Link Failure Prediction Analysis using Recurrent Neural Networks

Inspired by Language Modelling

Bharath Keshavamurthy and Imran Pasha

CISCO Systems, Inc.

June 2019

1 References

1. B. Keshavamurthy and I. Pasha,
"Correlation and Causation Analysis in Black Box Deep Learning Models: A Mathematical Framework", June 2019

2 System Model

- **Primary Task**: Binary Classification - The link under analysis progresses into a "Failed State" (1) or stays in a "Normal Operational State" (0)
- **Secondary Task**: Determine the progression of parameters essential for the primary task using time-series context-based estimation techniques
- In order to predict the failure of a link, we employ **Recurrent Neural Networks (RNNs)** to provide time-series estimates of the link parameters using historical context windows and use these look-ahead estimates to determine the progression of the state of the link.
- Additionally, the negative results of the primary prediction task are fed into the **Prediction Rationale Engine** (discussed in [1]) in order to determine the root cause(s) for the link progressing into a "Failed State".
- The primary task involves a standard Neural Networks based classification engine. This will not be discussed in this document.
- The causation analysis engine (Prediction Rationale Engine) is discussed in detail in [1].
- This document primarily deals with the design of the RNN-based time-series parameter estimation engine.

3 Time-Series Parameter Estimation using RNNs

- For the sake of this document and in order to generalize the application of this model, let us assume we are concerned with the time-series progression analysis of one design parameter/control variable denoted by α.
- We first construct a **vocabulary** for this parameter which helps us bound the possible values it can take on. This is a reasonable assumption because the possible values allowed for a particular feature in any given Machine Learning task is a finite set.
- The vocabulary needs to be processed in order to handle both numeric and categorical features. The entries in the feature vocabulary are mapped to integers using a one-to-one function $f : \mathcal{A} \rightarrow \mathbb{Z}$, where $\mathcal{A}$ is the vocabulary space of the feature.
- A few design parameters employed in this model are:
  - ω denotes the length of the **look-back context window**, i.e. the number of past samples considered for the prediction of the next (future) sample value
  - Ω denotes the length of the **look-ahead horizon**, i.e. the number of future samples to be predicted by this engine
  - λ denotes the length of the sample sequences before splitting them into input and target samples, i.e. $\lambda = \omega + 1$
  - β denotes the **batch size** for training
- The training dataset is first sequenced into samples of length λ and then split into input and target sequences each of length ω. These input and target pairs are allocated into batches of size β for training.
- The RNN model is built using TensorFlow leveraging modules and routines from the high-level Keras API.
  - The **embedding layer** is similar to Word2Vec embeddings used in language modelling - it projects the input context vector onto a dense, continuous, lower-dimensional vector space.
  - **GRUs (Gated Recurrent Units)** are used in the hidden layer and are activated with **sigmoid activation functions** and are initialized using **Xavier uniform initialization** ('*glorot_ uniform*' in Keras). GRUs are employed in this design instead of the widely popular LSTMs because GRUs are structurally simpler and hence, take smaller training times. Furthermore, GRUs do not have a forget gate and therefore, they expose the entire memory during their operation without having to set any additional control variables. Xavier uniform initialization sets up the RNN cells with the samples drawn uniformly at random from $[-\sqrt{\frac{6}{fan_{in}+fan_{out}}}, \sqrt{\frac{6}{fan_{in}+fan_{out}}}]$.

  - The design employs a **Hinton Dropout** layer for regularization (prevent over-fitting) so that the model generalizes well to unseen examples.
  - Finally, the output layer constitutes a fully-connected (**Dense**) NN-layer with neurons representing entries in the feature vocabulary.

- The RNN model employs a '**sparse_categorical_crossentropy**' cost function with an **AdamOptimizer**. In this categorical classification problem encapsulating the time-series parameter estimation procedure, since the design incorporates a single mapped integer for each entry in the feature vocabulary instead of a one-hot encoding for the class labels and since the number of available class labels is huge, we employ a sparse (cross entropy evaluation performed over a subset of the predicted probabilities vs the true class labels), categorical (non-binary classification problem) cross-entropy (measure the dissimilarity between the predicted probabilities and the true class labels) cost function.

- The designed model is then trained using the sequenced, split, and batched training dataset over many epochs.

- To leverage the CUDA and/or Tensor core capabilities of NVIDIA GPUs, we use CuDNNGRUs in our hidden GRU layer for faster training.

- After training the model, we use the most recent contextual samples of length ω to trigger the prediction process. During the course of the prediction process which estimates the feature values for a look-ahead horizon of Ω, the context window is moved to the right with every iteration of the prediction loop. The predicted values are iteratively added to an output vector and to a context vector which is employed as a look-back window for predicting the feature value in the next time step. The output vector is passed on to the primary classification engine where the time-series estimates of other relevant features are used in tandem to predict the progression of the state of the link across the given look-ahead time horizon Ω.