Chaos in Computer Performance

Shasvat Desai
University of Massachusetts Amherst
shasvatmukes@cs.umass.edu

Abstract

In this project, we endeavor to model the chaos in computer performance from the entropy signals of memory traces. We use variety of Machine Learning techniques of varying expressive powers to model these entropy signals. I have used sequntial autoencoders to achieve this goal. The method uses the power of LSTM to model sequential nature of our data and thus extracts the underlying chracterisitic-sof the data by learning the most salient features

1. Introduction

We were given the entropy signals as recorded from various pro- gram runs on multiple programming languages such as C,Java, Fortran. The choice of the entropy signal is justified since it is invariant to cache size, caching mechanism and addressing mechanism. The programs that were run were taken from various benchmarks that are used to bench- mark compilers and computer architectures. The spec benchmarks (SPEC Benchmarks[2]) and DaCapo Bench-mark were run. among others, with varying inputs. Entropy signals were generated using 64 bit and 4096 bit line sizes for memory ac- cess. Varying windows (1 million and 100k) for which the entropy of memory line accesses was evaluated. The set of traces comprised of instruction accesses, data accesses and combined in- struction and data accesses, each treated as a different access for the same program run. There were also multiple traces of the same program for some of the C programs and they were treated as dif- ferent traces and consequently the resulting entropy signals were treated differently too. All in all, the total number of entropy sig- nals amount to 1428. The goal was to explain the variance of the predictions using vari- ous supervised learning methods. We approach the problem without making any presuppositions about the nature of the signals. Having experimented with traditional Time Series Regression based techniques and Feedforward networks, we moved on to advanced Deep Learning techniques such as RNN and LSTM which are used to model Time-Series Data. The report is organized into four parts - Sequence autoencoders Experiements, future work and a summary of my learnings in the project.

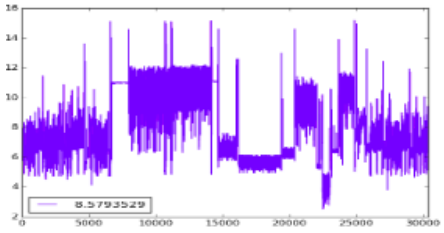

Figure 1. Sample Entropy signal for a memory trace

2. Seqeunce autoencoders

2.1. Purpose of using autoencoders

The primary problem which we are attempting to solve here is that of ascertaining , with a fair amount of certainty, the predictability of our entropy signlas. Now the entropy signals are generated through recording the sequence of memory access that a given program has made over a period of a time. Due to this the data over which we are trying to fit our model is essentially sequential in nature. This led me choose sequence autoencoders as my choice of model because they are immensely useful in modeling the underlying characteristics of a data which is sequential in nature.

2.2. Basic architecture

Autoencoding is the process of mapping the original input data to another space using a particular function , usually implemented by the Encoder part fo the model. This representation obtained after passing the input through a set of one or more encoders is known as the latent representation of the input data. This representation contains the most salient features of out input time series data. This latent represenation is again passed through a set of one or more decoders. A decoder is a part of the network which implements a function to map the latent representation of the data to output which is same as the input data in our case.
For our purposes, I am training the sequence autoencoders

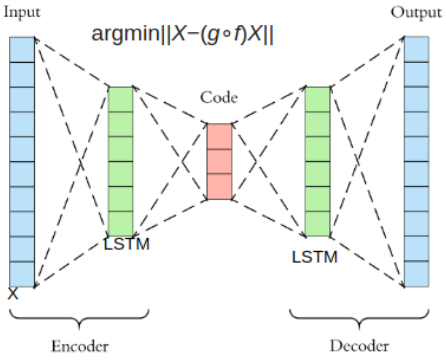

Figure 2. Model architecture

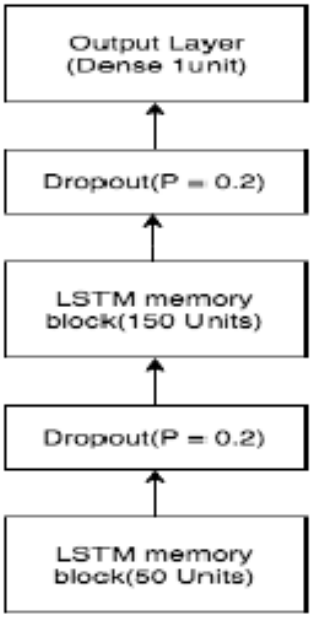

Figure 3.

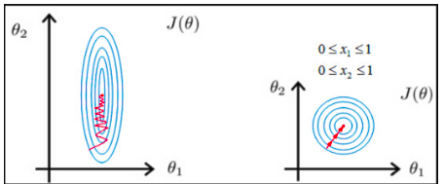

Figure 4.

by using the mean squared error as my objective function. What am I trying to do is essentially minimize the reconsturction error. The input block of my model is the encoder. I use LSTM network as my encoder as LSTMs[3] are known to work well with sequence data. The input is reshaped into the form $[n\_samples, timesteps, n\_features]$ The number of samples is equal to the batch size is our case which is a hyperparameter. The timesteps in another imprtant attribute as it determines how far back in hostory are we looking at to predict the next value in case of LSTM. In theory, LSTM are supposed to have memory of all the previous values due to the way the internal gates of a LSTM behaves, But in practice, we need to limit it certain number of timsteps as it cannot remember all previous values well and also due to increased computational overhead of remembering all the previous timesteps. The number of features we use in this is equal to one as we have only one value which is the entropy value. The decoder block is also LSTM.

2.3. Training Protocol

As a Machine Learning task, it is important for us provide the training protocol which entails the entire pipeline from setting up the data to modeling. For my purpose , I used all the entropy trace files with window sizes of 100k and cache line size of 64 bits. Thus in total, I use 274 trace files.

Then each file is divided into multiple blocks, with each block consisting of 200 entropy trace values. Thus if the file containes n trace values, the file will be split into n/200 chunks. 80% of the chunks of each file were taken for training and the rest 20% were kept for testing. Thus in this way, we ensure that our model sees a part of all the files atleast once.

An important point to make here is that after the chunks were formed, they were randomly selected for training and testing, instead of feeding first 80% of the chunk for training and the rest for testing. This is because of the nature of our data. Our data consists of entropy values generated from cache memory access of the computer program. Now since a computer program has phases asscoiated with it, each part of the entropy trace differs from each other greatly. Hence if the model sees only the first 80% of the chunk, it might not be able to generalize over the next 20% which might corresspond to some different phase of a program.

Given these chunks, we train our model over the entire set of 274 files with a batch size of 2048.

During testing phase, we remove the decoder part of the model. We load out trained weights and pass it through the Encoder part of the model to obtain the latent representation of the input test data for each of the trace file. Thus if our latent dimension is h, then a 274Xh output vector will be obtained.

One of the consideration which had to be taken care of during the training is that of normalizing the data between 0 and 1. This led to faster convergence and better learning(lower MSE) This might be because the LSTM network used in the Encoder part of our model, which makes use of tanh non-linearities. As show in figure above, convergence path becomes much smaller when normalization is done.

3. Experiments

3.1. Context length LSTM architecture

There is an inherent tradeoff between setting the context length of the LSTM, which would allow our model to look far back in history and the computational complexity of our model. This was a far more important questions in our case and I had to spend great deal of time in deciding this hyperparameter. This is because some trace files had around 10K points which were making LSTM too slow to work with.
Initilally, I set out my context length or the timesteps to be 50 with overalpping windows but soon figured, that since data points was highly correlated, skipping some of them would still enable us to get similar performance. Hence finally, I used context length of 200 with non -overlapping windows

3.2. Number of Hidden dimensions

Increasing the number of hidden dimension would imply ta model with a greater expressive power but at the same time it would also imply greater training time and danger of overfitting. I thus started out with 1024 dimensions, then went on to 512 and finally settled with 256 dimensions

3.3. Optimizer

I experiemented with 2 set of optimizers.First I tried to use SGD with momentum which failed to give good result as the netwotk failed to converge. Next I experimented with Adam optimizer which worked
In the end, I managed to obtain a MSE of 0.02 after training converged. Although, I was unable to obtain good clustering, this result shows that the model is promising and it might be useful to experiment with it in the future

4. Future Work

In future, several improvements can be experimented with to this architecture which might possibly yield better results.

4.1. Attention Mechanism in LSTM

With an attention mechanism, instead of encoding the full source sentence into a fixed-length vector ,we rather, allow the decoder to attend to different parts of the source input at each step of the output generation. Importantly, we let the model learn what to attend to based on the input sentence and what it has produced so far. Thus this might help the LSTM rememeber more and thus model the entropy trace in a better way

4.2. Multiple Layers of LSTM

In the current model, only one LSTM is used as encoder and a decoder. We could instead stack multiple layers of LSTM in both the encoder and the decoder to see what the model learns. It might be possible that the the stacking of 2 LSTM's together might enbale the model to learn different patterns or concepts in the signal and give us something different.

4.3. Variational Autoencoders

Currently, we have used sequence autoencoders which encode the input into a fixed length vector which is our latent representation. Instead a Variational autoencoder might be leveraged for the same purpose wherein instead we could find the parameters of the distribution corresponding to the latent representation and then sample values from the distribution to feed into the decoder. This might help us better understand the underlying characteristics of the signal.

4.4. Model input and output as distributions

Here we try to minimize the mean squared error betewen the values. This might point us in wrong direction in certain cases. Thus one could also experiment with modeling the input and the output as distributions. Thus rather than feeding in the raw entropy values,we could instead bin these values and find some distribution over them before feeding the input to the model. Over the ouput, we could just do a softmax which will again give us back the distribution.

5. Learning

I am thankful for the opportunity given to me to understand signal generated by a non-linear dynamical system. Obtaining an insight into the problem statement itself was an enriching experience. The subtelties involved and the nunaces that went into generating the signal itself was an intellectually enriching process. I learnt the importance of consistency in scientific process as we discussed and debated the best practices all of us should follow while dealing with time series data. I learnt to question all decisions- from what model should be used, why should that model would be useful, to the tradeoff between using overlapping windows and the computational complexity in LSTM. I delved deep into the study of seasonality and trend detection when dealing with time series data, even though they were not applicable in our particular case.
I got to train a Deep Learning model on a GPU. I tackled first hand the challenges of fitting such a model on a huge amount of data. Especially LSTM are known to be computationally expensive. generally, when LSTM's are used for sequence modeling tasks in Natural Language Processing, the input length involved in too short-as compared to the input length of our data. Sentence length in Natural Language rarely exceeds 200 while in our case , some of the input file had arounf 10K entropy values. I deeply understood and appreciated a wide variety of bells and whistles that go into making a Deep Neural Network trainable. Designing the

entire pipeline, starting from mapping my understanding of the problem to the model which I want to implement, to detailing the Training protocol and at the same time assessgin the tradeoffs involved in desiging the model architecture and understanding the impact this tradeoffs will have on the final outcome was a highly educating process

All in all it has been a great learning experience!

6. Acknowledgments

All the Entropy Signals were provided by Professor Eliot Moss and all the work to run, extract and formulate the entropy signals of memory accesses for the memory traces was done by Professor Moss. ValGrind (ValGrind) software was used in the process of extracting memory line accesses.

7. References

References

[1] F. Chollet et al. Keras. `https://keras.io`, 2015.

[2] J. L. Henning. Spec cpu2006 benchmark descriptions. *SIGARCH Comput. Archit. News*, 34(4):1–17, Sept. 2006.

[3] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, Nov. 1997.

[1] [3] [2]