Anomaly detection at CERN websites

CERN summer student programme
Julia Jansson, julia.jansson@gmail.com
Supervisor: Konstantinos Samaras-Tsakiris, IT-CDA-WF

July 30, 2021

1 Introduction

The World Wide Web (WWW) was invented by British scientist Tim Berners-Lee in 1989, while working at CERN [1]. The idea was to share information between scientists in universities and institutes around the world. Over time, web services became increasingly important for CERN for transferring large amounts of data to servers over the world. The aim of my summer student project was to detect anomalies in some of CERN's web services.

From Prometheus metrics [2] we have some information about the current state of a web service, like response latency and error rate. We want to avoid a high error rate over a long time period. Moreover the error rate is strongly connected to the response latency of the website: as the latency explodes we might run into errors.

The end goal is to predict upcoming errors or upcoming rapid changes to latency from our metrics. We do this by forecasting the error rate or response latency given the other metrics. It is also interesting to try to classify different states of the data into stable or anomalous states. In this case we have to choose some levels for the different metrics where some levels are treated as anomalous in contrast to stable. We explore different approaches to this problem including time series forecasting, t-SNE visualisation, recurrent LSTM networks and lastly Transformers.

2 Background

This section covers the tools and metrics used for the project. Then a survey of previous work in multivariate time series forecasting is given including time series analysis, multivariate time series visualisation, recurrent networks and transformers.

2.1 Tools

During this summer internship I gained insights in valuable tools such as working remotely on a Kubernetes cluster, using Kerberos and ssh for authentication and getting better at collaborating with git. I also started managing my references with Zotero. But most of all I started learning a whole new programming language: julia [3]. I did a tutorial in julia and learned how to use Pluto notebooks (similar to Jupyter). My previous experience of machine learning (ML) was in python using PyTorch so learning julia and ML packages like Flux was interesting. Sometimes when trying to implement papers and posts it would have been more straightforward to use python but in this way we contributed to growing ecosystem of julia packages.

2.2 Metrics

From Prometheus metrics we have some information about the current state of a web service like current sessions, incoming and outgoing throughput, session instantaneous rate and connection instantaneous rate. The current sessions correspond to the number of users currently using the web service. Throughput measures how many packets arrive at their destinations successfully. The session instantaneous rate is the current number of sessions per second over last elapsed second, and the connection rate depends on the internet connection.

Latency describes the amount of delay on a network or internet connection. Low latency implies that there are no or almost no delays. High latency implies that there are significant delays. One of the main aims of improving performance is to reduce latency.

The error rate is the number of times per second we are faced with an error when entering the websites. Often it is undefined or NaN, this is because there are no errors at the time. Then it increases and at some point the error rate is so high so that it is not correctly

measured and is stored as Inf. In our data preprocessing we take care of this, so that we do not have NaNs or Infs in our data.

The response latency and the error rate could be correlated, since sometimes high latency can cause errors and the other way around. We aim to detect anomalies in error rate or latency to avoid unexpected errors. However the error rate might experience some periodical effects for some of the websites, for example a brief shutting down period each midnight. Therefore we do not wish to detect just point anomalies of the error rate but times where the error rate is *unexpected*.

2.3 Time series analysis

Time series analysis and forecasting can be done using statistical methods without even going into machine learning. The two main methods for time series analysis are frequency-domain methods and time-domain methods, where the former include spectral and wavelet analysis and the latter include auto- and cross-correlation analysis [4].

An application where spectral analysis is used for Multivariate Time Series (MTS) analysis is in classifying EEG data, for example to classify human emotion [5, 6].

Auto- and cross-correlation analysis analysis consists of taking autocorrelation, partial autocorrelation and cross-correlation. Autocorrelation is like correlation but shifted for a series with different lags, and partial autocorrelation is autocorrelation for the difference between two random variables. Cross-correlation is for two different variables [4].

There are several different time series models, the autoregressive (AR) models, the integrated (I) models, and the moving average (MA) models. These three classes depend *linearly* on previous data points. Combining this give autoregressive moving average (ARMA) and autoregressive integrated moving average (ARIMA) models, or the seasonal autoregressive integrated moving average models (SARIMA) [4].

However these methods are only linear. Non-linear time series models can represent changes of variance over time (heteroskedasticity). These models represent autoregressive conditional heteroskedasticity (ARCH) and the collection comprises a wide variety of representation (GARCH etc.). These are often used for modelling financial data like the stock market [7].

2.4 MTS visualisation

Several techniques have been developed to visualize and analyze time series data [8, 9]. Since these techniques are made for univariate time series data they may not fully capture the correlations of the MTS data. However, multivariate data is hard to represent because of the difficulty of mentally picturing data in more than three dimensions. One way of visualising MTS data is Multivariate Time Series t-Distributed Stochastic Neighbor Embedding (m-TSNE) which is a modification of the t-SNE method [10, 11]. The t-SNE method is similar to Principal Component Analysis (PCA) in reducing dimensions of high-dimensional data and is used for visualisation, with the advantage that t-SNE is non-linear and stochastic compared to PCA which only captures linear dependencies [12]. The m-TSNE method first calculates the similarity between all pairs of MTS data points in high-dimensional space, based on Extended Frobenius norm (EROS) which is a similarity metric for MTS data [13]. Then, it computes the low-dimensional (2-D or 3-D) projection of the MTS data points using a gradient descent method by preserving the similarity relation between pairs of high-dimensional points [10].

2.5 Reccurrent Neural Networks

The recurrent neural network (RNN) is a common tool in Natural Language Processing (NLP). Recurrent networks have a hidden state that is used for generating a prediction from a sequence. This hidden state is passed on, so that it can be used when generating the subsequent prediction, which gives it a recurrent structure [14].

The original RNNs had numerous issues such as the vanishing gradients problem, and no long term memory. The Long Short-Term Memory network (LSTM), which is robust to the vanishing gradients problem, was then developed. Because memory was now maintained separately from the previous cell output it is capable of storing longer-term memory. [14]

Especially when the attention mechanism was invented on top of the regular LSTM long-term memory issues were diminishing rapidly and good results could be acquired for MTS forecasting. The attention mechanisms contain a weighted context vector that weighs the outputs of all previous prediction steps instead of the hidden state [15].

It is also possible to combine convolution with LSTMs and use it for similar MTS forecasting and classification tasks [16, 17]. However, the convolutional neural nets were more focused on image processing application such as pedestrian identification [18].

LSTMs still had some issues since processing had to be performed sequentially, imposing a significant resource bottleneck. This problem was solved when the Transformer was introduced. The Transformer is capable of maintaining the attention mechanism while processing sequences in parallel: all words together rather than on a word-by-word basis. Transformers also make it possible to transition from point forecasting to a sequence-to-sequence model [19].

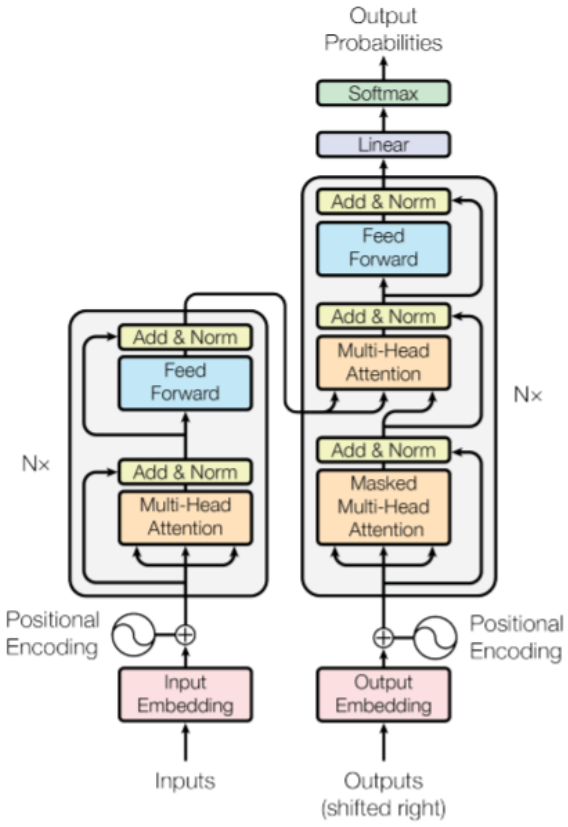


Figure 1: The transformer architecture. [19]

2.6 Transformers

Transformers consist of an encoder segment, which

1. takes inputs from the source language,
2. generates an embedding for them,
3. encodes positions,
4. computes where each word has to attend to in a multi-context setting,
5. outputs some intermediary representation.

The next part is a decoder segment, which takes inputs from the target language, generates an embedding for them with encoded positions, computes where each word has to attend to, and subsequently combines encoder output with what it has produced so far. The outcome is a prediction for the next token, by means of a Softmax and hence argmax class prediction (where each token, or word, is a class) [20].

The trick with the transformer to handle the sequence is the positional encoding which stores positions in the sequence as values. This time we do not have a strict ordering but the position is still taken into account due to the encoding. The other thing is the attention mechanism in the encoder and decoder, which chooses which elements in the sequence to focus on [20].

Transformers have had great success in NLP with large language models such as GPT-2, GPT-3, BERT and so on. But how do we use them for MTS data? Some ideas include Adversarial Space Transformer for time series forecasting [21] and Gated Transformer Networks for MTS classification [22].

3 Method

The methods used for our project is described from preprocessing and data analysis to our implementations of TPA-LSTM and transformers. The code for the project can be found on gitlab `https://gitlab.cern.ch/webservices/webservice-anomaly-detection/`.

3.1 Preprocessing

Data was collected from around 2000 web services using the prometheus metrics over the scope of two weeks. This formed a MTS with 7 metrics (features) over the time of 20064 minutes. First a significance metric is calculated for all the website data, excluding the ones with too little traffic to be interesting. The metric is simply the sample variance of the connection rate, and if it is too low the data is discarded. This narrowed down the number of websites to around 100, from where we choose to focus on those with most activity, for example `lhcbproject` and `cvmrepo`.

In the error rate we substituted NaNs to zeros and Infs to max of the data. The other metrics had no missing data so we left this unprocessed. We used this data in the data analysis such as autocorrelation functions. For the rest of the data analysis the data was then mean-centered and min-max normalized.

3.2 Data Analysis

The time series properties of the relevant websites were analyzed with autocorrelation and partial autocorrelation plots as well as crosscorrelation plots. We noticed slight seasonality

for the error rates but no clear trend. Overall the data did not look like the ARIMA model could explain it; it was quite irregular and occasionally had high anomalous spikes. We did not notice any significant correlation between the response latency/error rate with the other metrics. This means that regular time series forecasting would not be feasible and this is likely due to non-linear dependency.

We tried out some periodogram and spectrogram techniques but they were hard to interpret. The data was also visualised with m-TSNE plots, to try to find clusters in the data. However the representation of the data and implementation of the m-TSNE method proved to be quite tricky so we did not get any good results.

3.3 Temporal Pattern Attention Long Short-Term Memory Network (TPA-LSTM)

For the TPA-LSTM we based our code on the example on github `https://sdobber.github.io/FA_TPALSTM/`, which was already made for MTS forecasting for solar power data. Similarly to the solar power data we segmented the data into windows that are overlapping (stride = 1). In our case the windows were 15 minutes long. We simultaneously stored the target value to be predicted, a horizon ahead. We choose the horizon to be 5 minutes. So from the 15 minute window we wanted to do a point prediction of the error rate 5 minutes into the future. Iterated prediction is a possibility but we did not delve into that.

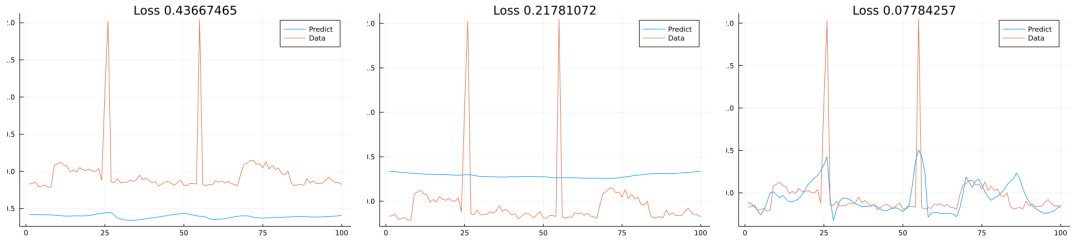
In practice we reshaped our 20064×7 array to some batches of $7 \times 15 \times 1 \times 100$ where 100 was our batchsize. The extra dimension was added for practicalities when using a LSTM.

3.4 Transformer for MTS forecasting

For implementing Transformers we followed the tutorial [23] and the paper [22]. We used the Transformers package in julia.

We created levels for discretization for each metric, to mimic a vocabulary. When used for NLP purposes, Transformers have a vocabulary with all the words and the output is given as a probability distribution of all the words in the vocabulary. However when we have a MTS instead of a word sequence, we do not have a vocabulary (that would be all the real numbers, an infinite set). Thus we created a discretized MTS with between 3 and 15 levels per metric depending on the metric (more details can be found in the code). The value at a given timestamp was in a given class if the current value is beneath that level but above the previous one.

The goal for the transformer was to predict the next minute's response latency class given the classes of all the metrics in the current 15 minute segment. The segments were stored in batches of size 64. We used Cross Entropy loss and trained about 10 epochs which took around 10-20 minutes. (Training on GPU was attempted but had numeric instability).



(a) 1 iteration. (b) 5 iterations. (c) 100 iterations.

Figure 2: Results from running different iterations of training of TPA-LSTM on 100 minutes of website data from cvmrepo.

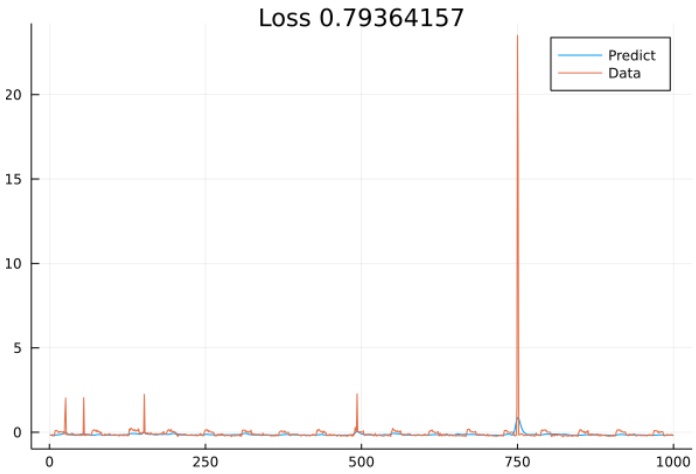
The positional encoding was standard for transformers and for the embedding we used a linear embedding layer. We used a single decoder for simplicity and repeated it 2 times. More implementation details can be found in the code.

4 Results

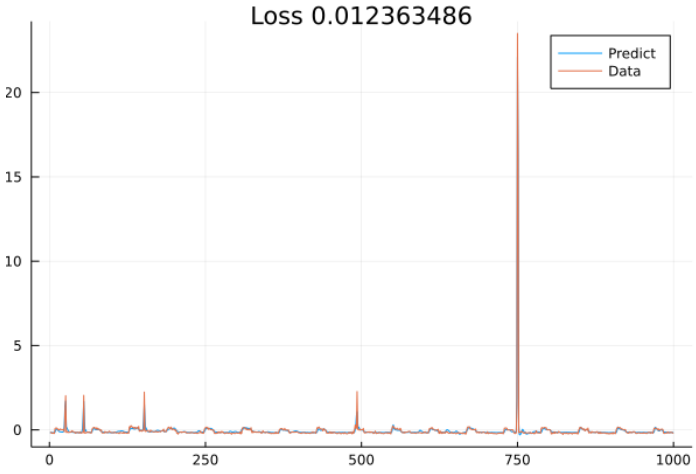
Here we present results for TPA-LSTM and Transformers training.

4.1 TPA-LSTM

When we tried TPA-LSTM we could only do small segments at a time, since it took a lot of time, and the processor ran out of memory. In figure 2 we see how the network learns for a 100 minute segment for the `cvmrepo` data. In figure 3 we see it corresponding for 1000 minutes. Unfortunately this is too short of an interval to see the clear daily dependence, which might have been easier to learn. We see quite good results regardless but it is important to keep in mind that it is only evaluated on training data so it could be overfitting.



(a) 10 iterations.

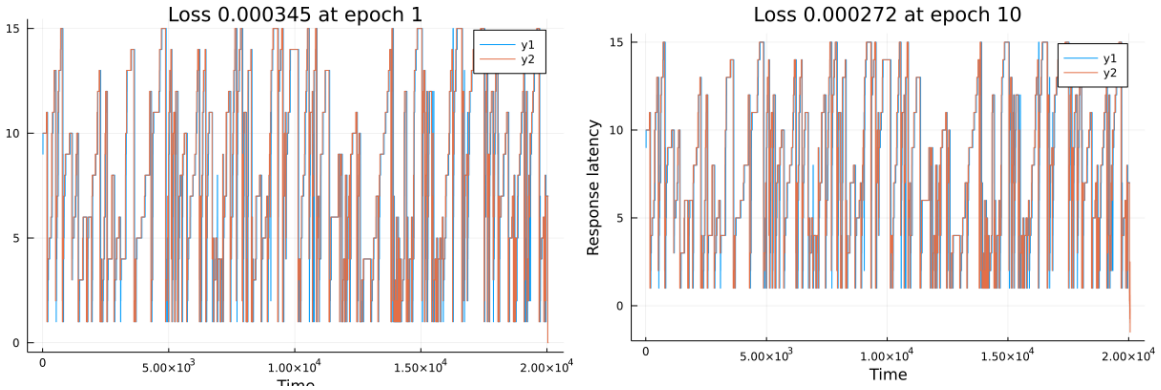


(b) 100 iterations.

Figure 3: Results from running different iterations of training of TPA-LSTM on 1000 minutes of website data from cvmrepo. The label to be predicted was the error rate 5 minutes into the future.

4.2 Transformers

The transformers training was much more efficient, so we could train the whole interval. We also changed to another website data (lhcbproject) and changed from error rate to response latency. Also our data was discretized as opposed to raw. This makes the two plots look quite different but also here we get a quite good result, even if it is hard to see. Again a problem is that we are evaluating on the training set and not a test set.



(a) Epoch 1.

(b) Epoch 10.

Figure 4: Results from running different iterations of training of Transformer on 20064 minutes of website data from lhcbproject. The label to be predicted was the response latency class 1 minute into the future.

5 Discussion

There is a lot of room for improvement for this project. First of all, we only focused on two of the websites and undeniably it would be interesting to also examine the other websites. For example, if we would find some interesting conclusion in the forecasting, it would be interesting to see if this also holds for other websites - so that the whole web server is affected, or if it is largely on a website to website basis.

Regarding the visualisation it would be interesting to see what clusters could be found with m-TSNE if we got it to work, because it seems promising.

The project was carried out in quite an explorative manner: we were not sure which method to use at first and only learned about Transformers at the end of the project. There is surely possible to get improvements in the TPA-LSTM part but I think Transformers are the way to go in this problem area.

We did not have time for hyper-parameter tuning, or separate training and test set for either TPA-LSTM or Transformers. This would surely improve the results a lot. It is also a good idea to vary the size of the segment from 15 to something else and to vary the horizon from 5 minutes in the TPA-LSTM. For the Transformer we had one-ahead prediction and it would be more interesting for a practical application to predict a larger window.

The Transformer architecture could be varied a lot, with different combinations of decoder and encoders and changing the embedding, positional encoding and segmentation. We choose to have levels in the Transformers but how would the result be if we removed them? Or if we had even cruder classes like anomalies or not.

Lastly, our data is sparse and this might affect the training results. Maybe special methods can be applied for sparse data.

5.1 Conclusion

The TPA-LSTM is a good model for forecasting error rate but not efficient enough. Therefore we use transformers for classifying response latency. This could be expanded for more data and by doing a more proper training.

6 Acknowledgements

I am thankful to CERN for giving this amazing opportunity to participate in the Summer Student Programme, including people from all over the world, great lectures and an interesting project. A big thank you to Konstantinos Samaras-Tsakiris for helping me in every step of the way going through with this project. I also want to thank Sofia Vallecorsa for providing helpful insights.

References

[1] *The birth of the Web*. https://home.cern/science/computing/birth-web.

[2] *From metrics to insight*. https://prometheus.io/.

[3] *The Julia Programming Language*. https://julialang.org/.

[4] Rob J Hyndman and George Athanasopoulos. *Forecasting: principles and practice*. OTexts, 2018.

[5] Xiao-Wei Wang, Dan Nie, and Bao-Liang Lu. "Emotional State Classification from EEG Data Using Machine Learning Approach". en. In: *Neurocomputing* 129 (Apr. 2014), pp. 94–106. ISSN: 09252312. DOI: 10.1016/j.neucom.2013.06.046.

[6] Murugappan Murugappan, Nagarajan Ramachandran, and Yaacob Sazali. "Classification of Human Emotion from EEG Using Discrete Wavelet Transform". en. In: *Journal of Biomedical Science and Engineering* 3.4 (Apr. 2010), pp. 390–396. DOI: 10.4236/jbise.2010.34054.

[7] Alexander J McNeil, Rüdiger Frey, and Paul Embrechts. *Quantitative risk management: concepts, techniques and tools-revised edition*. Princeton university press, 2015.

[8] J.J. Van Wijk and E.R. Van Selow. "Cluster and Calendar Based Visualization of Time Series Data". In: *Proceedings 1999 IEEE Symposium on Information Visualization (InfoVis'99)*. Oct. 1999, pp. 4–9. DOI: 10.1109/INFVIS.1999.801851.

[9] Conglei Shi et al. "RankExplorer: Visualization of Ranking Changes in Large Time Series Data". In: *IEEE Transactions on Visualization and Computer Graphics* 18.12 (Dec. 2012), pp. 2669–2678. ISSN: 1941-0506. DOI: 10.1109/TVCG.2012.253.

[10] Minh Nguyen et al. "M-TSNE: A Framework for Visualizing High-Dimensional Multivariate Time Series". In: *arXiv:1708.07942 [cs, stat]* (Aug. 2017). arXiv: 1708.07942 [cs, stat].

[11] Kwan Yeung Wong and Fu-lai Chung. "Visualizing Time Series Data with Temporal Matching Based T-SNE". In: *2019 International Joint Conference on Neural Networks (IJCNN)*. July 2019, pp. 1–8. DOI: 10.1109/IJCNN.2019.8851847.

[12] Geoffrey Hinton and Sam Roweis. "Stochastic Neighbor Embedding". en. In: (), p. 8.

[13] Kiyoung Yang and Cyrus Shahabi. "A PCA-Based Similarity Measure for Multivariate Time Series". In: *Proceedings of the 2nd ACM International Workshop on Multimedia Databases*. MMDB '04. Washington, DC, USA: Association for Computing Machinery, 2004, pp. 65–74. ISBN: 1581139756. DOI: 10.1145/1032604.1032616. URL: https://doi.org/10.1145/1032604.1032616.

[14] *From Vanilla RNNs to Transformers: A History of Seq2Seq Learning*. en-US. Dec. 2020. URL: https://www.machinecurve.com/index.php/2020/12/21/from-vanilla-rnns-to-transformers-a-history-of-seq2seq-learning/.

[15] Shun-Yao Shih, Fan-Keng Sun, and Hung-yi Lee. "Temporal Pattern Attention for Multivariate Time Series Forecasting". In: *arXiv:1809.04206 [cs, stat]* (Sept. 2019). arXiv: 1809.04206 [cs, stat].

[16] Tianjun Zhang et al. "Research on Gas Concentration Prediction Models Based on LSTM Multidimensional Time Series". en. In: *Energies* 12.1 (Jan. 2019), p. 161. DOI: 10.3390/en12010161.

[17] Joanna Waczyńska et al. "Convolutional LSTM Models to Estimate Network Traffic". en. In: (2021), p. 13.

[18] Yang Li et al. "Attention Based CNN-ConvLSTM for Pedestrian Attribute Recognition". en. In: *Sensors* 20.3 (Jan. 2020), p. 811. DOI: 10.3390/s20030811.

[19] Ashish Vaswani et al. *Attention Is All You Need*. 2017. arXiv: 1706.03762 [cs.CL].

[20] *Introduction to Transformers in Machine Learning*. en-US. Dec. 2020. URL: https://www.machinecurve.com/index.php/2020/12/28/introduction-to-transformers-in-machine-learning/.

[21] Sifan Wu et al. "Adversarial Sparse Transformer for Time Series Forecasting". In: *NeurIPS*. 2020.

[22] Minghao Liu et al. "Gated Transformer Networks for Multivariate Time Series Classification". en. In: (Mar. 2021).

[23] *Transformer Implementation for TimeSeries Forecasting | by Natasha Klingenbrunn | MLearning.Ai | MLearning.Ai*. https://medium.com/mlearning-ai/transformer-implementation-for-time-series-forecasting-a9db2db5c820.