Detecting Anomalies in the Baxter Collision Dataset - Team Greenarm

Dominik Durner
Technische Universität München
durner@cs.tum.edu

Philipp Dowling
Technische Universität München
dowling@cs.tum.edu

Atanas Mirchev
Technische Universität München
ga75lar@mytum.de

Benedikt Brandner
Technische Universität München
benedikt.brandner@tum.de

Abstract

Within this paper we present our results on the Baxter robot collision dataset, concerned with anomaly detection. We managed to provide an implementation of STORN in Keras that seems to perform very well. In the anomaly detection task, on coarse detection of samples as normal or anomalous we achieved state of the art accuracy, specificity and sensitivity. Furthermore we were able to generate better ground truth labels for individual anomalies based on torque data. On a fine-grained level of detecting individual anomalies we achieved over 88% accuracy and evaluated the validity of our labels by comparing them with different window sizes to the original labels.

1 Introduction

Anomaly detection is an important problem in many domains, ranging from data cleaning in general statistical applications to fraud detection in banking, or intrusion detection in computer networks. In this project, we deal with the use case of detecting events in sensor data. More specifically, we work with sensor data from a robotic arm with the goal of detecting collisions, i.e. obstacles or pushes to the robot arm.

2 Dataset and Preprocessing

We work on the Baxter Collision data set, which includes samples from 1020 "normal" sequences of measurements and 288 sequences that contain anomalies. Each sequence is around 40 seconds in duration, and sequences were recorded at different frequencies. We subsampled all sequences to 15hz and pre-padded them with zeros (ignored by our models) to bring them to the same length. Fine-grained anomaly labels are supplied as timestamps for when an anomaly occurred. However, these only indicate when a collision was requested to be triggered by a human, thus the actual anomaly in the data can occur significantly later than the labels indicate (or in rare cases, not at all).

The sequence data includes seven dimensions representing the angle configurations of the Baxter robot arm, and additionally includes measurements of measured and commanded torque-like motor efforts of each joint in the robot arm. However, our anomaly detection task is concerned only with detecting anomalies from the angle configuration data.

2.1 Dataset augmentation

In order to improve the quality of the given anomaly labels, we analyzed the measured and commanded torque data to generate more precise labels for where the anomalies actually occurred in the data.

We computed the mean squared error between the commanded and measured torque, which when plotted exhibits extreme spikes in places where anomalies occurred. We considered a maximum window of twelve seconds for each original label. The vast majority of corrected labels lie within two to six seconds after the original one, with some exceptions ranging up to a ten seconds delay. This augmentation through adaptive thresholding worked for every but two of the samples. The validity of these generated labels will be examined in the evaluation section, by comparing results between original and new labels.

3 Methodology

The task of anomaly detection is defined as finding observations in the data which do not conform to the *normal* behavior. There are a number of issues that arise in this context [5]. Arguably, the two most note-worthy problems are:

- the inherent sparsity of the anomalous data instances
- the lack of precise labels marking exactly when an anomaly has occurred

The first issue stems in the very nature of the problem: data outliers carry a lot of information and are therefore rare and improbable (information theory). Thus they are always bound to be underrepresented in any data set. The second problem can be explained with the unavoidable manual labor necessary to label the data, leading to inconsistencies and sometimes missing labels. [5] The undefined boundary between anomalous and normal behavior for most applications and the large variance in the anomaly duration further complicate the matter. Our Baxter Collision Dataset is no exception to these two rules.

For this reason, fitting a classical binary classifier directly to the data becomes a significant challenge. Therefore, we approach the problem by first fitting a model to normal data, that does not contain any anomalies. Once trained, such a model can produce a **normality measure** (e.g. the loss value) for any newly presented, unseen observations. Based on this measure, one can distinguish between anomalous or not anomalous instances.

For the following part, we always held out 20% of the data as a test set (both anomalous and normal), and used the rest for training. All of our models further had a separate validation set cut off from the training data, on which the model performance was evaluated during training. We used those scores for early stopping and checkpointing the best trained models.

3.1 Timeseries Predictor

Our first attempt at solving the anomaly detection task was to build a time-series model flexible enough to fit the arm movements of the Baxter robot. The purpose was to get an understanding of where the modelling difficulties are and, if possible, obtain a baseline accuracy score.

We settled on a recurrent deep learning model, comprising of a number of RNN units stacked on top of each other. Additionally we placed feed-forward layers right after the input and before the last output of the network, accounting for better feature extraction from the data. At each time step we fed x_{t-1} as 7-dimensional input and provided x_t, the next sequence element, as a target. That way the network attempts to learn a generative model for $p(x_t|x_{1:t-1})$. We trained the so-formed network on normal data, taking MSE as a loss function and the *rmsprop* optimizer. Using MSE corresponds to a negative log likelihood with a Gaussian assumption for the aforementioned distribution, without any prior distribution placed on the network weights.

Unsurprisingly, the model performed well on the **normal test set**, managing to follow the changes in the joint angle values very precisely. However, as shown in Section 4, this was not sufficient for proper anomaly detection. The model showed early signs of overfitting, approximating the anomalies in the data nearly as well as normal data, rendering the produced loss values useless in terms of outlier detection. Introducing 0.3 *dropout* as a counter-measure did not lead to any significant improvements.

3.2 STORN

We attribute the failures of the previous model to the lack of explicit probabilistic modeling of the inherent *structure* in the angle configurations data. Data sets, such as the Baxter robot arm movement, exhibit two main properties: there is high signal to noise ratio (SNR) and there are hidden factors (e.g. the human commands that drive the arm movement) that explain nearly all the variability in the *observed* data through complex relationships. Modeling the latent random variables explicitly is therefore adequate.[2]

Joint distribution of the complete data set
We denote the observed angles at time step t with $\mathbf{x_t}$, and the corresponding latent factor at the same time step with $\mathbf{z_t}$. Let $\mathbf{x}_{1:t}$ denote all elements in a sequence up to time step t. We first consider the (complete) joint probability distribution of the data in a sequence of length T:

$$p(\mathbf{x}_{1:T}, \mathbf{z}_{1:T}) = \prod_{t=1}^{T} p(\mathbf{x}_t, \mathbf{z}_t | \mathbf{x}_{1:t-1}, \mathbf{z}_{1:t-1}) = \prod_{t=1}^{T} \underbrace{p(\mathbf{x}_t | \mathbf{x}_{1:t-1}, \mathbf{z}_{1:t})}_{p(\mathbf{x}_t | \mathbf{z}_t, history)} \underbrace{p(\mathbf{z}_t | \mathbf{x}_{1:t-1}, \mathbf{z}_{1:t-1})}_{p(\mathbf{z}_t | history)} \tag{1}$$

where we have used the product rule of probabilities and Bayes‘ rule to decompose into 2-term products. We stress that the second term in the product is conditioned on $\mathbf{x}_{1:t-1}, \mathbf{z}_{1:t-1}$, **only up to step** $t-1$, making it a prior for the latent factor $\mathbf{z}_t$ at step t. Note that we do not have observations for $\mathbf{z}_t$, but we will need this distribution later when we specify the loss of our model.

Bayesian graphical model
We choose to implement the distribution $p(\mathbf{x}_t | \mathbf{x}_{1:t-1}, \mathbf{z}_{1:t})$ with a RNN unit. Let h_t denote the hidden state of the RNN, with inputs $\mathbf{z}_t$ and $\mathbf{x}_{t-1}$ at time step t:

$$h_t = f(\mathbf{x}_{t-1}, \mathbf{z}_t, h_{t-1}) \tag{2}$$

Since the state h_t is a *recursive deterministic function*, it encompasses $\mathbf{z}_{1:t}$, $\mathbf{x}_{1:t-1}$, therefore $p(\mathbf{x}_t | h_t) = p(\mathbf{x}_t | \mathbf{x}_{1:t-1}, \mathbf{z}_{1:t})$. Thus, the RNN implements the first term in the product in equation (1) correctly. This is depicted in the top part of the graphical model in Figure 1.

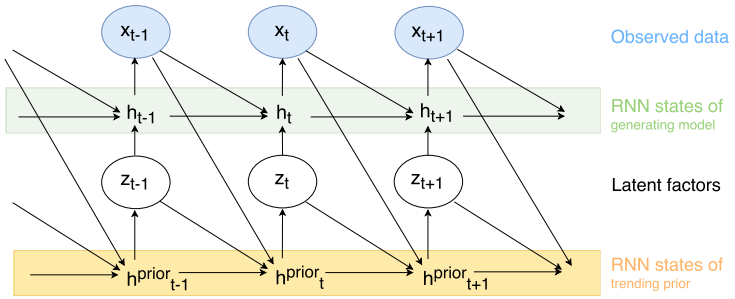


Figure 1: Graphical model for STORN with a trending prior.

For the latent prior distribution we essentially have 2 choices: we could assume that the latent factors $\mathbf{z}_t$ at step t are independent from any past data, simplifying $p(\mathbf{z}_t | \mathbf{x}_{1:t-1}, \mathbf{z}_{1:t-1})$ to $p(\mathbf{z}_t)$. This assumption was presented in the original STORN [1] paper. It simplifies the Bayesian graphical model greatly, and does not require the implementation of a *trending prior*. However, we argue that such modeling is too restrictive for our use case. It disregards any temporal dependencies between the latent states, which we would naturally want to model, as equation (1) already shows. Thus we decided to add a *trending prior* (courtesy to [4]) to our graphical model, depicted in *Figure 1*. The graphical model now follows the original equation (1) without any unjust simplifications. Note that we again implement the latent trending prior $p(\mathbf{z}_t | \mathbf{x}_{1:t-1}, \mathbf{z}_{1:t-1})$ with a RNN unit with hidden state h_t^{prior} at step t given by:

$$h_t^{prior} = f(\mathbf{x}_{t-1}, \mathbf{z}_{t-1}, h_{t-1}^{prior}) \tag{3}$$

Again, because of the recursion h_t^{prior} is a deterministic function of $\mathbf{x}_{1:t-1}, \mathbf{z}_{1:t-1}$, and thus $p(\mathbf{z}_t|h_t) = p(\mathbf{z}_t|\mathbf{x}_{1:t-1}, \mathbf{z}_{1:t-1})$. The final graphical model in Figure 1 now correctly reflects the joint distribution between observed and latent random variables.

The STORN model

With the correct modeling in hand, we can now build a deep learning model to train on the normal movements data. Because of the presence of *latent factors* $\mathbf{z}_t$, the most meaningful way to train the network is via a Variational Auto-Encoder (VAE) [3], replicated at each time step t. In short, a VAE first models the reverse (encoding) distribution $p(\mathbf{z}|\mathbf{x})$ with a *recognition model*, and then reconstructs (decodes) $\mathbf{x}$ back by modeling $p(\mathbf{x}|\mathbf{z})$ with a *generating model*. Since we have time-series data, we need to replicate this behavior across all time steps from 1 to T, which can be achieved with RNNs.

To this purpose, three RNN units come together to form a *Stochastic Recurrent Neural Network* (STORN), first introduced in [1]. The network structure of our STORN implementation is presented in *Figure 2*:

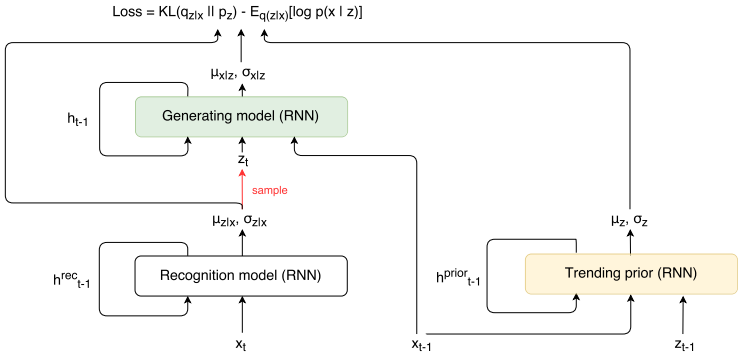


Figure 2: Network structure of the STORN model. The same color scheme is used as in the graphical model in Figure 1

We can already recognize the *trending prior model* and *generating model*, whose hidden states h_t^{prior} and h_t were already depicted in the graphical model in Figure 1. Additionally, we introduce a *recognition model* with the purpose of "encoding" $\mathbf{x}_t$ into $\mathbf{z}_t$ at every time step. It is also implemented with a RNN, with $\mathbf{x}_t$ as input in every step t. All of the three models output *sufficient statistics* of distributions. Given the nature of our data set, we assume multi-variate Gaussian distributions for all three of them. Between the *recognition model* and the *generating model* we first **sample** a specific $\mathbf{z}_t$ from the output statistics of the former model, and then feed this $\mathbf{z}_t$ as input to the generating model. The need for sampling will become clear in the next subsection.

Training

Since the generating distribution $p(\mathbf{x}_t|\mathbf{z}_t)$[1] is modeled via a strong non-linearity (a RNN, i.e. a neural net in every time step), the "inverse" distribution $p(\mathbf{z}_t|\mathbf{x}_t)$ is not analytically tractable. This renders techniques such as EM impossible to use in this setting. Instead we use the training method of *Stochastic Gradient Variational Bayes* (SGVB). Similar to [3], we use Jensen's inequality and equation (1) to obtain the following **upper bound** of the NLL, which we use as our loss:

[1] Omitting conditioning on previous time steps $\mathbf{x_{1:t-1}}, \mathbf{z_{1:t-1}}$ for clarity of the notation

$$
\begin{aligned}
NLL &= -log p(\mathbf{x}_{1:T}) = -log \int_{\mathbf{z}_{1:T}} p(\mathbf{x}_{1:T}, \mathbf{z}_{1:T}) d\mathbf{z}_{1:T} \\
&= -log \int_{\mathbf{z}_{1:T}} \prod_{t=1}^{T} \underbrace{p(\mathbf{x}_t|\mathbf{x}_{1:t-1}, \mathbf{z}_{1:t})}_{p(\mathbf{x}_t|h_t)} \underbrace{p(\mathbf{z}_t|\mathbf{x}_{1:t-1}, \mathbf{z}_{1:t-1})}_{p(\mathbf{z}_t|h_t^{prior})} d\mathbf{z}_{1:T} \\
&= -log \int_{\mathbf{z}_{1:T}} \frac{q_{\mathbf{z}|\mathbf{x}}(\mathbf{z}_{1:T})}{q_{\mathbf{z}|\mathbf{x}}(\mathbf{z}_{1:T})} \prod_{t=1}^{T} p(\mathbf{x}_t|h_t) \underbrace{\prod_{t=1}^{T} p(\mathbf{z}_t|h_t^{prior})}_{:=p_{\mathbf{z}}(\mathbf{z}_{1:T}|h_{1:T}^{prior})} d\mathbf{z}_{1:T} \\
&\leq E_{q_{\mathbf{z}|\mathbf{x}}(\mathbf{z}_{1:T})} \left(-\sum_{t=1}^{T} log p(\mathbf{x}_t|h_t) \right) - KL(q_{\mathbf{z}|\mathbf{x}}(\mathbf{z}_{1:T}) || p_{\mathbf{z}}(\mathbf{z}_{1:T}|h_{1:T}^{prior})) = LOSS
\end{aligned}
$$

where at line 2 we substitute for the hidden states of the *generating model* and *trending prior model* RNNs to simplify notation. At line 3 we define the "joint prior" distribution $p_{\mathbf{z}}(\mathbf{z}_{1:T}|h_{1:T}^{prior})$. Furthermore, $q_{\mathbf{z}|\mathbf{x}}(\mathbf{z}_{1:T})$ represents the distribution implemented via the *recognition model* RNN, which is an approximation of the true (and not tractable) latent factor posterior distribution.

Since we chose isotropic multivariate Gaussian distributions for both $q_{\mathbf{z}|\mathbf{x}}(\mathbf{z}_{1:T})$ and $p_{\mathbf{z}}(\mathbf{z}_{1:T}|h_{1:T}^{prior})$, there is a closed formula for the KL divergence based on the output statistics of the *recognition model* and the *trending prior model*.

Following SGVB, we approximate the expectation term in the loss with MCMC sampling from the recognition distribution $q_{\mathbf{z}|\mathbf{x}}$, setting the number of samples L to 1. This is the reason why we have the necessary sampling step in the STORN network on top of the statistics produced by the *recognition model* (Figure 2). Note that with the given distribution assumptions, the specified upper bound loss decomposes into separate terms for each time step t.

With this loss, we can train the STORN network using normal BPTT (Back-propagation through time) and any gradient descend optimization method (e.g. sgd, rmsprop).

Implementation
We implemented STORN entirely in **keras** (a theano framework). To our knowledge this is the first such implementation. We augmented the STORN network with deep feed-forward units (6 dense layers) around every RNN unit for optimal feature extraction. The biggest challenges we experienced were working around **keras**, since it was missing some functionality, and the very long training times because of the model complexity. We obtained our best results with *GRU* recurrent units and *tanh* activation functions, optimizing with *rmsprop*.

3.3 Anomaly detection from a loss curve

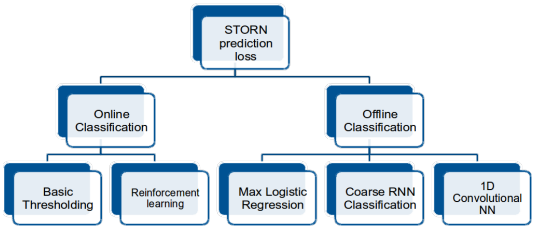


Figure 3: Anomaly detection approaches

The basic idea for detecting an anomaly is to consider the loss values of STORN (or the Time-series Predictor) to predict if a sequence occurred. We distinguish different classification approaches which are grouped in Figure 3.

3.3.1 Offline Detection

Offline detection for this dataset means to predict whether a given sample contains an anomaly or is free of those. Hence, we have a binary classification problem with the whole sequences as observations.

Max Value Logistic Regression A basic approach is to learn a linear separator to detect whether or not a time series sequence is anomalous. Instead of finding the loss threshold manually we can compute the threshold on the fly by using the maximum loss spike as representation of each sequence. We train the classifier on these values and are able to separate the two classes with this approach.

RNN Classification We attempted to train an RNN sequence classifer to detect anomalous sequences. This consisted simply of feeding the error at each time step into an RNN with a logistic regression classifier on the last hidden state and training the model with gradient descent on binary classification targets (sequence is normal vs. anomalous). Unfortunately, this idea did not work out, and we could not get the model to successfully predict anomalies, likely because we had too few anomalous sequences in the data, and anomalies within these occurred only very sparsely (only about 0.1% of all timesteps were anomalous).

1D Convolutional Network To further improve the results, we used a 1D Convolutional net to get better classification. We figured out this approach after trying a simple feed forward neural network. Since the anomalies occur on different positions it was hard to train such a simple feed forward network. Therefore, we introduced a simple 1D convolutional layer to be able to train the network such that the convolutional layer minimizes the problem of having anomalies on different positions and intensities.

3.3.2 Online Detection

In the online detection case, we try to predict an anomaly in an online fashion. This means, that we only have the information of the current and the previous timesteps without knowing the future. Hence, we want to rather detect each anomaly and not just binary classify whether or not a whole sequence contains any anomalies.

Basic Thresholding One of the most basic ideas is to just predict an anomaly online if the current spike is higher than a predefined threshold value. This simple approach gave us quite reasonable results in predicting the anomalies in an online fashion.

Reinforcement Learning We tried to use the Q-learning algorithm to train a simple reinforcement learning neural network to this task, by structuring it as a game wherein a series of loss values is presented, and the agent places "bets" on whether they are anomalous or not, and accordingly gets a payoff or penalty. Unfortunately, as in the offline-detection RNN case, we faced the problem of having extremely sparse positive anomaly labels. In the end, the agent would simply learn to always predict non-anomaly at every step, since that locally minimized the risk of false-positive penalties (which are much more common than true-positive payoffs).

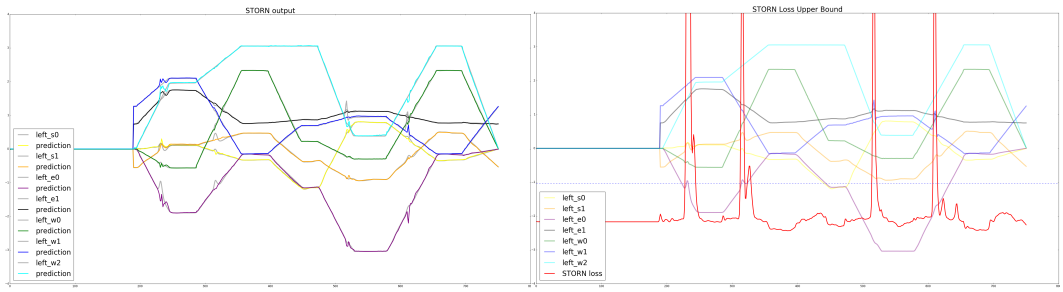
4 Evaluation

4.1 Approximating normal data

As seen in Figure 4a, the joint movements are predicted accurate as long as no anomalies occur. If there are anomalies then the actual movement and the predicted curve differ at the place of the anomaly. This is very important since the later steps of the anomaly detection rely on highly accurate **normal** data predictions. Otherwise, normal data cannot be distinguished from anomalous data.

4.2 Anomaly detection from a loss curve

Those gaps between the actual movement and the prediction result in high spikes in the loss curves. This is a result since the loss function explodes on such irregular behaviors. For the detection



(a) STORN fits the data perfectly but predicts different curve shapes at anomalies

(b) STORN loss is high on positions where STORN loss predicts a different curve

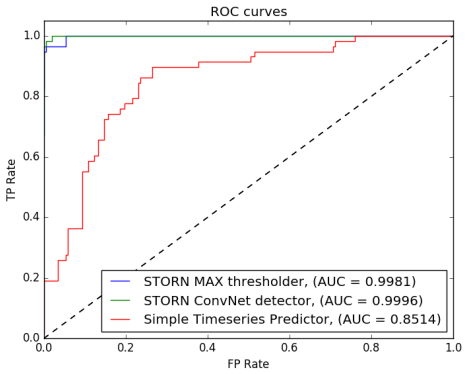
afterwards, it is crucial that non anomalous steps have a low loss. An example of the loss curve behavior of STORN can be found in Figure 4b.

4.2.1 Coarse Detection

	Accuracy	Specificity	Sensitivity
Max Logistic Regression on Timeseries Predictor Data	82.4%	99.9%	20.7%
Max Logistic Regression on STORN Data	98.9%	99.0%	98.3%
ConvNet classification on STORN Data	99.2%	99.5%	98.3%

STORN seems to fit the data better

(a) Coarse anomaly detection



(b) ROC curves, for the Max Value Log. Regression and ConvNet classifiers (with Area Under Curve metric)

As discussed in the methodology section, we tried different approaches for the offline detection. This anomaly detection, also called coarse detection, is easily measurable and comparable since we know the ground truth of whether a sample contains at least one anomaly. Hence, we could train the network on a train and validation set and test it on a test set. Comparing the ground truth with the binary classification output, we calculated various statistics for the different approaches. The most important values are highlighted in Figure 5a.

Max Value Logistic Regression We measured the max value logistic regression approach for both the STORN and the simple RNN Timeseries Predictor. We figured out, that there is a huge gap between those two models. The test set accuracy of the Simple RNN Predictor is with 82.4% quite low compared to the 98.9% of the STORN loss. This huge gap between the two prediction approaches lies in the fact that the sensitivity of the STORN approach is way higher. This is a result of the better fitting of STORN to the latent structure in the original data. Hence, the loss values differ significantly between anomalous and normal movements.

1D Convolutional Network By introducing a convolutional network we were able to further optimize the classification rates. Even though the differences to the Max Value Logistic Regression is very minor we can see that a more complex network is able to detect the anomalies slightly better.

As Figure 5b shows, STORN clearly outperforms the simple Timeseries Predictor, with AUC score of roughly 1.0 for both the ConvNet and the Max Value LogRegression classifiers.

4.2.2 Fine-Grained Detection

It must be made clear that the fine-grained anomaly detection quality is directly **limited** by the quality of the provided anomaly labels. Still, we tried to use our improved torque labels and provide some rough ballpark estimates of the quality of STORN's online capabilities. As already mentioned in

Section 3.3.2, we first considered more complex algorithms like Reinforcement learning or a RNN classifier running on *overlapping windows* of length 2s over the STORN loss. Unfortunately, these models quickly saturated during training as the gradient steps became too small to train.

Logistic Regression Thresholding Taking into account the training difficulties of more complex models, we eventually went for thresholds of the STORN loss in the fine-grained case as well. While simplistic in nature, this approach proved to provide good enough results. We used the threshold obtained from the *coarse detection* Logistic Regression classifier. We then considered two **heuristic** regions for where an anomaly should be: up to 5s after an *original* anomaly label, or -3s to +3s around an improved *torque* label. Loss spikes exceeding the threshold outside of these regions were considered false positives. Table 1 shows a summary of the results:

Table 1: Fine-grained anomaly detection with STORN

Interval (Heuristic)	**TP** (properly detected)	**FP** (falsely detected)	**FN** (missed)
[0, +5s] after original label	141	14	19
[-3s, +3s] around torque label	141	6	19

Both experiments provided similar results, with ca. 90% sensitivity in both cases. Using the torque labels led to fewer *false positives*. However, we **stress** that numbers like these are ambiguous and should be taken with great caution, as we do not have a proper ground truth for the fine-grained case.

5 Future Work

Here we discuss possible improvements to our results that seem worth looking into.

5.1 Improved labels

One of the biggest challenges with the dataset is that the anomalous ground truth labels are not accurate at all. The biggest room for improvement is thus with acquiring accurate labels, because it can both impact the results of the existing solution and ensure the validity of the approach of generating more accurate labels through data augmentation.

Of course also this approach can use more sophisticated methods than adaptive thresholding to achieve better results. The problem with generating ground truth labels yourself is that credibility of results based on a handpicked ground truth is very low. Thus having accurate labels from a third party or within the data set is preferable to generating them along with the solving approach.

5.2 Fine-Grained Detection

For our work we mainly focused on coarse detection, i.e. classifying a whole sample as anomalous or normal. We did try fine grained detection and achieved decent results. However, compared to coarse detection there is still a lot of optimizations we have not tried yet. This is mainly due to the difficulties we had when dealing with ground truth labels.

Apart from improving on ground truth data, which we consider the most desirable improvement, we would have liked to further investigate the reinforcement learning approach in particular.

References

[1] J. Bayer and C. Osendorfer. "Learning Stochastic Recurrent Networks". *arXiv*, 1411.7610v3 [stat.ML], 2015.

[2] J. Chung et al. "A Recurrent Latent Variable Model for Sequential Data". *arXiv*, 1506.02216v6 [cs.LG], 2016.

[3] D. P. Kingma and M. Welling. "Auto-encoding variational bayes". *arXiv*, 1312.6114, 2013.

[4] M. Ludersdorfer M. Sölch, J. Bayer and P. van der Smagt. "Variational Inference for On-line Anomaly Detection in High-Dimensional Time Series". *arXiv*, 1602.07109, 2016.

[5] A. Banerjee V. Chandola and V. Kumar. "Anomaly Detection: A Survey". *ACM Computing Surveys*, 41.3:15, 2009.