Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Pretrained Model for Understanding of Integer Sequences

Bachelor's Thesis

Thomas Lackner

tlackner@ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Ard Kastrati, Peter Belcak
Prof. Dr. Roger Wattenhofer

August 23, 2022

Abstract

The goal of this thesis is to use a deep bidirectional transformer model to learn representations for integer sequences. This technique already produced state of the art results in natural language processing for several downstream tasks as shown in BERT [1]. As in BERT, the model is first trained on a *masked model* task followed by an *is next prediction* task.

The BERTIS project, as in ”BERT used on Integer Sequences“, contains the model and a training loop implementation as well as several data preprocessing utilities.

BERTIS shows promising training results on a synthetic dataset of simple integer sequences but fails to find representations for a more comprehensive, natural dataset.

Contents

Abstract **i**

1 Introduction **1**

1.1 Transformers and BERT 1

1.2 What BERTIS does the same as BERT 2

1.3 What BERTIS does differently than BERT 2

2 BERTIS **3**

2.1 Preprocessing and Feature generation 3

2.2 Model architecture 4

2.3 Training tasks 8

3 Datasets **9**

3.1 Synthetic Dataset 9

3.2 FACT Dataset 9

4 Explorative Evaluation **10**

4.1 Synthetic Dataset 10

4.2 FACT Dataset 11

5 Missing Results **14**

Bibliography **15**

A Code **A-1**

Chapter 1

Introduction

What follows is a high level overview of transformer models, BERT and what was changed in BERTIS.

1.1 Transformers and BERT

Transformer are a design for deep neural networks for sequence modeling introduced by Attention Is All You Need [2]. Traditionally, recurrent encoder/decoder designs were used for this task, relying on LSTM, GRU or convolution units, to name a few. The major drawback of these techniques is the training effort. They require being executed in sequence by nature, yielding few performance benefits from parallelization. Transformers mitigate this problem by introducing a multi-head attention technique, where each attention head can be computed in parallel. Not only do the results show major improvements in training time but also in model performance in several downstream tasks. [2]

BERT [1] is a design built using transformers and is focused on natural language processing. It operates on sentences of word embeddings. The Bidirectional Encoder Representations from Transformers design is capable of fusing the context information from both the left and right side context of a word into its representation. It does so using a Masked Language Model (MLM) pretraining task, where some word embeddings are replaced by a mask token and the model is tasked with finding the word behind the mask. Additionally, it employs a Next Sentence Prediction (NSP) pretraining task, which aims to force the model to learn about relationships between sentences. During this task, the input consists of two sentences of word embeddings. The model is tasked with deciding whether sentence B follows sentence A or if sentence B was chosen randomly. BERT was shown to build high quality representations that can easily be utilized for several downstream NLP tasks. [1]

1.2 What BERTIS does the same as BERT

BERTIS is based on the BERT design. The architecture of the transformer block was taken over. Also, the idea of using one MLM and one NSP like task to pretrain the model was adapted. The mechanisms of masking tokens and selecting a next sentence stayed the same. The goal was to find a good representation for integer sequences which could be used for downstream tasks.

1.3 What BERTIS does differently than BERT

Instead of using word pieces embeddings for input sentences, BERTIS enhances the integers of the input sequences with generated features. It uses a different decoding of the transformer output. While BERT supports input of variable length up to a maximum, BERTIS operates on fixed size sequence windows. Since BERT operates on a word-pieces embedding of (relatively few) finitely many unique values, it uses a cross entropy loss for the MLM task to find the most probable one behind the mask [1]. BERTIS is using an MSE loss for the MLM type task since it is operating on integers.

Chapter 2

BERTIS

In addition to the model and the training loop implementations, the BERTIS project contains various preprocessing utilities.

2.1 Preprocessing and Feature generation

BERTIS preprocessing takes as input a list of integer sequences of varying lengths. For use in the model, the integers will be treated as float64 values. Sequences with values out of range of float64 will be filtered from the input. Subsequently, they will be split into windows of configurable size. If windows at the end of a sequence are not entirely full, they will be padded with 0 values. After feature generation, they will be split into training and validation data based on a configured fraction.

BERTIS will add various features to each integer based on the configuration:

- Order: the position of the integer within the sequence window
- Log: the natural logarithm of the integer
- Normalized: the value of the integer normalized across the sequence window
- Digits: each digit of the integer as a separate value in the feature vector

Before each epoch, the preprocessed sequences will be prepared for the MLM and NSP style tasks. In the context of BERTIS we use the terms Masked Sequence Model (MSM) and Next Sequence Prediction (NSP). The procedure is similar to the one used in BERT [1].

For the MSM task preparation, any integer within a sequence window is chosen 15% of the time. A chosen integer is replaced with a mask token 80% of the time. 10% of the time it will be replaced with a random integer from another sequence and otherwise it stays the same. The mask token is a vector the same dimension as the other values of the sequence that is filled with a configurable integer in all dimensions.

The NSP task preparation concatenates two sequence windows. 50% of the time it will be the actual continuation of the sequence and otherwise it will be a random sequence window picked uniformly out of all available sequence windows.

2.2 Model architecture

The model architecture is best explored on different levels of granularity. First a high level overview of the entire model will be given followed by a more in depth view on all building blocks.

High level BERTIS model

An example BERTIS model is shown in figure 2.1. The model input dimensions are a result of the window size and activated features configured during preprocessing. In this example, we are operating on inputs of at most 50 integers, each of which is enhanced by 21 features. The None dimension indicates the batch size.

For use within the transformer blocks the input vectors are expanded in the *transformer embedding* dense layer. Intuitively, this gives the transformer blocks more room to store a meaningful representation. The actual embedding dimension is configured to be 32 in this example.

What follows is the transformer stack. The design of an individual transformer block is the same as in [1] and is shown in more detail in figure 2.3. As in BERT a configurable amount of these blocks is stacked on top of each other, 3 in this case. Intuitively, a higher transformer stack allows the model to represent more nuanced dependencies between individual integers.

After the transformer stack, the dimension of the transformer output is reduced to one in preparation for the output. This is done by the *reverse transformer embedding* dense layer followed by the *slicing operation.*

At this point the model splits into the MSM and the NSP output paths. Both are a stack of two dense layers featuring the ReLu activation. The NSP output uses a sigmoid activation on the final layer. A figure of a sequential block can be seen in figure 2.2. Both paths are finally concatenated to form the output of the model.

Sequential Blocks

The schematic for a sequential block abstraction can be seen in figure 2.2. In BERTIS, this abstractions consists of 2 densely connected layers. The sequential block is parameterized by the input and output dimensions which correspond to

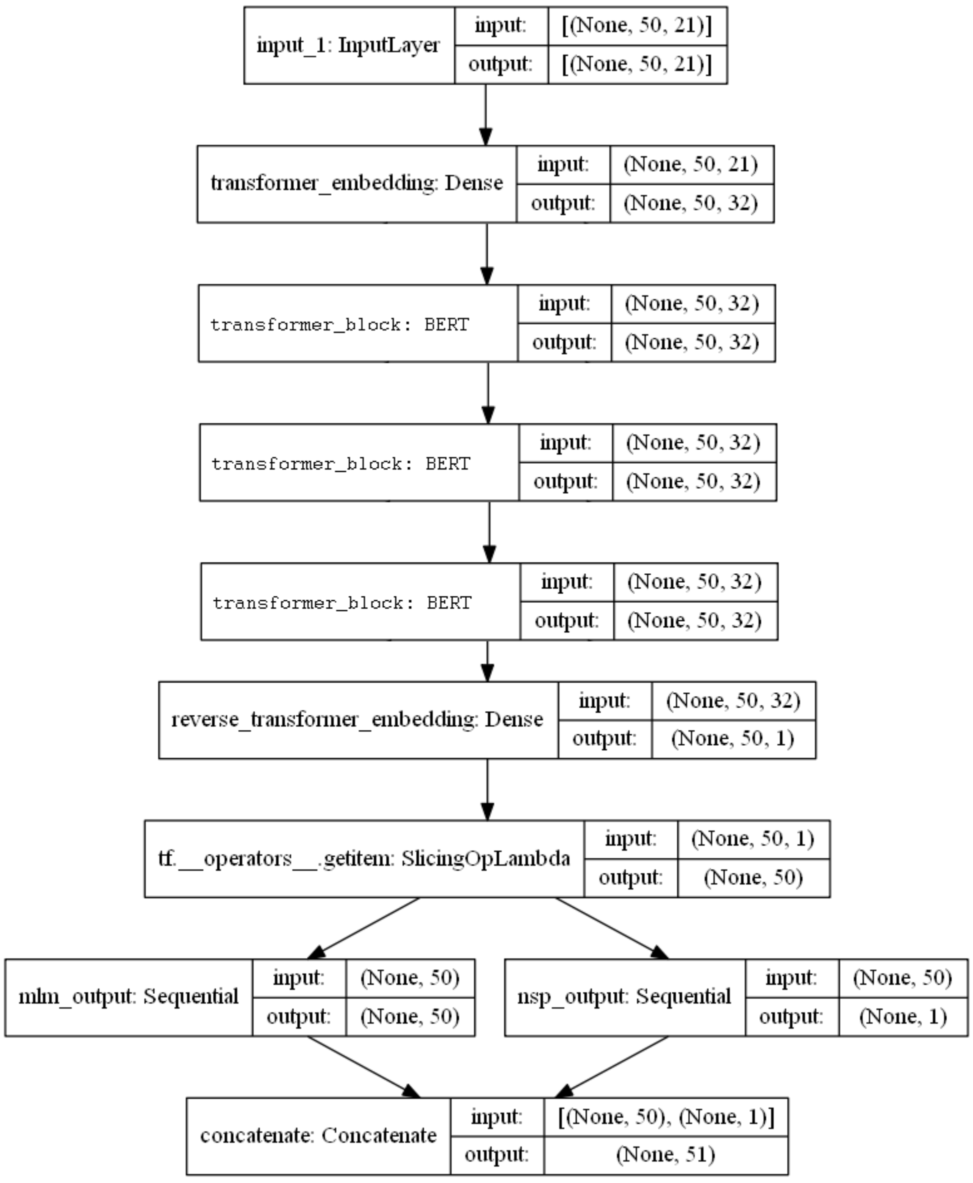

Figure 2.1: This is a full BERTIS model using three BERT transformer blocks.

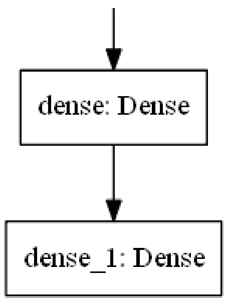

Figure 2.2: A simple sequential block of dense layers with varying activation functions.

the first and second layer dimensions. Additionally, the layer activation functions may vary.

Transformer Blocks

The schematic for the transformer block abstraction can be seen in figure 2.3. It follows the BERT [1] transformer design.

As an input, a transformer block takes an embedding vector. This may be the initial embedding dense layer as in the schematic, or the output of a previous transformer block.

What follows is a multi head attention block which is shown in more detail in figure 2.4. The result of the block is again a vector shaped like the transformer embedding. A dropout is applied to the result and finally combined with the input vector using elementwise addition of the entries in the last dimension of the vectors. The summed up vector is then normalized. The normalization again happens on last dimension of the vector.

This design of adding additional information to the input value and then normalizing the result is repeated within the transformer block. This time, instead of another multi head attention, a sequential block is used.

Attention and Multi Head Attention Blocks

A single scaled dot-product attention block can be seen in figure 2.4. The design is the same as described in [2]. It takes as input three vectors: Query, Key and Value. Depending on how these input vectors are filled, the block may model different concepts. The case where all three input vectors are the same vector corresponds to the concept of self attention. This is the relevant use case of this block for BERT as well as for BERTIS. Intuitively, the block finds a representation of how values influence each other. In other words, looking at an individual value

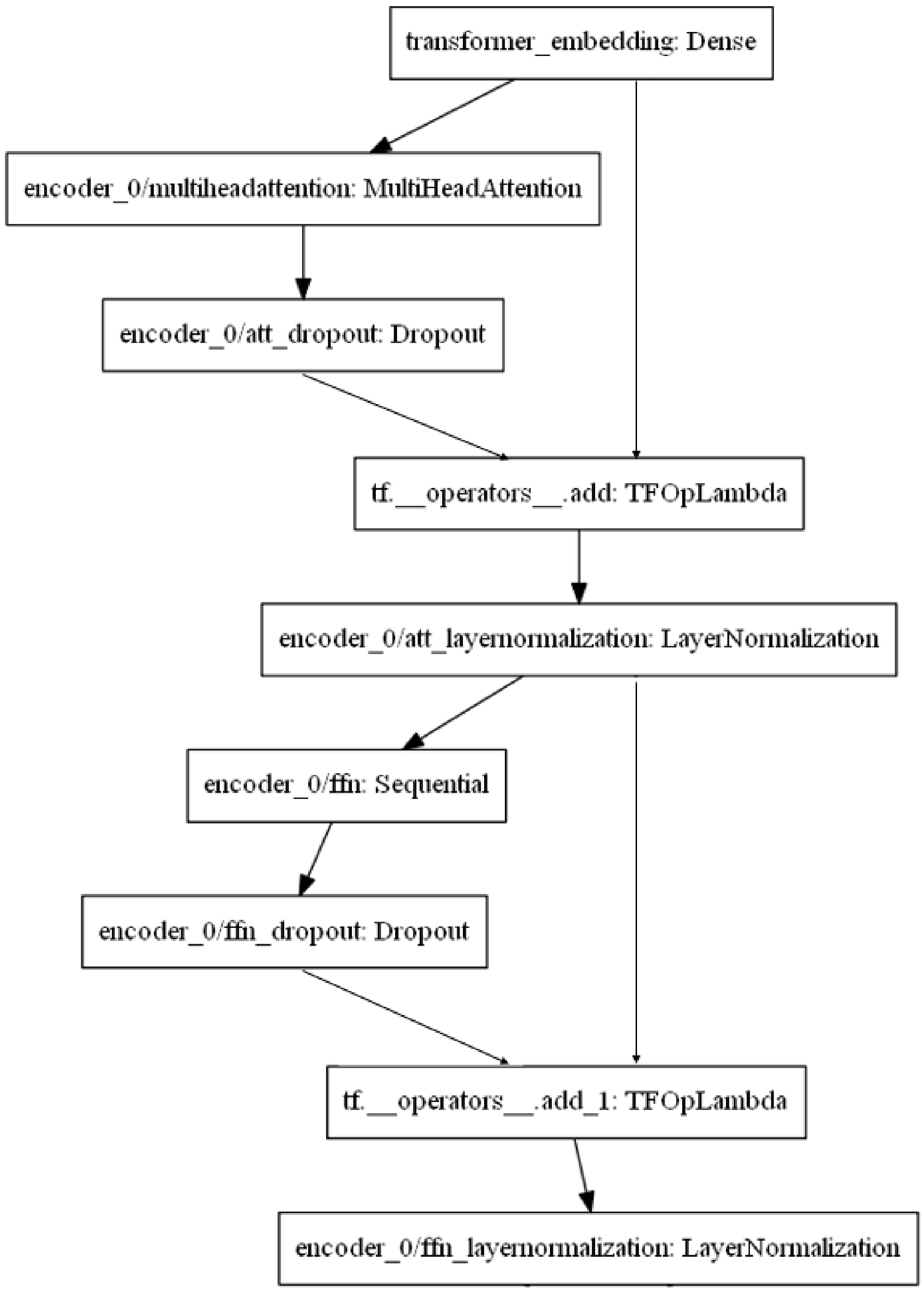
transformer_embedding: Dense
encoder_0/multiheadattention: MultiHeadAttention
encoder_0/att_dropout: Dropout
tf.__operators__.add: TFOpLambda
encoder_0/att_layernormalization: LayerNormalization
encoder_0/ffn: Sequential
encoder_0/ffn_dropout: Dropout
tf.__operators__.add_1: TFOpLambda
encoder_0/ffn_layernormalization: LayerNormalization

Figure 2.3: A BERT style transformer block.

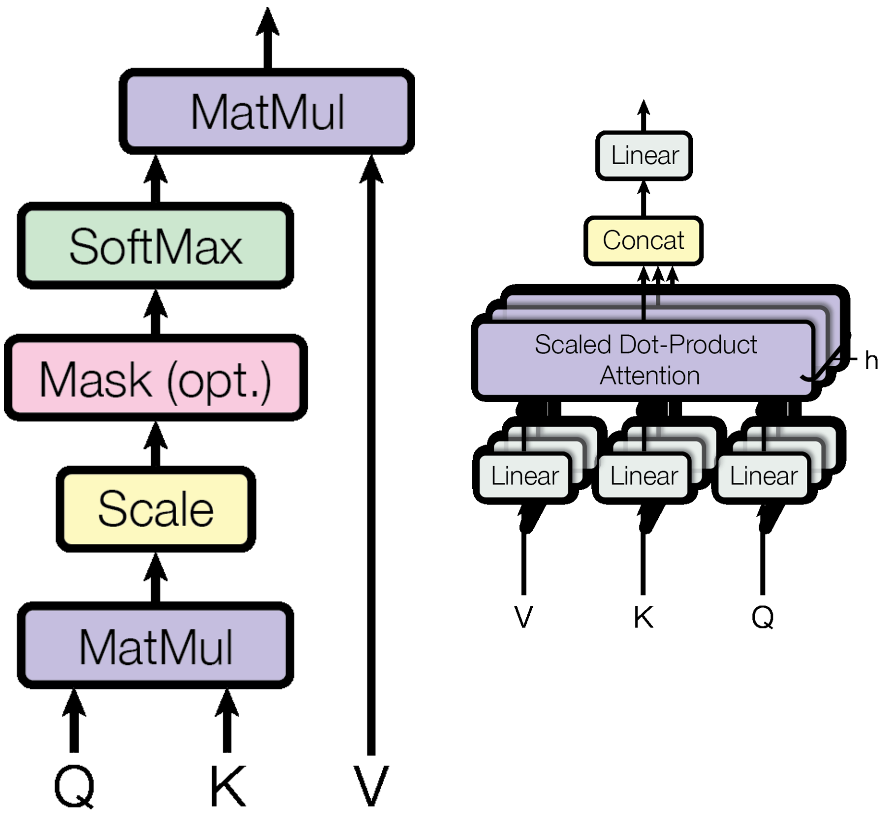

Figure 2.4: A scaled dot-product attention block and multi head attention block as presented in [2]

of the input, what other parts of the input does one have to pay attention to, in order to understand the context of the value.

Multi head attention, see figure 2.4, uses multiple attention blocks, also called heads, in parallel. This gives the model the opportunity to consider different attention models and choose the most fitting combination. The multiple attention heads are also the reason why the model can benefit from parallelization.

2.3 Training tasks

As in BERT, sequential training on MSM and NSP tasks is used to train the model.

For the MSM task an MSE loss function is used. BERTIS only consider the predictions for the masked values when calculating the loss.

The loss function used for the NSP task is also MSE. This time it operates only on the NSP output which is a single probability.

Chapter 3

Datasets

Two different datasets were used for training and testing BERTIS. One synthetically generated consisting of simple sequences and one based on the FACT dataset as described in [3]

3.1 Synthetic Dataset

The BERTIS project contains the code to generate this type of dataset. The synthetic dataset consists of configurable many integer sequences:

- of configurable length
- with values within a configurable range
- with a configurable fraction of constant sequences
- with ascending and descending sequences
- with a configurable fraction of periodic sequences (of period length within a configurable interval)

3.2 FACT Dataset

This dataset consists of preprocessed integer sequences from the OEIS dataset. The preprocessing was done by FACT [3]. Each sequence is labeled with its types (polynomial, exponential, etc.). This information is not used by BERTIS and is discarded. The preprocessed data consists of 1.75M sequence windows of 50 integers. For BERTIS training, the window size is configured to 25 such that there are valid next sequences for the NSP task.

Chapter 4

Explorative Evaluation

During the explorative evaluation of model hyperparameters, differences between datasets were strongly noticable.

4.1 Synthetic Dataset

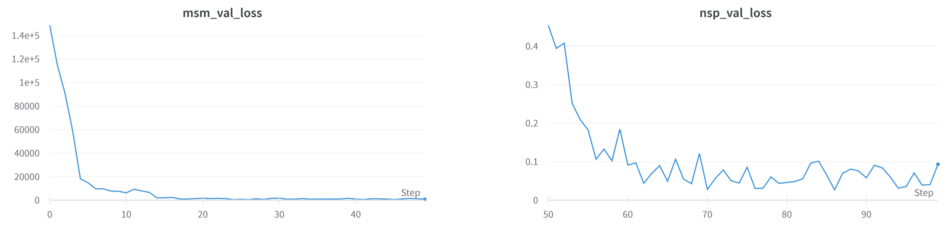

Figure 4.1: MSM and NSP task loss curves for a run on the synthetic dataset

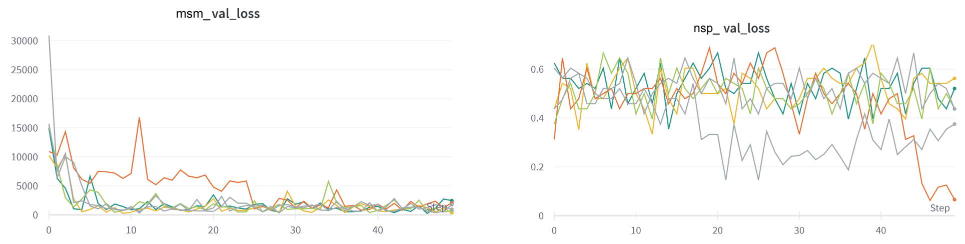

Figure 4.2: MSM and NSP task loss curves of runs with a varying number of features

Training on the simpler synthetic dataset was successful in the sense that losses are converging, see figure 4.1. Using hyperparameter search, the model configuration could be optimized, see figure 4.3. The most important parameter by far was found to be the optimizer. RMSProp showed the best results for both tasks. Other relevant parameters include the batch size and learning rate.

Training on the synthetic dataset showed that the generated features only have negligible effects on training losses. Given enough epochs, training on models who only differ in input features result in similar loss values as shown in figure 4.2. As can be seen in the NSP loss of figure 4.2, it even appears as if the run with no additional features (the orange line) performs best after 50 epochs of training. However, this is likely to change given more epochs. Unfortunately, the effect of features on training runs of the FACT dataset could not be investigated.

4.2 FACT Dataset

Training on the FACT dataset was unsuccessful. Since the model learned representations for the simpler dataset, one possible reason why training is failing is the model simply lacking the capacity to capture the more involved concepts of the organic dataset. Unfortunately, this hypothesis did not prove to be true as can be seen in figure 4.4. Increasing the model size in various dimensions had no effect on the training losses.

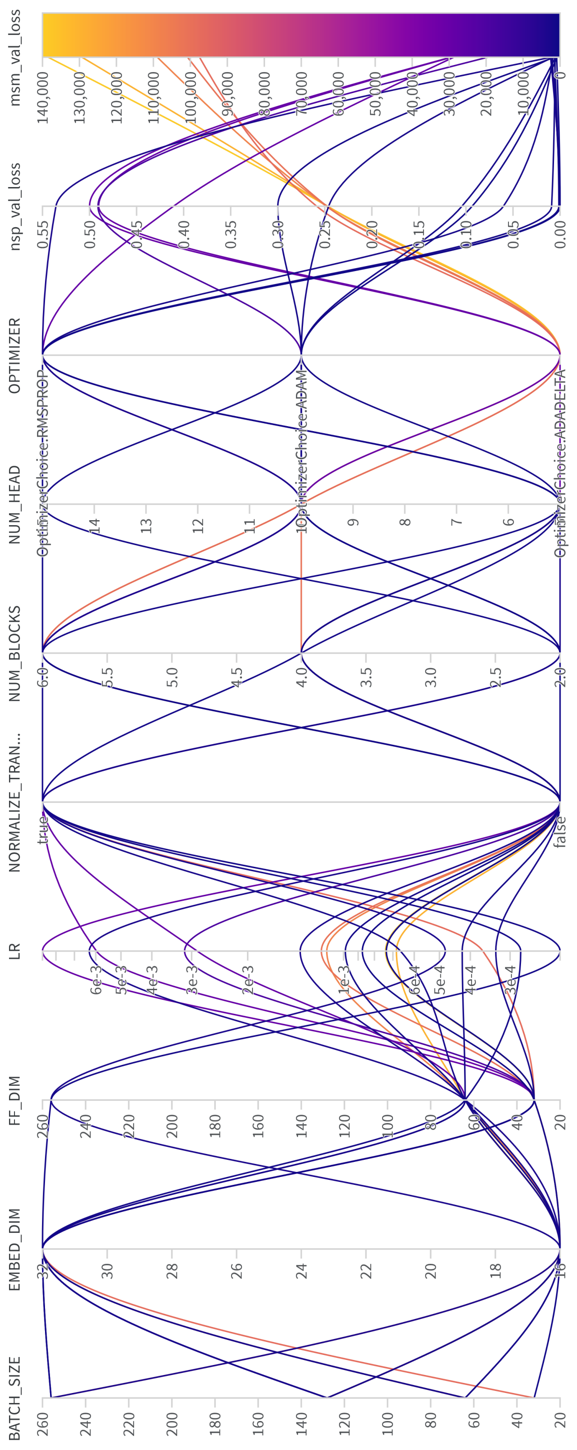

Figure 4.3: A sweep showing how runs using different hyperparameters performed. Based on the synthetic dataset.

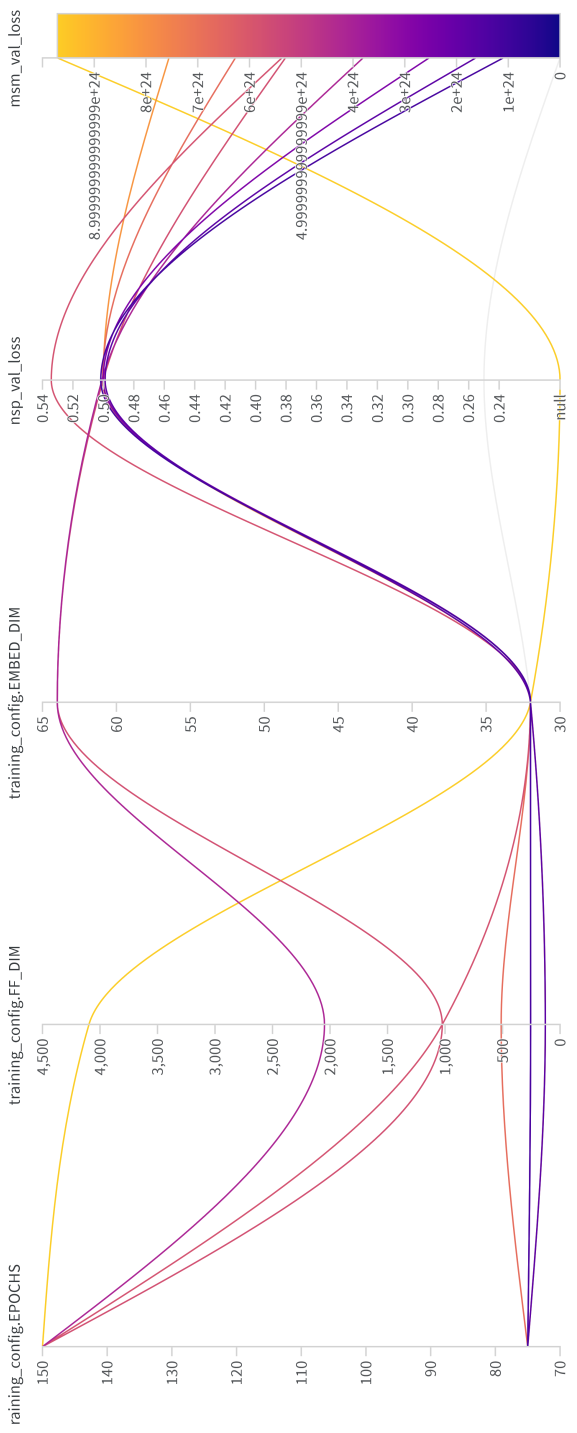

Figure 4.4: A sweep using the FACT dataset with increasingly large layer sizes.

Chapter 5

Missing Results

Unfortunately, training on the FACT dataset did not reduce the observed losses (see section 4.2). The downstream tasks we are interested in are however based on the FACT dataset. Therefore, we cannot report any meaningful results for downstream tasks.

One way to generate results would be to execute the FACT tasks over the synthetic dataset. However, adjusting the benchmark to the model would hardly yield results that are relevant to real world tasks.

Bibliography

[1] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *arXiv:1810.04805*, Oct. 2018.

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *arXiv:1706.03762*, Jun. 2017.

[3] P. Belcak, A. Kastrati, F. Schenker, and R. Wattenhofer, "Fact: Learning governing abstractions behind integer sequences," in *Pending...*, Aug. 2022.

Appendix A
Code

The BERTIS project code is maintained under https://gitlab.ethz.ch/disco-students/fs22/big-code-for-learning