International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 08 Issue: 02 | Feb 2021 www.irjet.net p-ISSN: 2395-0072

Evaluation of Different Optimizers in Neural Networks with Imbalanced Dataset

Santosh Raghavendraraju[1],Richard Martin.M[2], Keerthi Varman.G[3],Senbagavalli.M[4]

[1][2][3]Students, Department of CSE, Alliance University
[4]Assoc. Professor, Department of Information Technology, Alliance University

---

Abstract -*With an increasing number of embedded sensor systems and data collection units set up in production plants, machines, cars, etc., there are new possibilities to store, analyse and monitor the data from such systems. These development makes it possible to detect anomalies and predict the failures that affect availability of these systems and impact maintenance plans. Typical industry scenario points towards have very less failures and data points related to same being captured in systems making it difficult to predict a rare event. This paper would be focusing towards evaluating the different optimizers and impact they have on accuracy while trying to predict a rare event target in a time series-based data. We would be evaluating different built-in optimizer classes in by tensor flow for training neural networks.*

Key Words: **Failure Prediction, Neural Networks, Component Failure, Python, Optimizers, Adadelta, Adagrad, Adam, Adamax & Nadam**

1.INTRODUCTION

Optimizers have been widely used in deep learning. Although one of the most preferred algorithms has been Adam recently, its comparison with other optimization algorithms for large datasets with imbalanced targets for binary classification when training deep neural networks has not well evaluated and documented. The evaluation requires manual tuning of learning rate and validating the results for every run. Problems in initial optimization algorithms like SGD had given space for invention of more advanced algorithms. Now a days, the optimization algorithms used for deep learning adapt their learning rates during training. Current paper will focus towards evaluating Adam versus Adadelta, Adagrad, Adamax and Nadam in regard to model accuracy.

2. BACKGROUND AND RELATED WORK

In deep learning literature, working principles and performance analysis of optimization algorithms are widely studied. For example, theoretical guarantees of convergence to criticality for RMSProp and Adam are presented in the setting of optimizing a non-convex objective 9. They design experiments to empirically study the convergence and generalization properties of RMSProp and Adam against Nesterovs accelerated gradient method. In another study, conjugate gradient, SGD and limited memory BFGS algorithms are compared 5. A review is presented on numerical optimization algorithms in the context of machine learning applications 1. Additionally, similar to this work, an overview of gradient optimization algorithms is summarized 8. In this study, most widely used optimization algorithms are examined in the context of deep learning. On the other side, new variants of adaptive methods still have been proposed more recently. For example, new variants of Adam and AMSGrad, called AdaBound and AMSBound respectively, are proposed 6. They employ dynamic bounds on learning rates to achieve a gradual and smooth transition from adaptive methods to SGD. Also, a new algorithm that adapts the learning rate locally for each parameter separately and also globally for all parameters together is presented 3. Another new algorithm, called Nostalgic Adam (NosAdam), which places bigger weights on the past gradients than the recent gradients when designing the adaptive learning rate is introduced 4. In another study, two variants called SC-Adagrad and SC-RMSProp are proposed 7. A new adaptive:optimization algorithm called YOGI is presented 10.A novel adaptive learning rate scheme, called ESGD, based on the equilibration preconditioned is developed 2. Also, a new algorithm called Adafactor is presented 26. Instead of updating parameters scaling by the inverse square roots of exponential moving averages of squared past gradients, Adafactor maintains only the per-rowand

per-column sums of the moving averages, and estimates the per-parameter second moments based on these sums

3. METHODOLOGY

Below picture depicts high level workflow around the steps involved in developing and evaluating a failure prediction solution with different optimizers. Baseline model developed will be evaluated for accuracy and number of true failure events capture by changing the optimizer parameter.

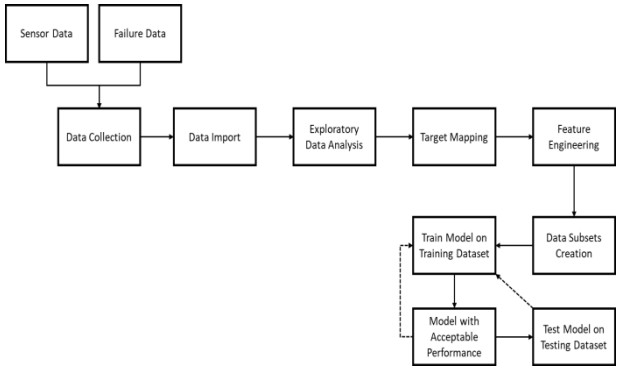

Fig -1:

4.IMPLEMENTATION AND RESULTS

Step 1: Data Collection

Azure Preventive analytics open user data around Predictive Maintenance has been used in the current analysis. The available dataset details are as follows,

Machine conditions and usage: Captures operating conditions of a machine e.g., data collected from sensors at hourly level for 4 different sensor tags. This is a telemetry time-series data. It consists of voltage, rotation, pressure, and vibration measurements collected from 100 machines in real time averaged over every hour collected during the year 2015.

Failure history: The failure history of a machine or component within the machine. These are the records of component replacements due to failures. Each record has a date and time, machine ID, and failed component type. Data for 100 machines (Two machine models), failures by 4 components are captured.

Maintenance history: The repair history of a machine, e.g., error codes, previous maintenance activities or component replacements. These are the scheduled and unscheduled maintenance records which correspond to both regular inspections of components as well as failures. A record is generated if a component is replaced during the scheduled inspection or replaced due to a breakdown. The records that are created due to breakdowns will be called failures which are explained in the later sections. Maintenance data has both 2014 and 2015 records

For the current study, we would be using only Telemetric& Failure data to build a neural network-based solution to predict failures.

Step 2: Data Import

Next step in process was to import the Telemetric and Failure data into the python work environment and create pandas data frame of it.

- Python (Pandas, Pandas SQL) using for importing data & shaping the data to required format
- Imported "PdM_telemetry" and "PdM_failures" data files into python
- Formatted "datetime" column to required format i.e. %Y-%m-%d %H:%M:%S format
- Sorting both datasets based on "machineID" and "date time" features
- Conversion of 'volt', 'rotate', 'pressure' & 'vibration' features to numeric format
- Above processing carried out separately on individual datasets before any merging
- Merging of failures data to Telemetry data
- Merging based on "date time" and "machineID" condition"
- "failure" column captures the failure component name- "comp1", "comp2, "comp3" or "comp4"
- Data is merged in a way to create 4 new indicator columns to base table (i.e. telemetry) - i.e. indicators of whether the related component failure happened or not for the machine &date time (Value =1, if there was a failure for machine on that particular timestamp)

Step 3: Exploratory Data Analysis

- Understand the distribution of individual source features
- Distribution of failures by machine & failure type
- Distribution of failures by Month-Year by Machine for each failure type
- Outlier identification
- Identification of percentiles (P1, P5, P10, P90, P95, P99)
- Finalization of lower and upper percentiles by component for creating Lower and Upper data thresholds
- Extreme outlier treatment- cap and floor of values

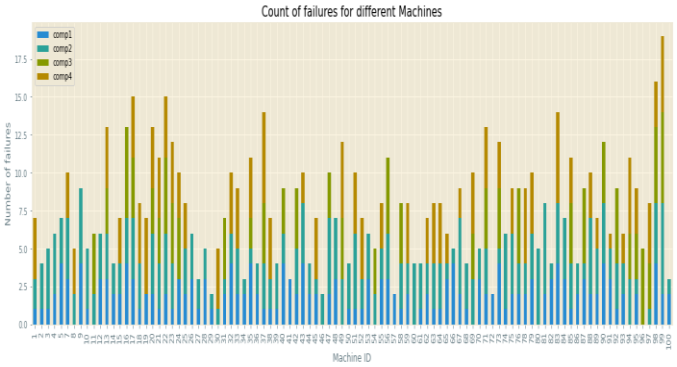

Fig -2:

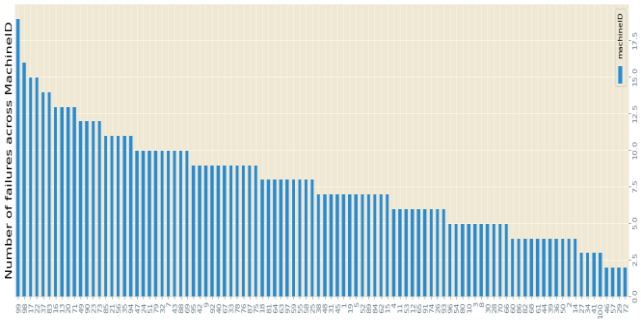

Fig -3:

Step 4: Target Mapping

Failure events mapping to telemetric data was carried out based on machineID and datetime variables. Since both failure and telemetric were represented at hour scale on datetime, data could be merged easily without loss of any failure data. This paper currently will focus on component 1 type failure and the count of failuresavailablefor analysis is limited to 192. Less than 1% of the total population i.e., telemetric data points.

Below table details the number of failures occurrence every month as captured in the failure dataset.

Year Month	Failure_count of Failure_comp1	Failure_count of Failure_comp 2	Failure_count of Failure_comp 3	Failure_count of Failure_comp 4
2015-01	25	30	17	22
2015-02	19	11	7	13
2015-03	18	15	11	15
2015-04	15	28	11	15
2015-05	16	21	10	17
2015-06	15	18	12	13
2015-07	18	24	11	11
2015-08	15	21	12	16
2015-09	12	19	12	16
2015-10	9	27	8	15
2015-11	15	23	12	12
2015-12	15	22	8	14
2016-01	0	0	0	0
GRAND TOTAL	192	259	131	179

Table -1:

Step 5: Feature Engineering

In view of limited sensor tags and the data being time series in nature, we undertook steps to engineer new features from the raw sensor readings to enrich the data. All the features derived are based on the statistical nature of the data and no domain engineered features were derived during this exercise. Below listed are details of some of the features derived for each of the individual sensor tag.

- Following features to be derived (Phase 1) for all the 4 source features ('volt', 'rotate', 'pressure' & 'vibration')
  - If feature value <= Lower threshold then feature_LT
  - If feature value >= Upper threshold then feature_UT
  - "Change over time Cm+1 = (Xm+1 – Xm)/(tm+1 – tm)"
  - "Rate of change over time RTm= (Cm+1 – Cm)/(tm+1 – tm)"
  - "Growth or decay Gm+1 = (Xm+1 – Xm)/Xm"
  - "Rate of growth or decay RGm= (Gm+1 – Gm)/(tm+1 – tm)"

- Count of obs above Threshold
- Count of obs Below Threshold
- Moving average = Average of (Xm-p to Xm)
- Moving standard deviation = Standard deviation of (Xm-p to Xm)
- Relative average = Moving average / Global average
- Relativestandard deviation = Moving standard deviation / Global standard deviation
- Ratio: Change Over time to SD
- Ratio: Rate of Change Over time to SD
- Ratio: Growth or Decay Over time to SD
- Ratio: Rate of Growth or Decay Over time to SD

Step 6: Data subsets creation

In view of limited sensor tags and the Target data being very limited and rare in our scenario, we decided to have at least 80% of the data for training the baseline model and remaining 20% for testing the model. Any data points falling under Jan'2016 was excluded from analysis as there were no targets captured for that month.

Year Month	Failure Count of Target_ Comp1	Failure Count of Target_ Comp2	Failure Count of Target_ Comp3	Failure Count of Target_ Comp4	Train/Test
2015-01	25	30	17	22	Train
2015-02	19	11	7	13	Train
2015-03	18	15	11	15	Train
2015-04	15	28	11	15	Train
2015-05	16	21	10	17	Train
2015-06	15	18	12	13	Train
2015-07	18	24	11	11	Train

Table -1:

2015-08	15	21	12	16	Train
2015-09	12	19	12	16	Train
2015-10	9	27	8	15	Test
2015-11	15	23	12	12	Test
2015-12	15	22	8	14	Test
2016-01	0	0	0	0	Excluded
Grand Total	192	259	131	179	

Table -1:

Step 7: Train Model

Defined a function that creates a simple neural network with a densely connected hidden layer, a dropout layer to reduce over fitting, and an output sigmoid layer that returns the probability of a event being failure. We trained the model and validated using different optimization algorithms by changing the parameter value in optimizer within the model. Compile class. We defined number of epochs to be 200 and batch size to its default value of 2048 records.

```
METRICS = [
    keras.metrics.TruePositives(name='tp'),
    keras.metrics.FalsePositives(name='fp'),
    keras.metrics.TrueNegatives(name='tn'),
    keras.metrics.FalseNegatives(name='fn'),
    keras.metrics.BinaryAccuracy(name='accuracy'),
    keras.metrics.Precision(name='precision'),
    keras.metrics.Recall(name='recall'),
    keras.metrics.AUC(name='auc'),
]
```

```
def make_model(metrics=METRICS,
output_bias=None):
  if output_bias is not None:
    output_bias =
tf.keras.initializers.Constant(output_bias)
  model = keras.Sequential([
     keras.layers.Dense(
        16, activation='relu',
      input_shape=(train_features.shape[-1],)),
     keras.layers.Dropout(0.5),
     keras.layers.Dense(1, activation='sigmoid',
                  bias_initializer=output_bias),
  ])

model. Compile(
     optimizer=keras.optimizers.Adam(lr=1e-3),
     loss=keras.losses.BinaryCrossentropy(),
     metrics=metrics)

  return model
```

5. RESULTS

Nadam, Adam and Adamax have good performance in detecting the true positive, though all had considerable false positive cases as well. Nadam seems to outform Adam in terms of performance and with reduced false positive cases while still maintaining same level of true positive capture. Study shows that Adam might be popular and been used heavily, but there are other optimizers that might fare well based on type of data and the model we trying to build for our use case.

Adam:

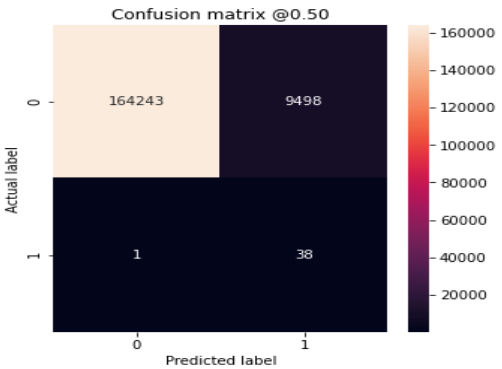

Fig -4:

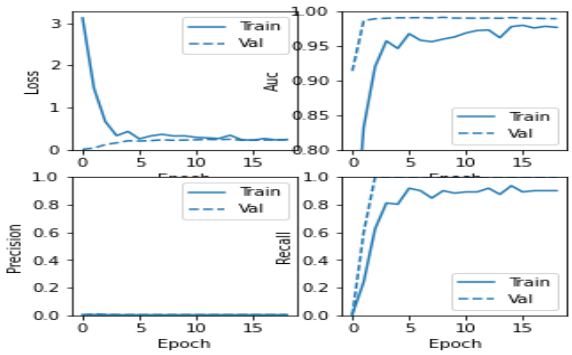

Fig -5:

Adadelta

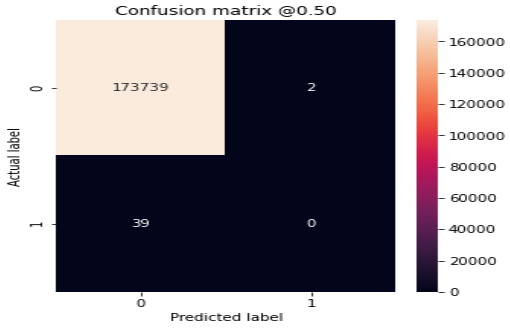

Fig -6:

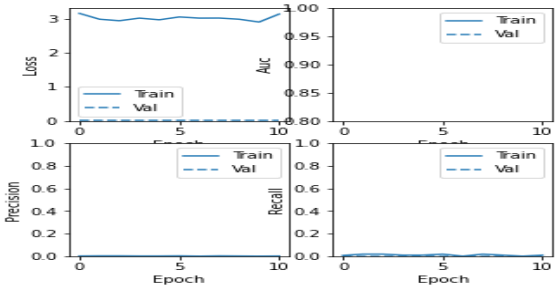

Fig -7:

Adagrad

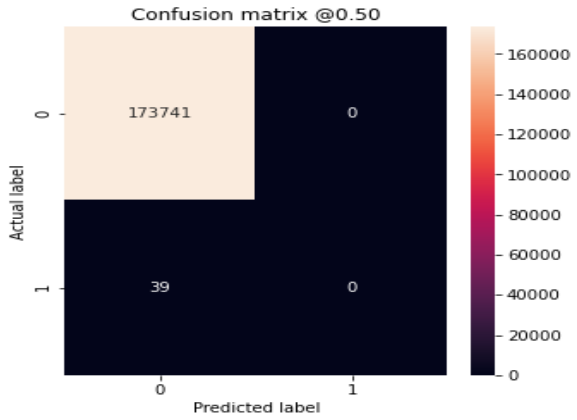

Fig -8:

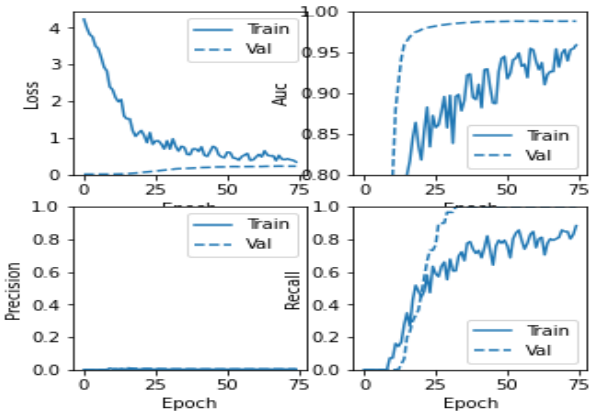

Fig -11:

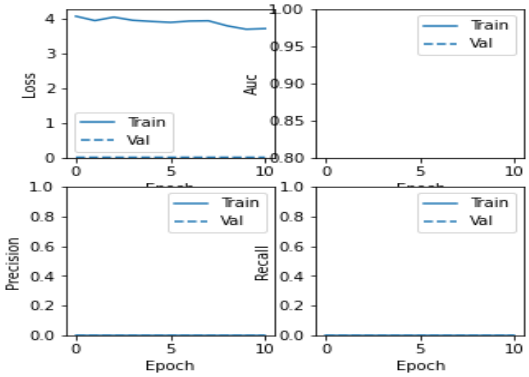

Fig -9:

Nadam

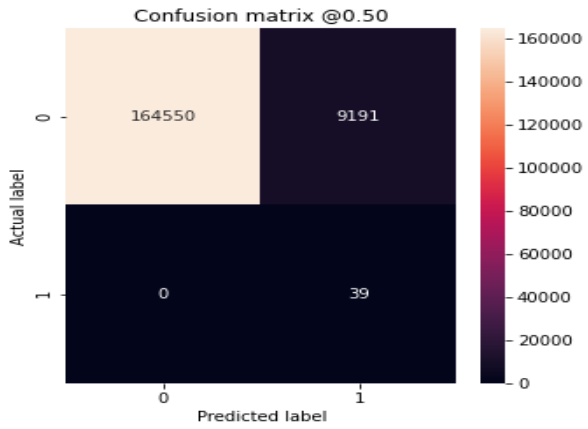

Fig -12:

Adamax

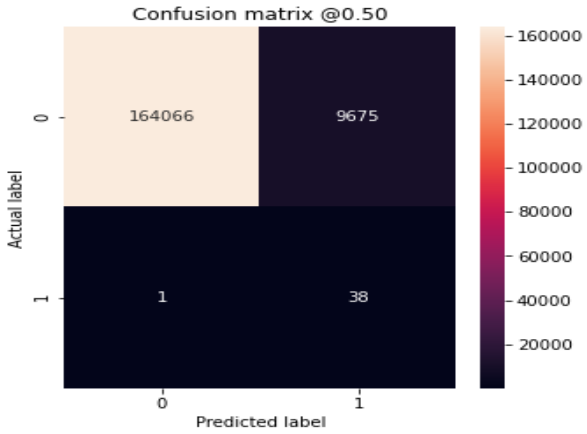

Fig -10:

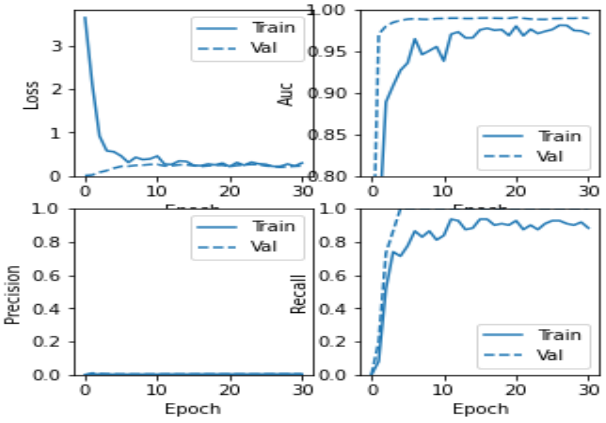

Fig -13:

References:

1. L. Bottou, F.E. Curtis and J. Nocedal, Optimization methods for large-scale machine learning, Siam Review, 60(2) (2018) 223{311.

2. Y. Dauphin, H. De Vries, J. Chung and Y. Bengio, RMSProp and equilibrated adaptive learning rates for non-convex optimization, CoRR abs/1502.04390, 2015.

3. H. Hayashi, J. Koushik and G. Neubig, Eve: A gradient based optimization method with locally and globally adaptive learning rates, arXiv preprint arXiv:1611.01505, 2016.

4. H. Huang, C. Wang and B. Dong, Nostalgic Adam: Weighing more of the past gradients when designing the adaptive learning rate, arXiv preprint arXiv:1805.07557, 2018.

5. Q.V. Le, J. Ngiam, A. Coates, A. Lahiri, B. Prochnow and A.Y. Ng, On optimization methods for deep learning, in Proc. 28th International Conference on Machine Learning, Bellevue, Washington, USA, 2011, pp. 265{272.

6. L. Luo, Y. Xiong, Y. Liu and X. Sun, Adaptive gradient methods with dynamic bound of learning rate, International Conference on Learning Representations, New Orleans, 2019.

7. M.C. Mukkamala and M. Hein, Variants of RMSProp and Adagrad with logarithmicregret bounds, in Proc. 34th International Conference on Machine Learning, Sydney,Australia, 70 (2017) 2545{2553.

8. S. Ruder, An overview of gradient descent optimization algorithms, arXiv preprintarXiv:1609.04747, 2016.

9. N. Shazeer and M. Stern, Adafactor: Adaptive learning rates with sublinear memorycost, arXiv preprint arXiv:1804.04235, 2018.

10. D. Soham, M. Anirbit and U. Enayat, Convergence guarantees for RMSProp andAdam in non-convex optimization and an empirical comparison to Nesterov acceleration, arXiv preprint rXiv:1807.06766, 2018.