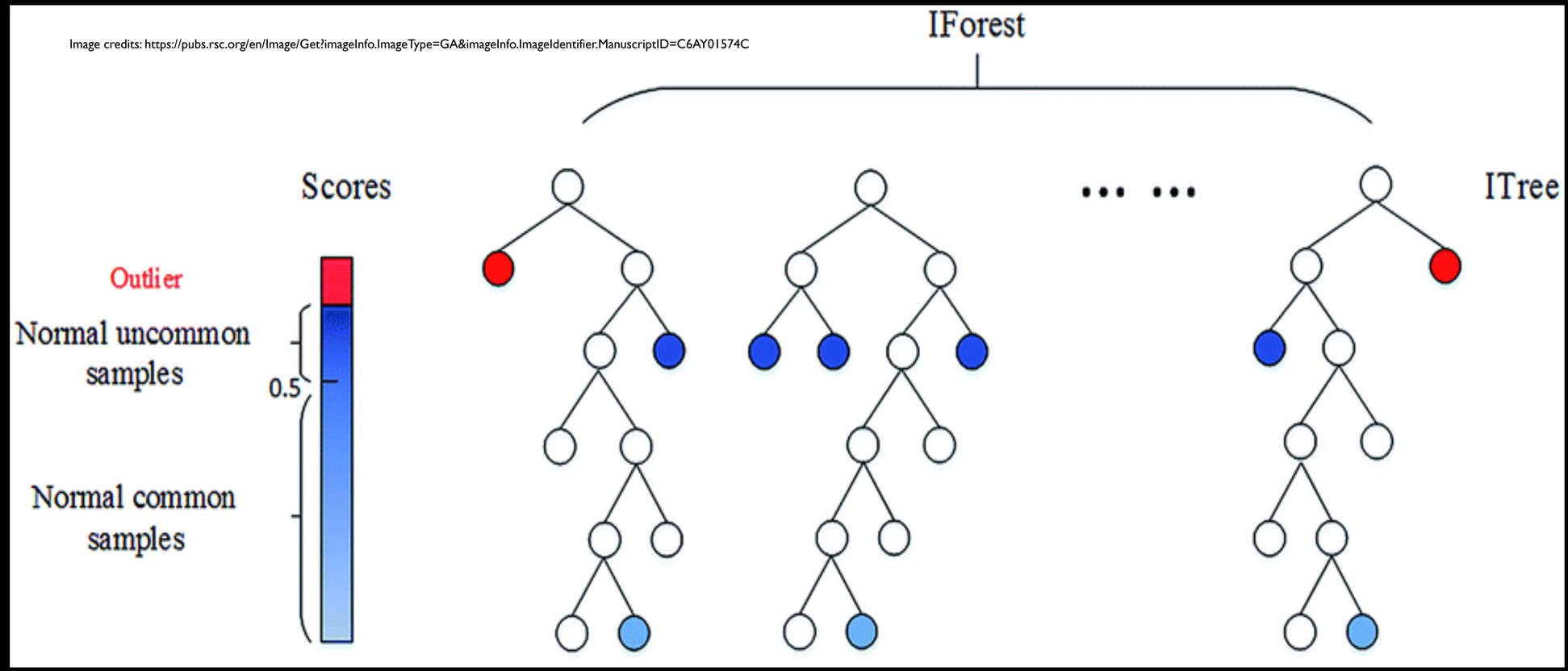
- **Mahalanobis distance** – Measures distance relative to the centroid (intersection of all the variables).

Isolation Forest

- Recursively partitions the data using a tree structure.
- The distance from the node to the root can be used as a measure of normality.
- Following that premise, when a collection of trees produce noticeably short paths for a certain data point, it is much more likely to be an anomaly.
- In general, anomalies tend to have the shortest path length because less partitions are needed to isolate them. Meanwhile, normal data requires more partitions to become isolated, hence they have longer path lengths to the root.

Isolation Forest

- From this graph of an isolation forest, the general trend of anomalous data being closer to the root node can be observed.

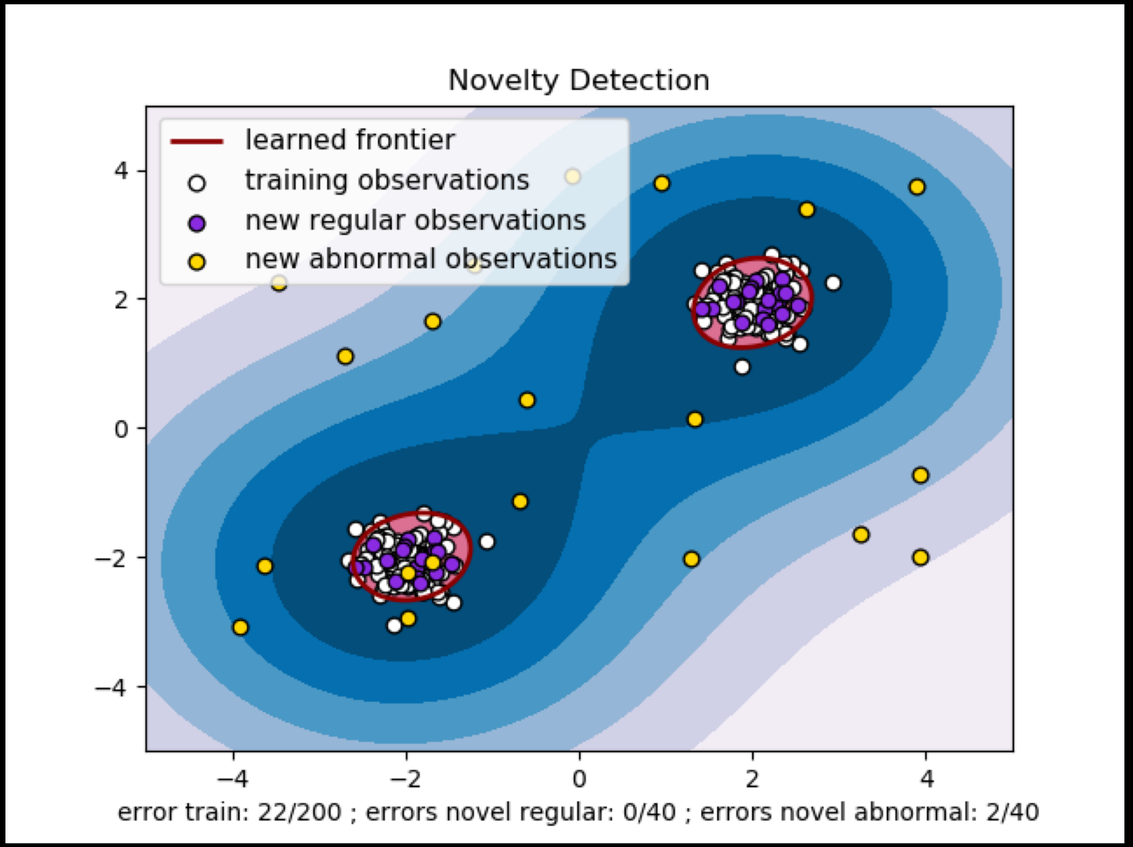


One-Class SVM

- Support vector machine model adapted to train on only one class, which in this case is the normal class.
- After being trained only on normal data, the SVM can be used for the task of novelty detection to classify a new data point as normal or as an anomaly.
- Training is about learning the frontier. Any points within the space enclosed by the frontier can be considered normal, while points that lie outside that space can be stated to be anomalous.

One-Class SVM

- From this graph, the learned frontier boundaries of the one-class SVM can be observed. With the frontiers approximating the training data set, predictions of new data points either lie inside or outisde the boundary.

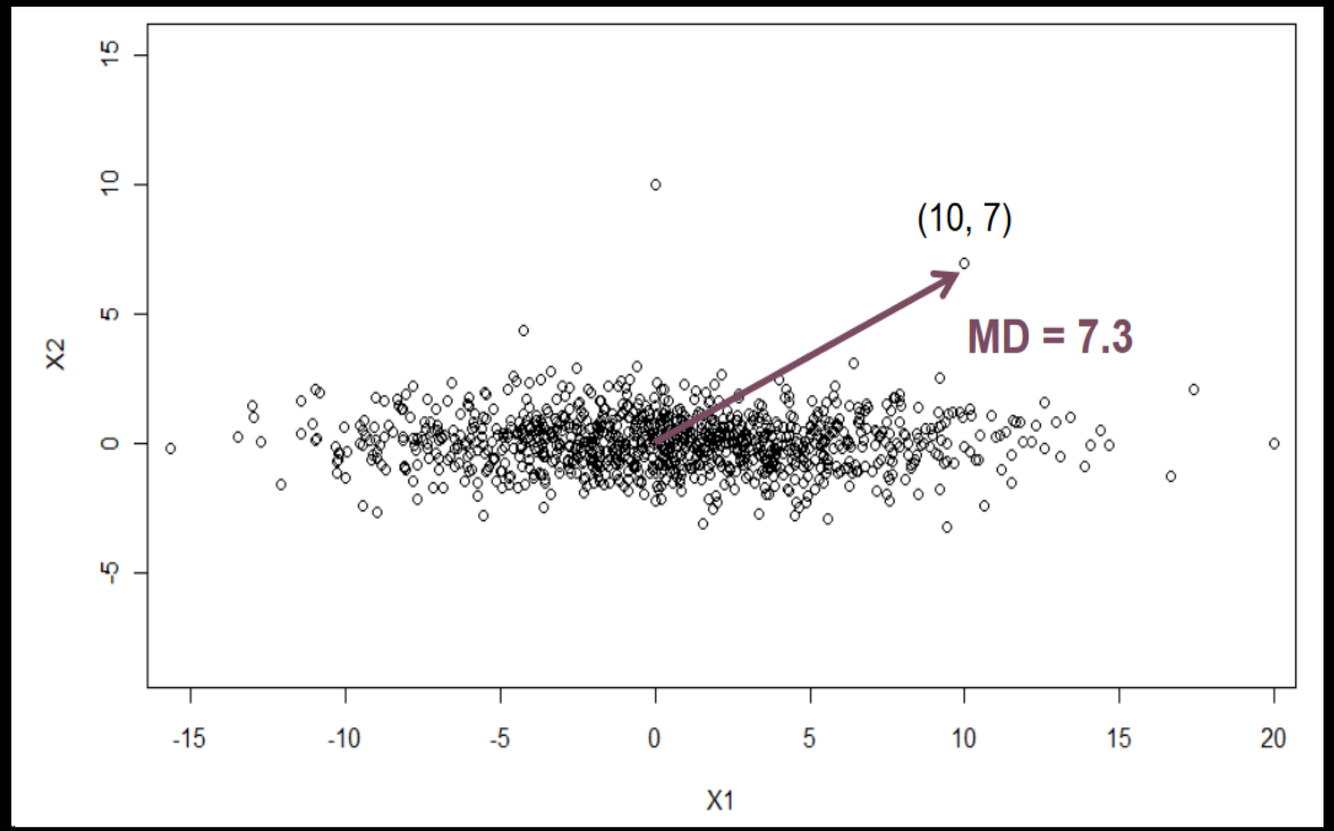


Mahalanobis Distance Ranked Algorithms

- Mahalanobis distance is the distance between a data point and the centroid, the point where the means from all variables intersect.
- After being trained only on normal data, the SVM can be used for the task of novelty detection to classify a new data point as normal or as an anomaly.
- Training is about learning the frontier. Any points within the space enclosed by the frontier can be considered normal, while points that lie outside that space can be stated to be anomalous.

Mahalanobis Distance Ranked Algorithms

- Calculation of the Mahalanobis distance is shown here in the graph. In two dimensions, it is simply the Euclidean distance between a given point and the centroid of the distribution. This same concept can be extended to multiple dimensions.



Deep Learning techniques

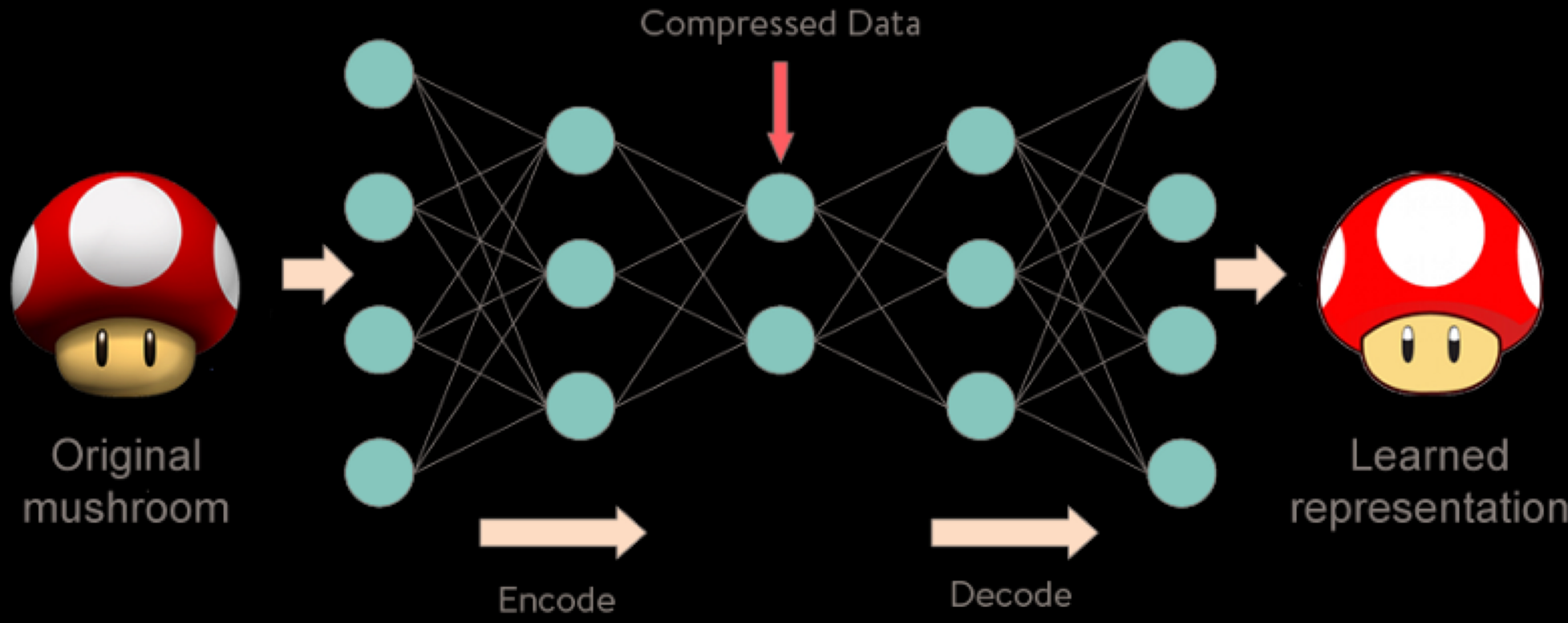
- **Autoencoders –** Commonly used in semi-supervised learning, and is the core structure for unsupervised deep learning algorithms. Compresses the training data and learns to reconstruct the original data from this encoded representation. Variations known as denoising autoencoders and variational autoencoders can also be applied to anomaly detection.
- **LSTM / RNNs –** For any data involving sequences or a time-series, LSTMs and RNNs are ideal. Can be built following an autoencoder architecture to allow for unsupervised anomaly detection.
- **Temporal Convolutional Network –** A relatively new architecture that shows great potential in applications to problems involving sequential data. Can be used to detect anomalies in image data.
- **Boltzmann Machines –** Several types of boltzmann machines including restricted boltzmann machines, deep boltzmann machines and deep belief networks can all be applied to anomaly detection.

Autoencoders

- Learns efficient data encodings in an unsupervised manner. Essentially compresses the input data and decompresses it in an attempt to reconstruct the data as accurately as possible.
- There are a variety of approaches to anomaly detection with autoencoders. One example would be a semi-supervised approach to novelty detection in which the model is trained only on normal data. When generating predictions on new points, normal data would be classified based on low reconstruction errors. On the other hand, anomalies would result in high reconstruction errors.
- In unsupervised anomaly detection using a variation autoencoder for example, anomalies would be predicted based on an anomaly score calculated from the probability of seeing such a data point x on a probability distribution p(x).
- For denoising autoencoders, they can be trained on noisy samples given the corresponding noiseless sample as the output. This way, anomalies as well as the channels causing the error can clearly be identified.

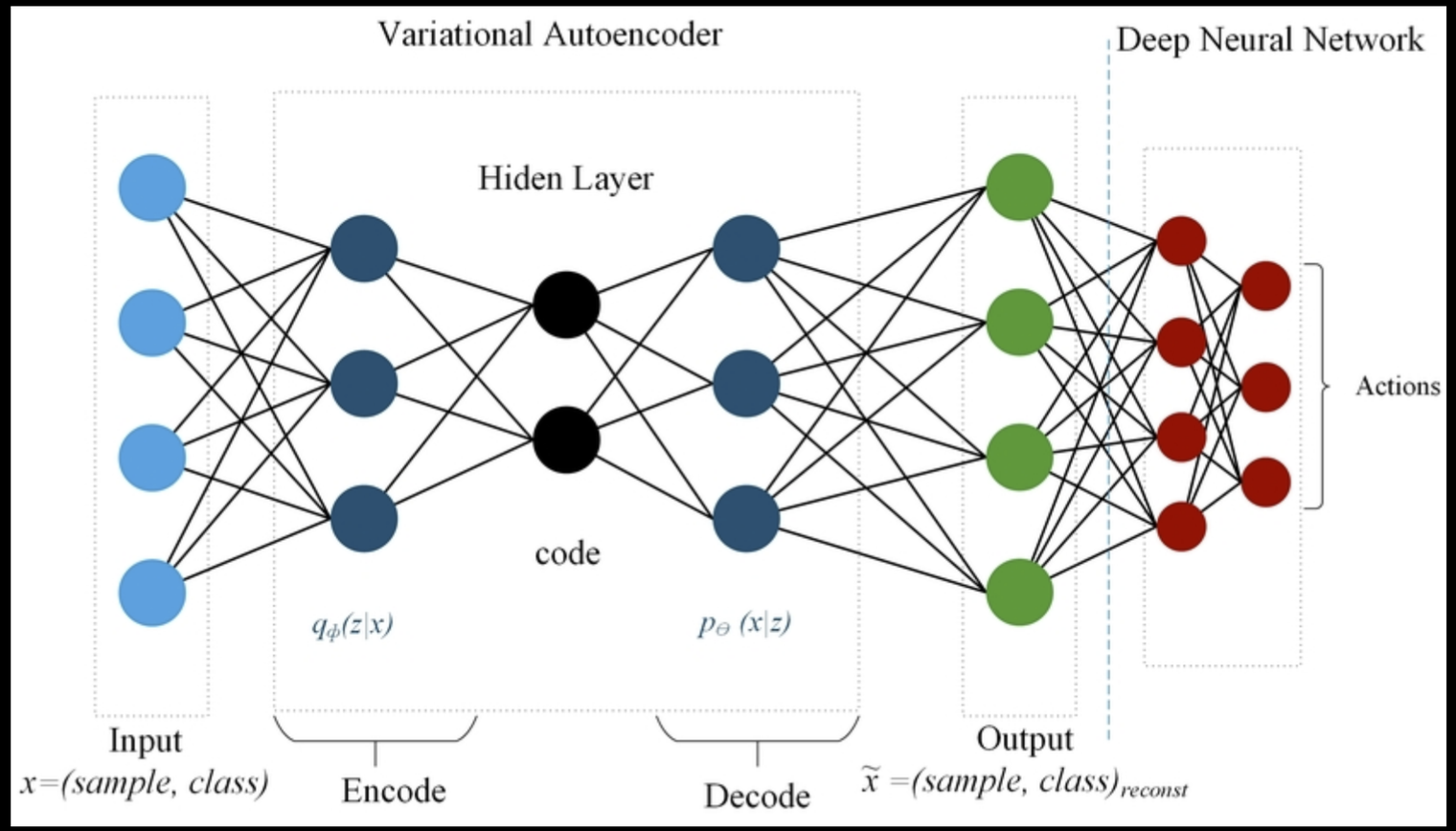
Autoencoders

- Essentially, the model compresses the original data down in the encoder stage and attempts to reconstruct the original data as accurately as possible in the decoder stage.



Variational Autoencoders

- Instead of taking the latent vector(encoded/compressed data), take a sample of the distribution (mean & standard deviations).

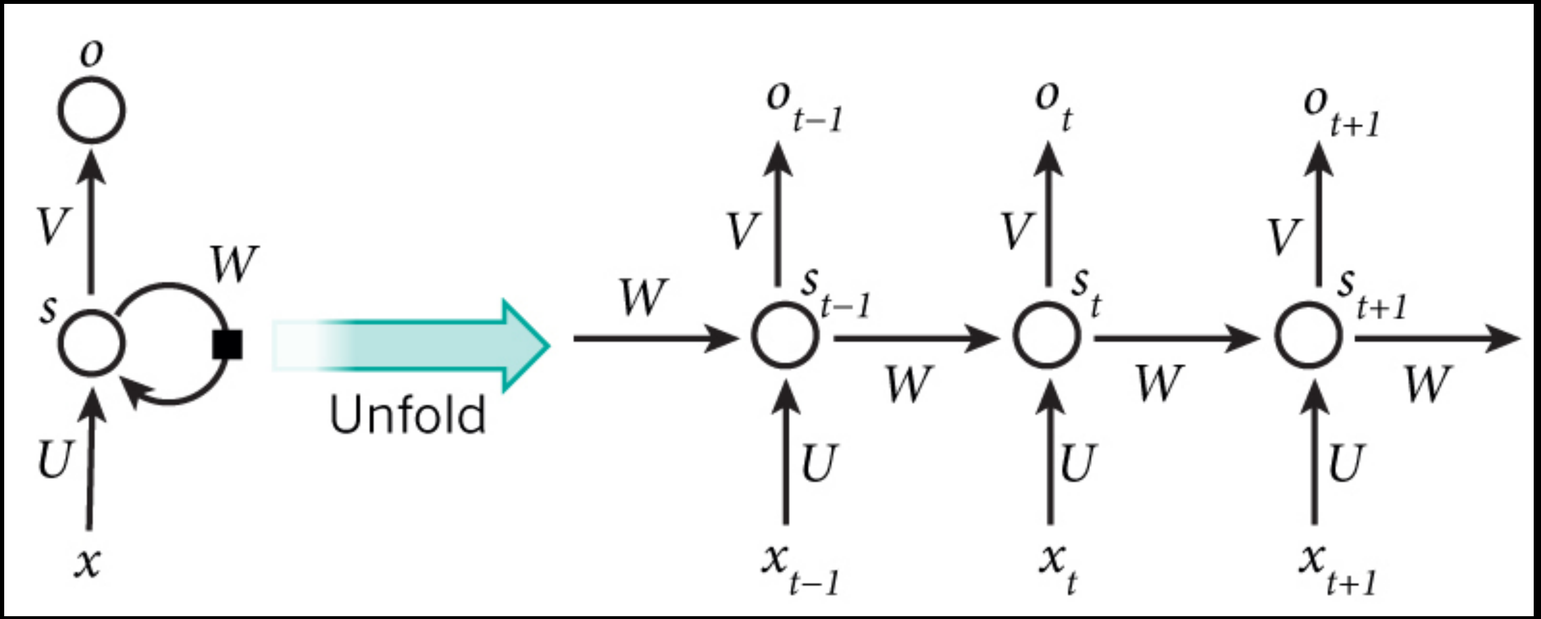


LSTMs / RNNs

- An LSTM is a type of RNN that can be used in time-series based anomaly detection. Essentially, an LSTM would train on time-series data and attempt to predict future time-steps. If there is a high error in the prediction, then it is likely to be an anomaly.
- LSTMs or other RNN based models can be integrated into an autoencoder style architecture to perform unsupervised anomaly detection. The option also exists to pair an LSTM with another type of network such as a CNN. In this case, the CNN would act as the encoder, and the LSTM as the decoder.
- While RNNs themselves can be used for anomaly detection, the LSTM offers better performance for many tasks.

LSTMs / RNNs

- Unfolded RNN network. S represents the hidden state at time t, x is the input at time t, and o is output at time t.



Temporal Convolutional Network

- Encoder-decoder temporal convolutional networks and dilated temporal convolutional networks (based on Google's Wavenet) are two examples of architectures involving temporal (one dimensional) convolutions.
- Dilated TCNs train significantly faster than LSTMs because they can take advantage of parallel computing on the GPU. Additionally, the dilated TCN can be applied to problems involving time-series, offering significantly improved performance compared to the LSTM in some cases.

Temporal Convolutional Network

- The dilated temporal convolutional network is named so because of the dilation involved within its layers, as seen in the graph above.

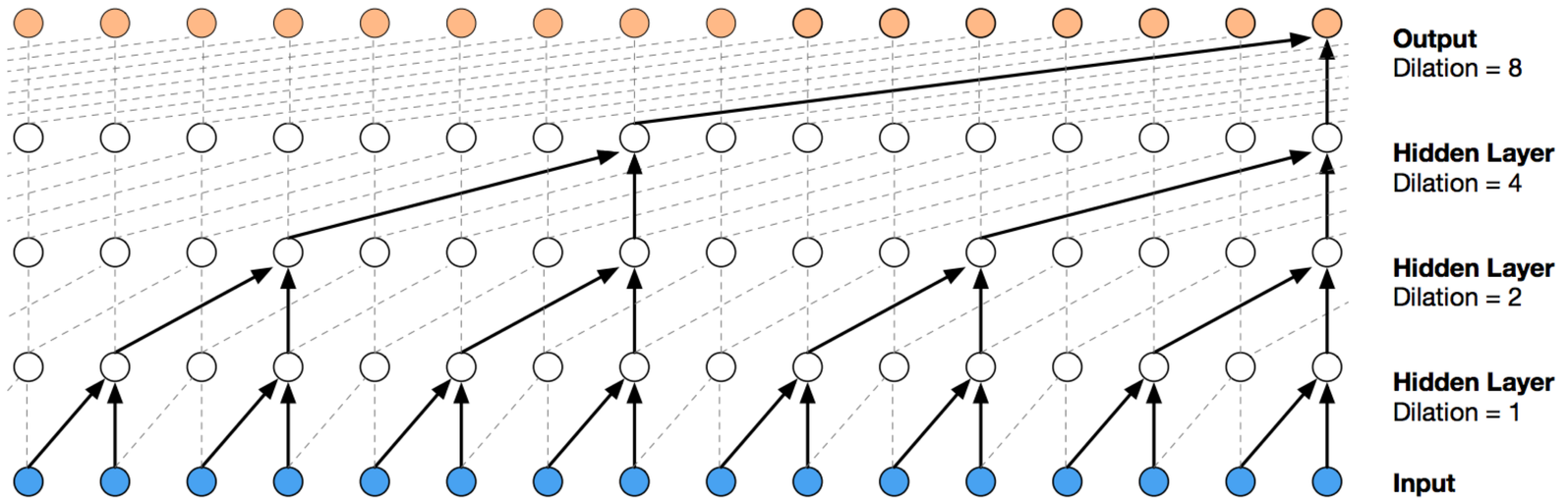


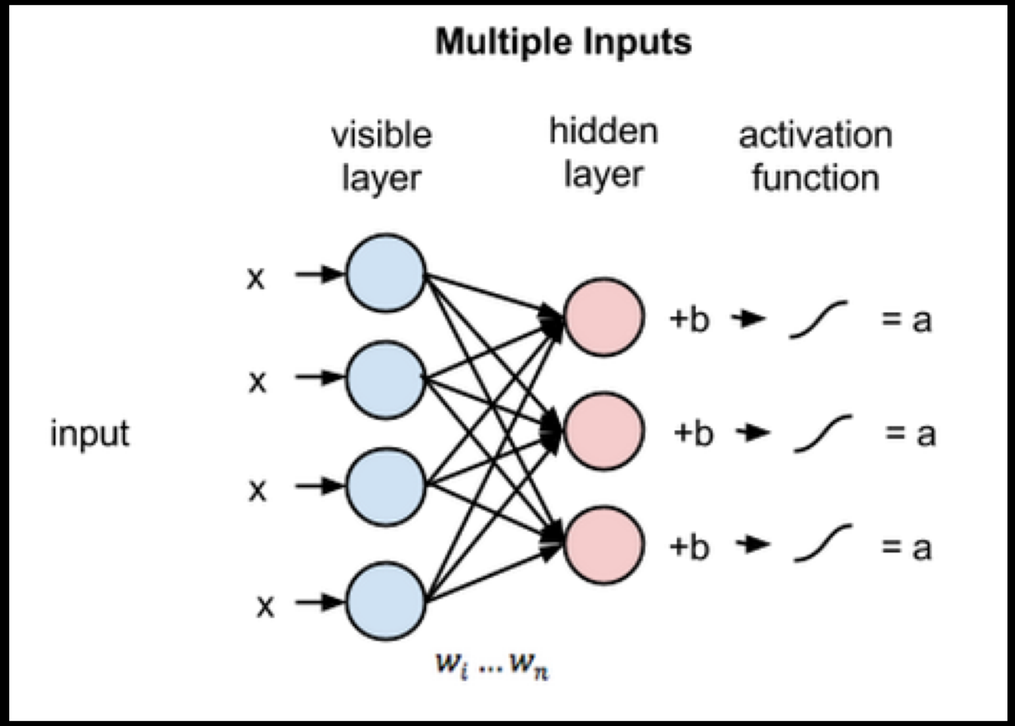
Figure 3: Visualization of a stack of *dilated* causal convolutional layers.

Boltzmann Machines

- A basic restricted Boltzmann machine consists of a visible layer and a hidden layer. Both layers can either be binary or have the visible layer be Gaussian and the hidden layer be Bernoulli.
- The free energy function can be derived from the RBM's energy function. The higher the free energy of some x, the higher the chance that x is an anomaly. Essentially, the RBM is stochastic, meaning it represents the data it trained on through a probability distribution. Anomalies would be detected by low probability scores, which are marked by high free energy by the free energy function.
- A deep boltzmann machine is similar to an RBM, but is an undirected network and has multiple hidden layers. Since the DBM is also a probablistic model, it would detect anomalies in a similar fashion to the RBM.
- Deep belief networks (DBN) are a generative, stochastic model that is comprised of stacks of RBMs.

Boltzmann Machines

- A restricted Boltzmann machine. The RBM is also generative, meaning it can generate new samples from what it has learned during the training process. By reversing the flow of data to start at the hidden layer and end at the visible layer, the RBM can generate reconstructions of the original input.



Autoencoder

In [2]:

```
# this is our input
input_img = Input(shape=(784,))
# "encoded" is the encoded representation of the input
encoded = Dense(encoding_dim, activation='relu')(input_img)

#this is the midpoint

# "decoded" is the lossy reconstruction of the input
decoded = Dense(784, activation='sigmoid')(encoded)

# this model maps an input to its reconstruction. the loss determines if this is anomaly or just little different
autoencoder = Model(input_img, decoded)

autoencoder.summary()
```

```
WARNING:tensorflow:From /Users/salla/miniconda3/lib/python3.7/site-packages/tensorflow/python/framework/op_def_library.py:263:
colocate_with (from tensorflow.python.framework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 784)               0
_________________________________________________________________
dense_1 (Dense)              (None, 32)                25120
_________________________________________________________________
dense_2 (Dense)              (None, 784)               25872
=================================================================
Total params: 50,992
Trainable params: 50,992
Non-trainable params: 0
```

1 Dimensional CNN

```
from keras.layers import Conv1D, GlobalMaxPool1D, Dense, Flatten
def createModel(window, metric):
    model = Sequential()
    model.add(Conv1D(filters=256, kernel_size=5, padding='same', activation='relu',
                     input_shape=(window, 1)))
    model.add(GlobalMaxPool1D())

    model.add(Dense(units=time_window_size, activation='linear'))

    model.compile(optimizer='adam', loss='mean_squared_error', metrics=[metric])
    print(model.summary())
    return model
```

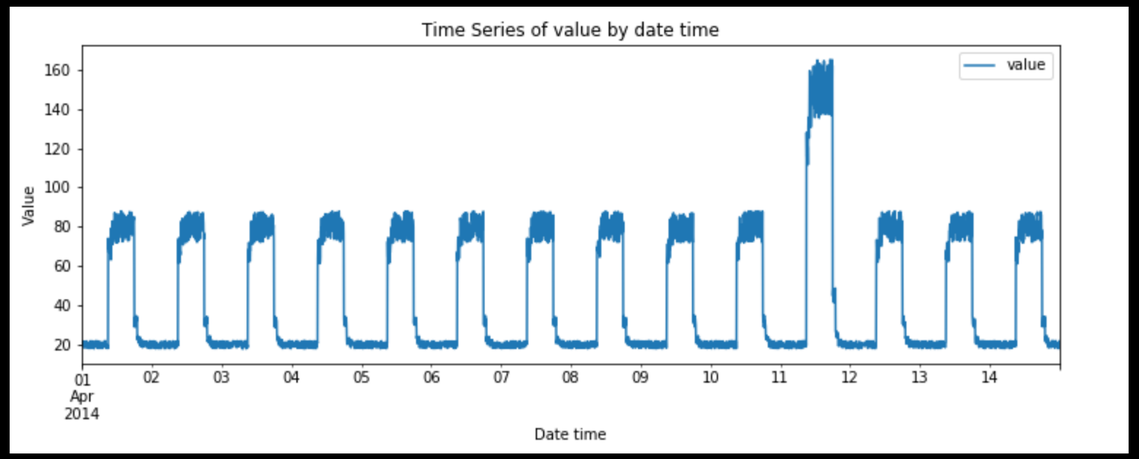
Bi-Directional LSTMs

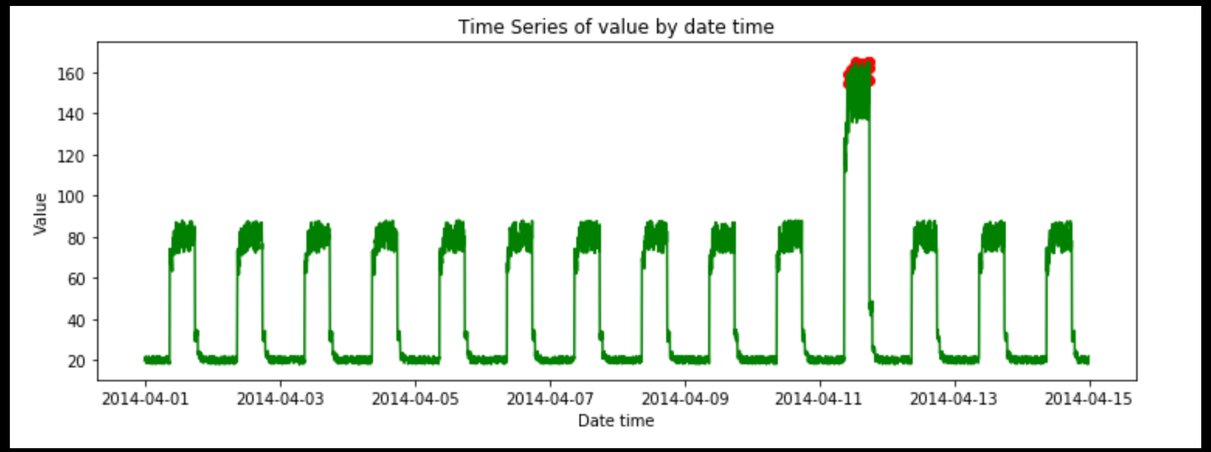
```
from keras.layers import Conv1D, GlobalMaxPool1D, Dense, Flatten, LSTM, Bidirectional, RepeatVector, MaxPooling1D
def createModel(window, metric):
    model = Sequential()

    model.add(Bidirectional(LSTM(units=64, dropout=0.2, recurrent_dropout=0.2), input_shape=(window, 1)))

    model.add(Dense(units=time_window_size, activation='linear'))

    model.compile(optimizer='adam', loss='mean_squared_error', metrics=[metric])

    print(model.summary())
    return model
```

CNN + LSTMs

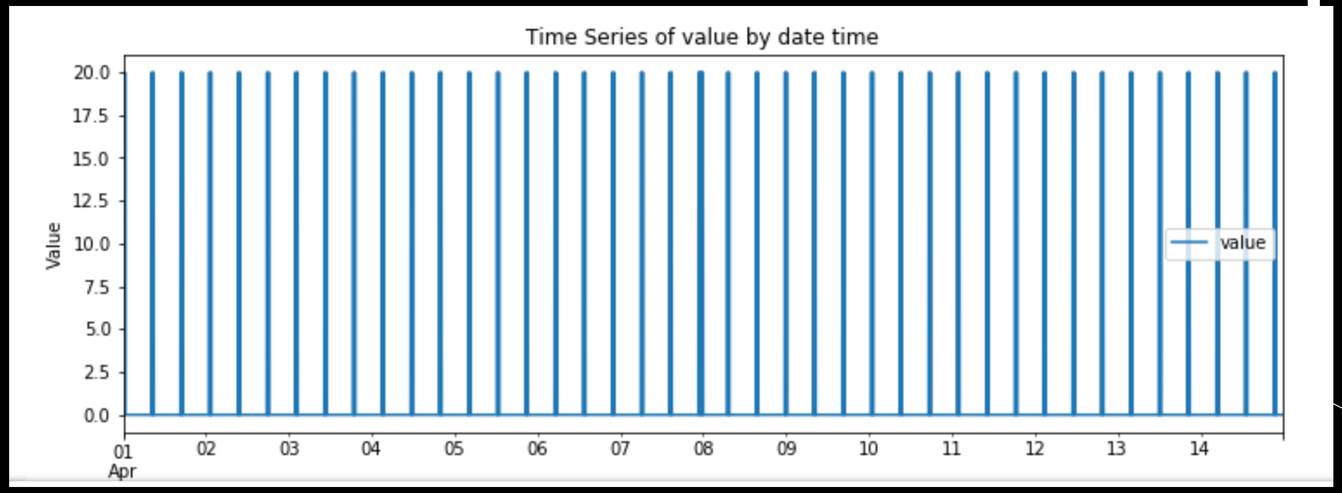
```
from keras.layers import Conv1D, GlobalMaxPool1D, Dense, Flatten, LSTM, Bidirectional, RepeatVector, MaxPooling1D

def createModel(window, metric):
    model = Sequential()

    model.add(Conv1D(filters=256, kernel_size=10, padding='same', activation='relu',
                     input_shape=(window, 1)))
    model.add(MaxPooling1D(pool_size=4))

    model.add(LSTM(64))

    model.add(Dense(units=time_window_size, activation='linear'))

    model.compile(optimizer='adam', loss='mean_squared_error', metrics=[metric])

    print(model.summary())
    return model
```

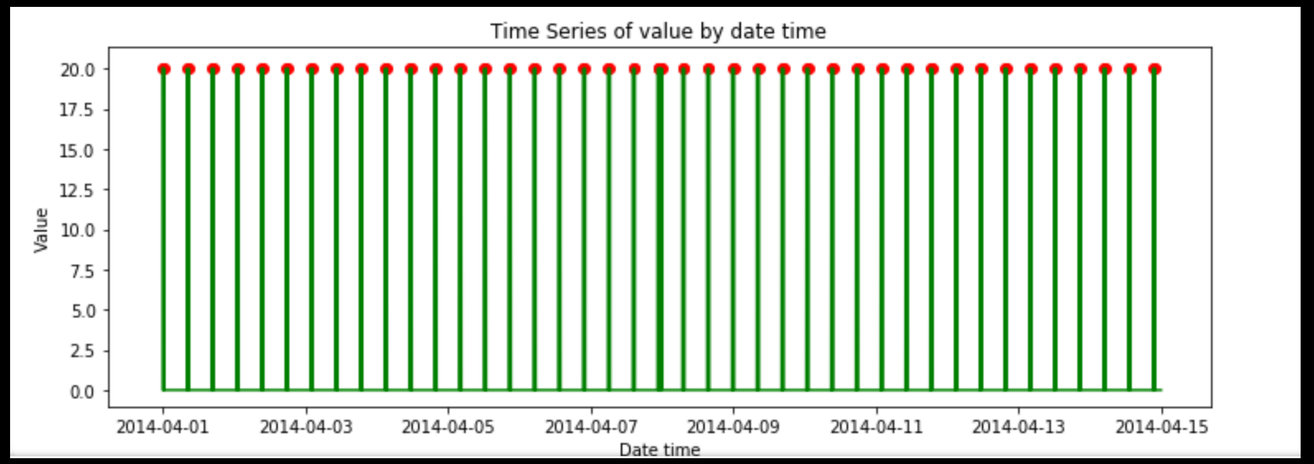
Examples


Time Series of value by date time
value
Value
Date time
01
Apr
2014

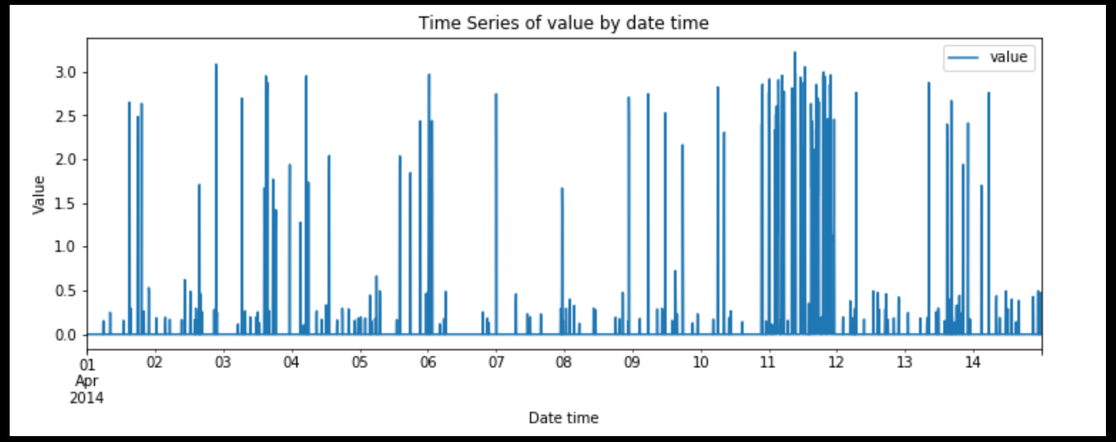

Time Series of value by date time
Value
Date time
2014-04-01
2014-04-03
2014-04-05
2014-04-07
2014-04-09
2014-04-11
2014-04-13
2014-04-15

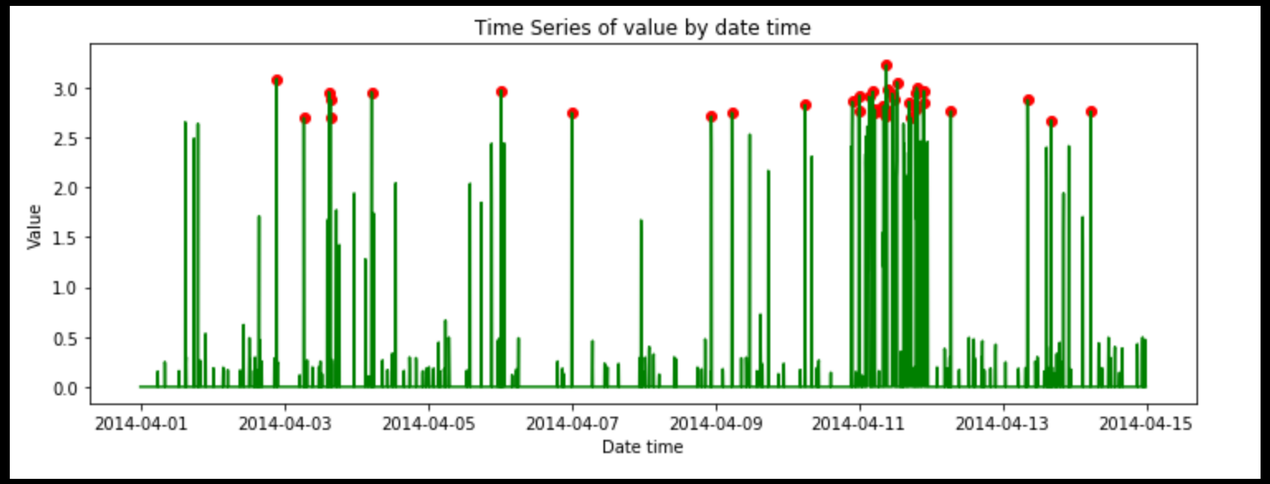
Examples


Time Series of value by date time
Value
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
value
01
Apr
02
03
04
05
06
07
08
09
10
11
12
13
14

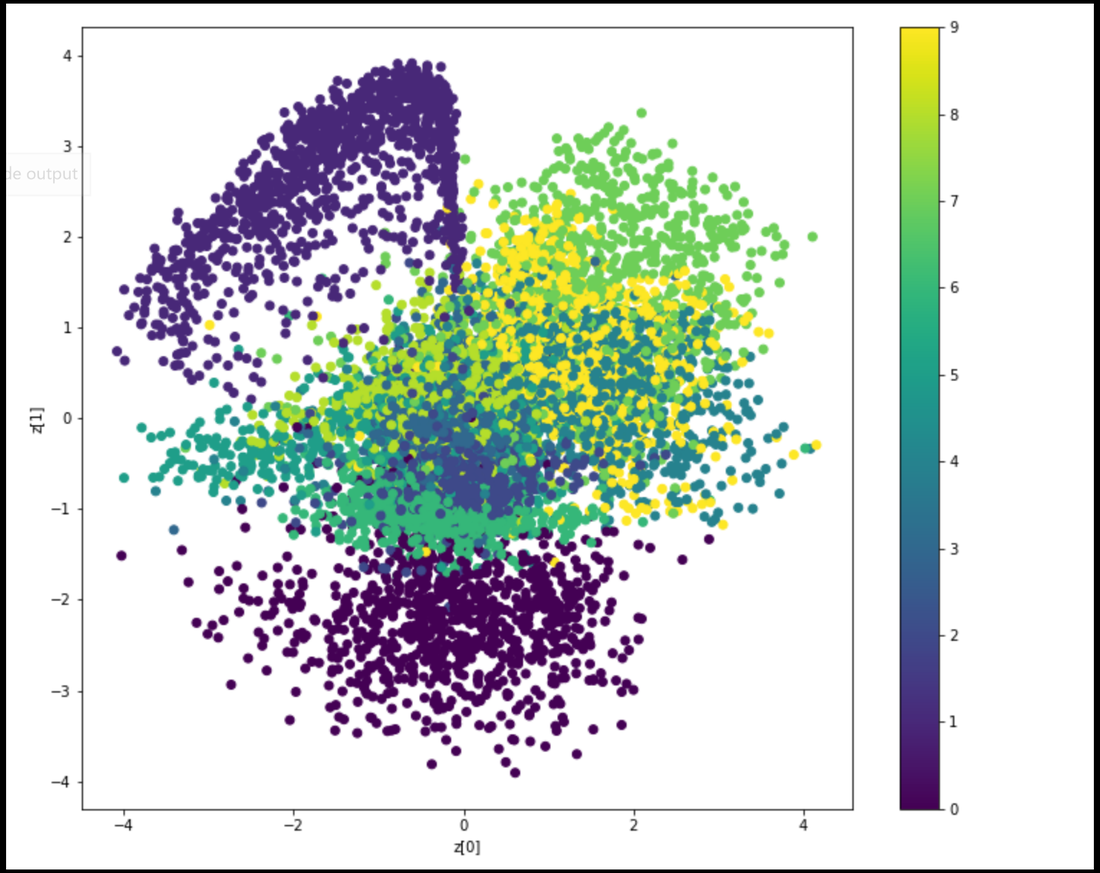

Time Series of value by date time
Value
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
2014-04-01
2014-04-03
2014-04-05
2014-04-07
2014-04-09
2014-04-11
2014-04-13
2014-04-15
Date time

Examples


Time Series of value by date time
value
Value
3.0
2.5
2.0
1.5
1.0
0.5
0.0
01
Apr
2014
02
03
04
05
06
07
08
09
10
11
12
13
14
Date time


Time Series of value by date time
Value
3.0
2.5
2.0
1.5
1.0
0.5
0.0
2014-04-01
2014-04-03
2014-04-05
2014-04-07
2014-04-09
2014-04-11
2014-04-13
2014-04-15
Date time

Examples



Key Observations

- For univariate data , Autoencoders work best
- For non time series data, Autoencoders work very well
- For time series data, Autoencoders could be used however CNN based (LSTM or TCN) works better depending on the historical aspect (30 days/365 days/3 years etc)

Questions ?

- Sridhar Alla : sid@bluewhale.one

  Check out http://bluewhale.one

- Syed Nasar: techtalk@nasars.com @techmazed

  Cloudera Machine Learning Presentations: cloudera.com/ml