Inference of packet-error rates using SNIR patterns via neural networks

Chimera Solutions
in partnership with OMNeT++ Team
The beginnings

Future…?

- Someone knowledgeable about neural networks could produce great results in a short time
- Proposal article: https://docs.omnetpp.org/articles/neuralnet-errormodels/
- Posted on Reddit: https://www.reddit.com/r/deeplearning/comments/fgb9yb/request_for_advice_on_neural_network_architecture/
Problem statement

Main goal:
Improve speed of current wifi error models while maintaining the baseline accuracy.
SNIRs, Error model, PER

<table>
<thead>
<tr>
<th><strong>SNIR (input):</strong></th>
<th>Commonly used in wireless communication as a way to measure the quality of wireless connections</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Error model (what we enhancing):</strong></td>
<td>Describes how the SNIR affects the amount of errors at the receiver.</td>
</tr>
<tr>
<td><strong>Packet Error Rate (output):</strong></td>
<td>is used to test the performance of a receiver. PER is the ratio, in percent, of the number of Test Packets not successfully received by the receiver.</td>
</tr>
</tbody>
</table>
Problem statement

Models we going to mention:

- **Scalar Radio Model**
  - Analytical formulas
  - Based on single SNIR values
  - fast but still inaccurate

- **Layered Radio Model (baseline)**
  - Based on the whole SNIR pattern
  - Slow but can be trusted as a baseline measurement in this study (lack of empirical data)

- **Neural network approach**
  - Can be quite accurate.
  - Can achieve very fast inference speeds.
  - Can generalize for different wifi modes.
  - Can take into account the whole SNIR pattern by design
Problem statement

- Layered model
- Neural Network

Training Dataset:
SNIR patterns ←→ packetError values

Trade-off:
- Much faster computation compared to layered model
- Cannot be more accurate than layered model (only converge to it)

Based on OMNET++ presentation:
Motivation: trade-off
Exploring the data

Raw data: SNIR values corresponding to a packetError value.

Borrowed from OMNET++ presentation:
Exploring the data

Conjecture: if the humans can recognize relationships between packet error values and the SNIR patterns, there is a great chance that neural network can do also and even doing better.

Increasing packet error values
Exploring the data

Conjecture: if humans can recognize relationships between packet error values and the SNIR patterns, there is a great chance that neural network can do also and even doing better.

Try yourself!

colab-notebook demo

Interactive 3D clustering demo:
https://skfb.ly/ozB6B
Methods we tried

Convolutional Neural Networks

**expectations:**
- compatible with different input sizes
- excellent in image like pattern recognition
- **limitation: limited long range interactions in the data**

**reality:**
- in this particular case it worked but it was very hard to achieve good accuracy
- it was not as robust as we thought for architectural changes
- it was hard to tune and train it with this data
- it generalized poorly
Methods we tried

---

**XGBOOST**

**expectations:**
- SOTA in tabular data inference
- fast and generalizes exceptionally
- **limitation: input can be fixed size data only**

**reality:**
- for fixed size SNIR matrices it achieved very good accuracy and generalized well, but we needed it to work for variable SNIR sizes, so we just could not use it.
Methods we tried

LSTM

**expectations:**
- recognizing long range interaction in data
- excellent for time dependent data
- excellent for variable data size
- **limitation:** longer training times due to more sequential structure

**reality:** it satisfied all of our expectations
LSTM network and settings which worked

Model configuration and topology:

*Lightweight, and fast*

Input:

- **timeDiv**
- **frequencyDiv**

**input size:**
timeDivision X frequencyDivision (fixed)
(it was fixed in the data as well...)

batched input -> batch size as hyperparameter

```
Model: "sequential"

<table>
<thead>
<tr>
<th>Layer (type)</th>
<th>Output Shape</th>
<th>Param #</th>
</tr>
</thead>
<tbody>
<tr>
<td>lstm (LSTM)</td>
<td>(None, None, 64)</td>
<td>29952</td>
</tr>
<tr>
<td>lstm_1 (LSTM)</td>
<td>(None, 64)</td>
<td>33024</td>
</tr>
<tr>
<td>dense (Dense)</td>
<td>(None, 10)</td>
<td>650</td>
</tr>
<tr>
<td>dense_1 (Dense)</td>
<td>(None, 5)</td>
<td>55</td>
</tr>
<tr>
<td>dense_2 (Dense)</td>
<td>(None, 1)</td>
<td>6</td>
</tr>
</tbody>
</table>
```

Total params: 63,687
Trainable params: 63,687
Non-trainable params: 0
LSTM network and settings which worked

Conjecture:

*LSTM-s are designed for recognizing patterns which are “far from each other”, e.g. time series data points distant from each other with respect to time, SNIR values can be distant in frequency...*

Example for our conjecture in the study of long-range amino acid interaction inference:
https://academic.oup.com/bioinformatics/article/33/18/2842/3738544

CNN locality by design:
https://abenezer-g.medium.com/part-1-convolutional-neural-network-in-a-nutshell-89f81a329ec3 -> “The reason why CNN is best at image classification” section
Neural Network integration to existing pipeline.
Results

Training on 160,000 lines,

- Split the data. 80% training, 20% testing.
- We place the data into batches with identical timeDivision.
- We transform the data to have a list with shapes of $\text{batch\_size} \times \text{timeDivision} \times \text{frequencyDivision}$.

Target and predicted values distributions are very similar. Predicted values plotted in the function of target values.

Predicted vs Target values:

- RMSE: 0.059539
- Corr coeff: 0.9893
Results

The closer the points to the green ones, “the better”, due to the lack of experimental data....
Results

Comparing the Scalar Radio Model (blue) and Neural Network approach (orange) to the Layered Radio Model (green).

Comparison with respect of computational time.
Results: trade-off is fulfilled

Neural network approach is in agreement with the simulated baseline

Neural Network approach can be orders of magnitude faster.
Conclusion

The novel neural network approach in agreement with the Layered Radio model baseline, while it has a significant speed up in computation time, especially when the packet length is large.

With the frugally-deep environment, our Keras-python implementation is compatible to the existing OMNET++ ecosystem, making this project potentially valuable for future studies and relevant use cases.

In the future we are planning to continue the collaboration and expand our model to generalize to multiple WIFI modes as for now it is only compatible with a single specific mode.
Tools used

- python
- Keras
- frugally-deep

Creating the architecture, train, inference

Omnet++ team integrated our model with it to their pipeline

- pandas
- matplotlib
- jupyter

Data visualization, showcasing, data management.
Thank you

Check out our other projects: [www.chimeramultimedia.com](http://www.chimeramultimedia.com)

Follow us on Twitter, Instagram, LinkedIn, YouTube, join our Discord:
[www.chimeramultimedia.com/contact](http://www.chimeramultimedia.com/contact)