Department of Mathematics and Computer Science
Data Mining Research Group

Introducing Evolutionary Sparsity in the Transformer Model Architecture

Seminar - Final Report

J.G.C. Brouns - 0856180

Supervisor:
dr. Decebal Constantin Mocanu

v1.0

Eindhoven, Januari 2019

Contents

Chapter 1

Introduction

Most of unpreceded new research focuses on attempting to innovate through the creation of new neural network architectures. Innovations in deep learning are being made in a radical rapid-pace, always attempting to disrupt and supersede the established state-of-the art model architectures. But what if there is something fundamentally wrong with the made assumptions on architectural elements and the therefore resulting architectural components that are used in current (state-of-the-art) architectures?

"Just because everyone around you is used to digging holes with their bare hands, it would be bizarre to assume that this is the most efficient way to dig holes."

Recent research [8] has shown that dense (fully connected) layers in Artificial Neural Networks (ANN) are superfluous. They showed that all dense layers in ANNs can be replaced with sparse ones before training using their sparse evolutionary training (SET) procedure, reducing quadratically the number of parameters, with no decrease in accuracy. SET was demonstrated on three popular ANN types (restricted Boltzmann machines, multi-layer perceptrons and convolutional neural networks), on two types of tasks (supervised and unsupervised learning), and on 15 benchmark datasets. SET is a finding that can potentially be very influential, yet unexplored and unutilized by deep-learning enthusiasts. Referring back to the given metaphor:

"It seems that the shovel is invented, but everyone keeps digging with their hands!"

1.1 The New Domain

One of the biggest challenges in Natural Language Processing (NLP) is the shortage of training data needed for the many distinct NLP-tasks. The current go-to method to counteract this shortage of data is the use of pre-trained embeddings; embeddings that are pre-trained on a large corpus (billions of annotated training examples) in a variety of tasks in order to incorporate some general word representation. Subsequently, these pre-trained models can be fine-tuned on a specific NLP task (e.g. classification, sentiment analysis) using a fairly small data-set, which results in a substantial accuracy improvement compared to training a model from scratch [1], [9], [6].

Pre-training of embeddings was usually done using deep recurrent neural networks (RNN), but a new architecture emerged: The Transformer [12]. In contrast to RNNs, the Transformer architecture uses no recurrence, but instead processes all words or symbols in a sequence in parallel while making use of a self-attention mechanism to incorporate context from words farther away. In October 2018, Google AI open-sourced its very promising Bidirectional Encoder Representations from Transformers (BERT) architecture [2], which is heavily based on the transformer architec-

ture. BERT outperforms previous state-of-the-art models in 11 NLP tasks. Moreover, in contrast with other models, BERT does not need complex architectural changes to fine-tune towards specific tasks and is therefore being referred to as *the ImageNet moment of NLP*.

1.2 Research Question

This paper presents a case-study that introduces evolutionary sparsity in the original Transformer model by means of the earlier mentioned SET-procedure. It was investigated whether the fully connected layers in the transformer really are necessary to create an embedding that correctly captures a corpus'contextual information. The adapted sparse architecture was used to train a new model by means of a small corpus. Subsequently, the model was evaluated with a similar English-to-German translation task, as proposed by the original authors of the Transformer model [12]. The main hypothesis can be formulated as follows:

> ***Hypothesis 1 (H1):*** *The adapted Transformer model M' will perform at least as good as the original Transformer model M in a German-to-English translation task.*

First, the SET-procedure and the Transformer model architecture will be elaborated briefly, in order to identify possible caveats and to identify where and how the SET-procedure can be applied. Subsequently, the evaluation framework, experiments and results will be presented. Lastly, a brief conclusion will be given.

1.2.1 Contribution to Science

Although this given evaluation and validation of SET by the original authors is strong, SET could potentially benefit of a case-study applied in other (more complex) deep-learning models and their coherent domains. This could potentially re-enforce SET's potential and provide insight in the technical roadmap, necessary to overcome current implementation challenges and realize large-scale implementation and adaption.

1.2.2 Repository of this project

The codebase for this seminar project can be found in the following public repository:

Introducing Sparsity in the Transformer model (a Keras Implementation):
`https://github.com/jgcbrouns/Introducing-Sparsity-in-the-Transformer`
Author: jgcbrouns

Chapter 2

Background Study

2.1 The Transformer Architecture

The Transformer architecture follows an encoder-decoder structure. The encoder maps a sequence input $X = (x_1, .., x_n)$ to a vector-representation input $Y = (y_1, .., y_n)$. Subsequently, the decoder takes this intermediate representation Y together with the previous representation (auto-regression) and generates an output sequence of symbols $Z(z_1, .., z_n)$. Both the encoder and decoder are composed of a stack of $N = 6$ identical layers. Please refer to Figure 3.1 for a schematical overview of the model architecutre. A brief elaboration on the main architectural components will now follow. Please note that the Transformer architecture is quite complex and that not all components can be discussed in detail. For more elaborate information on the Transformer, please take aid in the original paper of the authors [12].

2.1.1 Attention

Scaled Dot-Product Attention
This function essentially maps vector-sets of queries Q, keys K and values V to an output (see figure 2.2). Input conists of queries and keys of dimension d_k, and values of dimension d_v. Attention over the vector-sets will be calculated by means of a 'Scaled Dot-Product':

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}}V)$$

Multi-head Attention
It was found beneficial by the authors to linearly project Q, V and K, h times with other linear projections (see figure. 2.3). The attention function is then applied in parallel on these individual projected versions which are subsequently concatenated and again projected, resulting in the final values. In contrast with single-head attention, multi-head attention allows the model to jointly attend information from different representation subspaces at different positions.

2.1.2 Positionwise Feed-forward Network

The feed-forward networks consists of two linear transformations with a ReLU activation in between:

$$FFN(x) = max(0, xW_1 + b_1)W_2 + b_2$$

While linear transformations are the same across different positions, they use different parameters from layer to layer. This can also be described as two convolutions with a kernel-size of 1.

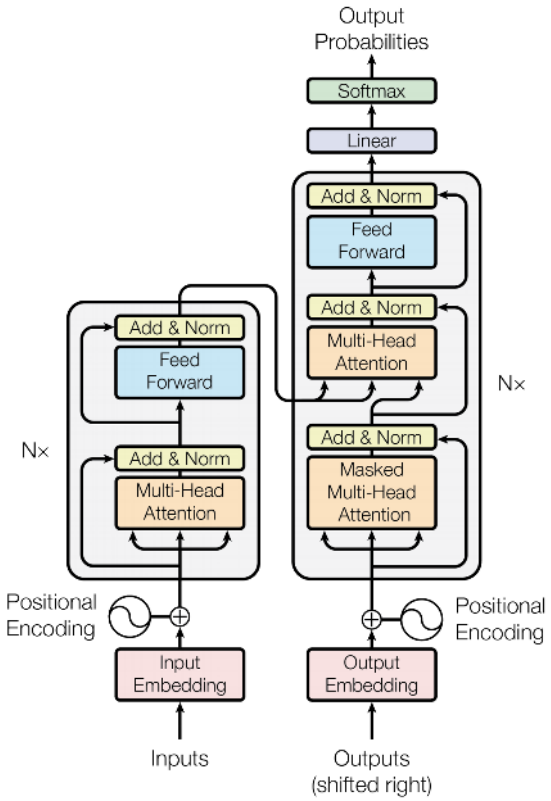


Figure 2.1: The Transformer - Model Architecture

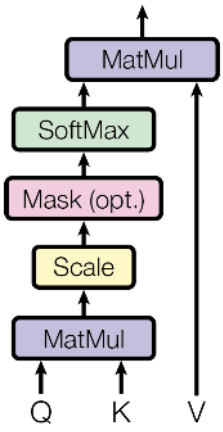


Figure 2.2: The Scaled Dot-Product

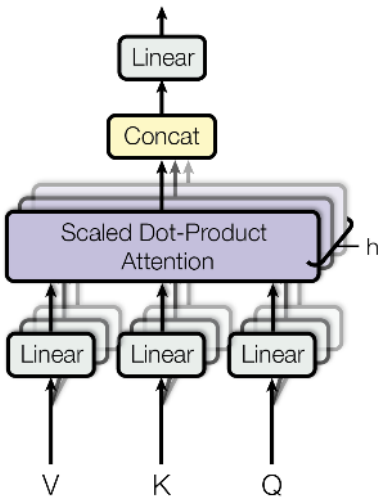


Figure 2.3: Multi-Head Attention

2.2 The Sparse Evolutionary Training Procedure

The SET training procedure is inspired by network properties of biological neural networks. Biological neural networks often are sparse and scale-free, yet their artificially created counterparts, generally use fully connected layers. Before training, SET introduces evolutionary sparsity by replacing the fully connected layers with sparse layers. During training, SET evolves a Erdős-Rónyi topology [4] into a scale-free topology (a network whose degree distribution follows a power law). What follows is a brief elaboration of the SET-procedure:

- **Before training (initialization):**
  - Define parameters:
    - $\epsilon \in \mathbb{R}^+$ - controls the sparsity level
    - ζ - fraction of closest weights to 0 that are removed and added
  - Replace every fully connected layer FC^k in the ANN with a sparse connected layer SC^k, where:
    - SC^k has n^k neurons, collected in a vector $h^k = [h_1^k + h_2^k, ..., h_{n^k}^k]$
    - SC^k has a Erdős-Rónyi topology in which the probability of a connection between neuron h_i^k and h_i^{k-1} is:
$$p(W_{ij}^k) = \frac{\epsilon(n^k + n^{k-1})}{n^k n_{k-1}}$$
- **During training:**
  For each normal training epoch e, the standard training procedure and weights update is being followed. Subsequently, for every sparse connected layer SC^k:
  (1) A fraction ζ of the weights closest to zero are removed.
  (2) *If e is not the last training epoch*, new weights (connections) are added in the same amount as removed previously under (1).

Chapter 3

Approach

3.1 Applying SET

3.1.1 Where to apply?

The SET procedure can be applied to any fully-connected layer. It was chosen to attempt the introduction of evolutionary sparsity into the attention mechanism, rather than the feed-forward network of the transformer, since the feed-forward network uses dense-layers that are implemented as convolutional layers. Moreover, since the attention-mechanism is 'the heart' of the Transformer, a good outcome will be more meaningful to the case-study of SET; the layers that encode the contextual meaning of every input token are now directly targeted by the sparsity introduction.

Recall from section 2 that in the Multi-head Attention component of the Transformer, the queries Q, keys K and values V are linearly projected h times with other linear projections. Dense layers are used to create an embedding for every input-token that is subsequently used to generate Q, V and K. The adaption will be performed on these dense layers: initially, SET converts these dense layers to sparse layers with a Erdős-Rónyi topology. In the training-phase, a fraction ζ of weights closest to zero will be removed and subsequently be randomly added in the same amount. The functioning of the the attention-mechanism should be similar when introducing evolutionary sparsity, since the attention-mechanism selects attention based on the highest corresponding Key-Query-Value tuple through scalar-products and Softmax. In contrast with this, SET is only interested in the lowest weights in these layers.

3.1.2 Mask Technique

According to the original authors of SET, as for now, all deep learning implementations are based on very well-optimized dense matrix multiplications on GPUs, while sparse matrix multiplications are extremely limited in performance. The original authors of SET performed proof-of-concept experiments in order to simulate and validate the functioning of SET. This involved using fully connected layers and a mask over weights in order to simulate sparsity. The same masking-technique will be used in this case-study, since truely (and efficient) sparse-matrix multiplications are still not properly implemented on GPUs.

3.2 Implementation details

3.2.1 Used Codebases

Parts of the following codebases was used:

- **Proof of concept Keras implementation of SET:**
  Author: Dcmocanu
  `https://github.com/dcmocanu/sparse-evolutionary-artificial-neural-networks`
- **A Keras+TensorFlow Implementation of the Transformer:**
  Authors: Lsdefine, JulesGM
  `https://github.com/Lsdefine/attention-is-all-you-need-keras`
- **Attention is all you need - A Pytorch implementation:**
  Author: Jadore801120
  `https://github.com/jadore801120/attention-is-all-you-need-pytorch`

3.2.2 Custom Transfer Method

After every trainings-epoch, the trained model unfortunately needs to be re-initialized. This is due to how the masking of weights is implemented in the proof-of-concept code. Masking weights in Keras is done via the *kernel_constraint* parameter upon creating a new layer. Once the layer is created, *kernel_constraint* cannot be set anymore. Since, SET needs to reapply masking of the weights after every trainings-epoch, the original authors of SET were forced to reinitialize the model after every epoch while manually applying the rewired weights and weighs-mask of the previous model to this new model upon creation of the sparse layers.

The original authors of SET used small models with only few sparse layers and could therefore hard-code the creation and re-initialization of the model. Because the Transformer architecture is very extensive and complex, a custom model transfer procedure was designed that dynamically merges the newly initialized model (that contains the masked weights) with the old model (that contains the weights of the remaining non-sparse layers).

The role of this custom transfer method in the implementation of SET is visually depicted in figure 3.1 and can be described more formally as follows:

1. After each epoch: for every sparse layer $s \in S$, the corresponding weights-matrix w gets extracted via the Keras API using a previously assigned unique identifier for that particular layer. Let W be the set of all weights.
2. $\forall w \in W$: weight matrix w is rewired (SET algorithm).
3. A new empty Transformer model *newModel* is constructed where, upon initialization via the Keras *weights* parameter, for each sparse layer $s \in S$, its corresponding adjusted weights are set as obtained in the previous step using the SET-procedure. The weights-mask is set via the Keras *kernel_constraint* parameter.
4. The *newModel* will be merged with the *oldModel* by means of a custom-designed model-transfer procedure. Pseudocode can be found below: Algorithm 1 Custom Model Transfer Procedure

Because of the introduction of this custom model transfer procedure, we need to validate that this procedure does not introduce any other dependent factors that could offset our experiment-results. In other words, one cannot directly compare the original validated Transformer model implementation with a variant of the Transformer model that features both a 'sparsity adaption' and a 'function to transfer models'; multiple independent factors can be considered and thus it would be hard to yield a valid conclusion.

Algorithm 1 Custom Model Transfer Procedure - Pseudocode

```
S ← list of unique identifiers for every sparse layer
newModel ← newly initialized Keras Transformer model
oldModel ← old Keras Transformer model obtained from previous epoch
index ← 0
for layer in newModel.layers do
    if layer.name not in S then
        weightsFromLastEpoch = oldModel.layers[index].get_weights()
         layer.set_weights(weightsFromLastEpoch)
        index++
    endif
endfor
```

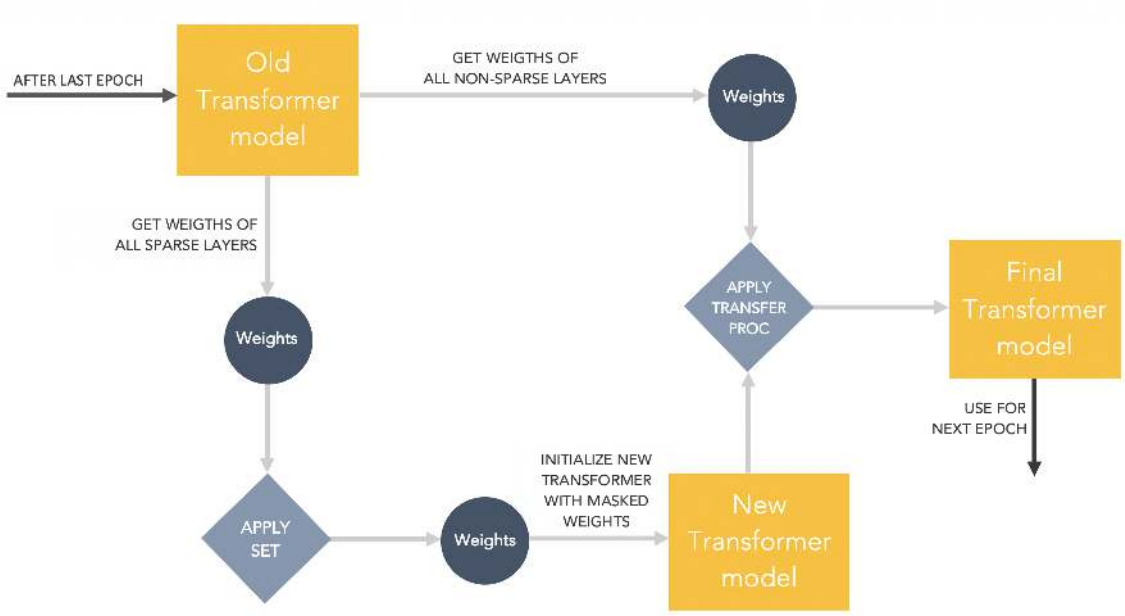


Figure 3.1: Schematic view of the custom model transfer procedure

3.3 Evaluation Method

The WMT'16 Multimodal Translation task [3] (Multi30k) will be used to validate the model's accuracy. This is a shared task aimed at the generation of image descriptions in a target language, given an image and one or more descriptions in a different (source) language. Evaluation will be performed against the German human translation on the test set using standard MT evaluation metrics, with METEOR[7] as the primary metric.

3.3.1 Dataset properties

The training data consists of 29,000 samples, each containing an English source sentence and its German translation (created by a human). The test data consists of 1,000 samples. An example sentence-pair sample is as follows:

Two men are playing music on a bench . Zwei Männer auf einer Bank musizieren .

Chapter 4

Experiments and Results

Google's Cloud Compute Engine was used to deploy 3 instances of the same virtual machine to ensure parallel training. Each VM trained one of the three Transformer variants on a NVIDIA Tesla P100 GPU and 8x vCPU 2.8GHz. The three versions of the transformer model, as discussed in section 3, were trained with the following noticeable parameters:

Variable	Value	Original Value*	Additional Description
batch_size	64	64	Batch size
dropout	0.1	0.1	Dropout
max_epoches	30	n.a	The number of epochs that were trained
d_model	512	512	Dimension of embedding-layers layers and residual connections
d_inner	**512**	**2048**	Dimension of inner layers
d_k	64	64	Dimensions of keys and queries
d_v	64	64	Dimensions of values
layers	**2**	**6**	# of identical stacked layers the encoder and decoder
n_head	8	8	The amount of stacked encoder and decoders
ζ	0.3	0.3	The fraction of the weights closest to zero that are removed
ϵ	20	20	Factor that determines the sparsity level

***** *Refers to the values as set in the paper of the original authors. Due to limited computational resources, lower values had to be used to ensure that the model fits in the GPU-memory.*

The original unadapted Transformer took 4 hours and 11 minutes to train. The Transformer with transfer function took 6 hours and 20 minutes to train and the Sparse Transformer with transfer function took 8 hours and 7 minutes to train. Results of the validation accuracy can be observed in figure 4.1. Results of the validation loss can be observed in figure 4.2. See Appendix A for a table of validation accuracies per epoch for all the three Transformer variants.

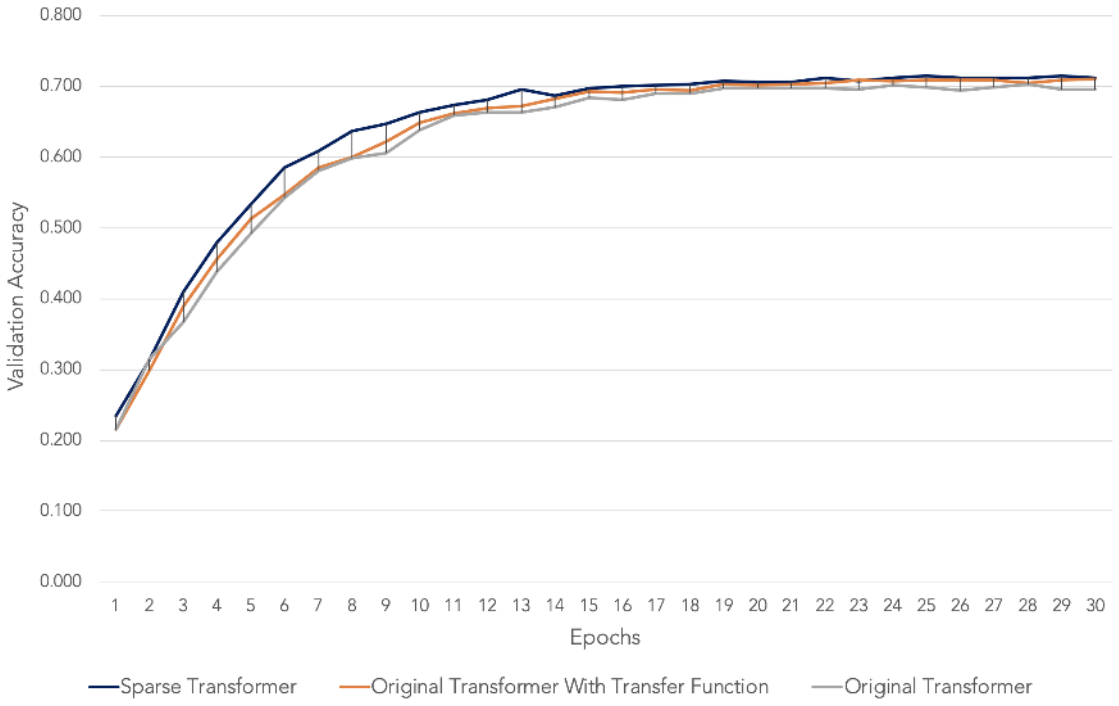
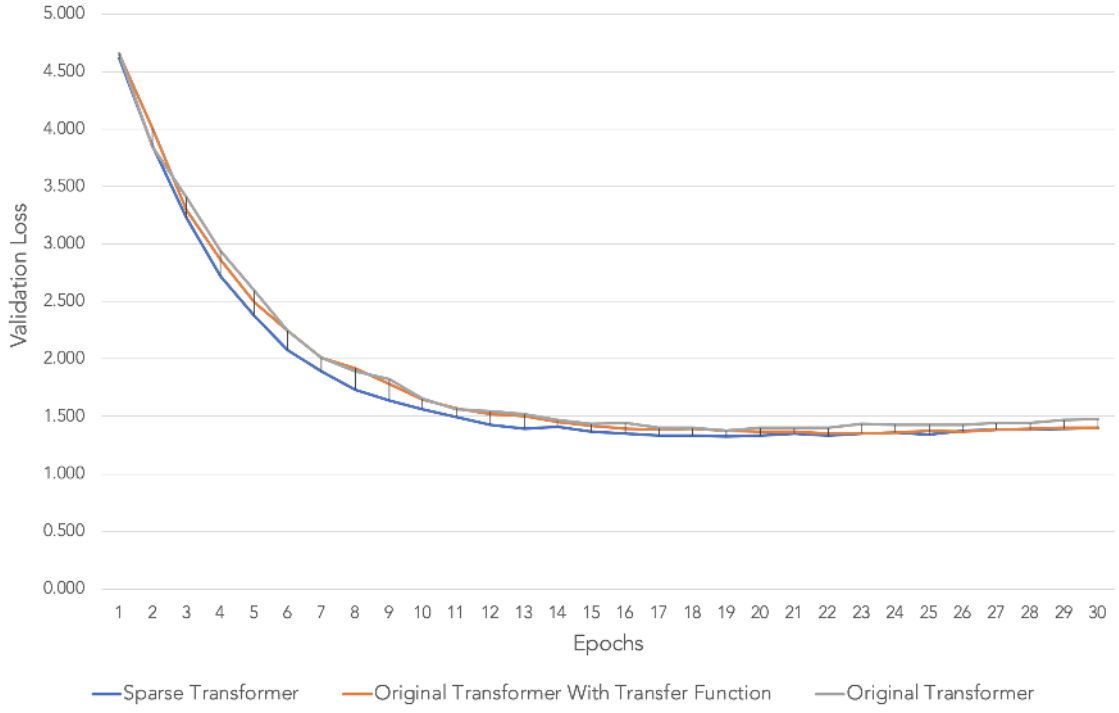


Figure 4.1: Sparse Transformer variants - validation accuracy



.png

Figure 4.2: Sparse Transformer variants - validation loss

Chapter 5

Conclusions

The original transformer (both with and without the transfer function) can be observed to have similar accuracy and loss functions. Hence a valid comparison of the results of the sparse Transformer and the original transformer is possible, since it is to be assumed that the custom model transfer procedure does not introduce additional factors that could contribute to different experiment outcomes.

As can be observed in figure 4.1, the sparse Transformer seems to outperform both the original Transformers, while having two orders of magnitude fewer parameters (for the selected fully-connected layers). Moreover, the sparse Transformer manages to obtain a higher starting accuracy that both original variants never seem to catch up with. Having an initial higher accuracy for the sparse model is notably different with was observed by the original authors of SET; for them, the sparse variants usually start with lower accuracies and at some point start to slightly outperform (and continue to outperform) the non-sparse variants.

In the final epochs, the models converge to similar accuracies; in the last 5 epochs on average 70.8% for the original Transformer (with the transfer function) and 71.2% accuracy for the sparse Transformer. To refer back to the hypothesis H1 that was defined in section 1, it can be concluded that H1 does indeed hold, that is: the adapted Transformer model M' indeed performs at least as good as the original Transformer model M in a German-to-English translation task.

5.0.1 Future work

Due to the limited extent of this seminar and the therefore short period of time to implement and experiment with different factors (such as model parameters), more research and experimentation is required to draw final conclusions on the degree of impact that sparsity can have on the Transformer model. In this sense, experiments could be conducted with other parameter-settings for the dimension of the inner layers, the amount of layers per encoder and decoder, different values of ζ and ϵ etc. Besides experimenting with different model parameters, multiple runs of the same model (with the same parameters) could be conducted in order to observe the average results of a model.

The Transformer model architecture forms the base of many recent developments that introduces new state-of-the-art architectures among which (the earlier mentioned) Google AI's BERT, the OpenAI Transformer [11] and ELMO [10]. Adapting these state-of-the-art models with the same sparse Transformer could be an interesting direction for future work. An intermediary step could be to apply block-sparsity to the Transformer; a promising technique that has seen recent developments in terms of overcoming software engineering related hurdles. [5].

Introducing evolutionary sparsity in fully connected layers has proven great potential, yet software-technical implementation issues (e.g. truly sparse matrix multiplications) are a current unavoidable obstacle to move from 'proof-of-concept' implementations to 'production usable' implementations. Showing SET's value to the deep-learning community by applying it to new popular state-of-the-art model architectures could speed things up and create the necessary traction for overcoming SET's practical boundaries.

Bibliography

[1] Andrew M. Dai and Quoc V. Le. Semi-supervised sequence learning, 2015. 1

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018. 1

[3] D. Elliott, S. Frank, K. Sima'an, and L. Specia. Multi30k: Multilingual english-german image descriptions. pages 70–74, 2016. 9

[4] P. Erdös and A. Rényi. On random graphs i. *Publicationes Mathematicae Debrecen*, 6:290, 1959. 5

[5] Scott Gray, Alec Radford, and Diederik P. Kingma. Gpu kernels for block-sparse weights. 2017. 12

[6] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification, 2018. 1

[7] Alon Lavie and Abhaya Agarwal. Meteor: An automatic metric for mt evaluation with high levels of correlation with human judgments. pages 228–231, 07 2007. 9

[8] Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H. Nguyen, Madeleine Gibescu, and Antonio Liotta. Evolutionary training of sparse artificial neural networks: A network science perspective. *CoRR*, abs/1707.04780, 2017. 1

[9] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations, 2018. 1

[10] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke S. Zettlemoyer. Deep contextualized word representations. In *NAACL-HLT*, 2018. 12

[11] Alec Radford. Improving language understanding by generative pre-training. 2018. 12

[12] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017. 1, 2, 3

Appendix A

Validation Accuracies

Validation accuracies per epoch for all the three Transformer variants. Delta (Δ) refers to the absolute difference between the original with transfer procedure and the original without the transfer procedure. In bold are Deltas that are equal or greater than 0.01

Sparse	**Original With Transfer**	**Original**	Δ
0.235	0.215	0.215	0.000
0.313	0.298	0.315	**0.017**
0.410	0.389	0.367	**0.022**
0.480	0.456	0.438	**0.018**
0.533	0.513	0.493	**0.020**
0.584	0.547	0.543	0.004
0.609	0.585	0.580	0.005
0.636	0.600	0.599	0.002
0.647	0.622	0.605	**0.017**
0.664	0.648	0.638	0.011
0.673	0.662	0.658	0.003
0.681	0.669	0.663	0.006
0.695	0.672	0.663	0.009
0.686	0.682	0.671	**0.011**
0.697	0.693	0.684	0.009
0.700	0.691	0.681	**0.010**
0.701	0.695	0.689	0.006
0.703	0.694	0.690	0.004
0.707	0.702	0.697	0.005
0.705	0.701	0.697	0.004
0.706	0.703	0.696	0.007
0.711	0.704	0.697	0.006
0.708	0.709	0.696	**0.013**
0.711	0.708	0.701	0.007
0.715	0.709	0.698	**0.011**
0.712	0.709	0.694	**0.014**
0.711	0.708	0.699	0.009
0.712	0.704	0.702	0.002
0.715	0.709	0.696	**0.013**
0.711	0.709	0.696	**0.014**