ECE418 Final Project
Using a LSTM network for Sequence Classification on Memory Sequences

Koffas George [1], **Tsivgiouras Konstantinos**[1]

[1]Department of Electrical and Computer Engineering – University of Thessaly (UTH)
Volos, Greece

Abstract. *This report covers our team's approach on Sequence Classifiaction, with a given dataset of memory sequences. Our approach utilized LSTM nodes.*

1. Introduction

Sequence Classifiaction is a section of Machine Learning problems where the data is presented in time steps, and can be either continuously updating or static. The problem lies in predicting future values from these types of datasets, whilst the challenge lies in the fact that there is a lot of *seasonality* and, in some cases, *sparsity* of the data. For our specific problem, we had to deal with both seasonality, as memory accesses in programs are both spatially and temporally local, and sparse due to how programs use data structures and the heap and stack respectively.

2. Related Work

The idea of using neural network architectures instead of the more traditional prefetching algorithms in caches has been around for more than a decade. An approach similar to ours was taken by [Hashemi et al. 2018], where they used an LSTM model embedding both ΔN_i (difference between memory access at timestep N and timestep $N + 1$), and the corresponding Programm Counter (PC) to a cache miss leading to accessing memory. Other researchers used Convolutional Neural Networks [Koch et al. 2018] and multivariate popularity prediction for music video caching in YouTube, or even the more traditional back-propagation-based neural networks [Cobb and Elaarag 2008].

3. Architecture

For out solution we utilized a simple RNN model, and more specifically an LSTM architecture (hyperparameters discussed in section 5). The Embedding layer reads the input sequence and summarizes the information in something called the internal state vectors (in our case, these are the hidden and cell state vectors of the LSTM), which it then passes on to the LSTM layer. The hidden state h_i is computed using the formula

$$h_t = f(W^{hh}h_{t-1} + W^{hx}x_t)$$

where h_{t-1} is the hidden state at the previous timestep, x_t is the input and W are the weight matrices. The Dense layer uses a linear activation function to produce the output, which is then backpropagated to update the weights of the network.

4. Datasets & Libraries

The dataset we were provided with was a univariate sequence of 400000 memory accesses. The libraries we used for our code were the following:

- numpy (https://numpy.org/doc/stable/reference/)
- pandas (https://pandas.pydata.org/pandas-docs/stable/reference/index.html)
- sklearn (https://scikit-learn.org/stable/modules/classes.html)
- keras (https://keras.io/api/)
- category_encoders (https://github.com/scikit-learn-contrib/category_encoders)

5. Experiment

The original dataset was not suitable for training with our methodology, so we had to do some preprocessing. Our method included using the *OrdinalEncoder* module of the *category_encoders* package to give each unique address in our dataset an ID. After that, we divided the dataset into train and test datasets, and created deltas sequences of length 999, so that our prediction would be the 1000^{th} delta to occur, thus being able to predict the next memory access. Before feeding the data in the model, we used sklearn's MinMaxScaler to scale our data since LSTM models generally fare better when data is scaled from 0 to 1 or -1 to 1. Finally we built the model and trained it. The training algorithm was back-propagation in Recurrent Neural Networks. A more detailed analysis of the hyperparameters and other experimetal data can be seen in Table 1.

Network Size	128×1 LSTM
Learning Rate	0.01
Embedding Size	128
Batch Size	16
Optimizer	ADAM
Loss function	CategoricalCrossentropy
# Training epochs	4
Training dataset %	80%

Table 1. Training hyperparameters for our model

6. Results & Conclusions

Our results were indicative of how efficient LSTM models are in Sequence Classification problems; after training, the model had an average accuracy of 100% and a loss of 0.02. The overall training time for our network over the entire training dataset size was 1 minute and 19 seconds. The final weight parameters of our network can also be seen in the *final_weights.txt* file.

7. Future Work

To expand on our work, we could do some revisions. For example, we could label this problem as a sequence prediction one instead of a sequence classification one, and instead use either a Seq2Seq Encoder-Decoder model, or an auto-Encoder model. Nonetheless, this project has been a great learning experience, and should the opportunity arise, we would love to tackle one such problem once again.

References

Cobb, J. and Elaarag, H. (2008). Web proxy cache replacement scheme based on back-propagation neural network. *Journal of Systems and Software*, 81:1539–1558.

Hashemi, M., Swersky, K., Smith, J. A., Ayers, G., Litz, H., Chang, J., Kozyrakis, C., and Ranganathan, P. (2018). Learning memory access patterns. *CoRR*, abs/1803.02329.

Koch, C., Werner, S., Rizk, A., and Steinmetz, R. (2018). Mira: Proactive music video caching using convnet-based classification and multivariate popularity prediction. In *2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 109–115.