Auto Encoder Decoder-Based Anomaly Detection with the Lakehouse Paradigm

Yinxi Zhang
Sr. Data Scientist, Databricks
Who am I?

Yinxi Zhang

- Sr Data Scientist @ Databricks
- ML Development and Deployment
We are going to talk about

- Challenges in Anomaly Detection
- The Autoencoder Approach
- Train Autoencoders Distributedly
- Deploy the Models
Brief on Anomaly Detection
Anomaly Detection

Use Cases

Predictive Maintenance

Fraud Detection

Medical Image Diagnose

Surveillance Video Monitor

![Predictive Maintenance Icon](image1)

![Fraud Detection Icon](image2)

![Medical Image Diagnose Icon](image3)

![Surveillance Video Monitor Icon](image4)

![Predictive Maintenance Chart](image5)

![Fraud Detection Chart](image6)

![Medical Image Diagnose Chart](image7)

![Surveillance Video Monitor Chart](image8)
Anomaly Detection

Challenges

• Anomaly labels are subjective and expensive to acquire
• Even when labels are available, anomalies are rare
→ Boundaries between normal and abnormal data are unclear

• Correlation with time + High dimensional features
→ non-linear, unsupervised model is preferable
Autoencoder
Autoencoder
Encoder + Decoder Hourglass Architecture

**Encoder**
It translates the original high-dimension input into the latent low-dimensional code.

*Non-linear Dimension Reduction*

**Decoder**
The decoder network mirrors the encoder architecture recovers the data from the input.

*Image source – https://lilianweng.github.io/posts/2018-08-12-vae*
Autoencoder

Encoder + Decoder Hourglass Architecture

Flexible Architecture

- Dense
- CNN
- LSTM
- Custom

Latent Space

- Lower dimensional
- Non-linear data representation
- Suitable for downstream tasks, e.g. cluster failure types

Image source - https://lilianweng.github.io/posts/2018-08-12-vae
Autoencoder

Anomaly Detection Modeling Steps

1. Train the AE on normal data only
2. Compute reconstructed error distribution of normal data
3. Choose reconstructed error threshold
4. Test the model and threshold, reconstructed error of anomalies need to be higher than chosen threshold

Image source – https://lilianweng.github.io/posts/2018-08-12-vae
Autoencoder

Limitations and variants

AE limitations

- Mapping between latent space and data space is deterministic
- Interpolating/Extrapolating latent space is challenging

Extensions

- **Variational Autoencoder** (generative model)
  - Encoder maps inputs to parametric latent distribution
  - Minimize KL divergence between parametric posterior and true posterior
  - [Keras implementation](https://keras.io)

- **Conditional Variational Autoencoder**
  - Condition data and latent variables with label information
  - Improve control over generated results
Pitfalls when implementing AE?
Develop model

Build, train, and evaluate model

1. Which network type should I use? Dense, CNN or LSTM?
2. How to select reconstructed error threshold?
Develop model

Build model

1. Which network type should I use? Dense, CNN or LSTM?

- **Time series**
  - a. Dense (Engineer time series features)
  - b. LSTM (N_samples_per_window x N_features)

- **Image**
  - a. CNN
  - b. Concatenate frames or CNN LSTM

**Data insights is the key**

- Appropriate window size
- Extract key features from window aggregations or rolling statistics
Choose AE model Architecture

Data Centric - Wind Speed Example (Time series)

Wind Speed of a month

Zoom in to a hour

daily window (bike sharing abnormal behavior)

Gust
Develop model

Build, train, and evaluate model

1. Which network type should I use? Dense, CNN or LSTM?
   a. Data insights are the key

2. How to select the reconstructed error threshold?
   a. Use both normal and abnormal data
   b. Aggregate prediction outputs (rolling stats)
   c. Use multiple metrics
      i. Duration metrics: Mean absolute percentage error (on normal periods), precision/recall (on anomaly duration time)
      ii. Instance metrics: precision/recall (on occurrence of alerts are sent)
Devop model

Choose reconstructed error threshold

more realistic decision boundary with both healthy and anomaly data

Duration Metrics Example:

Instance Metrics Example:

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>num of alerts sent in healthy period/total num of alerts sent</td>
<td>3/20</td>
</tr>
<tr>
<td>num of failure instances detected by model/total num of failures</td>
<td>6/7</td>
</tr>
</tbody>
</table>
Scale AE
Training and Deployment
Scale model training

Train an individual model per group instance

1. Naive approach
   a. use a for loop, iterate through data of each group instance
   a. manage training experiments per instance manually
   b. slow process and cumbersome codes

2. Pandas Function API
   a. Write users defined python functions (train, pred, eval)
   b. `groupby.applyInPandas()` maps each group of the current DataFrame and execute the user defined pandas udf function in parallel
Scale model training

Grouped map Pandas UDF

def train(self, df:DataFrame)->DataFrame:
    schema =  <return-schema>

def train_udf(df_pandas: pd.DataFrame) -> pd.DataFrame:
    '''Trains an Autoencoder model on grouped instances'''
    device_id = df_pandas['device_id'].iloc[0] # Pull metadata
    # Train the model
    X = df_pandas[features]
    ae_model = build_auto_encoder_decoder(df_pandas)
    ae_model.fit(X, X, **model_fit_kwargs)
    artifact_uri = f"{self.train_model_path}{device_id}.pickle"
    cloudpickle.dump(ae_model, open(artifact_uri, 'wb'))
    returnDF = pd.DataFrame([[device_id, artifact_uri]],
                              columns=["device_id", "model_path"])
    return returnDF

return df.groupby("device_id").applyInPandas(train_udf, schema)
Deploy Model

**Offline Features**

<table>
<thead>
<tr>
<th></th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
</table>
| **Prior to Model Training** | - Compute once and reuse  
- Leverages full dataset | - Slower to iterate |
| **Within the Model**          | - Easier to iterate                      | - Model latency depends upon transformation overheads  
- Data visibility            |

**Online Features (Context Features)**

- Encapsulate in model object
Deploy model with MLflow and Pandas UDFs

1. Custom MLflow model
   a. feature engineering + post-processing encapsulated in the model

2. Batch Inference
   a. groupby.applyInPandas(pred_udf)
   b. scheduled jobs

3. Streaming Inference
   a. unified API with batch
   b. integrate with streaming DLT pipeline

4. Real-time inference
   a. multi-model ensemble
   b. serve as a REST endpoint
Demo
To summarize

- Benefits of using AutoEncoders for anomaly detection
  - suitable for highly imbalanced data
  - flexible choices of network architectures
  - generates non-linear embeddings

- Scale AutoEncoders
  - train one model per group instance in parallel with Pandas UDFs
  - simplify deployment with MLflow pyfunc model
Questions?