You make **possible**

My Machine is an Honor Student at Cisco!

Quality of machine learning experience

Mike Mikhail, Architect
mamikhai@cisco.com
@MikeMikhail

DEVLIT-4002

cisco Live!
Barcelona | January 27-31, 2020

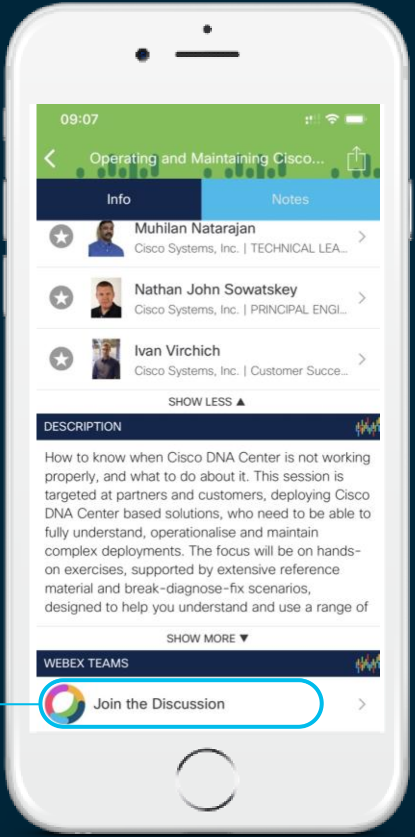
Cisco Webex Teams

Questions?

Use Cisco Webex Teams to chat
with the speaker after the session

How

1. Find this session in the Cisco Events Mobile App
2. Click “Join the Discussion”
3. Install Webex Teams or go directly to the team space
4. Enter messages/questions in the team space



09:07

Operating and Maintaining Cisco...

Info | Notes

Muhilan Natarajan
Cisco Systems, Inc. | TECHNICAL LEA...

Nathan John Sowatskey
Cisco Systems, Inc. | PRINCIPAL ENGI...

Ivan Virchich
Cisco Systems, Inc. | Customer Succe...

SHOW LESS ▲

DESCRIPTION

How to know when Cisco DNA Center is not working properly, and what to do about it. This session is targeted at partners and customers, deploying Cisco DNA Center based solutions, who need to be able to fully understand, operationalise and maintain complex deployments. The focus will be on hands-on exercises, supported by extensive reference material and break-diagnose-fix scenarios, designed to help you understand and use a range of

SHOW MORE ▼

WEBEX TEAMS

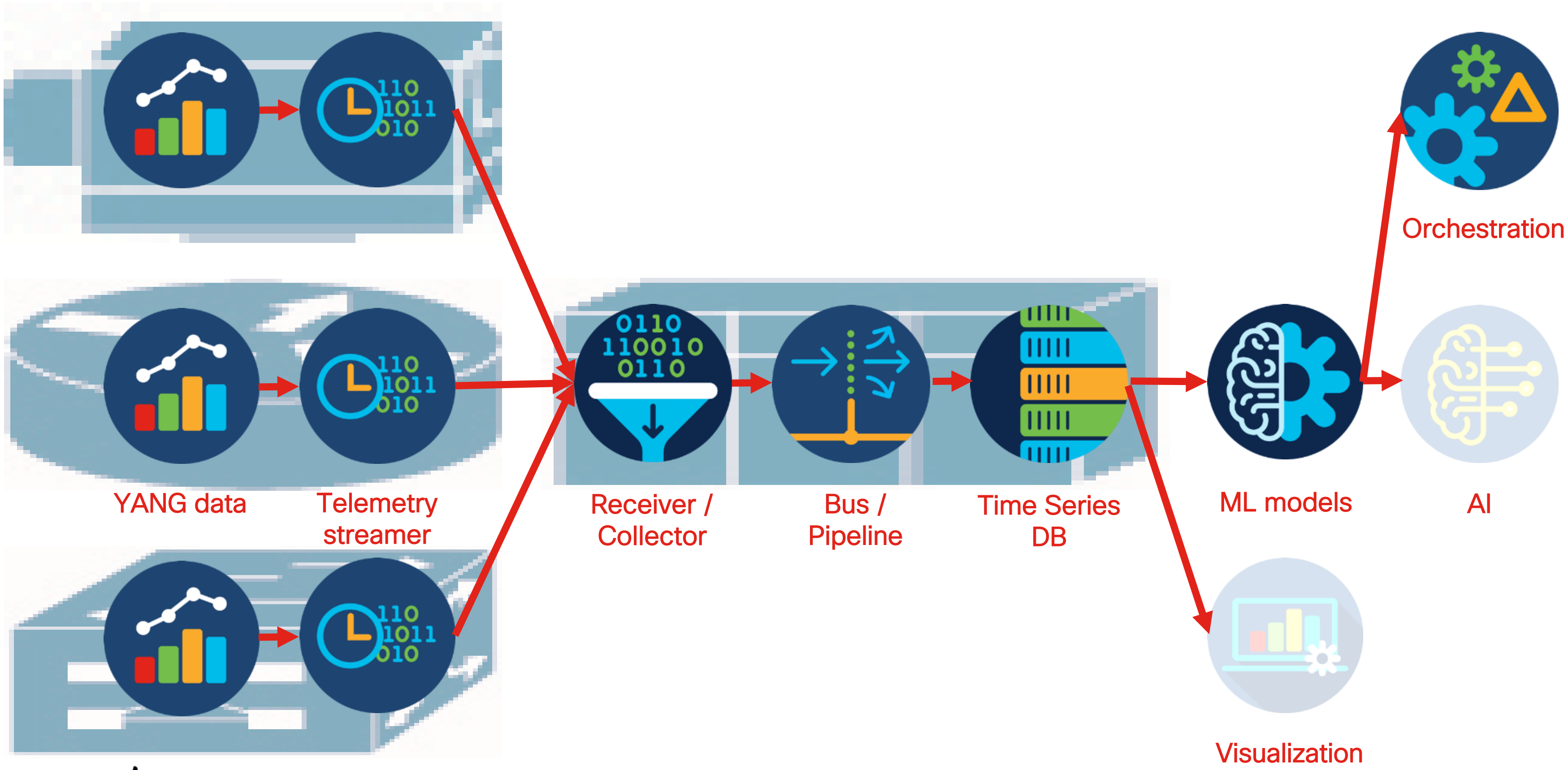
Join the Discussion

Agenda

- Data analysis and quality
- The process
- Data conditioning
- Visualization
- Quality assurance

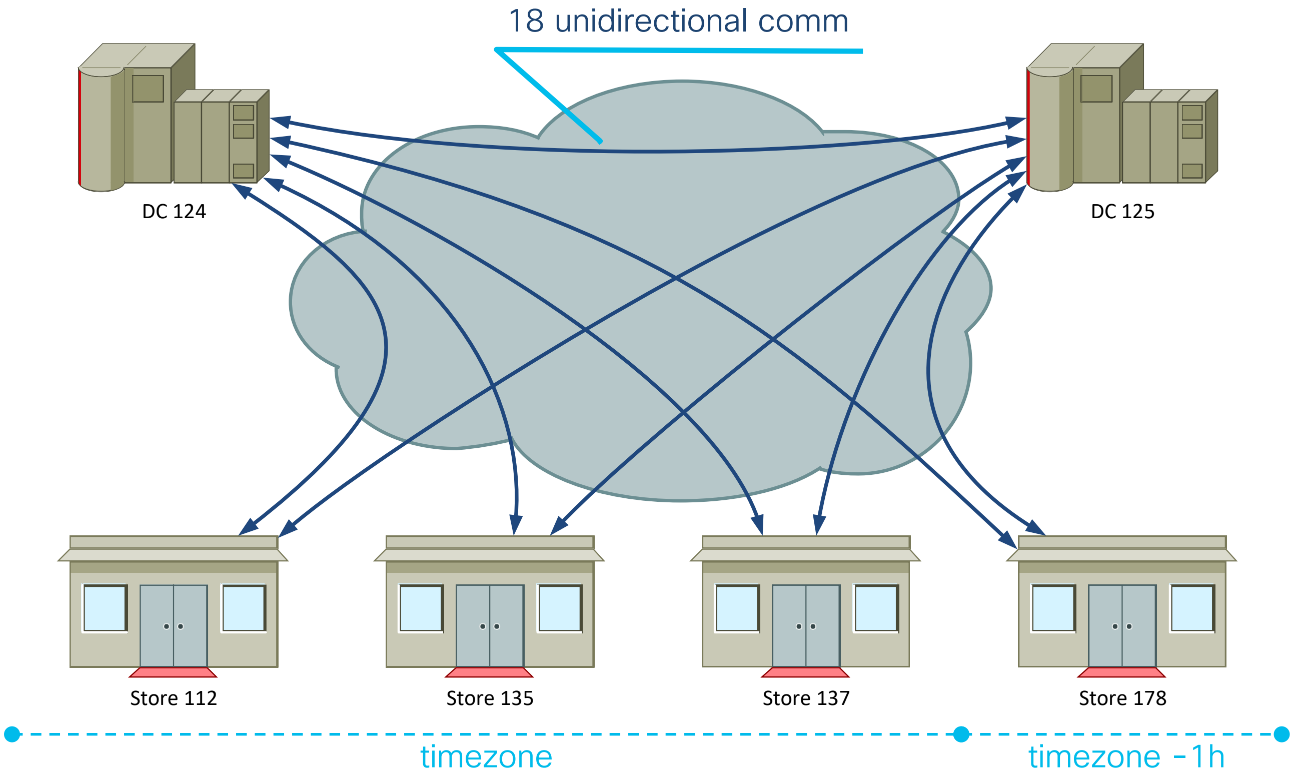
Initiate ML model, initial training...

Production Line

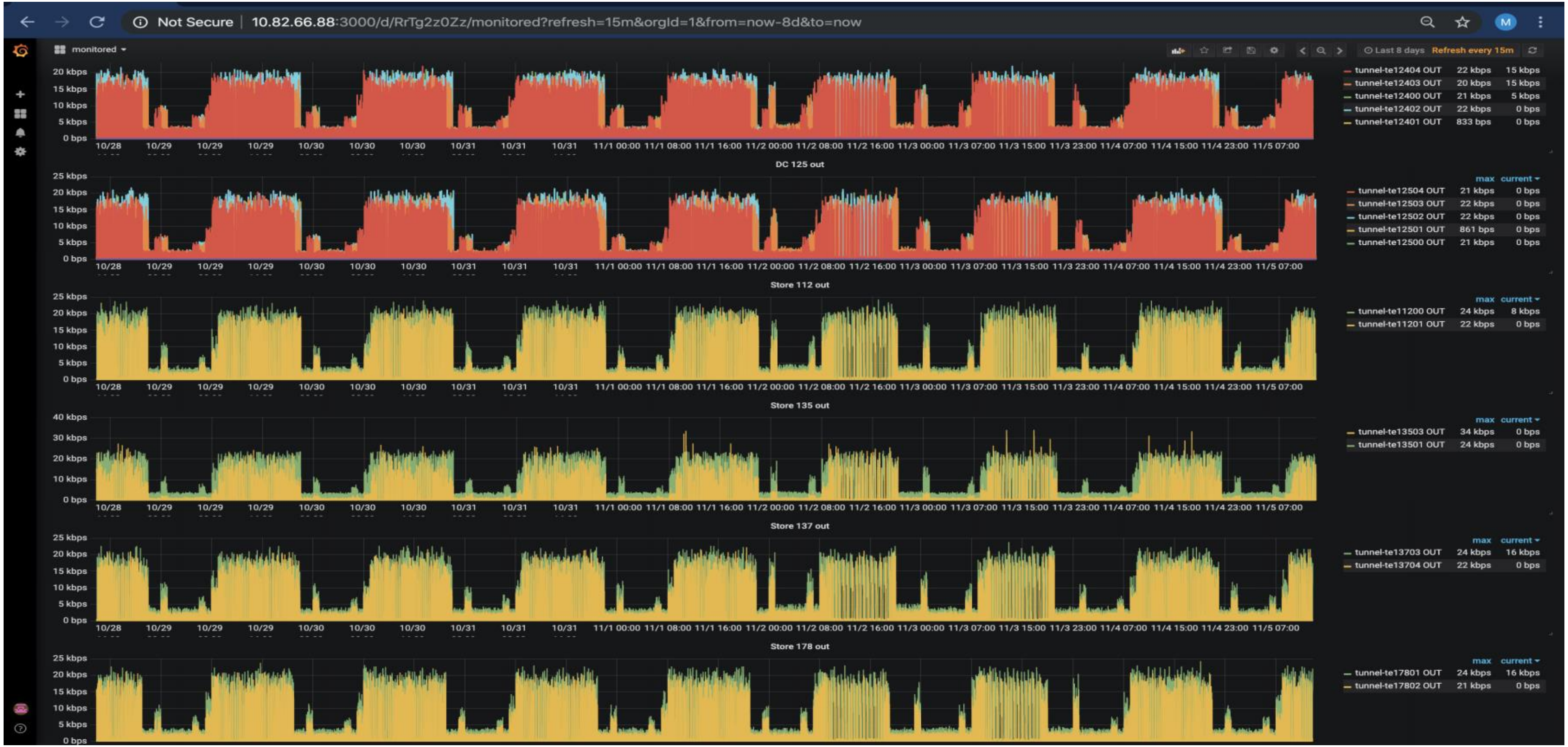


Data Source

Business app data



Visualized – 8days



tunnel-te12404 OUT	22 kbps	15 kbps
tunnel-te12403 OUT	20 kbps	15 kbps
tunnel-te12400 OUT	21 kbps	5 kbps
tunnel-te12402 OUT	22 kbps	0 bps
tunnel-te12401 OUT	833 bps	0 bps

DC 125 out

	max	current
tunnel-te12504 OUT	21 kbps	0 bps
tunnel-te12503 OUT	22 kbps	0 bps
tunnel-te12502 OUT	22 kbps	0 bps
tunnel-te12501 OUT	861 bps	0 bps
tunnel-te12500 OUT	21 kbps	0 bps

Store 112 out

	max	current
tunnel-te11200 OUT	24 kbps	8 kbps
tunnel-te11201 OUT	22 kbps	0 bps

Store 135 out

	max	current
tunnel-te13503 OUT	34 kbps	0 bps
tunnel-te13501 OUT	24 kbps	0 bps

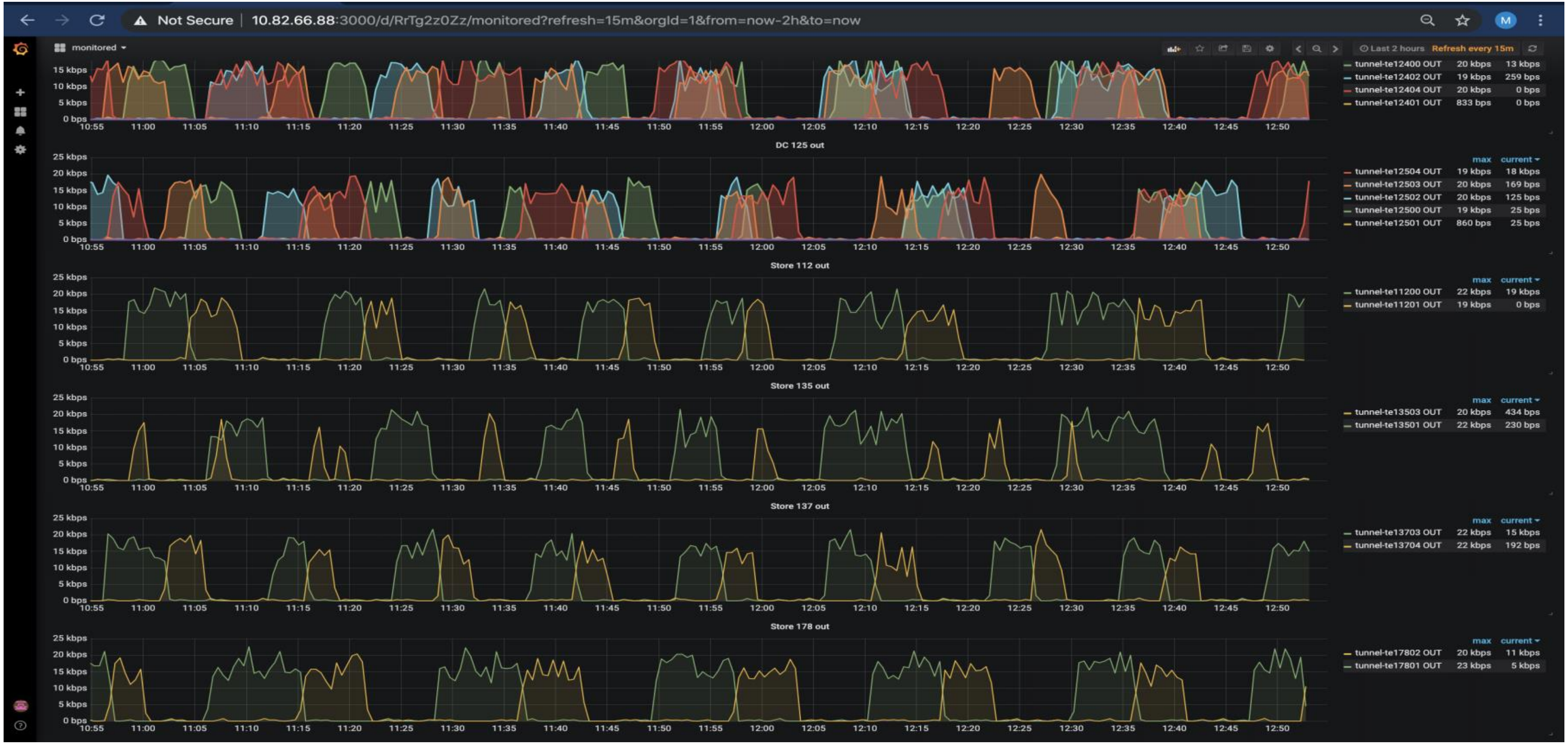
Store 137 out

	max	current
tunnel-te13703 OUT	24 kbps	16 kbps
tunnel-te13704 OUT	22 kbps	0 bps

Store 178 out

	max	current
tunnel-te17801 OUT	24 kbps	16 kbps
tunnel-te17802 OUT	21 kbps	0 bps

Visualized – 2 business hours



Not Secure | 10.82.66.88:3000/d/RrTg2z0Zz/monitored?refresh=15m&orgId=1&from=now-2h&to=now

monitored

Last 2 hours Refresh every 15m

tunnel-te12400 OUT	20 kbps	13 kbps
tunnel-te12402 OUT	19 kbps	259 bps
tunnel-te12404 OUT	20 kbps	0 bps
tunnel-te12401 OUT	833 bps	0 bps

DC 125 out

	max	current
tunnel-te12504 OUT	19 kbps	18 kbps
tunnel-te12503 OUT	20 kbps	169 bps
tunnel-te12502 OUT	20 kbps	125 bps
tunnel-te12500 OUT	19 kbps	25 bps
tunnel-te12501 OUT	860 bps	25 bps

Store 112 out

	max	current
tunnel-te11200 OUT	22 kbps	19 kbps
tunnel-te11201 OUT	19 kbps	0 bps

Store 135 out

	max	current
tunnel-te13503 OUT	20 kbps	434 bps
tunnel-te13501 OUT	22 kbps	230 bps

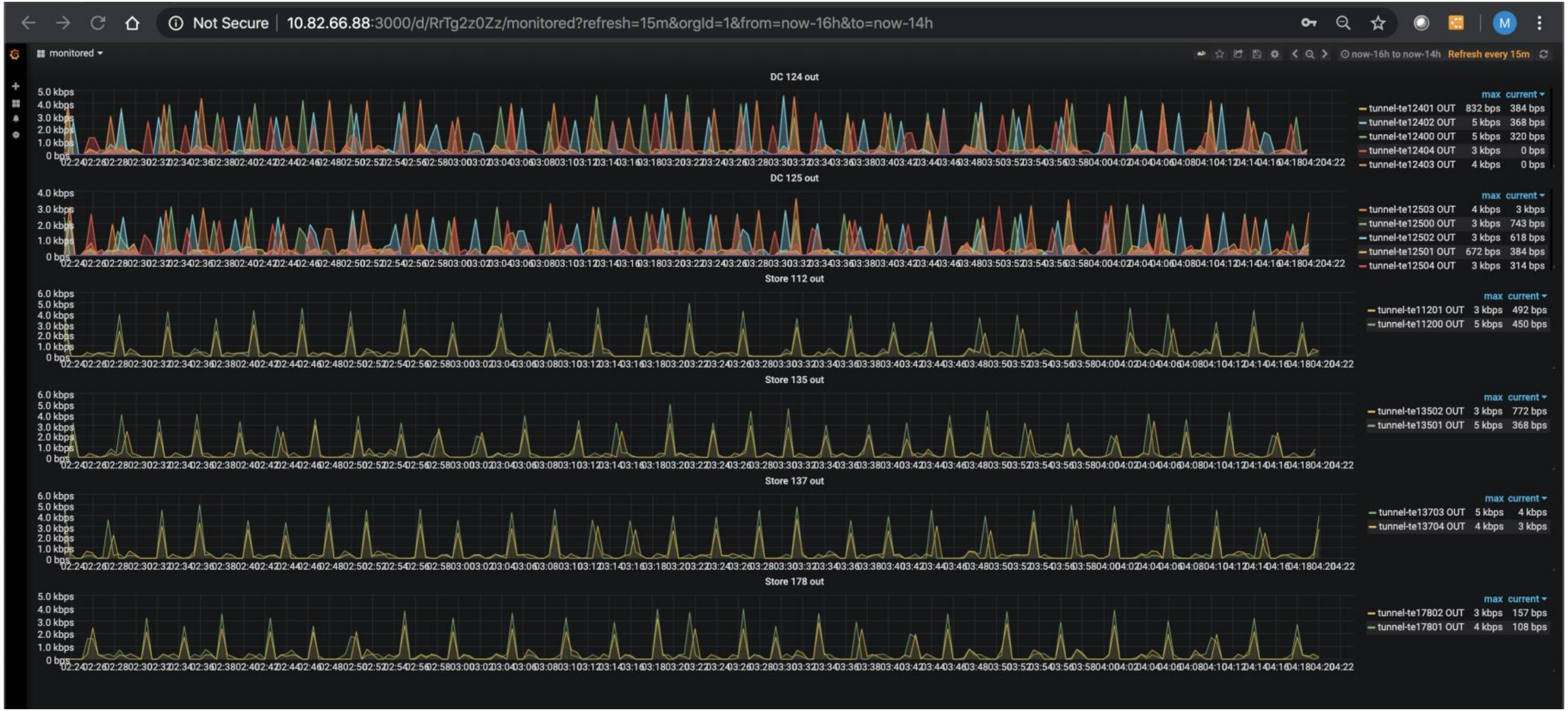
Store 137 out

	max	current
tunnel-te13703 OUT	22 kbps	15 kbps
tunnel-te13704 OUT	22 kbps	192 bps

Store 178 out

	max	current
tunnel-te17802 OUT	20 kbps	11 kbps
tunnel-te17801 OUT	23 kbps	5 kbps

Visualized – 2 off hours



DC 124 out

	max	current
tunnel-te12401 OUT	832 bps	384 bps
tunnel-te12402 OUT	5 kbps	368 bps
tunnel-te12400 OUT	5 kbps	320 bps
tunnel-te12404 OUT	3 kbps	0 bps
tunnel-te12403 OUT	4 kbps	0 bps

DC 125 out

	max	current
tunnel-te12503 OUT	4 kbps	3 kbps
tunnel-te12500 OUT	3 kbps	743 bps
tunnel-te12502 OUT	3 kbps	618 bps
tunnel-te12501 OUT	672 bps	384 bps
tunnel-te12504 OUT	3 kbps	314 bps

Store 112 out

	max	current
tunnel-te11201 OUT	3 kbps	492 bps
tunnel-te11200 OUT	5 kbps	450 bps

Store 135 out

	max	current
tunnel-te13502 OUT	3 kbps	772 bps
tunnel-te13501 OUT	5 kbps	368 bps

Store 137 out

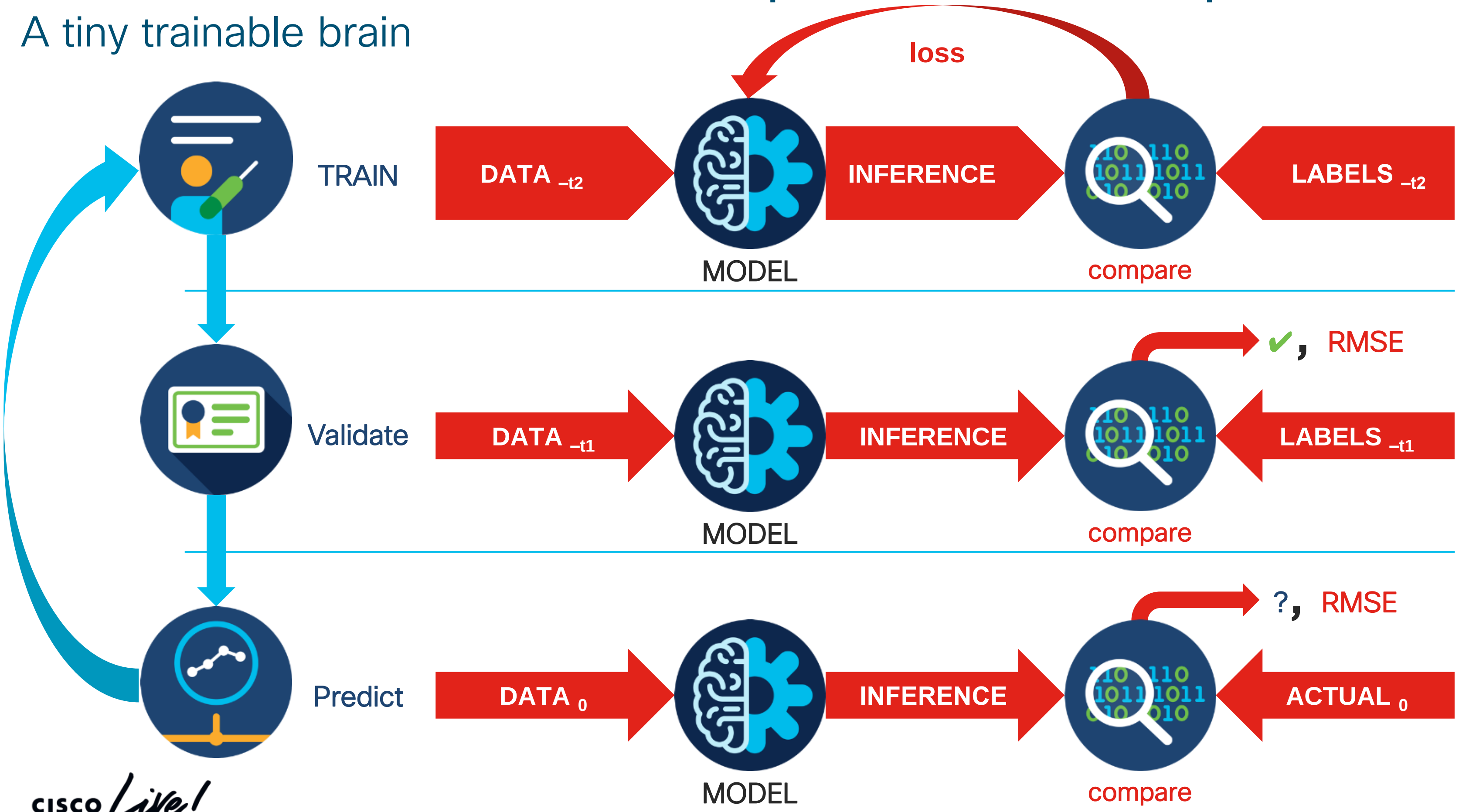
	max	current
tunnel-te13703 OUT	5 kbps	4 kbps
tunnel-te13704 OUT	4 kbps	3 kbps

Store 178 out

	max	current
tunnel-te17802 OUT	3 kbps	157 bps
tunnel-te17801 OUT	4 kbps	108 bps

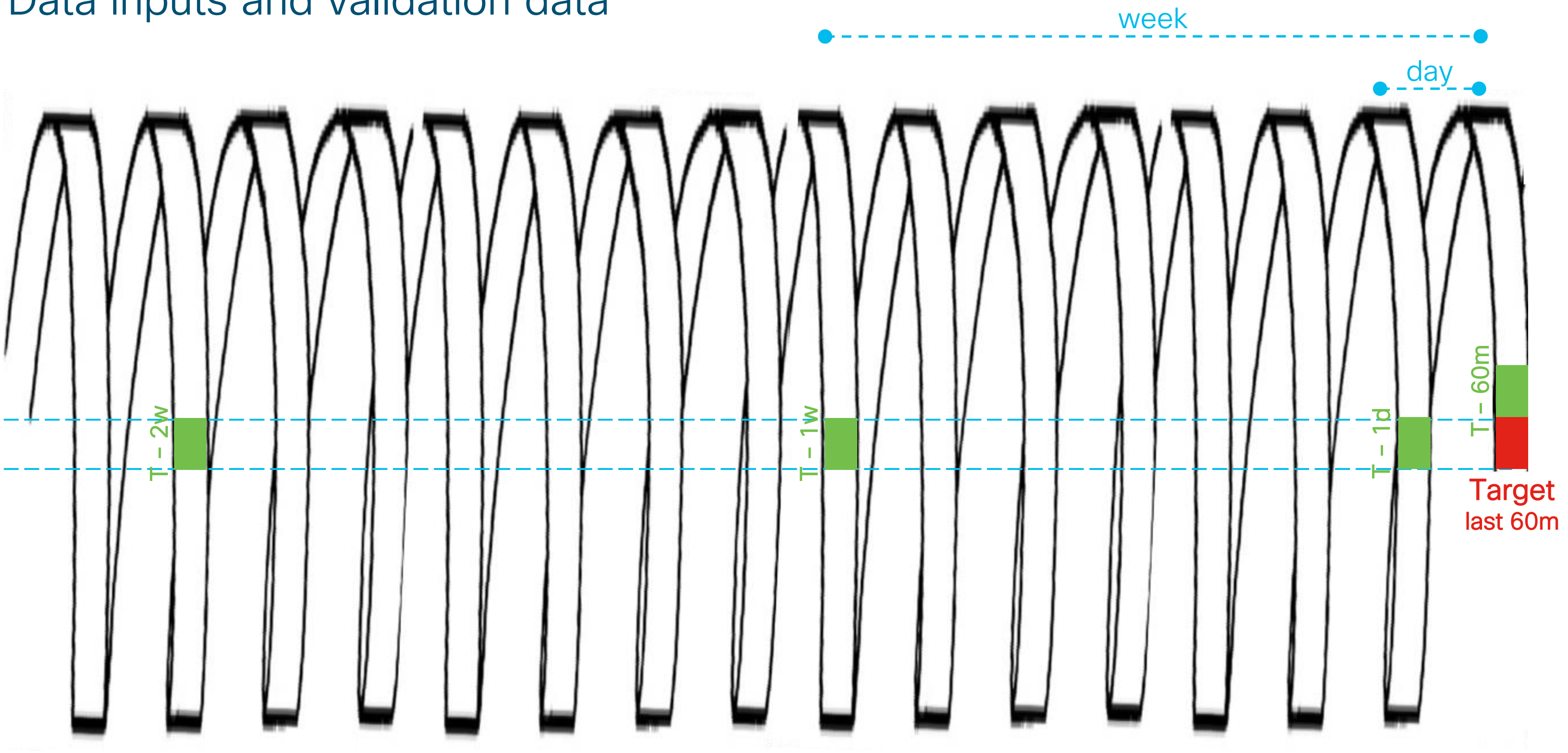
What’s an ML Model? Supervised example

A tiny trainable brain



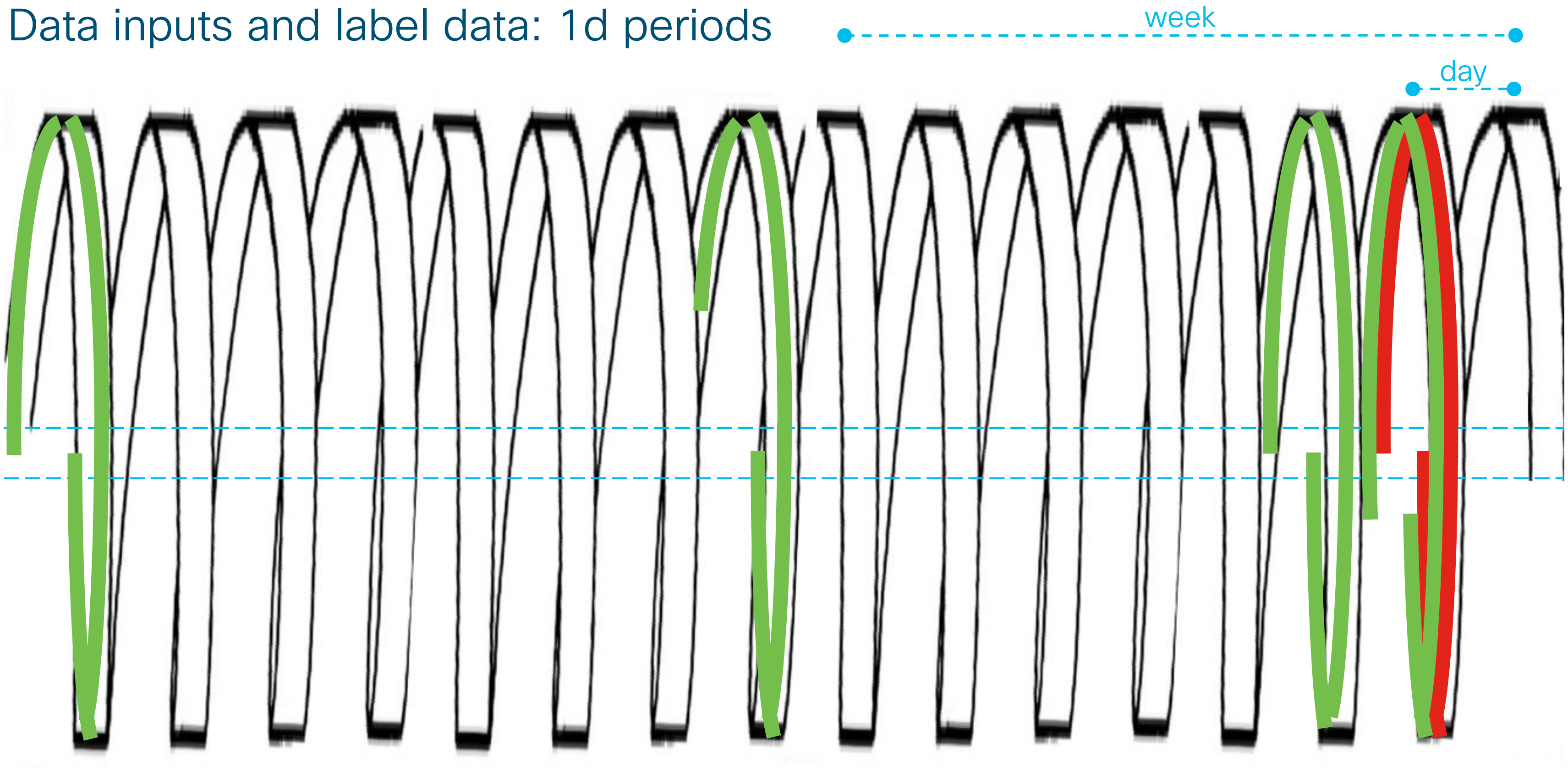
Basis for Inference

Data inputs and validation data



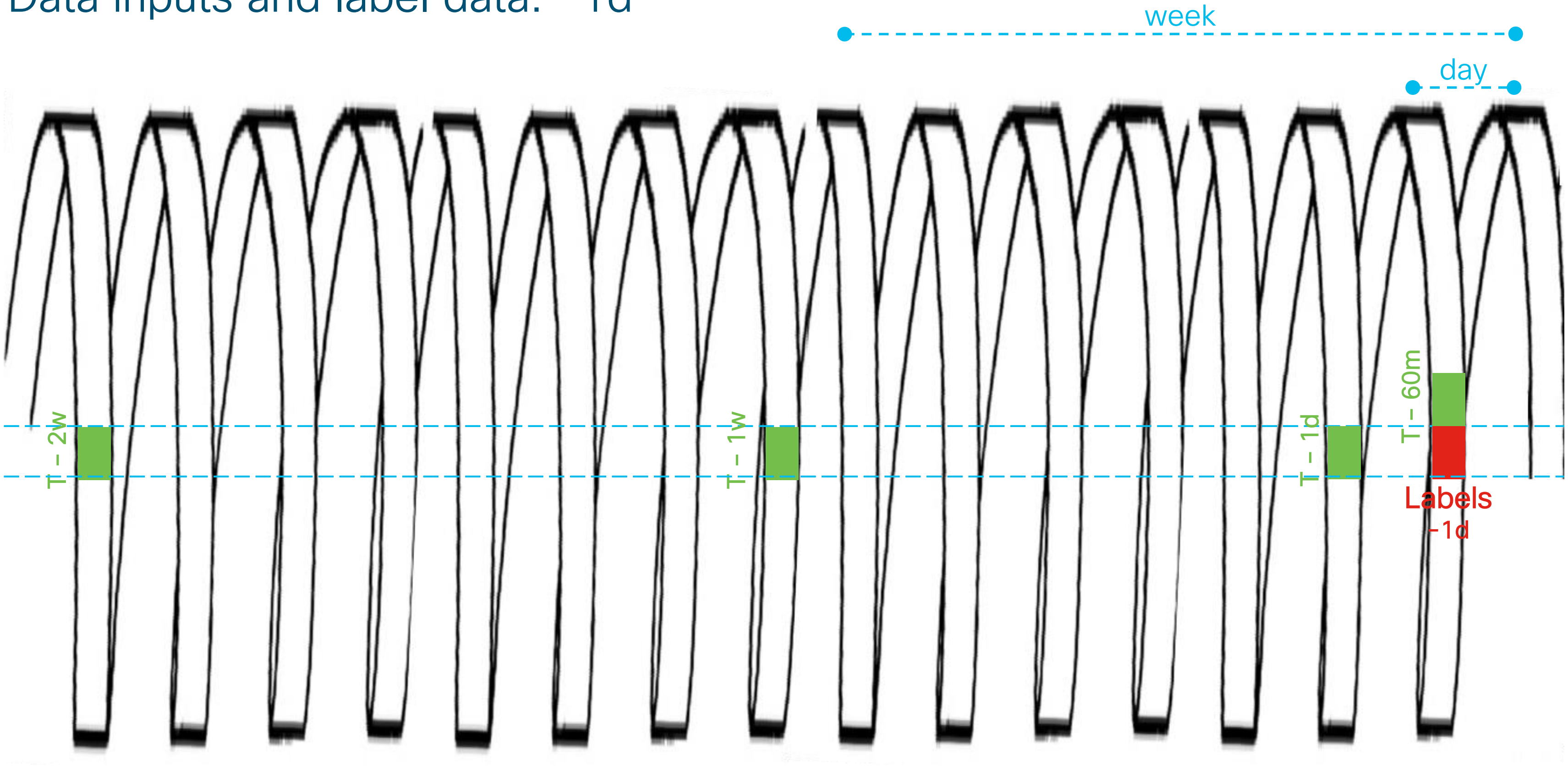
Training: Larger Datasets

Data inputs and label data: 1d periods



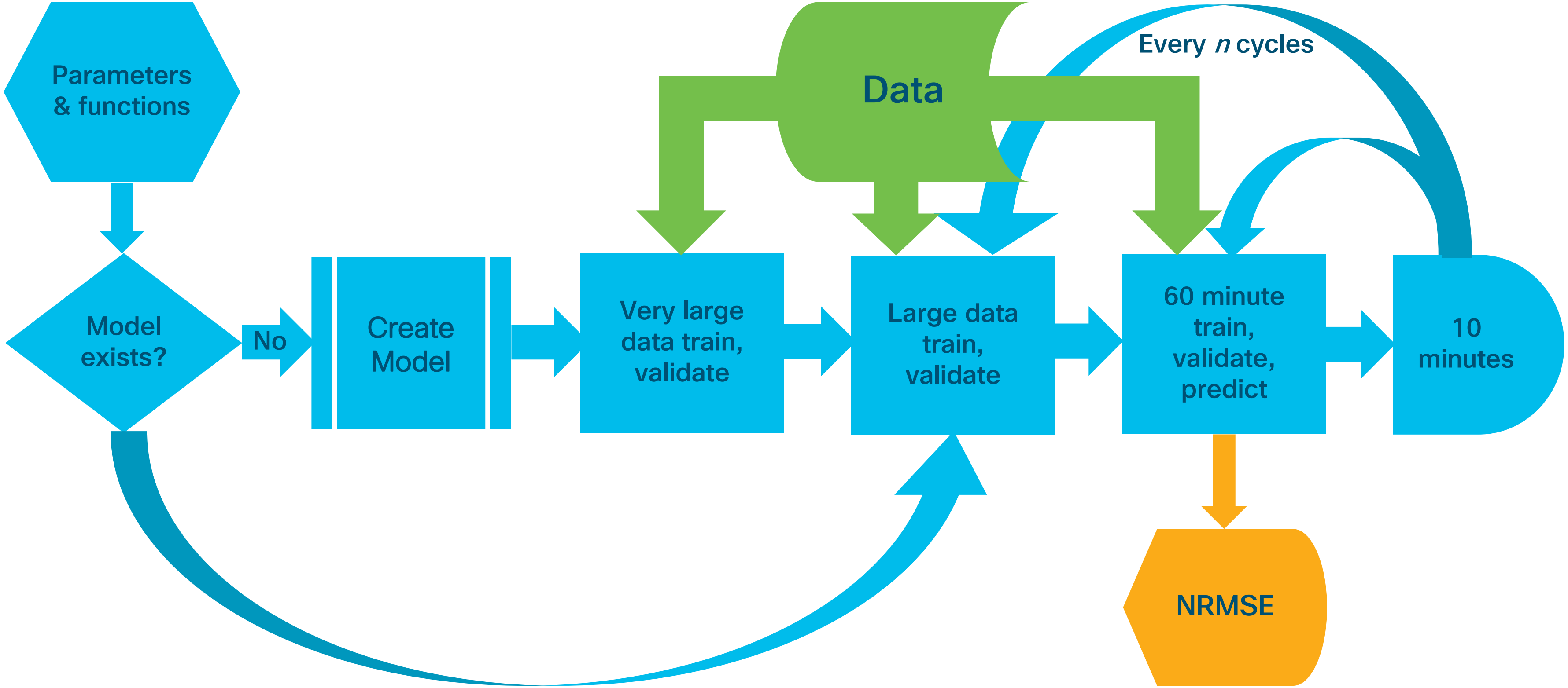
Training

Data inputs and label data: -1d

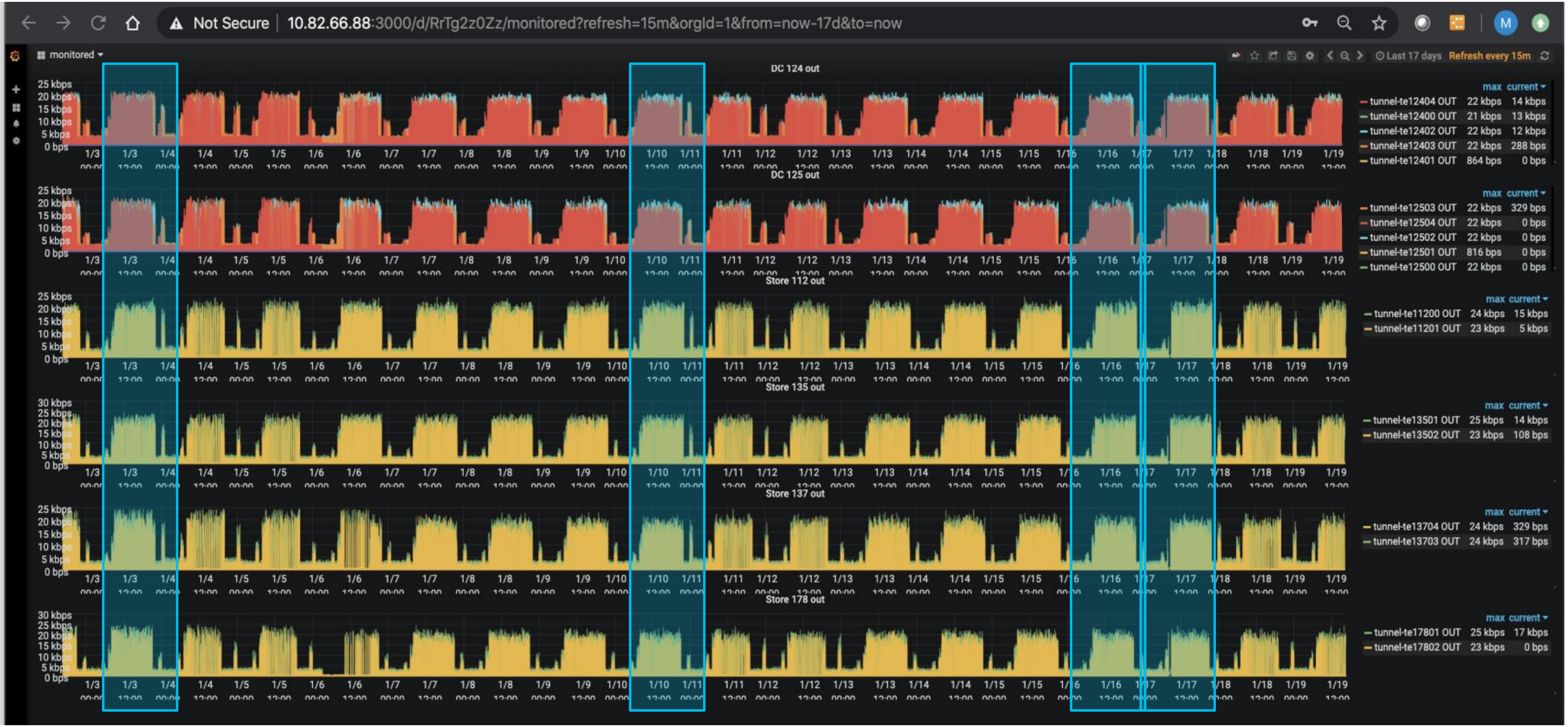


Training: Initial, and Periodic; Predict

“Monitor” flowchart


Parameters & functions
Model exists?
No
Create Model
Very large data train, validate
Data
Large data train, validate
Every *n* cycles
60 minute train, validate, predict
10 minutes
NRMSE

The Data in 4x 24-hour DataFrame, [4x18]x2880



Not Secure | 10.82.66.88:3000/d/RrTg2z0Zz/monitored?refresh=15m&orgId=1&from=now-17d&to=now

monitored ▾

Last 17 days Refresh every 15m

	max	current ▾
tunnel-te12404 OUT	22 kbps	14 kbps
tunnel-te12400 OUT	21 kbps	13 kbps
tunnel-te12402 OUT	22 kbps	12 kbps
tunnel-te12403 OUT	22 kbps	288 bps
tunnel-te12401 OUT	864 bps	0 bps

	max	current ▾
tunnel-te12503 OUT	22 kbps	329 bps
tunnel-te12504 OUT	22 kbps	0 bps
tunnel-te12502 OUT	22 kbps	0 bps
tunnel-te12501 OUT	816 bps	0 bps
tunnel-te12500 OUT	22 kbps	0 bps

	max	current ▾
tunnel-te11200 OUT	24 kbps	15 kbps
tunnel-te11201 OUT	23 kbps	5 kbps

	max	current ▾
tunnel-te13501 OUT	25 kbps	14 kbps
tunnel-te13502 OUT	23 kbps	108 bps

	max	current ▾
tunnel-te13704 OUT	24 kbps	329 bps
tunnel-te13703 OUT	24 kbps	317 bps

	max	current ▾
tunnel-te17801 OUT	25 kbps	17 kbps
tunnel-te17802 OUT	23 kbps	0 bps

4 24-hour_period x 18 path x 2880 30-second_field = 72 column x 2880 row DataFrame

DataFrame

```
def read_train_long(record_count, label_prefix, verbose=True):
.
        print('\ntraining long data')
        print(train.describe())
```

```
training long data
       d_te11200_previous  d_te11200_1d  d_te11200_1w  d_te11200_2w  \
count        2.880000e+03  2.880000e+03  2.880000e+03  2.880000e+03
mean         1.093179e+07  1.210950e+07  1.215980e+07  2.038959e+07
std          1.072837e+07  1.140074e+07  1.141777e+07  1.141292e+07
min          0.000000e+00  1.365000e+03  1.087200e+04  2.841000e+03
25%          1.663094e+06  1.665956e+06  1.689180e+06  1.086874e+07
50%          5.952412e+06  8.211694e+06  8.303540e+06  2.399129e+07
75%          1.959832e+07  2.178496e+07  2.212936e+07  3.107957e+07
max          3.269652e+07  3.305397e+07  3.308679e+07  3.262037e+07


       d_te11201_previous  d_te11201_1d  d_te11201_1w  d_te11201_2w  \
count        2.880000e+03  2.880000e+03  2.880000e+03  2.880000e+03
mean         7.527995e+06  8.306467e+06  8.337991e+06  1.384803e+07
std          7.289889e+06  7.747967e+06  7.757351e+06  7.804774e+06
min          0.000000e+00  1.380000e+03  8.182000e+03  1.001000e+03
```

Column “label”

df is 72 x 2880

Math description of column

DataFrame: same format, smaller size

```
validation data
       d_te11200_previous  d_te11200_1d  d_te11200_1w  d_te11200_2w  \
count        1.200000e+02  1.200000e+02  1.200000e+02  1.200000e+02
mean         1.358485e+06  9.795289e+05  1.179847e+06  1.246457e+06
std          8.000545e+05  6.292864e+05  6.797141e+05  7.001280e+05
min          1.380000e+03  7.731200e+04  9.600000e+01  1.928000e+03
25%          4.905140e+05  3.478915e+05  6.696790e+05  8.413410e+05
50%          1.362629e+06  1.087073e+06  1.240100e+06  1.377682e+06
75%          2.190353e+06  1.559045e+06  1.827315e+06  1.777683e+06
max          2.840542e+06  2.067730e+06  2.336623e+06  2.262407e+06


       d_te11201_previous  d_te11201_1d  d_te11201_1w  d_te11201_2w  \
count        1.200000e+02  1.200000e+02  1.200000e+02  1.200000e+02
mean         7.821397e+05  1.026548e+06  7.012429e+05  7.430594e+05
std          5.078806e+05  4.558566e+05  4.654541e+05  5.078973e+05
min          0.000000e+00  0.000000e+00  0.000000e+00  1.350000e+03
25%          3.321160e+05  5.965380e+05  4.511570e+05  3.695710e+05
50%          9.229315e+05  9.878900e+05  8.389290e+05  7.320545e+05
75%          1.225480e+06  1.511388e+06  1.236286e+06  1.205447e+06
max          1.499793e+06  1.857222e+06  1.556056e+06  1.526029e+06
.
```

df is 72 x 120

Fetching, Formatting, Conditioning the Data

```
def read_data(field_key, measurement_name, condition1, condition2, condition3, limit,
label):
  query_db = str('SELECT "%s" FROM "%s" WHERE %s AND %s AND %s LIMIT %d ' % (field_key,
measurement_name, condition1, condition2, condition3, limit+1))
  data_db = client.query(query_db)
  print('\ndata_db:\n', data_db)
  data_df = pd.DataFrame(data_db[str(measurement_name)])
  print('\ndata_df:\n', data_df)
  print('\ndata_df description:\n', data_df.describe())
  data_df.columns = [label]
  data_df.reset_index(drop=True, inplace=True)
  data_df.fillna(method='ffill', inplace=True)
  data_df.fillna(method='bfill', inplace=True)
  data_df -= data_df.min()
  data_df.drop(data_df.index[0], inplace=True)
  print('\ndata_df:\n', data_df)
  print('\ndata_df description:\n', data_df.describe())
  sys.exit()
  # data_df = data_df.sub(data_df.shift(fill_value=0))
  # print('\n', query_db, '\n', data_df.describe())
  return data_df
```

Raw Data

```
data_db:
 defaultdict(<class 'list'>, {'Cisco-IOS-XR-infra-statsd-oper:infra-
statistics/interfaces/interface/latest/generic-
counters':                                            bytes-sent
2020-01-03 06:34:04.032000+00:00  2269840208
2020-01-03 06:34:34.031000+00:00  2269840208
2020-01-03 06:35:04.027000+00:00  2269840208
2020-01-03 06:35:34.031000+00:00  2269840208
2020-01-03 06:36:04.032000+00:00  2269840820
2020-01-03 06:36:34.041000+00:00  2269842350
2020-01-03 06:37:04.032000+00:00  2269842554
2020-01-03 06:37:34.036000+00:00  2269842554
2020-01-03 06:38:04.032000+00:00  2269842554
2020-01-03 06:38:34.033000+00:00  2269842554
2020-01-03 06:39:04.090000+00:00  2269842554
2020-01-03 06:39:34.035000+00:00  2269853583
2020-01-03 06:40:04.036000+00:00  2269854129
2020-01-03 06:40:34.033000+00:00  2269854129
2020-01-03 06:41:04.035000+00:00  2269854228
2020-01-03 06:41:34.033000+00:00  2269855713
2020-01-03 06:42:04.035000+00:00  2269856604
.
```

In Single Column DataFrame

```
data_df:
                                   bytes-sent
2020-01-03 06:34:04.032000+00:00  2269840208
2020-01-03 06:34:34.031000+00:00  2269840208
2020-01-03 06:35:04.027000+00:00  2269840208
2020-01-03 06:35:34.031000+00:00  2269840208
2020-01-03 06:36:04.032000+00:00  2269840820
2020-01-03 06:36:34.041000+00:00  2269842350
2020-01-03 06:37:04.032000+00:00  2269842554
2020-01-03 06:37:34.036000+00:00  2269842554
2020-01-03 06:38:04.032000+00:00  2269842554
2020-01-03 06:38:34.033000+00:00  2269842554
2020-01-03 06:39:04.090000+00:00  2269842554
2020-01-03 06:39:34.035000+00:00  2269853583
2020-01-03 06:40:04.036000+00:00  2269854129
2020-01-03 06:40:34.033000+00:00  2269854129
2020-01-03 06:41:04.035000+00:00  2269854228
2020-01-03 06:41:34.033000+00:00  2269855713
2020-01-03 06:42:04.035000+00:00  2269856604
2020-01-03 06:42:34.034000+00:00  2269866268
2020-01-03 06:43:04.034000+00:00  2269866916
.
```

```
data_df description:
          bytes-sent
count   2.881000e+03
mean    2.284429e+09
std     1.233722e+07
min     2.269840e+09
25%     2.271348e+09
50%     2.283434e+09
75%     2.296978e+09
max     2.303451e+09
```

Column Label Changed, Baselined, Conditioned

```
data_df:
       d_te11200_previous
1                       0
2                       0
3                       0
4                     612
5                    2142
6                    2346
7                    2346
8                    2346
9                    2346
10                   2346
11                  13375
12                  13921
13                  13921
14                  14020
15                  15505
16                  16396
17                  26060
18                  26708
19                  28328
.
```

```
data_df description:
        d_te11200_previous
count         2.880000e+03
mean          1.459358e+07
std           1.233636e+07
min           0.000000e+00
25%           1.507463e+06
50%           1.362955e+07
75%           2.713798e+07
max           3.361109e+07
```

Constructing Multi-Column DataFrame

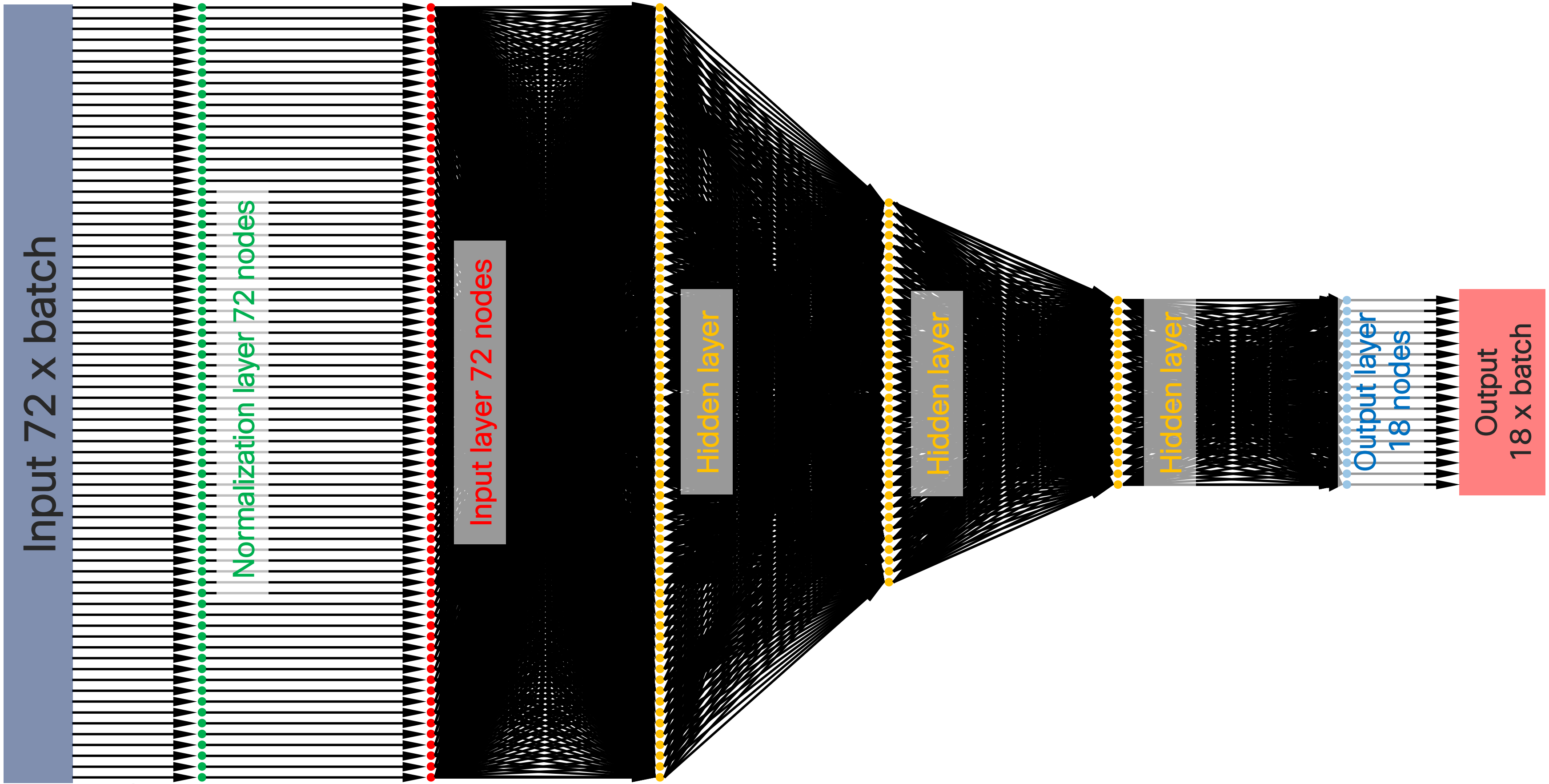
```
def read_train(record_count, label_prefix, verbose=True):
    for interface in tunnel_ifs:
        query_if = str('("interface-name" = \'%s\')' % (interface))
        label = str(label_prefix + interface[-7:] + "_previous")
        read_if = read_data('bytes-sent', 'Cisco-IOS-XR-infra-statsd-oper:infra-
statistics/interfaces/interface/latest/generic-counters', query_if, 'time >= now() - {} - 2h -
1m'.format(previous), 'time <= now()', record_count, label)
        if interface == tunnel_ifs[0]:
            train = read_if
        else:
            train = pd.concat([train, read_if], axis=1, sort=False)
        label = str(label_prefix + interface[-7:] + "_1d")
        read_if = read_data('bytes-sent', 'Cisco-IOS-XR-infra-statsd-oper:infra-
statistics/interfaces/interface/latest/generic-counters', query_if, 'time >= now() - {} - 1d - 1h -
1m'.format(previous), 'time <= now()', record_count, label)
        train = pd.concat([train, read_if], axis=1, sort=False)
.
        label = str(label_prefix + interface[-7:] + "_2w")
        read_if = read_data('bytes-sent', 'Cisco-IOS-XR-infra-statsd-oper:infra-
statistics/interfaces/interface/latest/generic-counters', query_if, 'time >= now() - 3w - {} - 1h -
1m'.format(previous), 'time <= now()', record_count, label)
        train = pd.concat([train, read_if], axis=1, sort=False)

    train.fillna(method='ffill', inplace=True)
    return train
```

Unique column label

Concatenate columns

The Model


Input 72 x batch
Normalization layer 72 nodes
Input layer 72 nodes
Hidden layer
Hidden layer
Hidden layer
Output layer
18 nodes
Output
18 x batch

The Neural Network, and Sample Cycle

```
.
  my_optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
  my_optimizer = tf.contrib.estimator.clip_gradients_by_norm(my_optimizer, 5.0)
  dnn_regressor = tf.estimator.DNNRegressor(
      feature_columns=construct_feature_columns(training_examples),
      hidden_units=hidden_units,
      optimizer=my_optimizer,
      model_dir= model_directory,
      label_dimension= len(tunnel_ifs) + len(physical_ifs)
.
hidden_units = [72, 36, 18]      # probably an overkill for our small scale
.
    cycle += 1
    print('cycle number ', cycle)
    dnn_regressor = train_nn_regression_model(
        learning_rate = 0.0003,
        steps = 1000,
        batch_size = 120,
        hidden_units = hidden_units,
        training_examples = read_train(120, 'd_'),
        training_targets = read_train_target(120, 'l_'),
        validation_examples = read_validate(120, 'd_'),
        validation_targets = read_last_target(120, 'v_'),
        prediction = True
.
```

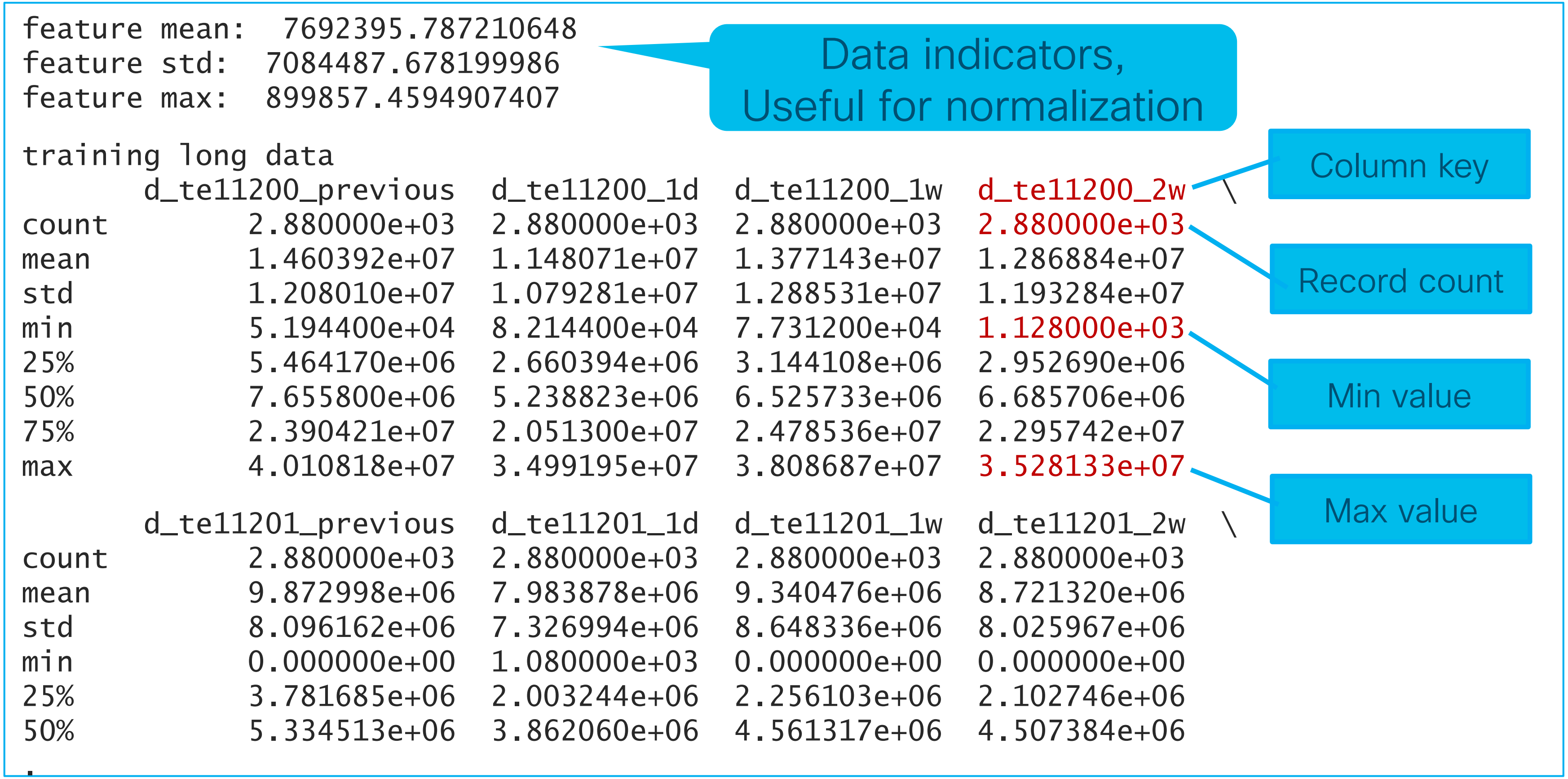
Set once, for a new model

Can change every call

+ Normalization Layer

```
def read_train_long(record_count, label_prefix, verbose=True):
    global feature_mean
    global feature_std
    global feature_max
.
    if feature_mean == 0:
        feature_mean = train.mean().mean()
        print('feature mean: ', feature_mean)
    if feature_std == 0:
        feature_std = train.std().mean()
        print('feature std: ', feature_std)
    if feature_max == 0:
        feature_max = train.max().mean() / 24 # The mean max per 1 hour
        print('feature max: ', feature_max)
.
def construct_feature_columns(input_features):
.
  # epsilon = 0.000001
.
  # choose best normalization of input data
.
  return set([tf.feature_column.numeric_column(my_feature, normalizer_fn=lambda val: (val) /
(feature_max))
              for my_feature in input_features])
```

The data - inputs



```
feature mean:  7692395.787210648
feature std:  7084487.678199986
feature max:  899857.4594907407

training long data
       d_te11200_previous  d_te11200_1d  d_te11200_1w  d_te11200_2w  \
count        2.880000e+03  2.880000e+03  2.880000e+03  2.880000e+03
mean         1.460392e+07  1.148071e+07  1.377143e+07  1.286884e+07
std          1.208010e+07  1.079281e+07  1.288531e+07  1.193284e+07
min          5.194400e+04  8.214400e+04  7.731200e+04  1.128000e+03
25%          5.464170e+06  2.660394e+06  3.144108e+06  2.952690e+06
50%          7.655800e+06  5.238823e+06  6.525733e+06  6.685706e+06
75%          2.390421e+07  2.051300e+07  2.478536e+07  2.295742e+07
max          4.010818e+07  3.499195e+07  3.808687e+07  3.528133e+07

       d_te11201_previous  d_te11201_1d  d_te11201_1w  d_te11201_2w  \
count        2.880000e+03  2.880000e+03  2.880000e+03  2.880000e+03
mean         9.872998e+06  7.983878e+06  9.340476e+06  8.721320e+06
std          8.096162e+06  7.326994e+06  8.648336e+06  8.025967e+06
min          0.000000e+00  1.080000e+03  0.000000e+00  0.000000e+00
25%          3.781685e+06  2.003244e+06  2.256103e+06  2.102746e+06
50%          5.334513e+06  3.862060e+06  4.561317e+06  4.507384e+06
.
```

Data indicators,
Useful for normalization

Column key

Record count

Min value

Max value

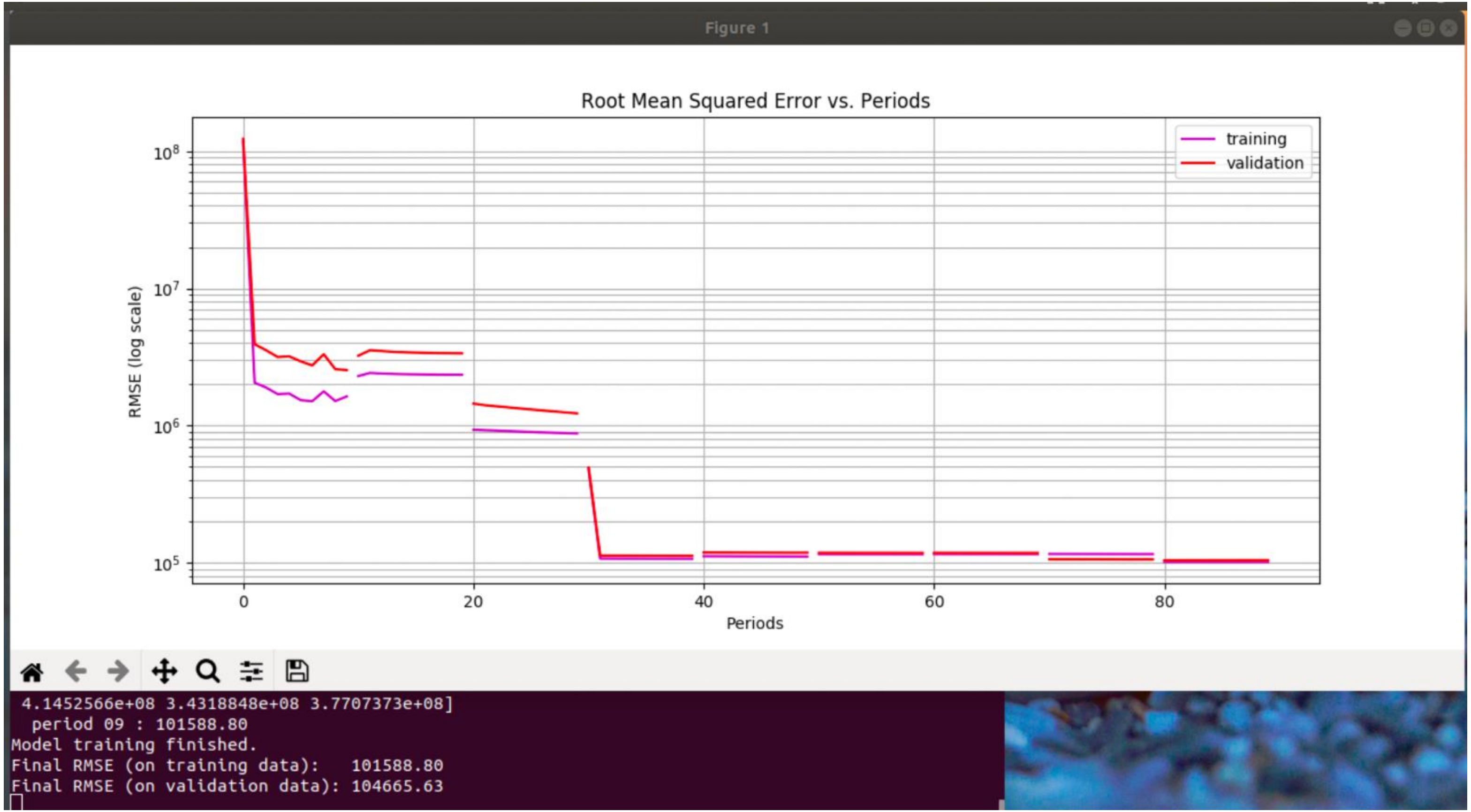
Check the Data: Model Output

```
.
training_predictions baoundaries
[ -9676.203    43134.777    30126.502    49435.23    -32687.848
  53458.71    -41498.527     2386.6956   -7947.478      841.848
  25334.666   -29982.818   -35702.715    39189.33       822.5124
  43476.05     48315.46       459.85422]
[2567748.5   1706653.8   1980904.5   2473544.2   2354506.8   2123372.
 2704984.5   1812864.2   2203519.      67233.766 1694452.8   1959683.9
 2268782.5   1549306.6     66137.98  2231304.2   1900274.9   1607212.8  ]
validation_predictions boundaries
[  9514.773    37849.348     3103.1733   43236.957    -5364.3784
  22101.646   -17330.65     16654.14      8049.155      765.1315
   3570.572    -6731.2876  -15120.497    33285.02       766.89764
  37848.816    19647.463    13101.8    ]
[2846600.   2264547.2  3210291.   2034950.   3650229.2  1717540.4
 1242460.5  1751227.8  2441805.8    68177.71 2717810.5  3045913.5
 1010453.4  2042300.1    66662.13 1811712.9  1519572.6  1552536.8 ]
  period 08 : 55037.89
.
```

The Training

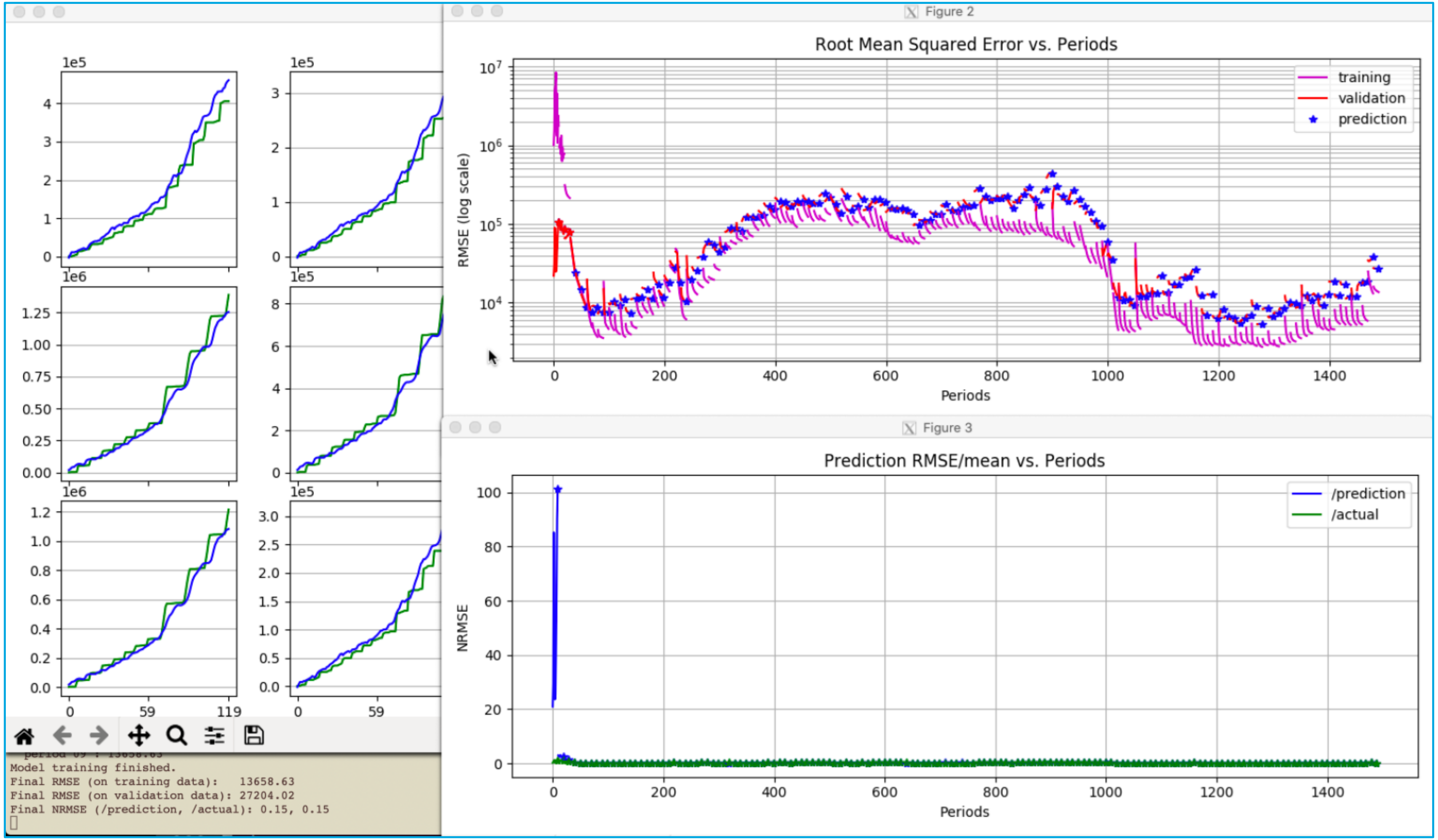
```
.
 period 08 : 3582.59
training_predictions baoundaries
[ 7208.3516   3251.541    7443.031    4104.243    4939.107    4307.9517
 -4889.6     -2828.935    5941.4014    726.0451   7325.6025   4610.5356
 -3598.1191   3449.1812    635.07806  4949.       4667.8926  -1990.8337 ]
[228423.4  177642.34 234673.56 181417.75 247542.95 188667.36 242869.14
 185446.6  211784.83  67781.5  213718.23 222849.58 222998.84 172893.55
  67602.49 174859.56 182980.92 178175.17]
validation_predictions boundaries
[ 6515.7734   3770.0752   8726.353    3760.6396  13467.836    3967.8716
 -5080.483   -3837.6772   5243.997     539.8233   8261.776   11545.85
 -3959.705    3852.7988    432.32617  4614.4375   4369.189   -3105.7751 ]
[193788.67  157614.84  150174.69  216238.94  187304.67  213302.8
 288468.47  197036.98  181182.31   51306.69  143677.8   166325.1
 265046.75  152532.97   51454.254 206665.23  205994.08  187957.39 ]
  period 09 : 3548.03
Model training finished.
Final RMSE (on training data):   3548.03
Final RMSE (on validation data): 22812.22
Final NRMSE (/prediction, /actual): 0.25, 0.25
cycle number  788
```

Training Makes Perfect!



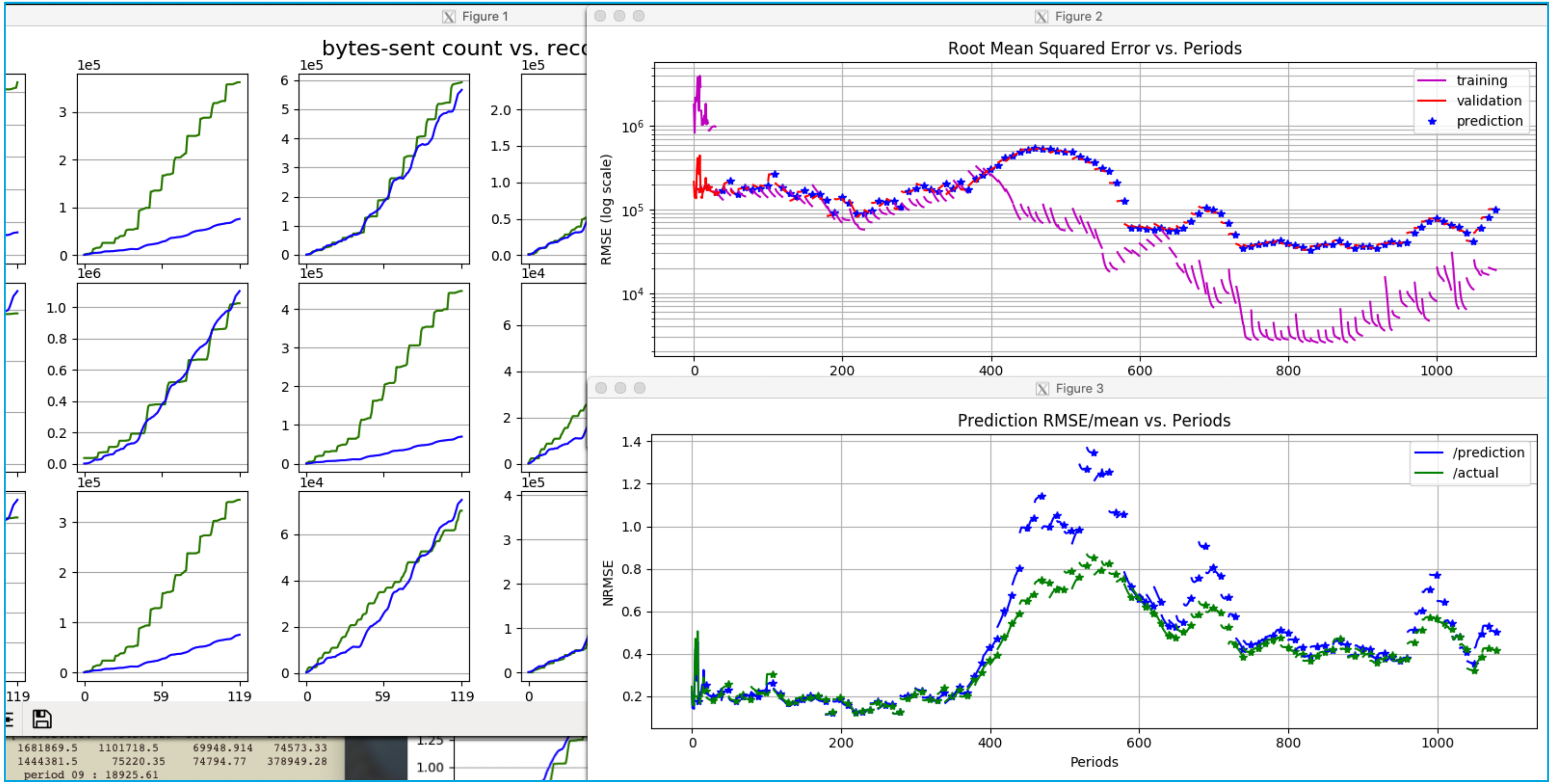
```
 4.1452566e+08 3.4318848e+08 3.7707373e+08]
  period 09 : 101588.80
Model training finished.
Final RMSE (on training data):   101588.80
Final RMSE (on validation data): 104665.63
```

Best Indicator: NRMSE



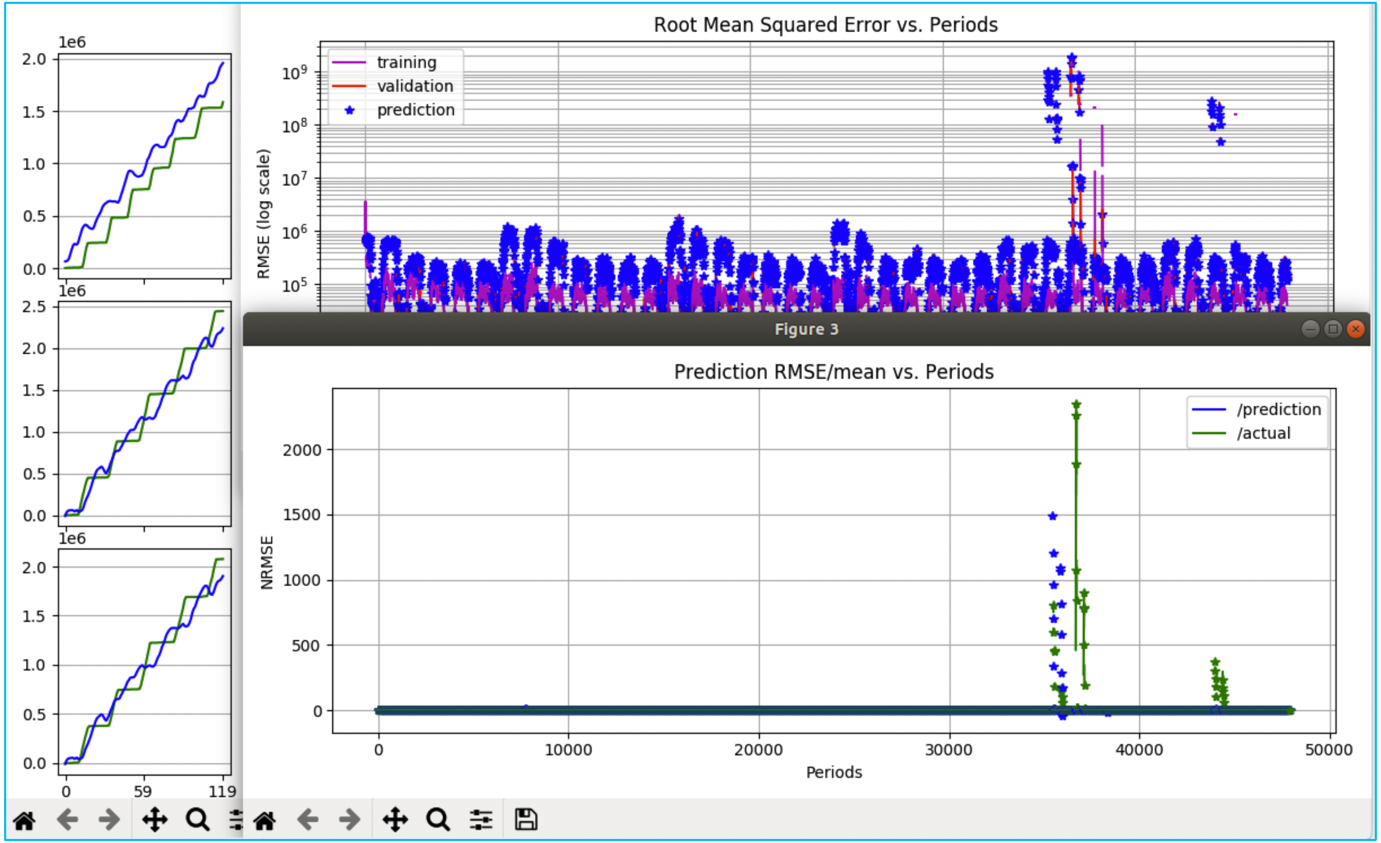
```
  period 09 : 13658.63
Model training finished.
Final RMSE (on training data):   13658.63
Final RMSE (on validation data): 27204.02
Final NRMSE (/prediction, /actual): 0.15, 0.15
```

A Short Term, Small Issue, & the Morning After



```
 1681869.5   1101718.5     69948.914    74573.33
 1444381.5     75220.35    74794.77   378949.28
  period 09 : 18925.61
```

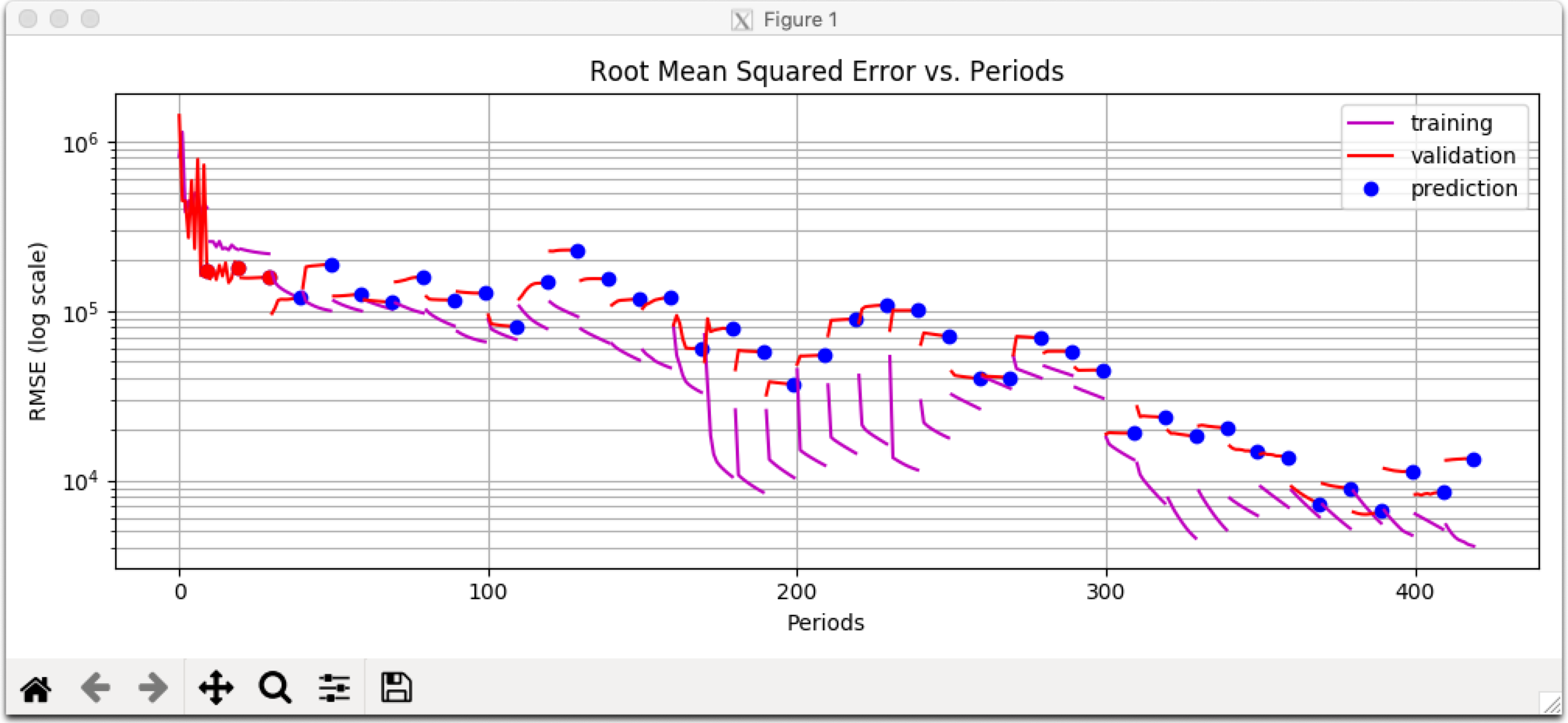
A Severe Issue



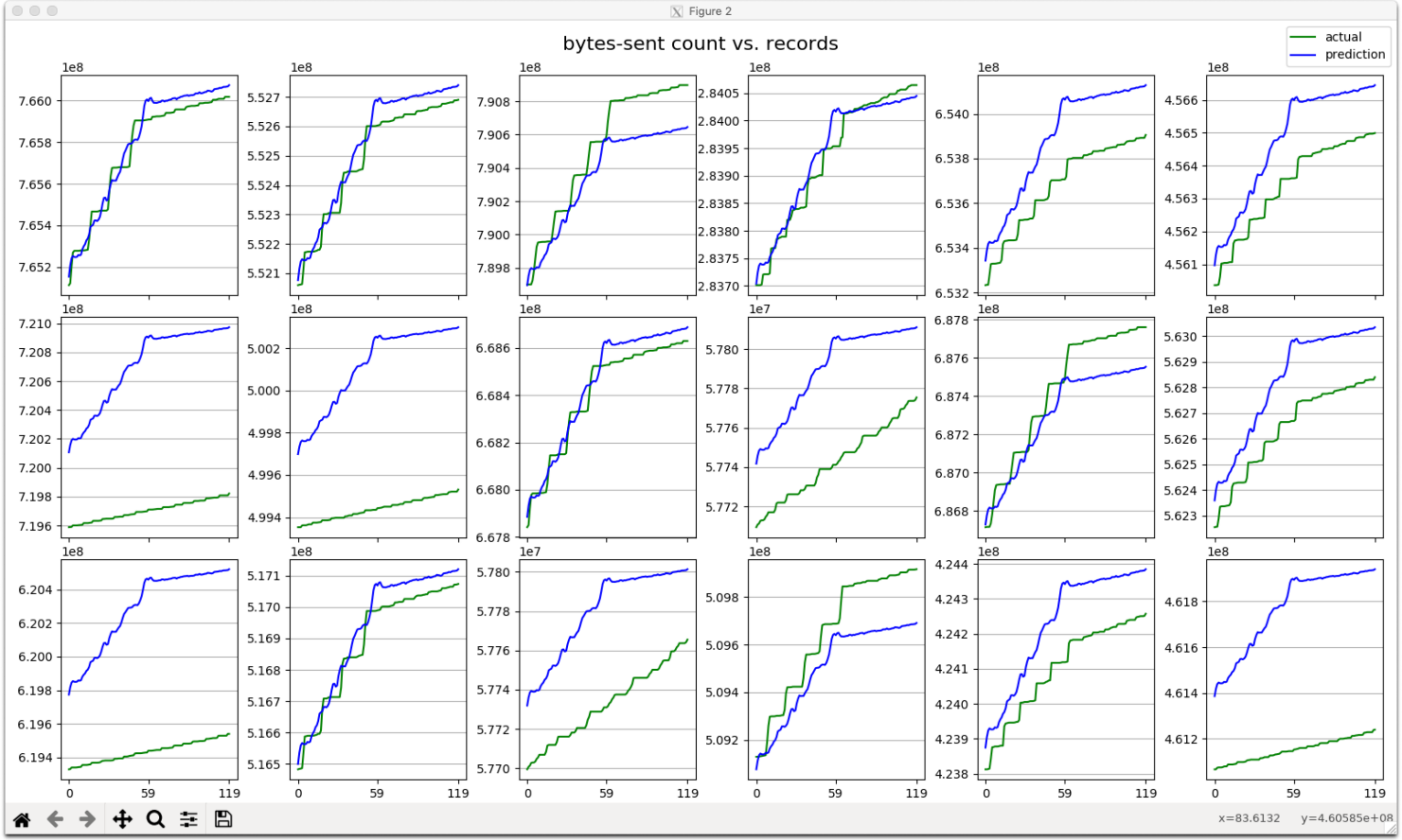
Visualize Progress: RMSE

```
.
 for period in range (0, periods):
    # Train the model, starting from the prior state.
    dnn_regressor.train(
        input_fn=training_input_fn,
        steps=steps_per_period
    )
    # Take a break and compute predictions.
    training_predictions = dnn_regressor.predict(input_fn=predict_training_input_fn)
    training_predictions = np.array([[item['predictions'][i] for i in range(0,
len(tunnel_ifs) + len(physical_ifs))] for item in training_predictions])
.
    validation_predictions =
dnn_regressor.predict(input_fn=predict_validation_input_fn)
    validation_predictions = np.array([[item['predictions'][i]for i in range(0,
len(tunnel_ifs) + len(physical_ifs))] for item in validation_predictions])
.
  if if_plot:
    # RMSE values and graphs
    global x_periods
    x_periods += periods
    plt.ion()
.
```

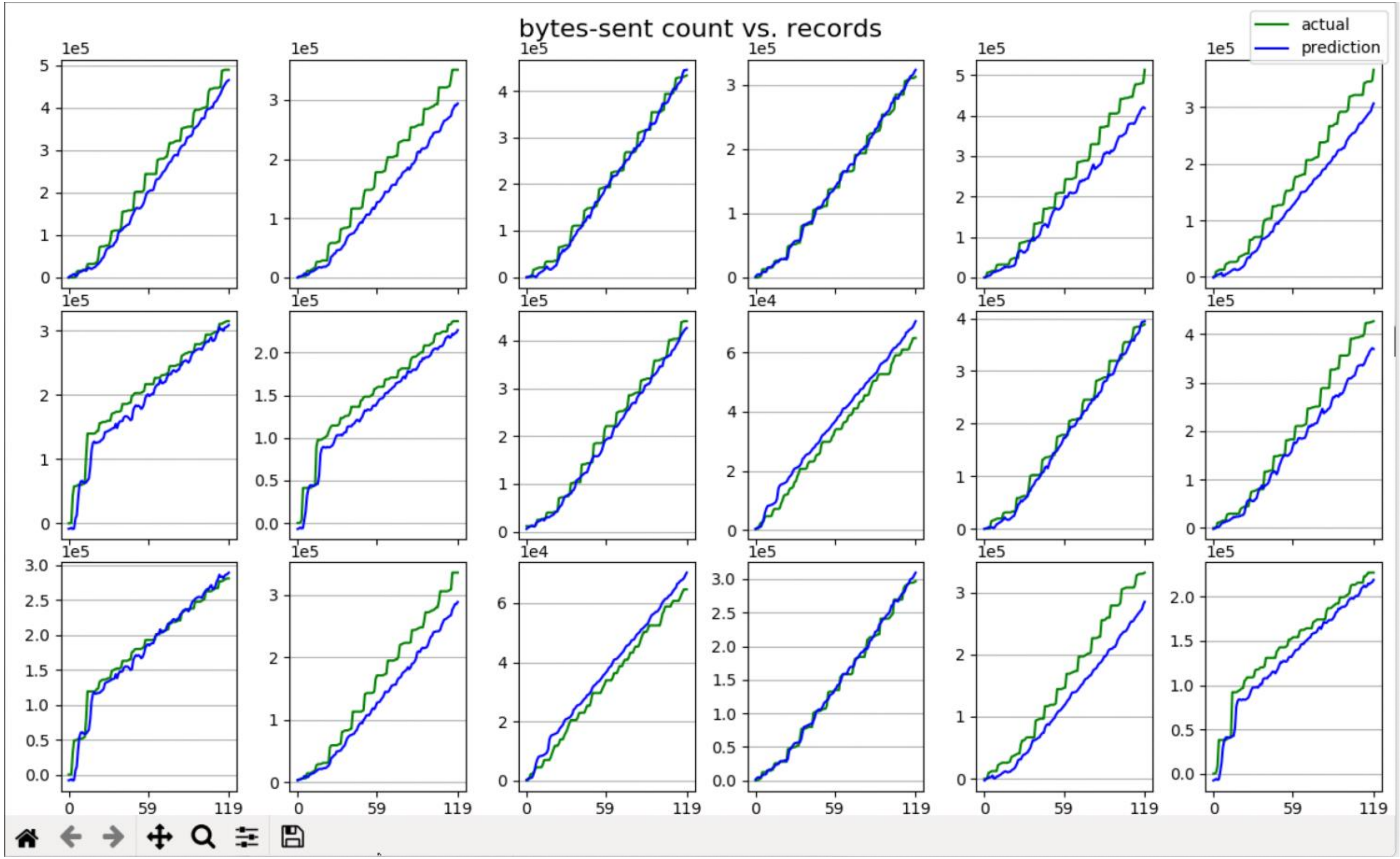
Visualize Progress: RMSE



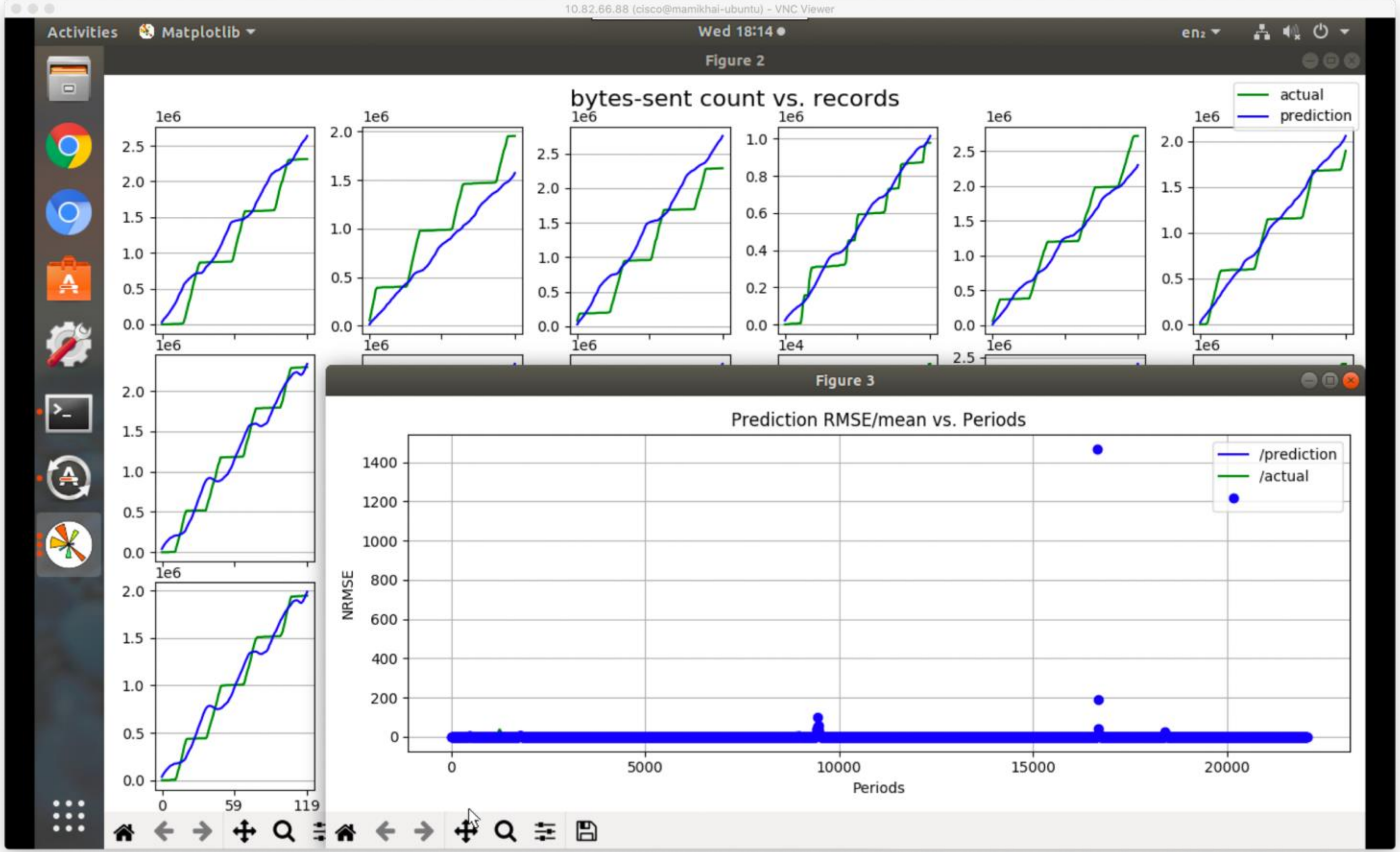
Validate: Visualize Prediction vs. Actual



Validate: Visualize Prediction vs. Actual



Eyes are Open?



Complete your online session survey

- Please complete your session survey after each session. Your feedback is very important.
- Complete a minimum of 4 session surveys and the Overall Conference survey (starting on Thursday) to receive your Cisco Live t-shirt.
- All surveys can be taken in the Cisco Events Mobile App or by logging in to the Content Catalog on ciscolive.com/emea.

Cisco Live sessions will be available for viewing on demand after the event at ciscolive.com.

cisco Live!

Continue your education

Demos in the Cisco Showcase

Walk-In Labs

Meet the Engineer 1:1 meetings

Related sessions

cisco Live!

Thank you

You make **possible**