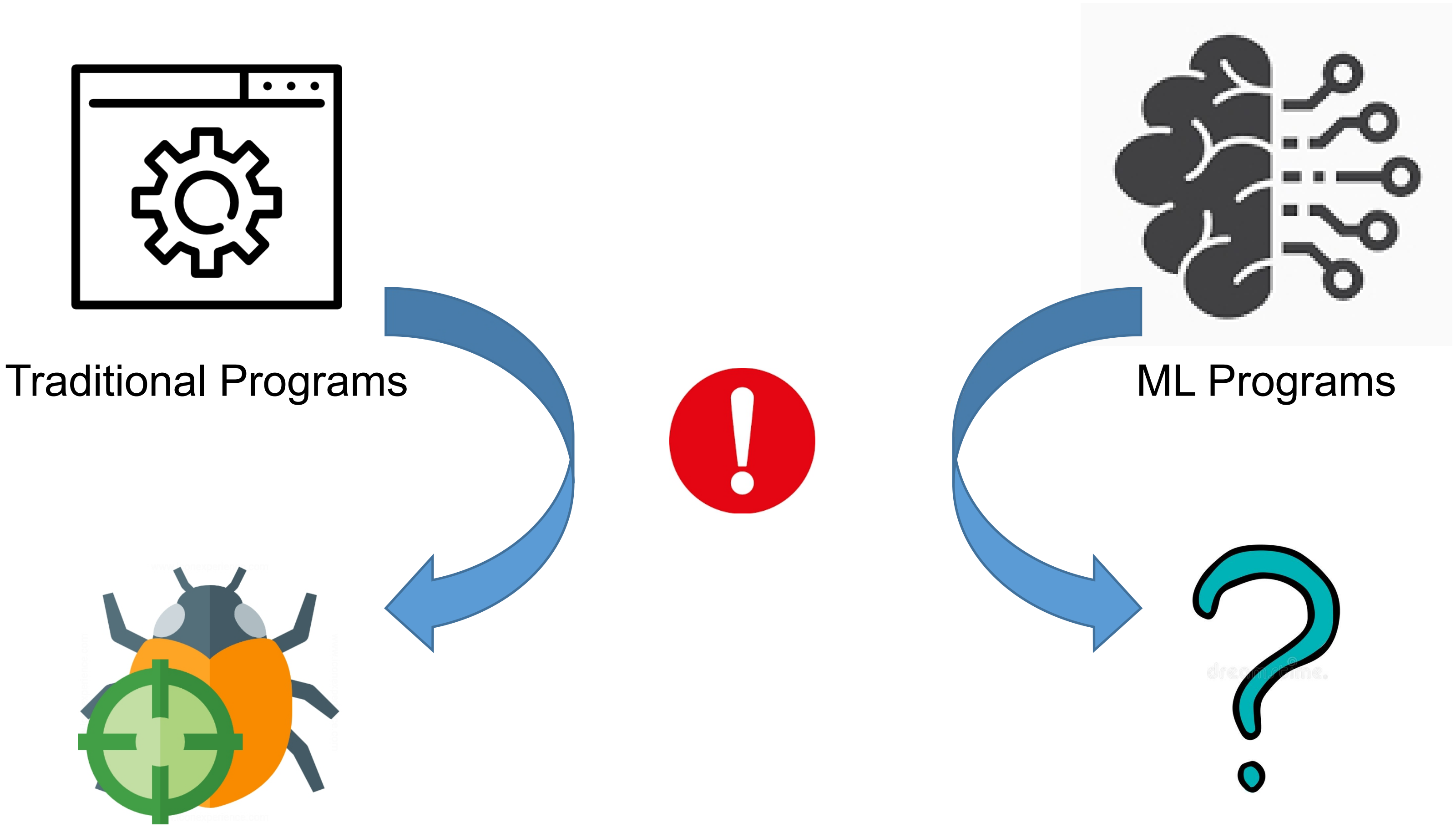
Amazon sagemaker debugger: A system for real-time insights into machine learning model training

Debugging in machine learning programs?

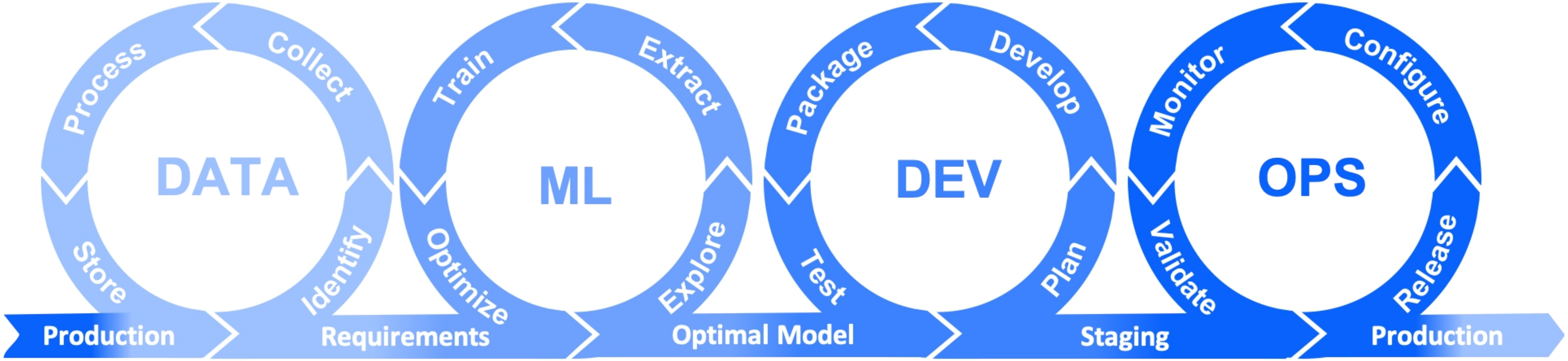


Debugging in machine learning programs?

- Bugs in traditional programs:
  - Segmentation fault
  - Division by zero
  - Business logic related error
  - ...
  - Symptom: Program crash / Error code (Exception thrown) ...
- Bugs in machine learning programs:
  - Low model capacity
  - Bad hyperparameter settings
  - Biased training data
  - Numerically unstable operations
  - ...
  - Symptom: Poor accuracy

Use rule-based method to detect common failure types in all stages of ML lifecycle.

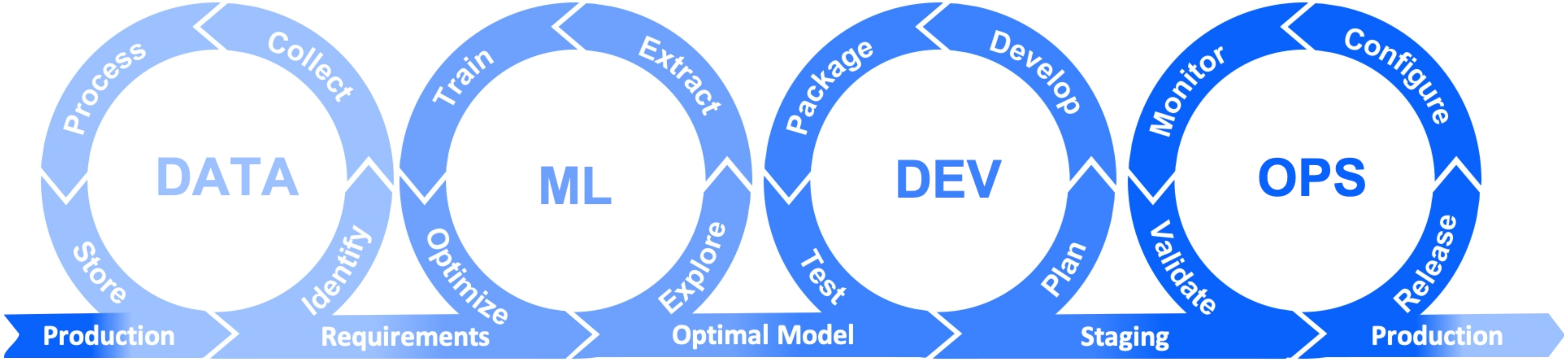
Potential bugs in ML lifecycle



Data preparation: Data cleaning, Data pre-processing, Feature engineering

Data don't contain representative samples: Underfit the dataset
Data imbalance: Overfit on parts of the data
Data not normalized

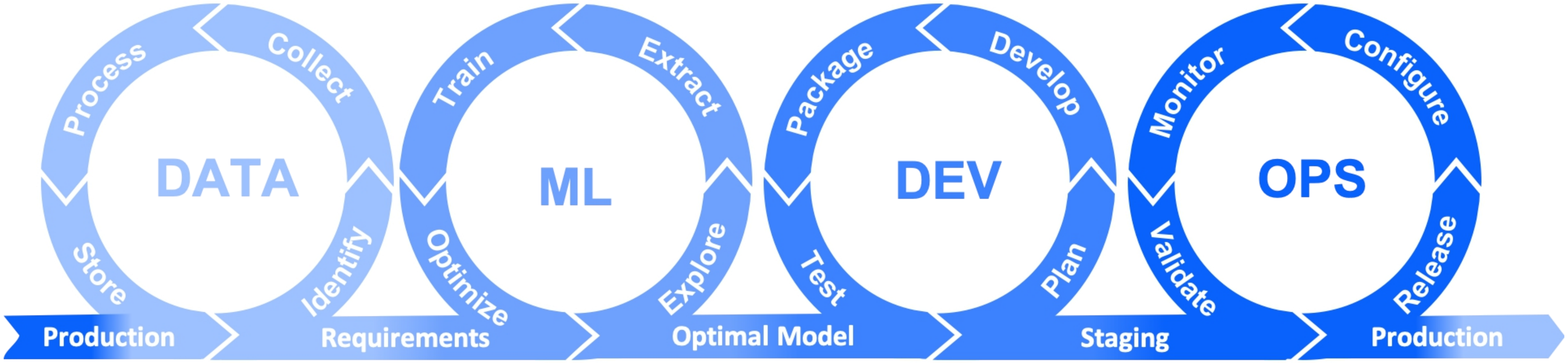
Potential bugs in ML lifecycle



Model training: Different configurations and model architecture is applied.

The model has too few parameters: Cannot converge
The model has many parameters: Overfit
Inproper {initialization schema, optimizer siettings, layerr configurations, hyper-parameters}: Cannot converge

Potential bugs in ML lifecycle

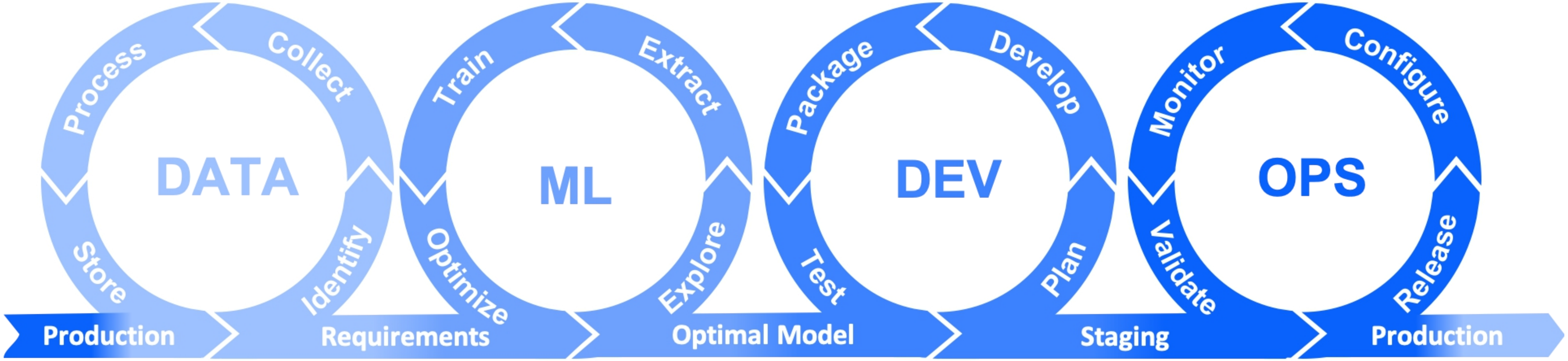


Hyperparameter tuning: Further refine a good model configuration

Non-optimal combination of hyper-parameters: The model has sub-optimal performance

ML model doesn't use full computation resources: Sub-optimal performance (a new feature that wasn't mentioned in the paper)

Potential bugs in ML lifecycle

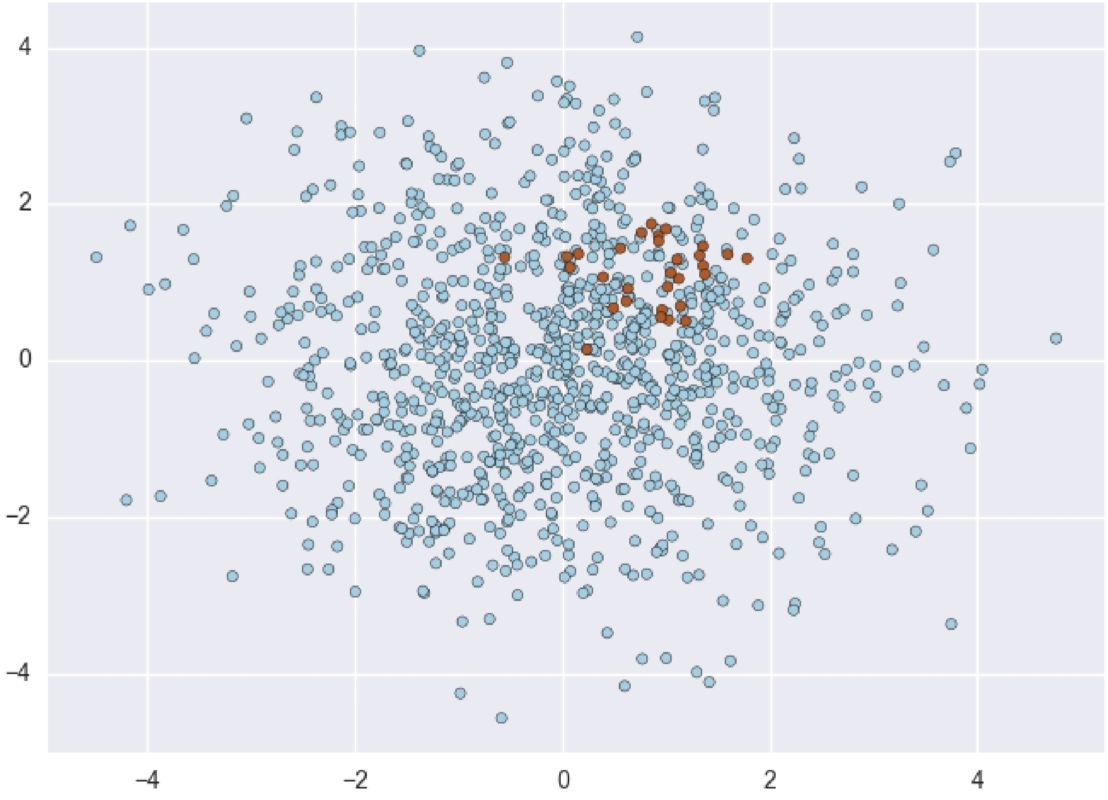


Deployment: Deploy ML model into a computation cluster

Distribution of the data inference is significantly different from the distribution of training data: Wrong prediction results

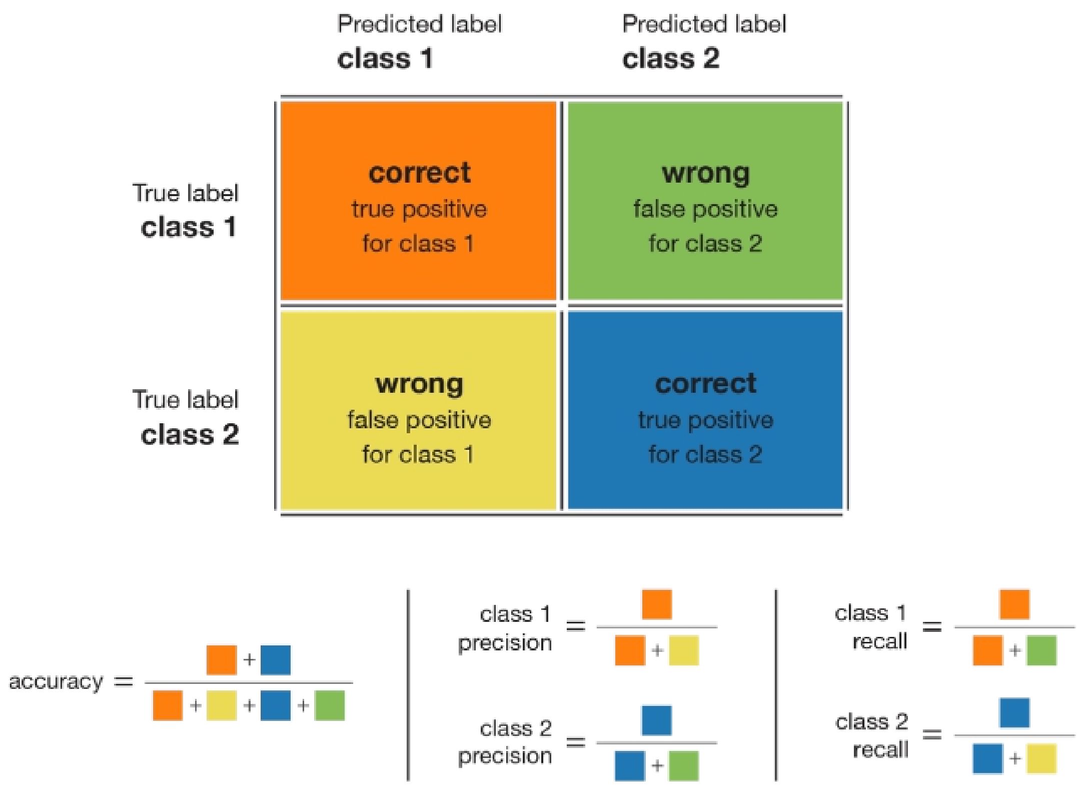
Rule-based bug-detection: Data preparation

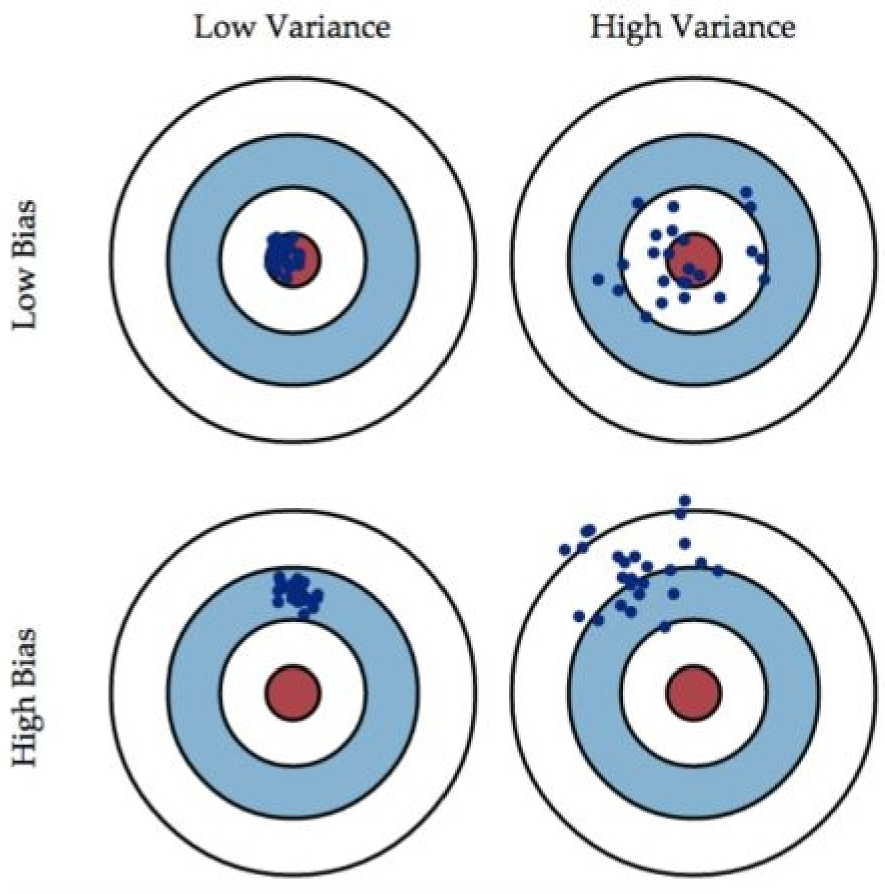
- Problem: Data imbalance



Rule-based bug-detection: Data preparation

- Problem: Data imbalance





Rule-based bug-detection: Data preparation

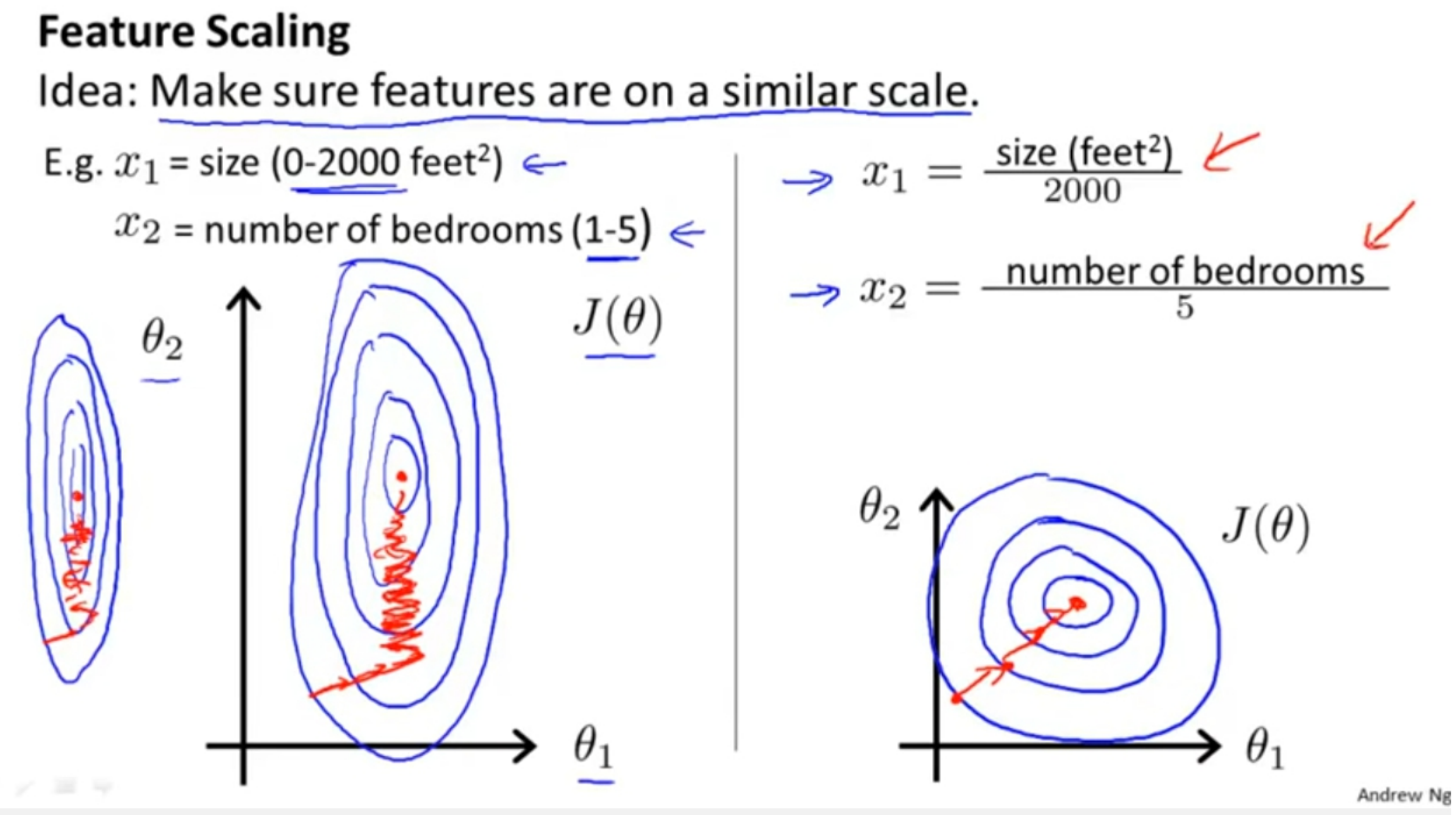
- Solution to Data imbalance

    Imbalance ratio+threshold

$$IR = \frac{max_i \zeta_i}{min_j \zeta_j}$$

Rule-based bug-detection: Data preparation

- Data not normalized

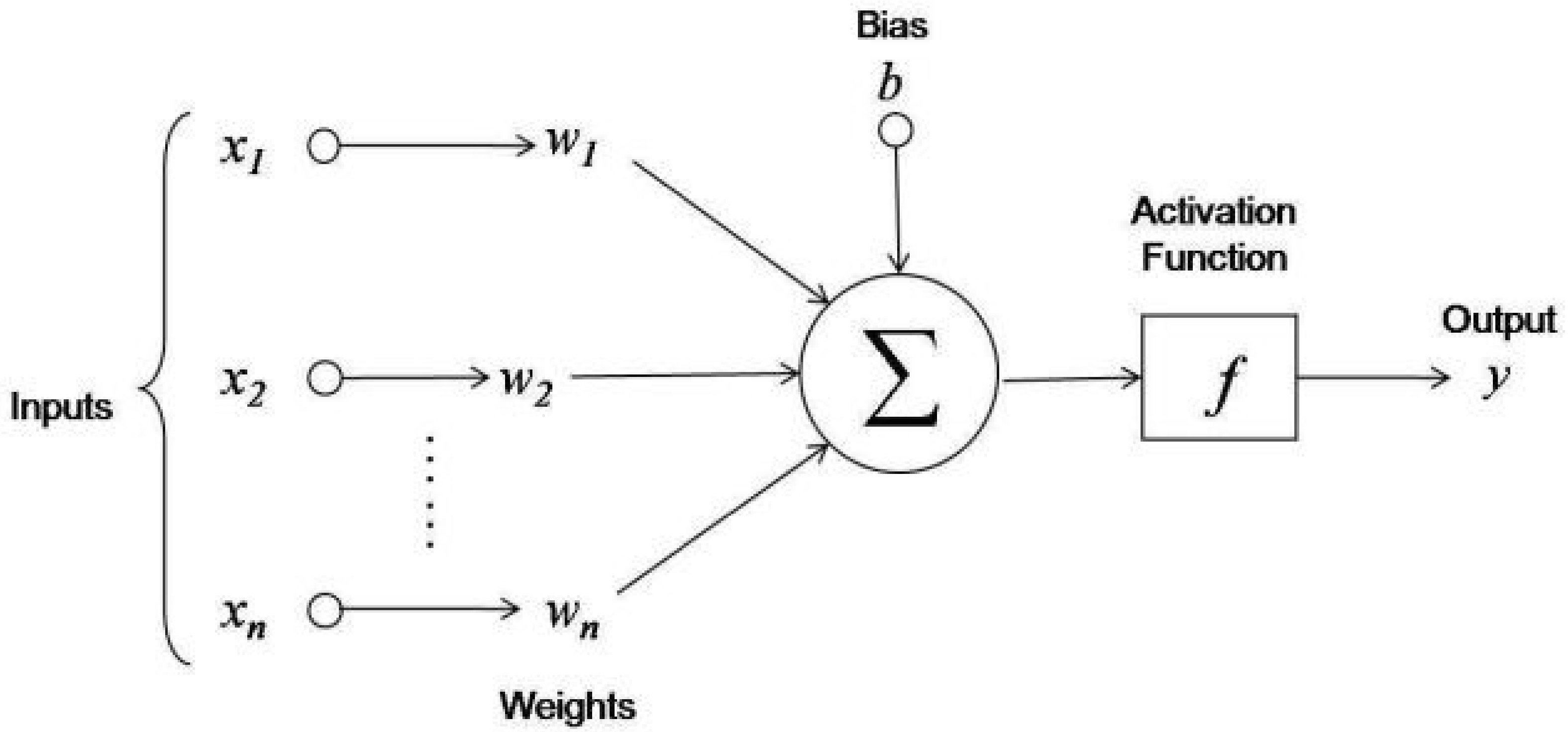


Rule-based bug-detection: Data preparation

- Solution to Data not normalized
  - Check if the data has zero mean
  - Check if the data has zero variance

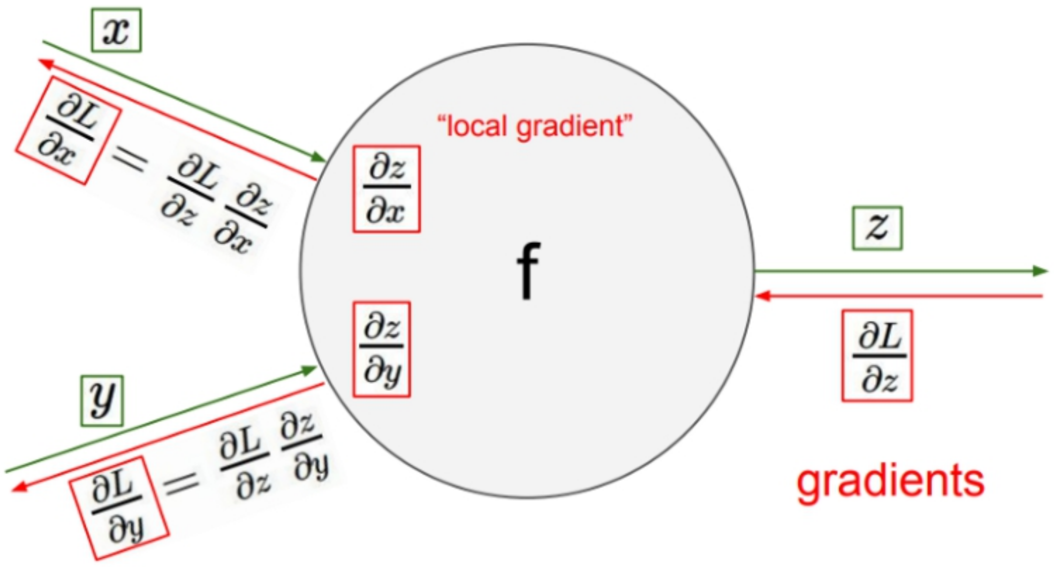
Rule-based bug-detection: Model training

- Activation function chosen improperly



Rule-based bug-detection: Model training

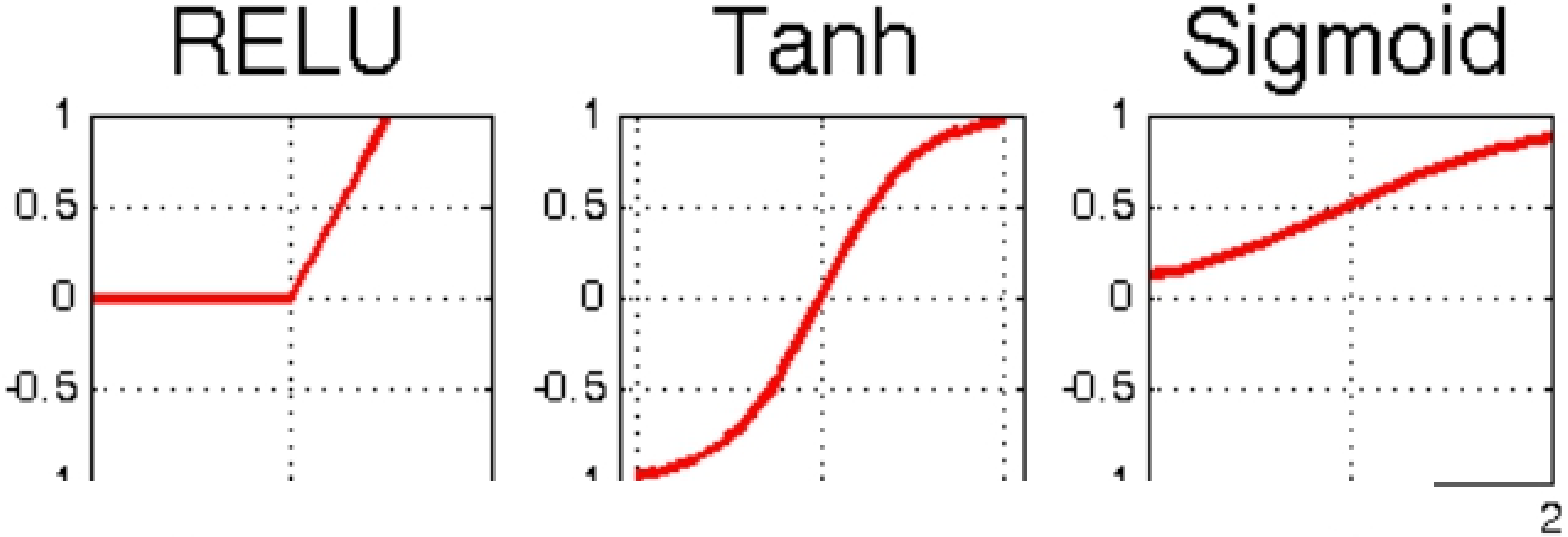
- Activation function chosen improperly



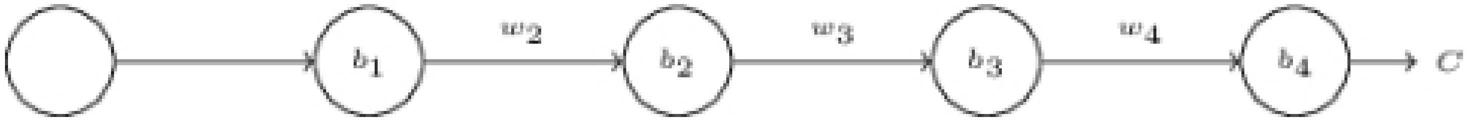
$$\Delta w = -\alpha \frac{\partial Loss}{\partial w}$$

Rule-based bug-detection: Model training

- Activation function chosen improperly
- Vanishing gradient (Activation function saturation), Gradient explode (Gradient too large)



$$\frac{\partial C}{\partial b_1} = \sigma'(z_1) \times w_2 \times \sigma'(z_2) \times w_3 \times \sigma'(z_3) \times w_4 \times \sigma'(z_4) \times \frac{\partial C}{\partial a_4}$$



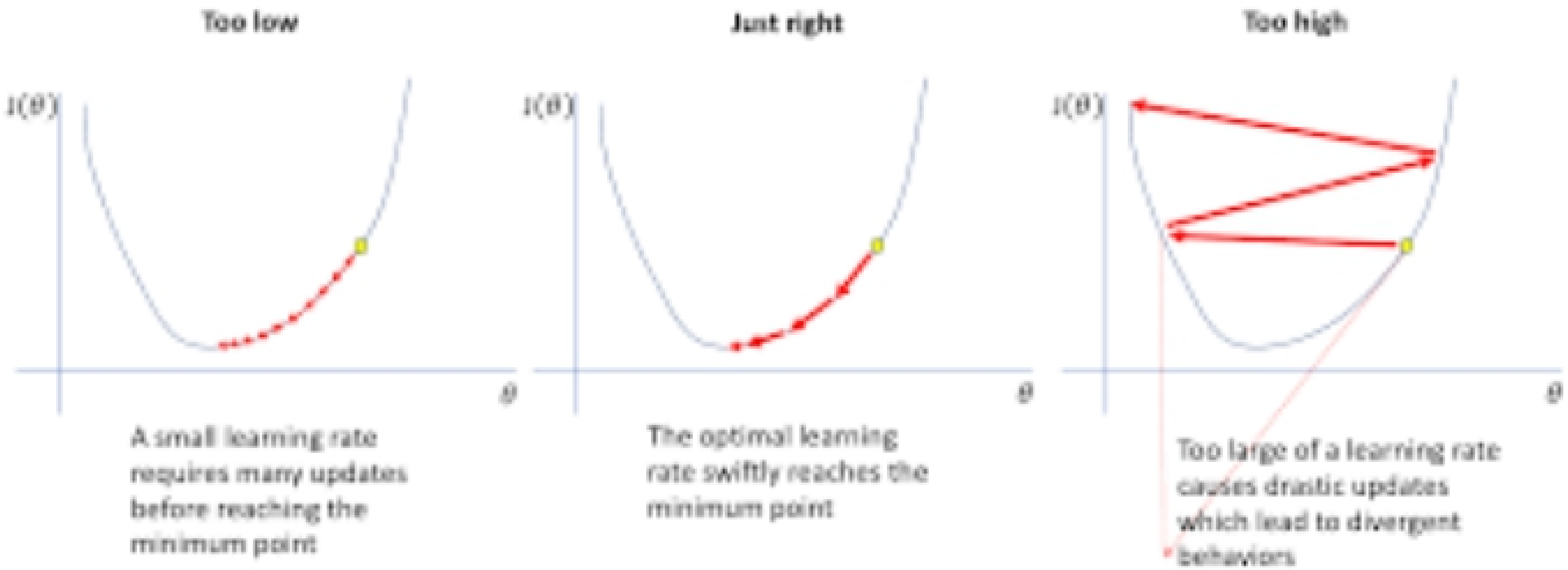
Rule-based bug-detection: Model training

- Solution to Gradient Vanishing: Retrieve activation outputs across steps and determine how many neurons in a model output zero values

- Solution to Gradient Explode: Retrieve activation outputs across steps and determines how many neurons have very large gradients

Rule-based bug-detection: Model training

- Optimizer options not chosen properly

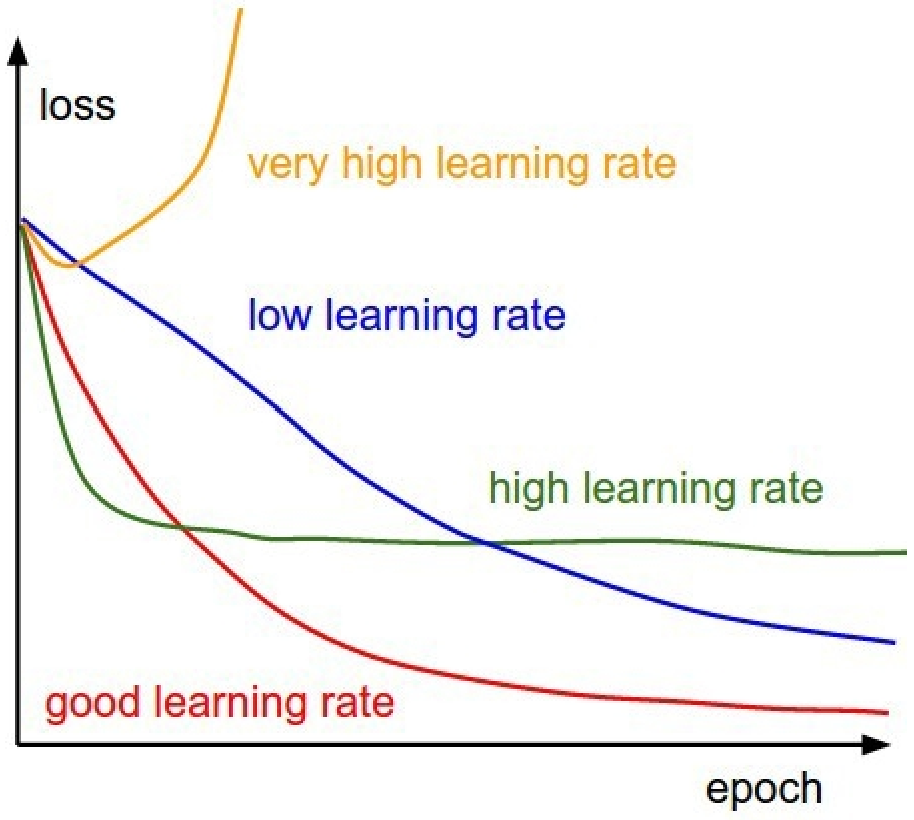


Rule-based bug-detection: Model training

- Solution to optimizer options not chosen properly:
    - Check if output changes too fast

Rule-based bug-detection: Model training

- Model won’t converge/model already converge but are still training

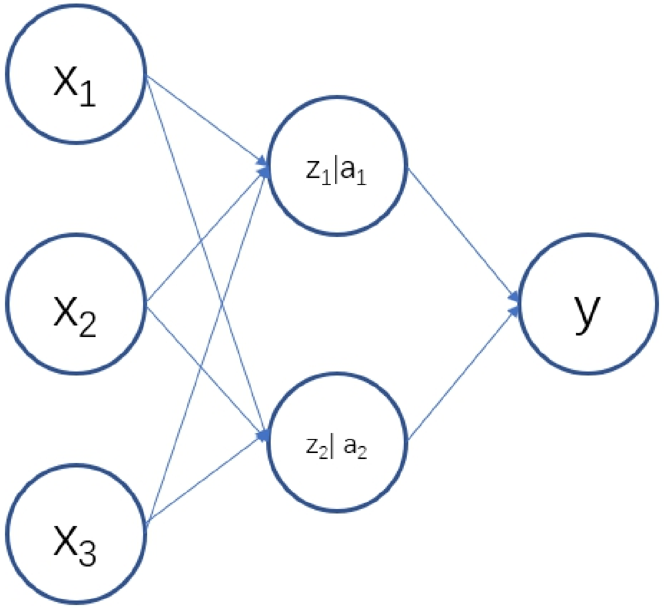


Rule-based bug-detection: Model training

- Solution to model won’t converge/model already converged but are still training
  - Stop training when loss exceed certain limit
  - Stop training when loss doesn’t change very often

Rule-based bug-detection: Hyperparameter tuning

- Parameter initialization wrong
  - eg: Initialization all weights to zero



z1 = w10 * x0 + w11 * x1 + w12 * x2 +w13 * x3

z2 = w20 * x0 + w21 * x1 + w22 * x2 +w23 * x3

Rule-based bug-detection: Hyperparameter tuning

- Solution to parameter initialization wrong
  - Check variance of output layers. A large variance may mean incorrect results.

Rule-based bug-detection: Tree model related

- Problem1: Tree model will overfit dataset when the tree depth will reach certain height.
- Problem2: Feature redundancy will occur if features are linearly dependent

- Solution:
  - Check tree depth
  - Check whether features are linearly dependent

System design

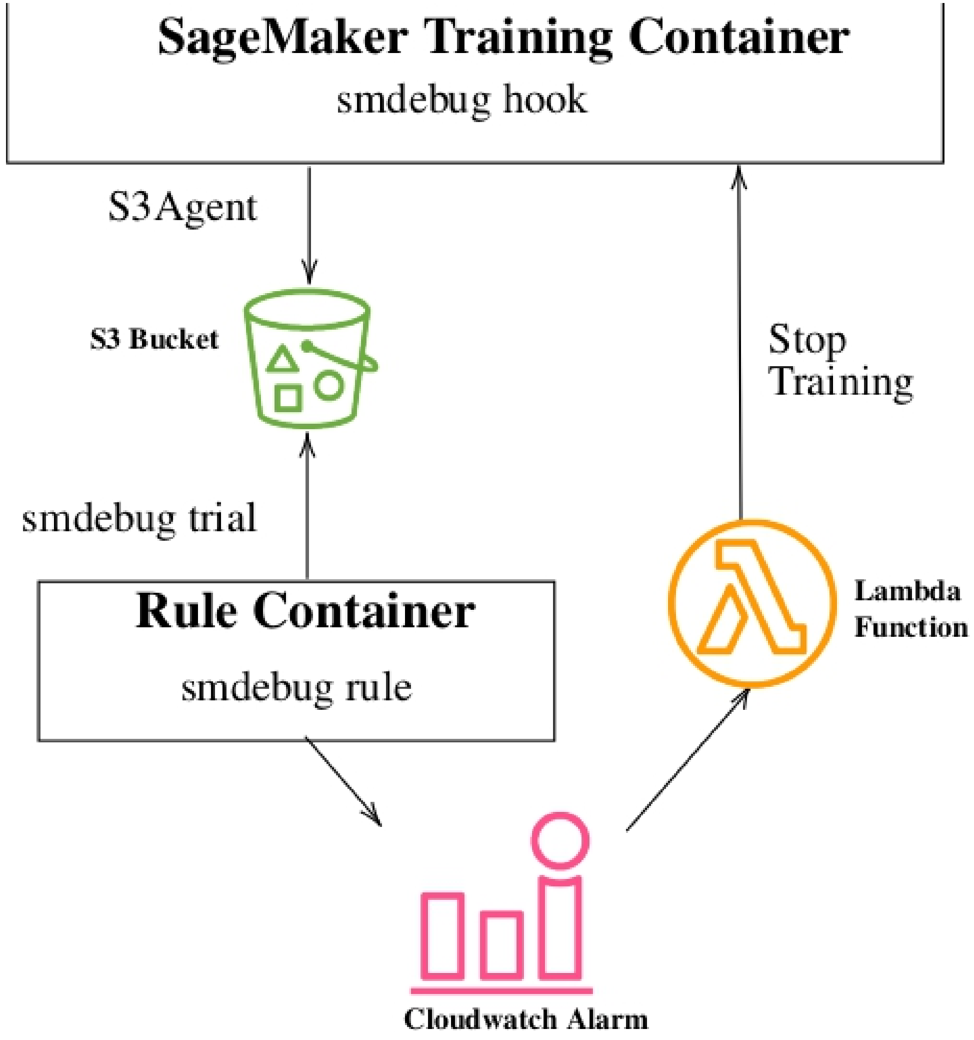


Figure 2. Debugger workflow

Technical challenges

- 1. Scale rule analysis by offloading into seperate containers
- 2. Reduce overhead when recording and fetching tensors
- 3. Separate compute and storage and minize impact on training

Case study

Check rule status

```
[ ]: pytorch_estimator.latest_training_job.rule_job_summary()
```

Read Debugger data

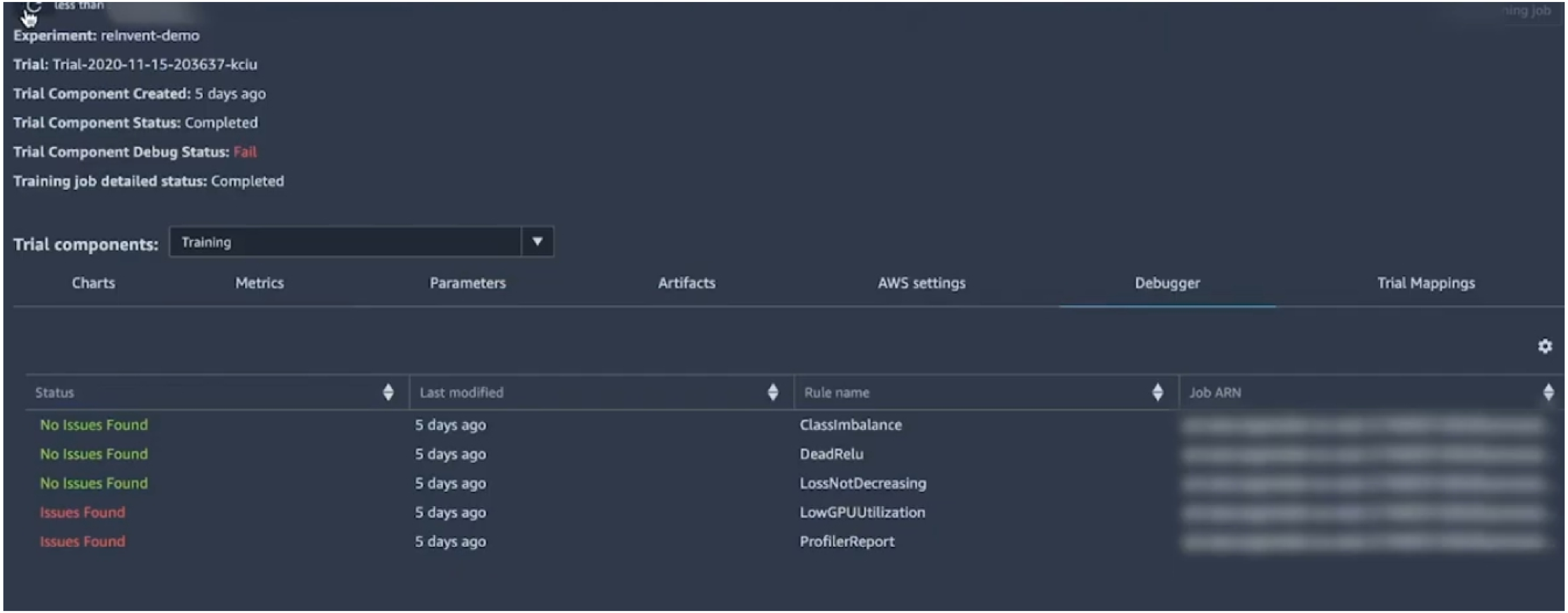
```
[ ]: ! pip install smdebug
```

```
[ ]: from smdebug.trials import create_trial

path = pytorch_estimator.latest_job_debugger_artifacts_path()
print('Tensors are stored in: {}'.format(path))

trial = create_trial(path)
```

Case study


Experiment: reInvent-demo
Trial: Trial-2020-11-15-203637-kciu
Trial Component Created: 5 days ago
Trial Component Status: Completed
Trial Component Debug Status: Fail
Training job detailed status: Completed
Trial components:
Training
Charts
Metrics
Parameters
Artifacts
AWS settings
Debugger
Trial Mappings
Status
Last modified
Rule name
Job ARN
No Issues Found
5 days ago
ClassImbalance
No Issues Found
5 days ago
DeadRelu
No Issues Found
5 days ago
LossNotDecreasing
Issues Found
5 days ago
LowGPUUtilization
Issues Found
5 days ago
ProfilerReport

Case study

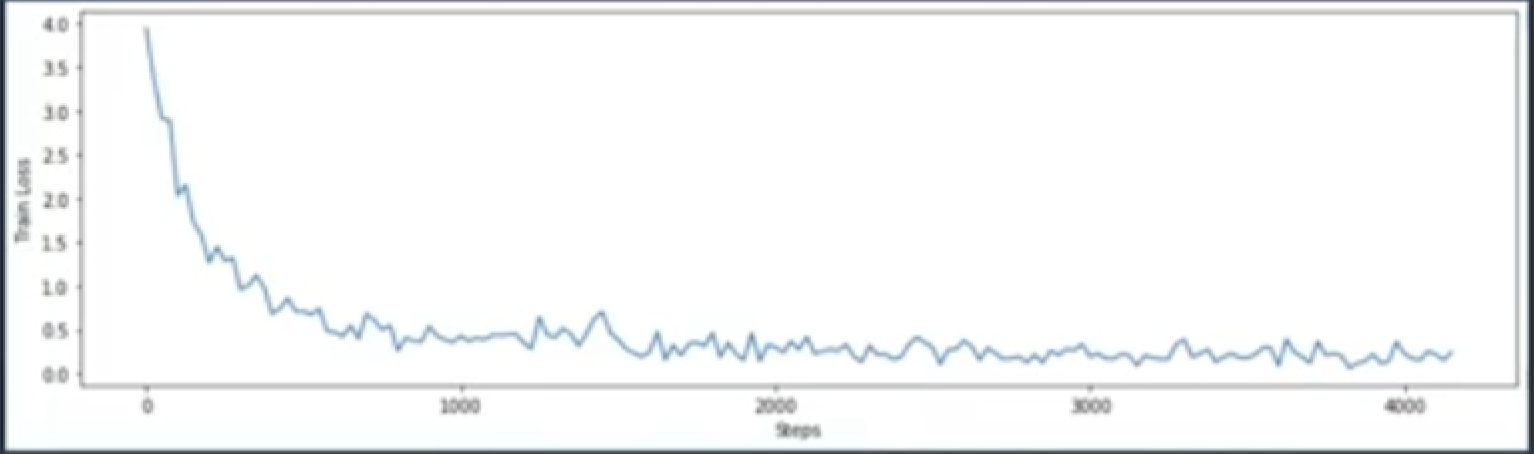
Investigate why 'Loss not decreasing' rule triggered

We can now easily visualize the loss values as training is still in progress.

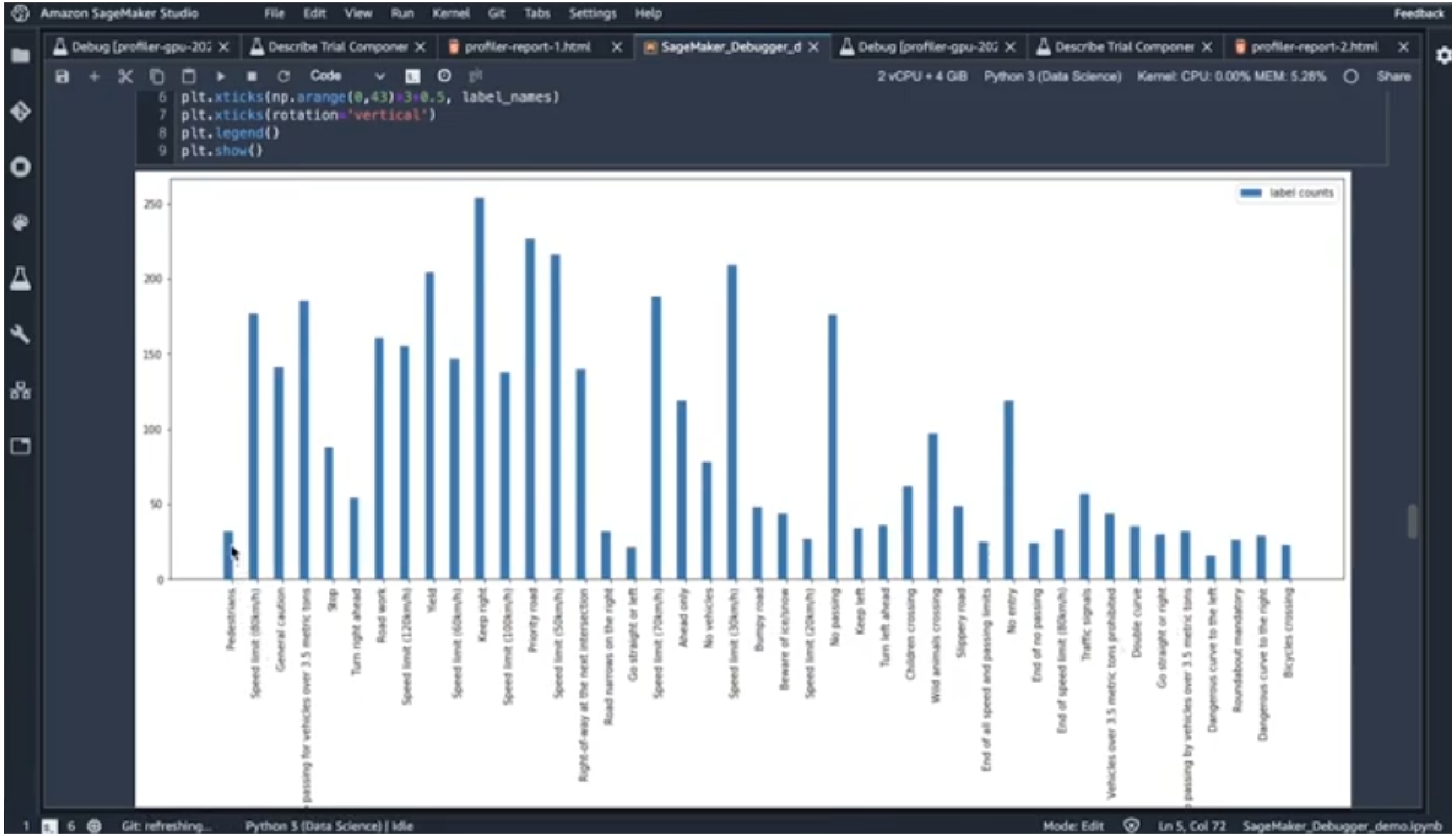
[186]:

```
import matplotlib.pyplot as plt
from smdebug import modes

plt.ylabel('Train Loss')
plt.xlabel('Steps')
plt.plot(trial.steps(mode=modes.TRAIN),
         list(trial.tensor('CrossEntropyLoss_output_0').values(mode=modes.TRAIN).values()))
plt.show()
```



Case study


Amazon SageMaker Studio
File Edit View Run Kernel Git Tabs Settings Help
Feedback
SageMaker_Debugger_d
profiler-report-1.html
profiler-report-2.html
Code
2 vCPU + 4 GiB
Python 3 (Data Science)
Share
plt.xticks(rotation='vertical')
plt.legend()
plt.show()
label counts
Pedestrians
General caution
Stop
Road work
Yield
Keep right
Priority road
No vehicles
Bumpy road
No passing
Keep left
Children crossing
Slippery road
No entry
Traffic signals
Double curve
Roundabout mandatory
Bicycles crossing
Python 3 (Data Science) | Idle
Mode: Edit
SageMaker_Debugger_demo.ipynb

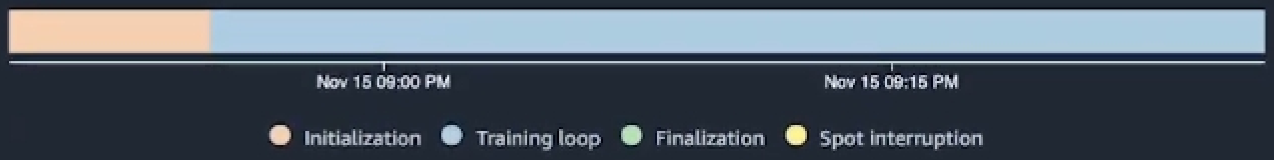
Case study


for prediction, label, image in zip(predictions, labels, images):
if prediction != label:
plot(image)
Predicted: 'Turn left ahead' Groundtruth: 'Turn right ahead'
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Predicted: 'Keep right' Groundtruth: 'Keep left'

Case study

Training loop duration	1866.820 seconds
Initialization	353.937 seconds
Finalization	0.043 seconds
Initialization (%)	15.94%
Training loop (%)	84.06%
Finalization (%)	0.00%

Training progress over time



Resource utilization summary

System usage statistics

Node	Metric	Unit	Max	p99	p95	p50	Min
algo-1	Network		69642384.61	0	0	0	0
algo-1	GPU		91	76.92	66	50	0
algo-1	CPU		100	54.88	41.38	21.94	0
algo-1	CPU memory		7.47	6.74	6.3	6.15	1.63
algo-1	GPU memory		30	25	21	16	0
algo-1	I/O		63.98	32.04	8.41	0	0

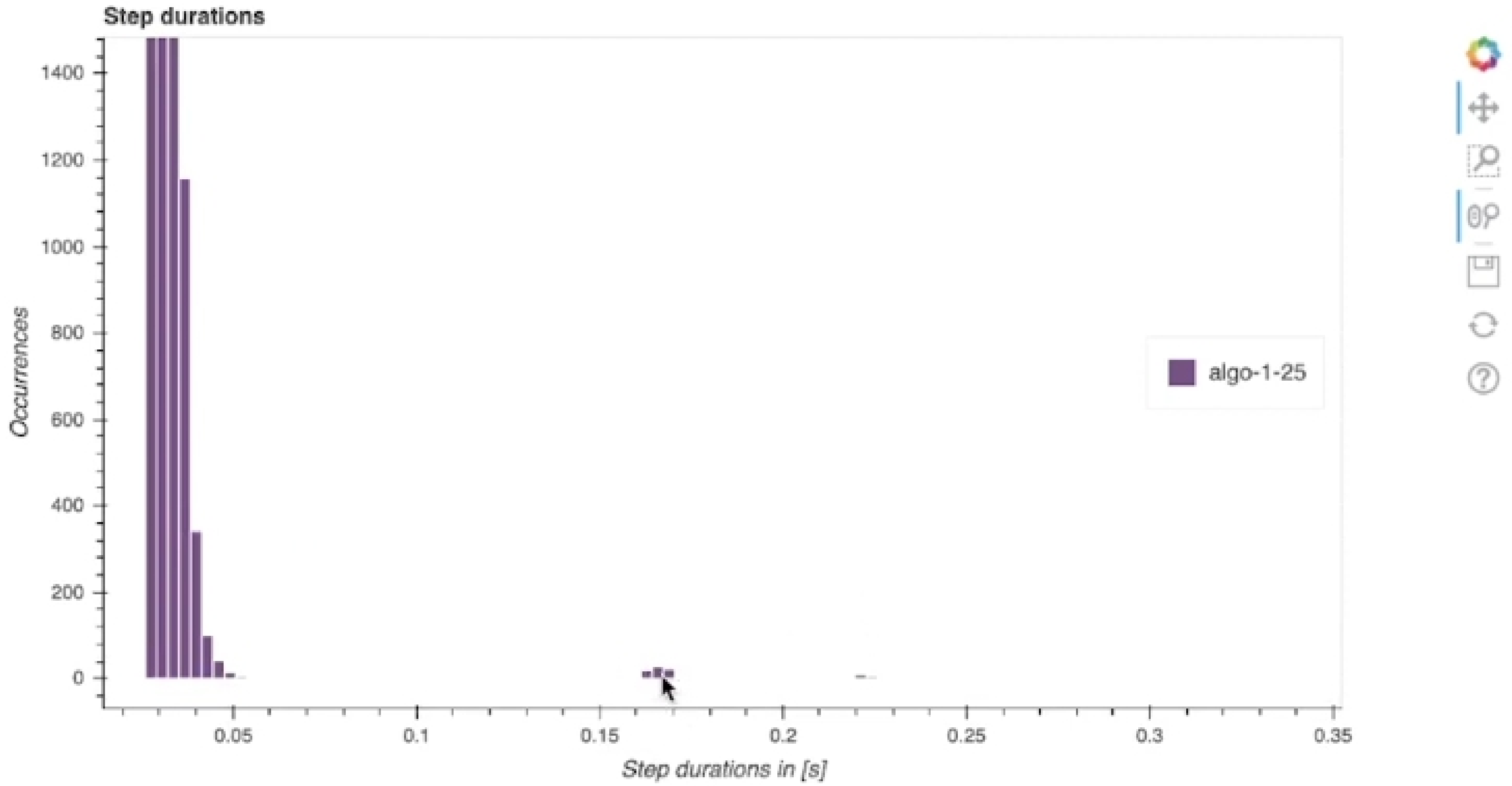
Case study

Resource intensive operations

Top operations on GPU

Percentage (%)	Cumulative time	GPU operator
22.81	209926	CudnnConvolutionBackward
22.74	209312	cudnn_convolution_backward
10.43	96020	conv2d
10.34	95166	convolution
10.24	94253	_convolution
9.97	91768	cudnn_convolution
3.82	35201	to
2.51	23121	CudnnBatchNormBackward
2.42	22239	batch_norm
2.38	21903	cudnn_batch_norm_backward
2.34	21542	_batch_norm_impl_index

Case study

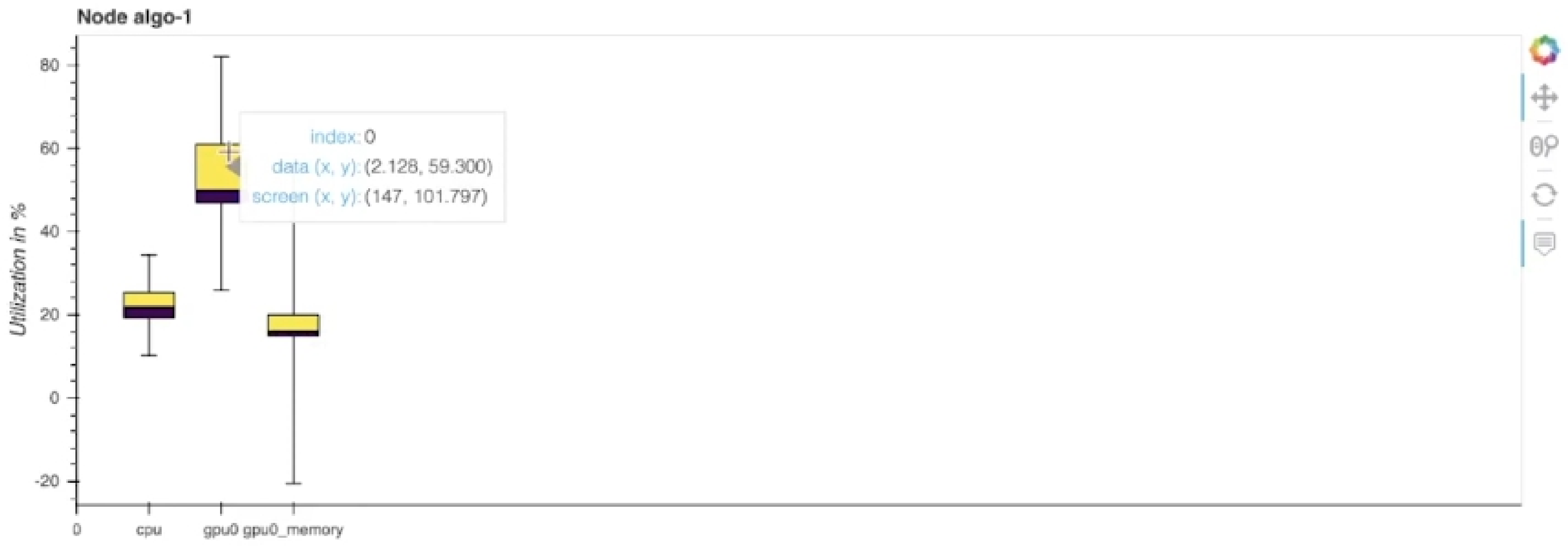

Step durations
1400
1200
1000
800
600
400
200
0
Occurrences
0.05
0.1
0.15
0.2
0.25
0.3
0.35
Step durations in [s]
algo-1-25

Case study

Batch size

The BatchSize rule helps to detect if GPU is underutilized because of the batch size being too small. To detect this the rule analyzes the GPU memory footprint, CPU and GPU utilization. The rule checked if the 95th percentile of CPU utilization is below cpu_threshold_p95 of 70%, the 95th percentile of GPU utilization is below gpu_threshold_p95 of 70% and the 95th percentile of memory footprint below gpu_memory_threshold_p95 of 70%. In your training job this happened 31 times. The rule skipped the first 4000 datapoints. The rule computed the percentiles over window size of 1000 continuous datapoints. The rule analysed 22208 datapoints and triggered 31 times.

Your training job is underutilizing the instance. You may want to consider either switch to a smaller instance type or to increase the batch size. The last time the BatchSize rule triggered in your training job was on 11/16/2020 at 04:48:00. The following boxplots are a snapshot from the timestamps. They the total CPU utilization, the GPU utilization, and the GPU memory usage per GPU (without outliers).



Case study

Rules summary

The following table shows a profiling summary of the Debugger built-in rules. The table is sorted by the rules that triggered the most frequently. During your training job, the BatchSize rule was the most frequently triggered. It processed 22208 datapoints and was triggered 31 times.

	Description	Recommendation	Number of times rule triggered	Number of datapoints	Rule parameters
BatchSize	Checks if GPUs are underutilized because the batch size is too small. To detect this problem, the rule analyzes the average GPU memory footprint, the CPU and the GPU utilization.	The batch size is too small, and GPUs are underutilized. Consider running on a smaller instance type or increasing the batch size.	31	22208	cpu_threshold_p95:70 gpu_threshold_p95:70 gpu_memory_threshold_p95:70 patience:4000 window:1000
LowGPUUtilization	Checks if the GPU utilization is low or fluctuating. This can happen due to bottlenecks, blocking calls for synchronizations, or a small batch size.	Check if there are bottlenecks, minimize blocking calls, change distributed training strategy, or increase the batch size.	31	22209	threshold_p95:70 threshold_p5:10 window:500 patience:4000
StepOutlier	Detects outliers in step duration. The step duration for forward and backward pass should be roughly the same throughout the training. If there are significant outliers, it may indicate a system stall or bottleneck issues.	Check if there are any bottlenecks (CPU, I/O) correlated to the step outliers.	25	12519	threshold:3 mode:None n_outliers:10 stddev:3
Dataloader	Checks how many data loaders are running in parallel and whether the total number is equal the number of available CPU cores. The rule triggers if number is much smaller or larger than the number of available cores. If too small, it might lead to low GPU utilization. If too large, it might impact other compute intensive operations on CPU.	Change the number of data loader processes.	1	84	min_threshold:40 max_threshold:200
IOBottleneck	Checks if the data I/O wait time is high and the GPU utilization is low. It might indicate IO bottlenecks where GPU is waiting for data to arrive from storage. The rule evaluates the I/O and GPU utilization rates and triggers the issue if the time spent on	Pre-fetch data or choose different file formats, such as binary formats that improve I/O	0	22249	threshold:50 io_threshold:50 gpu_threshold:10

Performance of recording tensors