

Study on Log Anomalies

—

Razi Mahmood

UC Berkeley Data Science

Problem scope

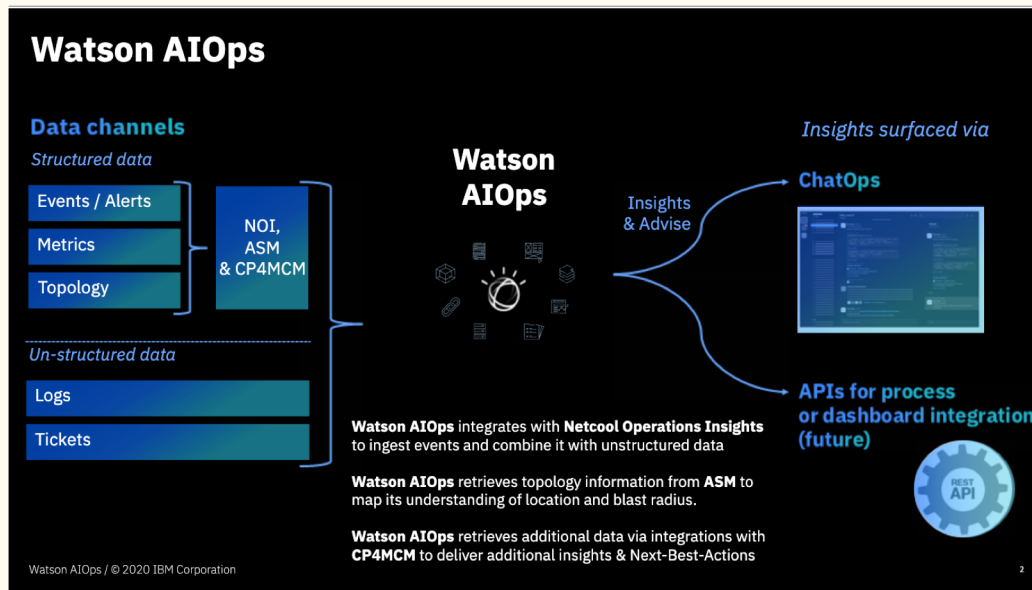
- In the general domain of AIOps, i.e. optimizing IT for operations management using AI
 - IT performance management, services, operations
- Problems in Log anomalies from application and infrastructure logs (many similarities to disease understanding problems)
 - Detection of issues early leading to events (anomaly detection)
 - Prediction of events before they occur (event prediction -> difference between anomaly and event?)
 - Scoping the incident (anomaly localization)
 - Recommending next best action for problem resolution
 - Reducing noise in doing the analysis (reduce false positives in detection and prediction)
 - Notification (involves grouping and reducing unnecessary alerts)

The log anomaly problem space - Summary

- What is the problem?
 - AI for ITOps log anomalies -> anomaly detection, prediction, resolution
- Why is it important?
 - IT infrastructure 24x7 maintenance, mission critical apps
- Why is it difficult?
 - Variety in problems is too large (many systems, infrastructure, code dumps, log types, etc.) so the same method may not work for all
 - Labeling data is difficult (needs self-supervised or unsupervised methods)
 - Incidence distribution is very skewed (A lot of normal and few abnormal makes training skewed)
 - Data preparation itself may need work
- How is it being solved?
 - Machine learning techniques (classical, deep learning)
 - Parsing for essential feature extraction, and learning
 - Direct learning from textual encoding
 - Any attention methods?
 - Which method(s) have proven to work well?
- Where is innovation needed?
 - Avoid manual feature extraction
 - Capture context and order in the log data
 - Better model the difference between normal and anomalous logs in the presence of imbalanced data

Watson AIOps – A tool for log anomaly detection

- Open questions:
 - How does it handle the enormous variations in logs, and input data?
 - How does it do this totally automatically?
- Is the basic functionality?
 - Anomaly prediction
 - Event grouping and clustering
 - Search for similar incidents for recommendation generation



Log Analysis Pipeline in Watson AI Ops

- **Tasks:**

- Training
- Inference
- Detection

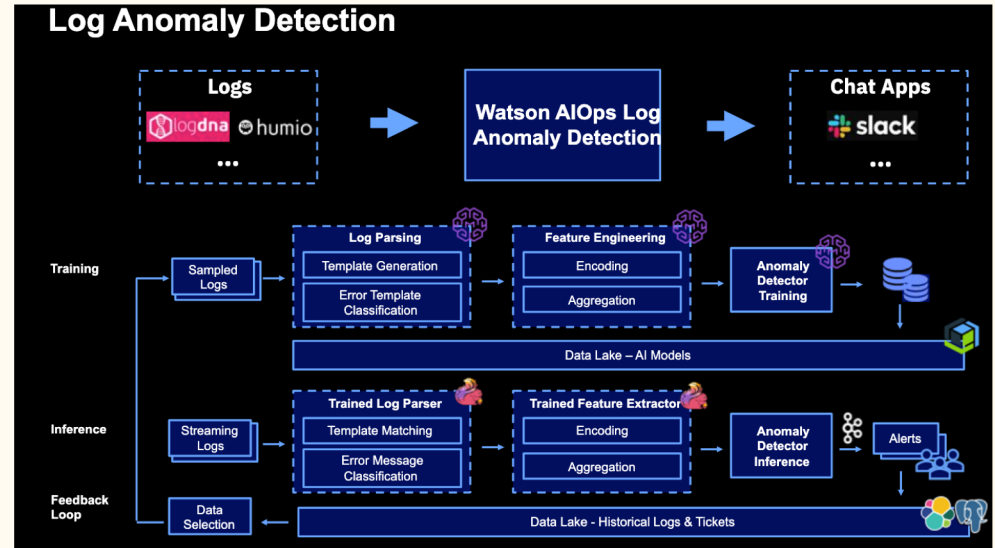
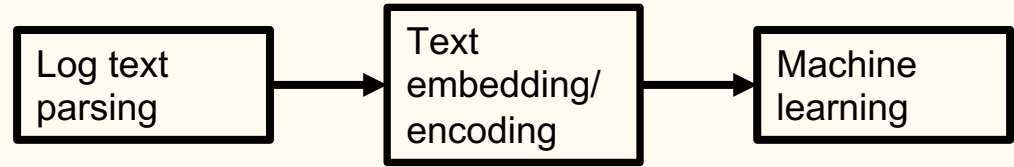
- **Pipeline:**

- Log parsing
 - Convert to structured space
- Feature engineering
- Anomaly detector training

- **How does prediction work currently?**

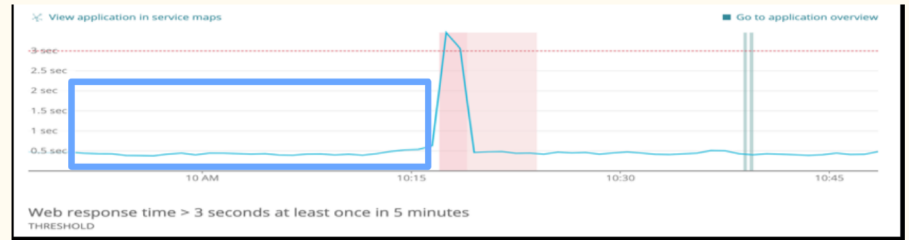
- **Why do text embedding?**

- Robustness to log variations?
 - Is it still per system type?



Questions & ideas

- What are some open problems in this space?
- I see color being used in the logs, can this be a cue in the analysis?
- Since the logs are produced by code, why not do something at coding time to make it easy to detect?
- For time series logs, can we use signal detection techniques with or without ML?
- Which is more important?
 - Precision or recall?



Research

- Experiments with existing models
- Analysis of representative papers
- Proposed new model

Conducting baseline experiments with existing models.

- I had to setup a new environment for the notebook, then install pytorch
- Git Clone the repository from
<https://github.com/logpai/loglizer/blob/master/README.md>
- Started a notebook to do the various model testing experiments
- Ran the decision tree model on the HDFS dataset
- Tuned the parameters to get F-score of 0.732 but saw that I can do better with more feature tuning

Experiments with decision trees

Total Data Size	train_ratio	Train amt	Test amt	Precision	Recall	F-measure value
7940	0.9	7145 (~90%)	749 (~10%)	0.938	0.469	0.625
7940	0.7	5557 (~70%)	2383 (~30%)	0.949	0.596	0.732
7940	0.5	3969 (~50%)	3971 (~50%)	0.985	0.427	0.596
7940	0.4	3175 (~40%)	4765 (~60%)	0.988	0.436	0.605
7940	0.3	2381 (~30%)	5559 (~70%)	0.991	0.486	0.652

Dataset tested: HDFS Logs

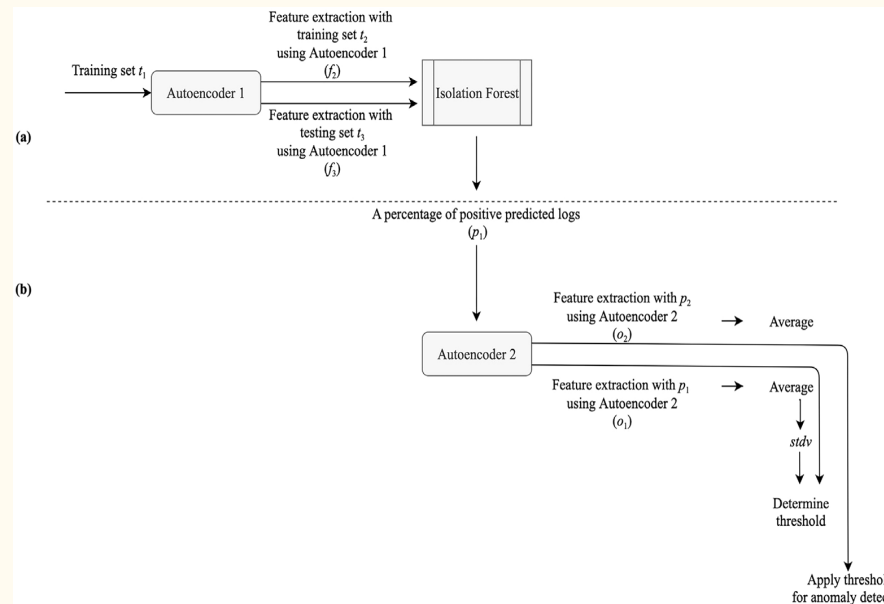
- For Decision Trees, decreasing training ratios correlate with higher precision values
 - Occasional low recall, the F-measures are fluctuating roughly around 0.6 and 0.7

Experiments with Logistic Regression

Total Data Size	train_ratio	Train amt	Test amt	Precision	Recall	F-measure value
7940	0.9	7145 (~90%)	795 (~10%)	0.941	0.500	0.693
7940	0.7	5557 (~70%)	2383 (~30%)	0.95	0.606	0.740
7940	0.5	3969 (~50%)	3971 (~50%)	0.986	0.433	0.602
7940	0.4	3175 (~40%)	4765 (~60%)	0.988	0.441	0.610
7940	0.3	2381 (~30%)	5559 (~70%)	1	0.259	0.412

Analysis of related papers

- Amir Farzad*, T. Aaron Gulliver, “Unsupervised log message anomaly detection”
- Seems to use a chain of encoders
 - Encoder 1 to learn features of normal and abnormal data
 - Encoder 2 to learn features of normal data only (Isolated forest)
 - Encoder 3 to learn features of normal data only
 - Detect anomaly by comparing the feature averages of normal data versus mixed data
- Is this necessary?
 - Why can't the features extracted from Encoder 2 be directly used for separation?
- The F-score is lower for anomaly detection



Code2Vec Paper summary

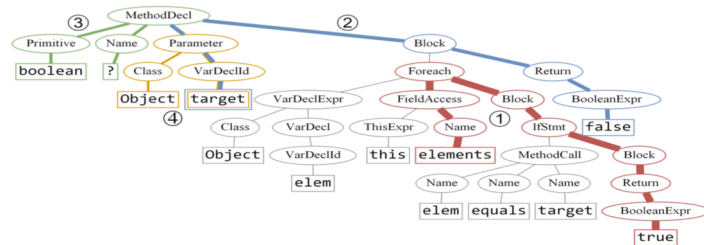
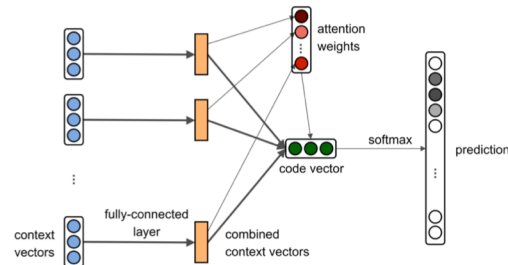
- Input: method snippet, semantic label for the snippet (presumably from the actual name itself abstracted)
- Pre-processing:
 - Parse the code snippet to extract an abstract syntax tree (AST)
 - Extract vocabulary and context vectors as paths between terminal nodes
 - Label the snippet with a method label (may match the actual name or some abstraction of it)
- Train an encoder that uses :
 - A fully connected layer for each context vector
 - Weighted combination of the encodings of context vectors as a way of applying attention
 - A classifier at the end
 - Learn the attention weights end-to-end
- Accuracy:
 - Precision 63.3%, recall 56.2% F-score 59.5%
- Question:
 - How is this useful in anomaly detection?

```
boolean f(Object target) {  
    for (Object elem: this.elements) {  
        if (elem.equals(target)) {  
            return true;  
        }  
    }  
    return false;  
}
```

(a)

Predictions:

contains		99.93%
matches		3.54%
canHandle		1.15%
equals		0.87%
containsExact		0.77%



DeepLog Summary

- Parses logs into log keys (aka key terms)
- Puts them in a sequence, and trains LSTM model
- Predicts the next log key as normal or abnormal

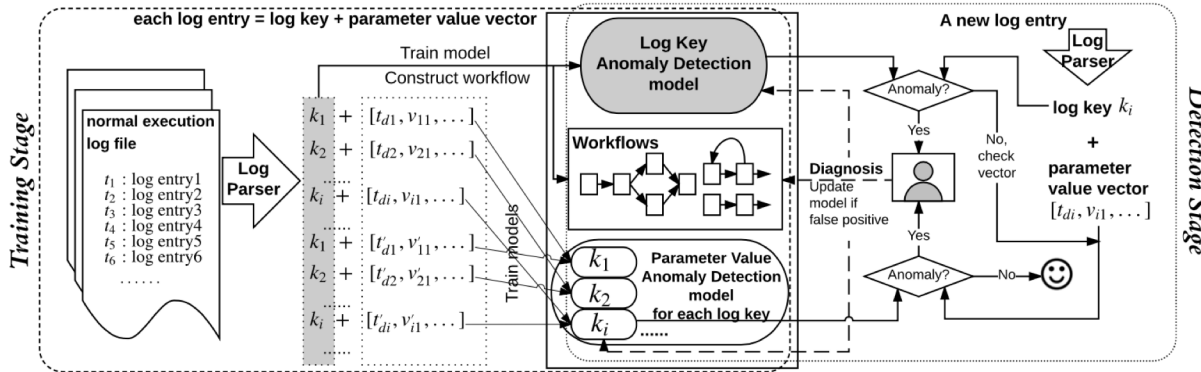
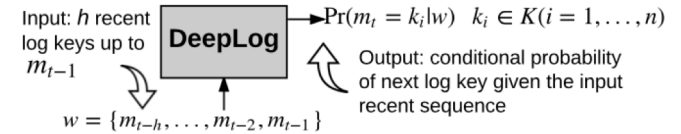
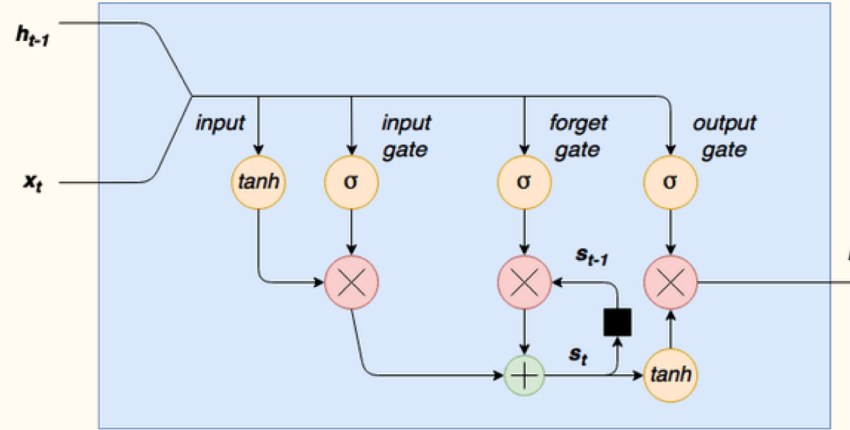


Figure 1: DeepLog architecture.

- How to choose window size?
- Need to see good examples of normal and abnormal sequences
- How to classify a log key as anomaly?

LSTM Model – My understanding

- A model to learn pattern information from sequence data
 - E.g. sentences, event streams
- The model has 4 gates:
 - Controls how much information from past cell to remember/forget
 - How much of the current data to learn from
 - How much of the recurrent state to weigh-in
 - How much learned information to pass onto the next
- A special case of a recurrent neural network (RNN)



What is DeepLog doing under the covers

- The example works with pre-extracted log keys
 - HDFS_100k.log_structured.csv column EventID
 - Block IDs are used to cross-reference and get the labels from anomaly_label.csv
- The event sequence is being modeled through LSTM

1	81109	203518	143	INFO	dfs.DataNode\$DataReceiver	Receiving block blk_-1608999687919862906 src: /10.250.19.102:54106 dest: /10.250.19.102:50010	E5	[8, 8, 0, 8, 3] -> 0
2	81109	203518	35	INFO	dfs.FSNamesystem	BLOCK* NameSystem.allocateBlock: /mnt/hadoop/mapred/system/job_200811092030_0001/job.jar.	E22	[8, 0, 8, 3, 2] -> 0
3	81109	203519	143	INFO	dfs.DataNode\$DataReceiver	Receiving block blk_-1608999687919862906 src: /10.250.10.6:40524 dest: /10.250.10.6:50010	E5	[0, 8, 3, 2, 3] -> 0
4	81109	203519	145	INFO	dfs.DataNode\$DataReceiver	Receiving block blk_-1608999687919862906 src: /10.250.14.224:42420 dest: /10.250.14.224:50010	E5	[8, 3, 2, 3, 2] -> 0
5	81109	203519	145	INFO	dfs.DataNode\$PacketResponder	PacketResponder 1 for block blk_-1608999687919862906 terminating	E11	[3, 2, 3, 2, 4] -> 0
6	81109	203519	145	INFO	dfs.DataNode\$PacketResponder	PacketResponder 2 for block blk_-1608999687919862906 terminating	E11	[2, 3, 2, 4, 4] -> 0
7	81109	203519	145	INFO	dfs.DataNode\$PacketResponder	Received block blk_-1608999687919862906 of size 91178 from /10.250.10.6	E9	[3, 2, 4, 4, 3] -> 0
8	81109	203519	145	INFO	dfs.DataNode\$PacketResponder	Received block blk_-1608999687919862906 of size 91178 from /10.250.19.102	E9	[2, 4, 4, 3, 2] -> 0
9	81109	203519	147	INFO	dfs.DataNode\$PacketResponder	PacketResponder 0 for block blk_-1608999687919862906 terminating	E11	[4, 4, 3, 2, 4] -> 0
								[4, 3, 2, 4, 5] -> 0

DeepLog Experiments – As a function of Window Size

Hidden size	Window size	Train ratio	epochs	Batch size	Recall	Precision	F-score
32	5	80%	2	32	0.17	0.47	0.255
32	10	80%	2	32	0.158	0.714	0.259
32	15	80%	2	32	0.126	0.888	0.22
32	20	80%	2	32	0.015	1.0	0.03

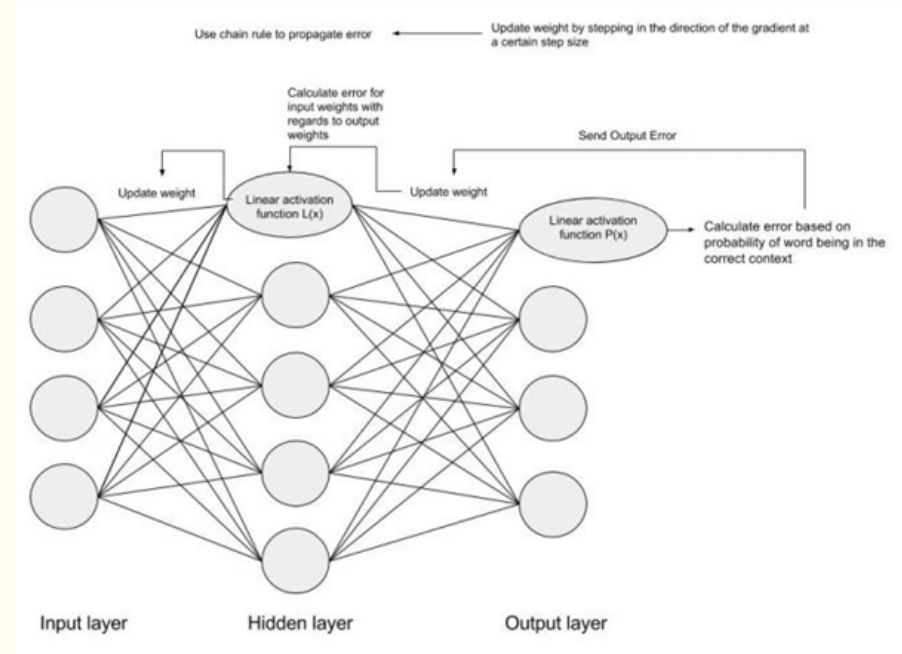
Window too large degrades performance. Size of 10 seems to be about optimal for this choice of hyper parameters

DeepLog Experiments – As a function of Training data

Hidden size	Window size	Train ratio	epochs	Batch size	Recall	Precision	F-score
32	10	80%	2	32	0.158	0.714	0.259
32	10	60%	2	32	0.22	0.667	0.33
32	10	50%	2	32	0.192	0.68	0.298
32	10	30%	2	32	0.132	0.5	0.20

Word2Vec experiments

- Word2Vec embedding for representing words
 - Words from a vocabulary
- To get word2vec working
 - Installed gensim library
 - Install pretrained model from Amazon AWS on Google News
 - Installed NLTK for word processing
- Built a fresh model for the HDFS data
 - Pre-processed the HDFS records
 - Ran CBOW and N-gram models



BERT (Bi-directional Encoder Representations from transformers)

- An embedding model that captures the use context of words better than Word2Vec
 - Word2Vec gives the same encoding for a word regardless of its use in a sentence
 - BERT gives a different encoding for the word depending on the sentence in which it occurs
 - I arrived at the **bank** after crossing the river
 - I went to the **bank** to deposit my check
- BERT models extract high quality language features from text data
- We can fine-tune these models on a specific task such as:
 - Sentiment analysis
 - question answering
 - Sentence completions.
 - Sentence similarity for retrieval
- There are a lot of variants on BERT!

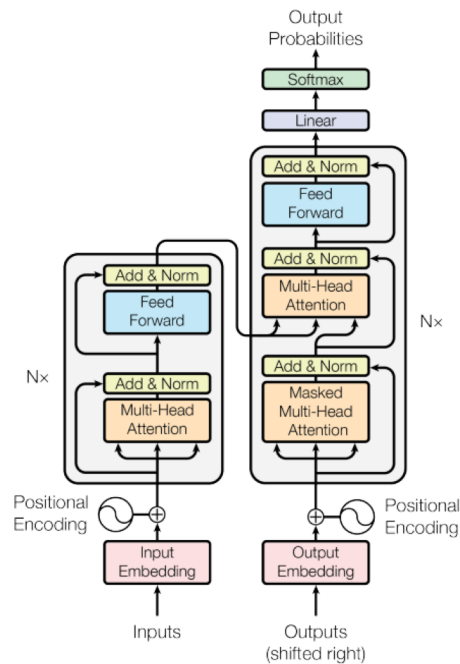







Figure 1: The Transformer - model architecture.

Studies with BERT

- Installed BERT from Huggingface
 - <https://github.com/huggingface/transformers>
- Also installed SentenceBERT
 - <https://github.com/UKPLab/sentence-transformers>
- Training on how to use the pipelines:
 - <https://www.kaggle.com/funtowiczmo/hugging-face-transformers-how-to-use-pipelines>

Applying BERT to our HDFS classification problem

- Key idea: Encode sentences of log traces in BERT and try to classify the vectors as normal or abnormal
- Larger HDFS dataset supplied:

 HDFS_abnormal_test.json	Jun 11, 2020 at 1:43 PM	70.6 MB	JSON
 HDFS_normal_test.json	Jun 11, 2020 at 1:43 PM	80 MB	JSON
 HDFS_train.json	Jun 11, 2020 at 1:42 PM	2.57 GB	JSON
 normal_test_blocks.txt	Jun 11, 2020 at 1:55 PM	411 KB	Plain Text
 normal_train_blocks.txt	Jun 11, 2020 at 1:56 PM	13.2 MB	Plain Text

- Only normal training data is provided here!
 - HDFS_train has 10,558,588 million traces for 541,385 block_ids
 - Abnormal_test has 16,838 block_ids (around 3% abnormal it looks like)
 - Baseline system should have 97% accuracy for normals
 - Can I use some of them to make the abnormal set and learn to separate? – For example, using 30% of the abnormal data and add to the classifier?

Ideas to build the classifier

- Direct classification of BERT embeddings:
 - Take the content field of the HDFS logs for both abnormal and normal
 - Encode using sentence BERT
 - Feed it to a classifier
- Build an intermediate embedding modeling the similarity between abnormal samples and separate them from normal samples.
 - Since we have very large normal samples and very abnormal samples, can we use an approach that takes the abnormal examples as positive examples of the anomaly and the large negative examples to build an embedding based on contrastive loss?
 - Learns a new embedding that captures the similarity between positive samples to better expose their patterns
 - Classify using the new embedding focused on the abnormal class
 - <https://arxiv.org/abs/2004.11362> (NeurIPS 2020)

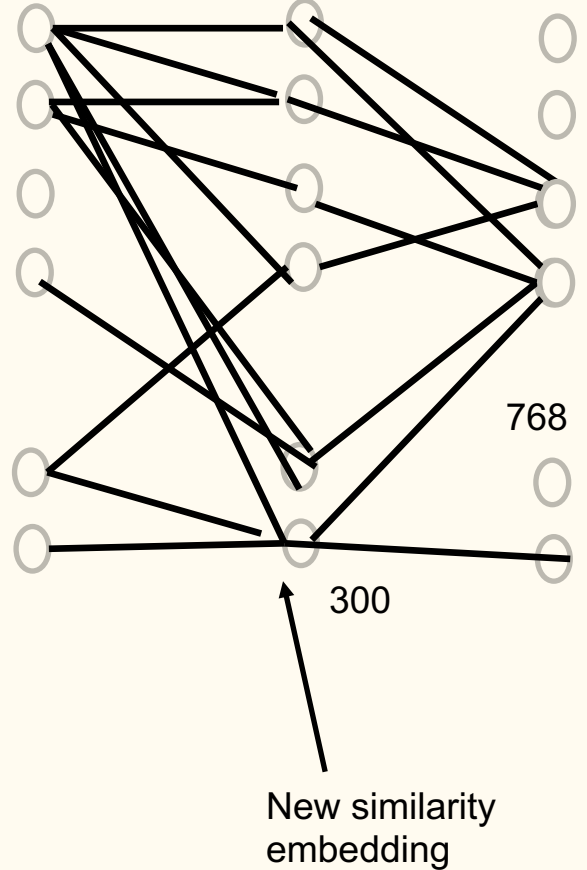
$$\mathcal{L}^{self} = \sum_{i \in I} \mathcal{L}_i^{self} = - \sum_{i \in I} \log \frac{\exp(\mathbf{z}_i \cdot \mathbf{z}_{j(i)}/\tau)}{\sum_{a \in A(i)} \exp(\mathbf{z}_i \cdot \mathbf{z}_a/\tau)}$$

Algorithm

- **Training:**
 - Build a batch of 1000 (200 abnormal and 800 normal?)
 - 768 size vector input
 - Same size hidden layer embedding
 - Output layer with two-class or just a softmax or ReLU operator?
 - In each batch, put some abnormal samples and lots of negative samples (i.e. normal samples) and learn to classify
 - Go through all the positive examples in multiple batches to develop the embedding
 - Chain it with a regular classifier (e.g. fully-connected network)
- **Inference:**
 - Feed the remaining abnormal and normal vectors at a time, use this embedding for classification
- **Two loss functions:**
 - Binary cross-entropy (direct learning from BERT)
 - Contrastive loss (Evolve a new embedding using BERT encoding)
- Does that look reasonable to try?

768 input vector per trace (normal or abnormal)

Binary output vector (1=abnormal 0=normal)

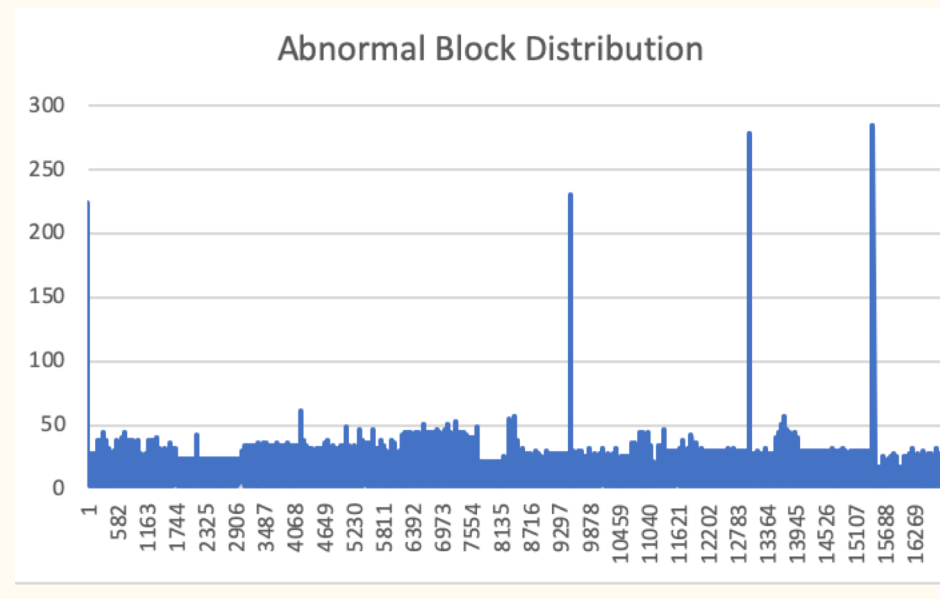
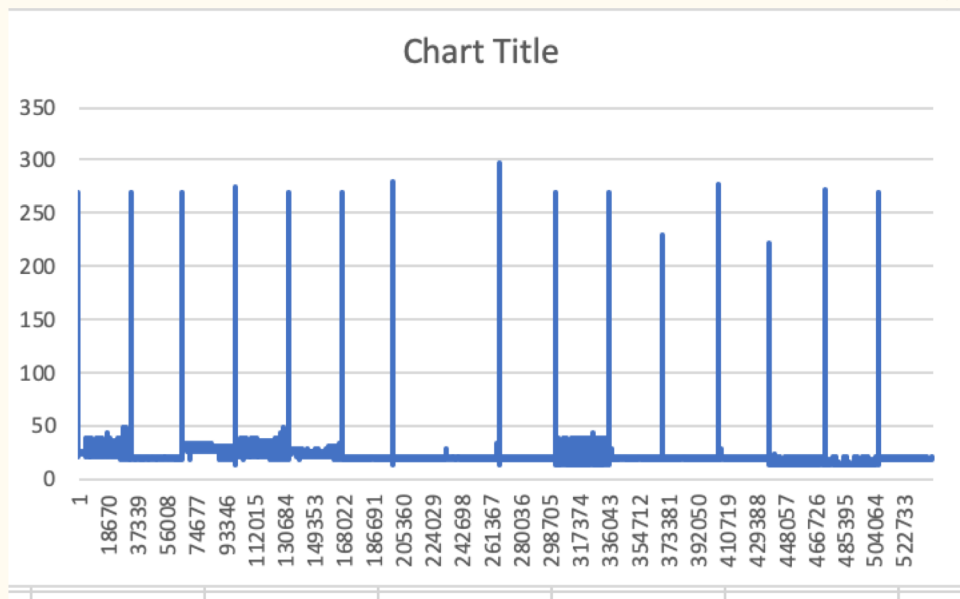


Update March 12, 2021

- Implementation of the contrastive learning of log anomalies
- Data preparation:
 - Processed HDFS_train.json, 541,385 samples
 - Processed HDFS_abnormal_test.json, 16,838 samples
 - Grouped all text per block
 - Formed BERT encoding per word, sentence, and group
 - Process takes a long time (1000 blocks per 20min or so)
 - Current experiments retained 1000 blocks each from normal and abnormal
- Contrastive model design
 - Input is a batchsize x BERT-encoded vector, each row represents a block encoding
 - 1000 x 768
 - Single hidden layer with RELU, projection layer for building the contrastive encoder
 - 20% abnormal and 80% normal in each batch
 - Trained on 100 epochs per batch
- To test the model, took an abnormal vector and ranked the nearest vectors from the contrastive encoding. If the nearest vectors were within the abnormal space, we could use this to classify later, but from the point of finding if it is normal or abnormal, this could be sufficient.

Block text size patterns for normal and abnormal blocks

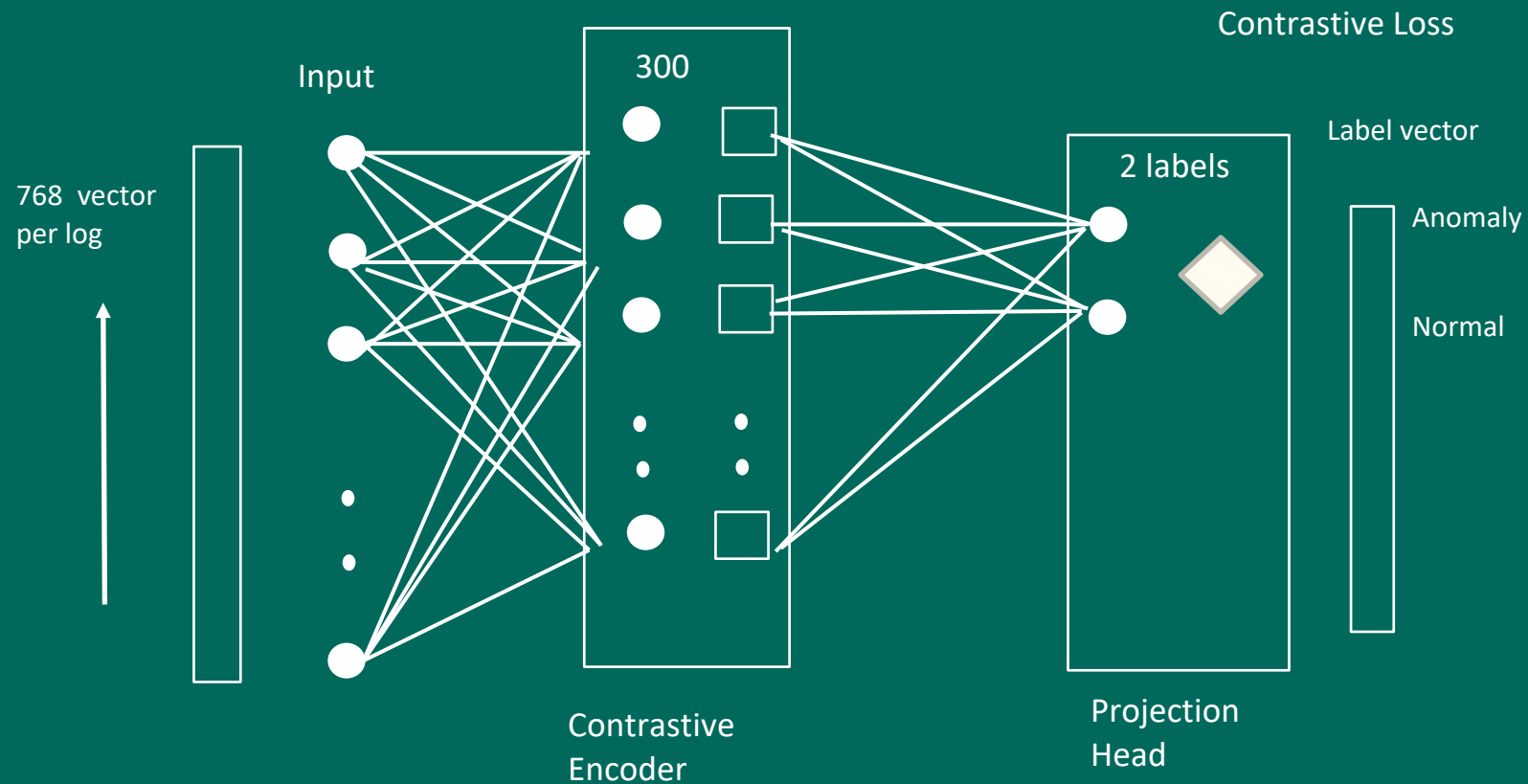
- Can these patterns be somehow exploited?
 - X-axis is sequence of blockIDs and y-axis is number of lines of text (left side is normal)



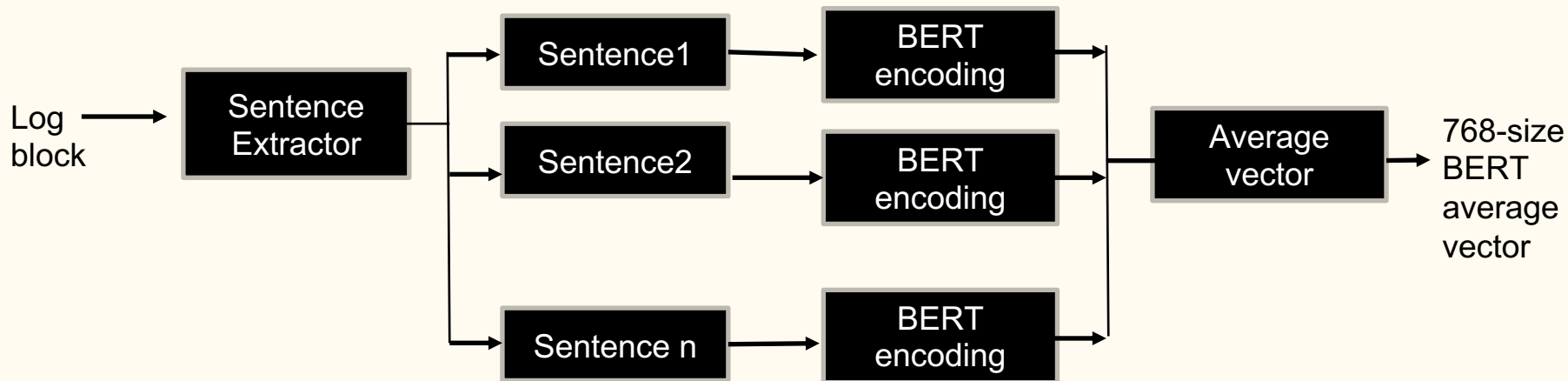
Train-test experiments

- 1000 abnormal blocks
- 2000 normal blocks
- Train-test split (80%,20%)
- During contrastive encoder learning
 - 800 abnormals, 1600 normal blocks
 - Batch size=768,
 - Projection head output=batch size
 - Encoder size=300
 - Learning rate =0.001
 - Temperature =0.05
- During contrastive classifier training
 - 800 abnormals, 1600 normal blocks (same blocks as above)
 - Batch size=300
 - Number of classes = 2
 - Dropout=0.5, hidden layer=150, same learning rate as encoder
- Accuracy on the test split: 99.67%

Supervised contrastive learning- Inference

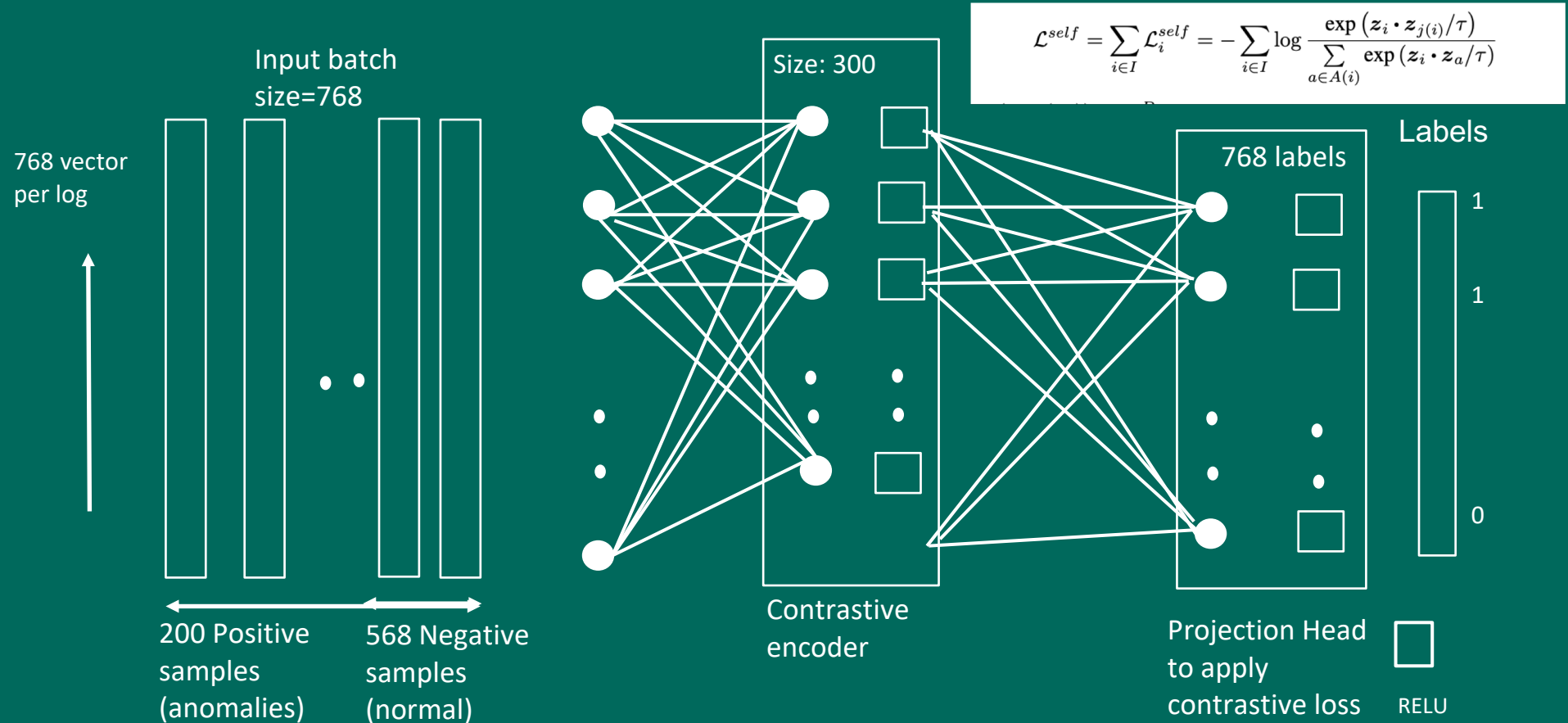


Supervised contrastive learning with BERT for HDFS logs – BERT-based encoding of blocks

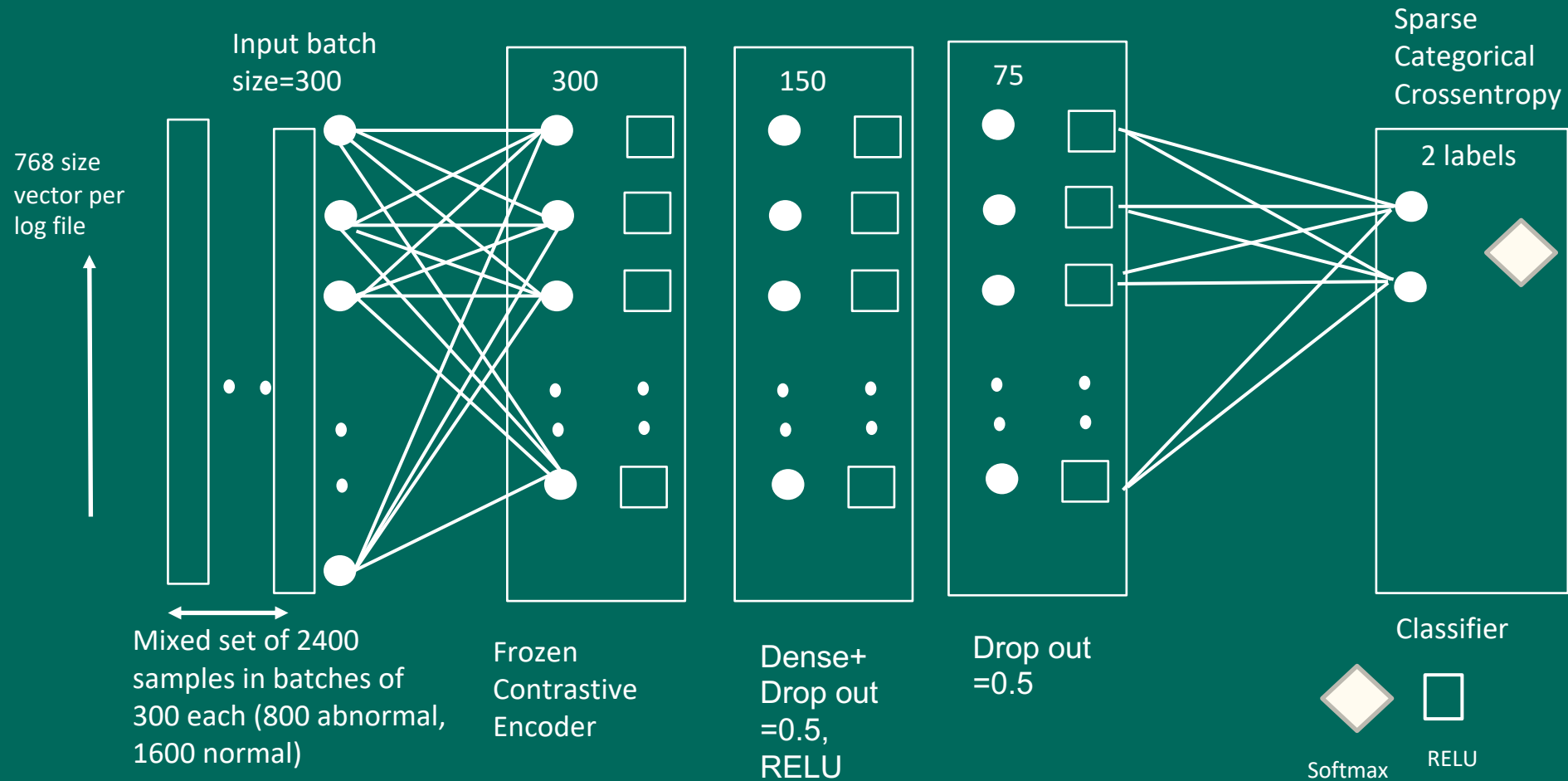


Log block Encoding

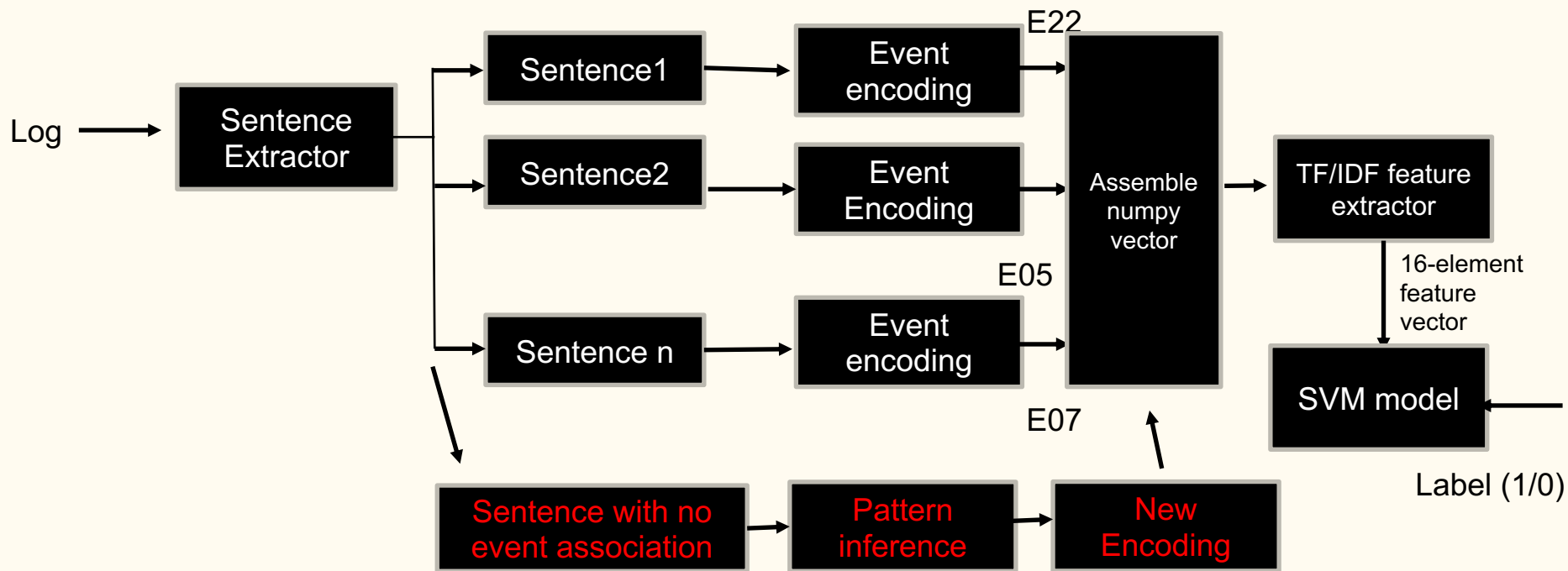
Contrastive Encoder Training



Supervised contrastive learning- Classifier with frozen encoder



Comparison with SVM model



Comparison with other models

- The models in loglizer used event IDs corresponding to each line in a block of text for each `block_id`.

1	81109	203518	143	INFO	dfs.DataNode\$Dat aXceiver	Receiving block blk_-1608999687919862906 src: /10.250.19.102:54106 dest: /10.250.19.102:50010 BLOCK* NameSystem.allocateBlock: /mnt/hadoop/mapred/system/job_200811092030_0001/job.jar.	E5
2	81109	203518	35	INFO	dfs.FSNamesystem	blk_-1608999687919862906	E22
3	81109	203519	143	INFO	dfs.DataNode\$Dat aXceiver	Receiving block blk_-1608999687919862906 src: /10.250.10.6:40524 dest: /10.250.10.6:50010	E5
4	81109	203519	145	INFO	dfs.DataNode\$Dat aXceiver	Receiving block blk_-1608999687919862906 src: /10.250.14.224:42420 dest: /10.250.14.224:50010	E5
5	81109	203519	145	INFO	dfs.DataNode\$Pac ketResponder	PacketResponder 1 for block blk_-1608999687919862906 terminating	E11
6	81109	203519	145	INFO	dfs.DataNode\$Pac ketResponder	PacketResponder 2 for block blk_-1608999687919862906 terminating	E11
7	81109	203519	145	INFO	dfs.DataNode\$Pac ketResponder	Received block blk_-1608999687919862906 of size 91178 from /10.250.10.6	E9
8	81109	203519	145	INFO	dfs.DataNode\$Pac ketResponder	Received block blk_-1608999687919862906 of size 91178 from /10.250.19.102	E9
9	81109	203519	147	INFO	dfs.DataNode\$Pac ketResponder	PacketResponder 0 for block blk_-1608999687919862906 terminating	E11

But these event Ids
are not available
for our dataset
<HDFS_train.json>

HDFS_100k.log_structured.csv

Mapping text to event IDs

- BLOCK* ask 10.251.91.84:50010 to replicate blk_-764842520721893215 to datanode(s) 10.251.39.209:50010
 - BLOCK* ask <*> to replicate <*> to datanode(s) <*>
 - (as given in HDFS_100k.log_structured.csv)
 - But to match the pattern I used
 - BLOCK* ask <regex> to replicate <regex> to datanode(s) <regex>
 - Regex= "[a-zA-Z0-9-_.\./]"
- Results:
 - 16838 abnormal blocks
 - 104,330 sentences that didn't match to the given patterns
 - 192470 out 288250 sentences or 66.7% coverage with the given event IDs
 - Extracted 10 new evnt patterns
 - Next step is to repeat the pattern extraction for normal blocks (10 million of those!)

New patterns extracted

- 10 new patterns from abnormal text

<*>:Got exception while serving <*> to <*>

<*>:Exception writing block <*> to mirror <*>

<*>:Failed to transfer <*> to <*> got java.io.IOException: Connection reset by peer

Adding an already existing block <*>

BLOCK* ask <*> to replicate <*> to datanode(s) <*>

BLOCK* ask <*> to replicate <*> to datanode(s) <*> <*>

BLOCK* NameSystem.addStoredBlock: blockMap updated: <*> is added to <*> size <*>

BLOCK* NameSystem.delete: <*> is added to invalidSet of <*>

Unexpected error trying to delete block <*> BlockInfo not found in volumeMap.

BLOCK* NameSystem.addStoredBlock: addStoredBlock request received for <*> on <*> size <*> But it does not belong to any file

Plan of next steps

- Extract all new event ID patterns for normal and abnormal train/test blocks
- Construct the sequence of ID patterns for each block of text
- Form a numpy array of these string literals and feed this to the conventional models
 - They will do their own feature extraction (e.g. SVM model is using TF/IDF feature extraction)
 - Run the model and obtain results
 - For the DeepLog using these sequences to train the LSTM model
- Comparison ideas:
 - Event ID-based features + SVM/DeepLog (hand-crafted features)
 - Directly encode text through BERT + SVM/Deeplog (deep learned features)
 - Directly encode text through BERT + Multi-label contrastive learning model (mine, pairwise learning of normal/abnormal patterns)
 - Trained and tested on full HDFS dataset

Event ID sequence recovery from text blocks

- The regex pattern matcher gave about 66.7% coverage with the given event IDs
- 91842 sentences out of 10,500,000 sentences still didn't find a match.
- Wondering if I can work with partial patterns, i.e. some sentences in a block may not match any pattern, and I can skip their event IDs and model this inherently as missing data?

`['E22', 'E5', 'E5', 'E5', 'E26', 'E26', 'E26', '?', 'E9', 'E11', 'E9', '?', 'E9']`

Larger scale testing of contrastive learning with BERT encoding

- Randomly sampled the 541385 normal blocks into 5 batches of 2000 blocks.
- Randomly sampled the 16838 abnormal blocks into 5 batches of 1000 blocks.
- Saved the BERT encodings of 100 blocks at a time into numpy arrays (took several hours)
- Trained the contrastive learning with cross-validation on these batches.
- During contrastive encoder learning
 - 800 abnormal, 1600 normal blocks, Batch size=768, Projection head output=batch size, Encoder size=300, Learning rate =0.001, Temperature =0.05
- During contrastive classifier training
 - 800 abnormal, 1600 normal blocks, Batch size=300, Number of classes = 2, Dropout=0.5, hidden layer=150, same learning rate as encoder
- Average accuracy across the batches for test splits: 97.32%

Contrastive learner ablation experiments

- From the 541385 normal training and 16838 abnormal, select randomly, 2000 normal and 1000 abnormals.
- 80% for training (=1600 normal, 800 abnormals).
- 20% for testing (=400 normal, 200 abnormals)
- Using trainable parameters for the contrastive encoder during classifier training (i.e. end-to-end training)
- Vector length=768 (BERT vector size)
- Batch size=768
- Epochs=10 (encoding, =100 for classifier)
- Effect of positive batch size = 0.03%, 0.01%, 2
 - 71.83 (2), 71.7% (8), 67.67% (23)

Smaller batch size gives improved accuracy
 Smaller positive batch size gives improved accuracy
 All of these were with trainable encoder even after self-training
 Batch size for classifier = 30 throughout

Batch size	Encoder epoch	Classifier epoch	Bert vector length	Contrastive encoder size	Training (normal)	Training (Abnormal)	Test (normal)	Test (abnormal)	Positive batch size	Learning rate/temperature	Accuracy
768	10	100	768	300(trainable)	1600	800	400	200	23	0.001/0.05	71.33
768	10	100	768	300(trainable)	1600	800	400	200	8	0.001/0.05	71.7
768	10	100	768	300(trainable)	1600	800	400	200	2	0.001/0.05	71.83
30	10	100	768	300 (trainable)	1600	800	400	200	2	0.001/0.05	72.67
30	10	100	768	300 (fixed)	1600	800	400	200	2	0.001/0.05	71.33
300	10	100	768	300 (trainable)	1600	800	400	200	2	0.001/0.05	70.33

