UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Tècnica Superior d'Enginyeria
de Telecomunicació de Barcelona

Disentangling neural network structure from the weights space

Master Thesis
submitted to the Faculty of the
Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona
Universitat Politècnica de Catalunya
by

Pol Caselles Rico

In partial fulfillment
of the requirements for the master in
Advanced Telecommunication Technologies

Advisors: Konstantin Schürholt (ICS-HSG), Damian Borth (ICS-HSG), Xavier Giró-i-Nieto (UPC)
Barcelona, Date 20/01/2021

Contents

List of Figures

List of Tables

Abstract

Deep Neural Networks have been used to tackle a wide variety of tasks achieving great performance. However, there is still a lack of knowledge of how the training of these models converge and how weights relate to their properties. In this thesis we investigate the structure of the weight space and try to disentangle its properties. Attention mechanisms are introduced to capture relations among neurons' weights that help in weight reconstruction, hyper-parameter classification and accuracy prediction. Our approach further has the potential to work with variable input size allowing different network width, depth or even architecture types.

Acknowledgments

I would like to thank my tutor, Xavier Giro-i-Nieto, for granting me the opportunity to do my master's internship at ICS-HSG in Sant Gallen, Switzerland. I want to acknowledge Konstantin Schürholt for his huge help during the development of the thesis. I would also thank to Damian Borth for the kind hospitality I received at all times and for having accepted me in the ICS-HSG lab.

1 Introduction

In recent years, Deep Neural Networks (DNN) have been used to tackle a wide variety of tasks, achieving incredible results. Looking at the amount of papers where these methodologies have been used, it has not stopped growing. DNN have been applied successfully to a wide different domains such as language, 3D reconstruction and analysis, image and video classification, creation and understanding, speech analysis and synthesis and so much more. DNN are increasingly employed to real-world use-cases and have become the state of the art of a lot of approaches. However, their structures and operations are still poorly understood. There is a lot of information and intuition but more research into this direction is needed to fully describe and define them in depth.

Mathematical fundamentals and a lot of techniques are know to train DNN, but due to the high dimension of the space in which these technologies work (in some cases up to 100B trainable parameters) it is difficult to explore it with the computation power that there is nowadays. There are still countless open questions about the solutions reached and how and why they really work: why do some networks generalize, while others do not? Why do some networks learn a bias, while others do not? Looking directly at the weight space of the networks, are different networks trained on the same task learning the same features or are different solutions? How unique are the solutions in the weight space? Can we map or group these unique solutions to some latent space that we can then take advantage to be applied into any other downstream task? How do different architectures trained on the same task relate to each other? Do different domains and tasks guide how the model ends up to different weights? Nowadays it is difficult to answer all these questions without uncertainty, and more explanations are still needed.

Gaining insight into the model weight structure may change the way new models are trained. Having a better understanding can lead us to potentially develop new strategies during the model design processes. This knowledge can be also used in the field of intellectual property to define metrics to certificate and patent them. Versioning and diagnostics can also be potentiated by the influence of models' knowledge providing explanations of their behaviour and how the appropriate structure should be. Having full knowledge of models' weights trajectories and how they relate to accuracy we could save time and computing power consumption. This knowledge can be used to predict which models are or are not going to flourish. Being able to compress model weights can also be useful in weight deployment transmission and storage savings. Another interesting topic that can take advantage in the field of the DNN is called interpretability. Fully understanding them

can lead us to discover new properties of the neural networks. Nowadays, in order to fine tune our models we usually retrain them by applying small changes in a wide variety of hyper-parameters. There is a field called meta-learning that studies the evolution of the models during training in order to learn how to learn. The explainable artificial intelligence field is concerned with explaining how models work beyond the theoretical part. It attempts to answer questions of a moral nature or of interpretation of their functioning and to explain the reason why some decisions are made or others not. Given that the field of artificial intelligence is the mathematics that defines the criteria of decision, investigating the question can help to understand the solutions obtained.

The data for this thesis are the models trained on different datasets, tasks, architectures, properties of training and the randomness of the iterative procedures (split of the dataset, batch composition, dropout and so on). Our goal is to generate model-level embeddings that are rich enough to define the entire architecture in time (including current state in training, e.g. epoch), dataset on which the model was trained on and properties defined during training as well as the structure of the architecture. Achieving this goal would mean that it will be able to understand the underlying structure. Due to the dimensionality of the solution space, finding analytical solutions is generally infeasible. Therefore, we have opted for the use of deep learning methodologies as they are designed to work with large volumes of data.

The main idea is to focus on a subset of models with a limited number of tunable parameters and try to disentangle the structure from the weight perspective. There is a wide range of architectures and possibilities during training. For this reason we have worked on vanilla convolutional and feed forward networks, of model sizes between 100 to 5000 weights for each sample.

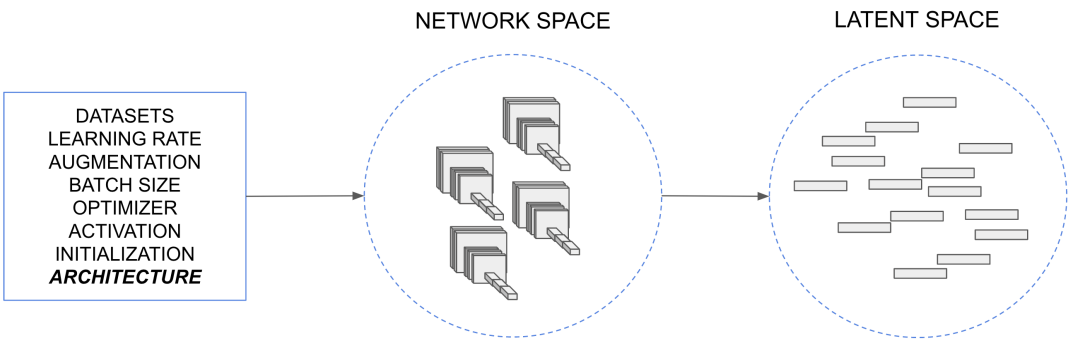


Figure 1: The main setup: define a bidirectional mapping between model architecture and its properties and a model-level embeddings that are rich enough to define it.

Recently, different datasets of neural networks trained on vision tasks datasets MNIST, MNIST-FASHION, SVHN, CIFAR10 for image classification tasks have been published. They are composed of models of different sizes and varying hyper-parameters for different trainings. In this domain, a sample of the dataset is the model's weights for a given hyper-parameters and checkpoint properties. Unterthiner et al.[1] released a dataset composed of a fixed architecture trained for image classification on the aforementioned datasets. The authors investigated the accuracy prediction from the weights and the statistics transformation space.

In our work we present an study on representation learning of neural network models from the weights perspective. We first introduce an analysis of the Google dataset. The investigation reveals that the characteristics of the networks have been impregnated in its weights. In order to exploit its weights' relationships we propose a novel attention-based approach to learn representation of weights space of neural networks. We demonstrated that this approach outperforms the DNN baseline as presented in Unterthiner et al.[1] and it is competitive against GBM. Same approach has been used to extend it to learning hyper-parameters as well as self supervised weights' reconstruction.

Attention mechanism has shown its capacity to work well in recurrent state approaches [4], [5] outperforming in some tasks well known approaches such as LSTM or GRU and the issues with exploding and vanishing gradients are solved. Besides, by looking at the attention score maps, it provides means for interpretability that can be useful in order to disentangle how the weights of a model interact. The transformer encoder is a well known architecture that implements attention mechanism, therefore it has been decided to incorporate it in our model as a core blocks to capture relations.

1.1 Gantt diagram

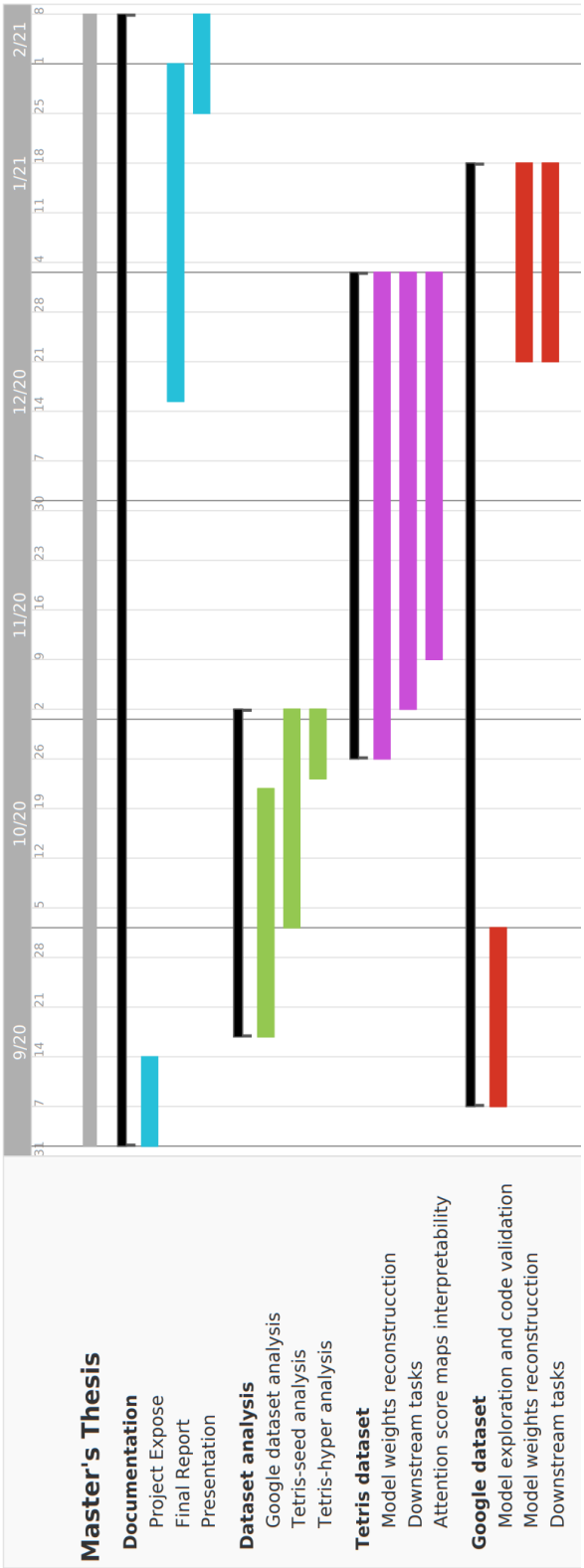


Figure 2: Gantt diagram

2 State of the art and related work

Currently there are a wide variety of methodologies and algorithms for a wide range of applications. In recent years, technologies based on deep neural networks have grown significantly. Systems based on DNN achieve very good results in many areas. However, they are the most difficult systems to understand in comparison with other mathematical methodologies such as decision trees [19]. For this reason, it is important to understand why deep neural networks obtain one outcome or another. It is also interesting which procedure the weights follow to reach a good solution and which characteristics they acquire. In this line of research, in this thesis we have focused on the representation learning from weights. Currently there are few publications concerning the prediction of accuracy and hyper-parameters using only the information of the weights that compose the models. Furthermore, we have opted for attention-based architectures for data processing and the use of linear dimensionality reduction systems and manifold learning.

2.1 Representation learning in NN weight space

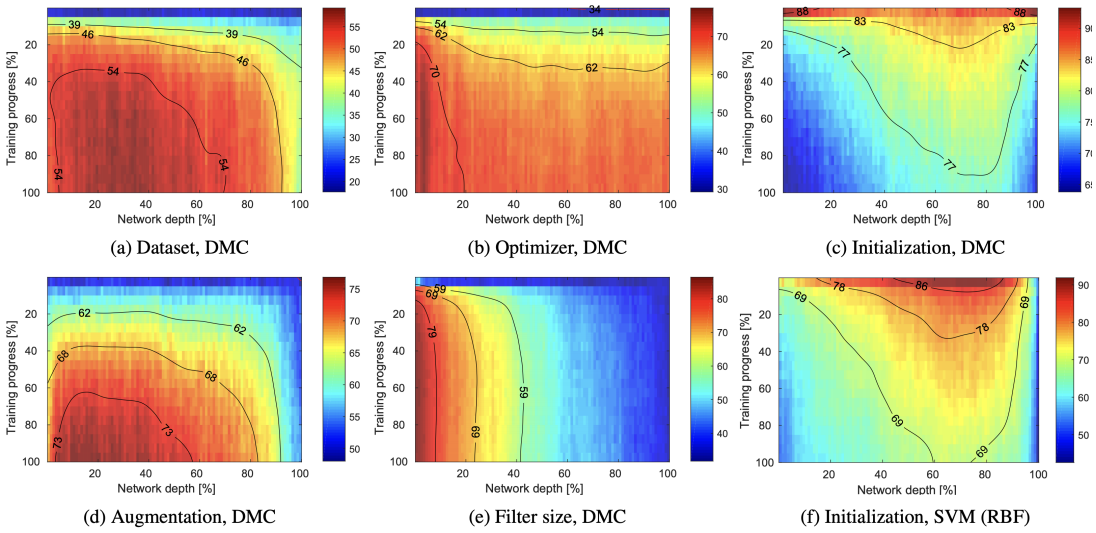
There have been published works [2], [1], [3] that tackle directly the field of neural networks representation from the weights' perspective. Unterthiner et al.[2] published a full dataset of neural networks trained on four well known vision datasets on classification tasks. They published two splits of the datased called C_fixed (in this case the architecture used is fixed around 90K parameters) and C_main (the architecture used also varies as another hyper-parameter, up to 300K parameters). In both datasets the authors trained the models during 86 epochs and they provide checkpoints at epochs 0, 1, 2, 3, 4, 20, 40, 60, 80, 86.

In their work Eilertsen et al. [2] predict models' accuracy with a linear regression predictor. They showed how important is each hyper-parameter to achieve good test predictions. Initialization was one of the most important parameters followed by the size of the convolution filters as well as the fully connected layers size. In order to apply different predictors their input vector of the models was defined as a vectorized form of all the trainable parameters of the network.

Eight basic statistics were computed : mean, variance, skewness (third standardized moment), and five-number summary (1, 25, 50, 75 and 99 percentiles) over the weights and their gradients. This process was applied for each layer of the network. SVM and SVM with Radial Basis Kernel as well as deep meta-classifers (DMC, applied directly on the

model's weights) where used in order to predict the hyper-parameters in each training. Results showed that this task is possible by looking at the weight space. They also proposed to apply DMC in chunks of 5000 weights (5% of the total number of the model's weights) and it had been able to predict them surprisingly well. This means that even with a little amount of weights it keeps information of the entire model and how it was trained.

Due to the fact that DMC can be applied in different positions along the vectorized weights of the model as well as there are model weights' checkpoints during training (time axis), the authors showed heatmaps (Figure 3) of how good the predictions can be for each of the hyperparameters depending on training time and model's weights depth. For example Figure 3(c) shows that in order to predict which initialization procedure was used, results show that best prediction accuracy is achieved when looking at the beginning of the training checkpoints.



(a) Dataset, DMC (b) Optimizer, DMC (c) Initialization, DMC

(d) Augmentation, DMC (e) Filter size, DMC (f) Initialization, SVM (RBF)

Figure 3: Taken from [2]. Meta-classification performance maps, visualizing the average test accuracy at different model depths and training progress steps. The y-axis illustrate training progress, from initialization (0%) to converged model (100%), while x-axis is the position of the weight vector where evaluation has been made, from the first weights of the convolutional layers (0%) to the last weights of the fully connected layers (100%). Since the results are evaluated over models with different architectures, it is not possible to draw separating lines between individual layers. (c) and (f) show how similar trends are found by DMCs and SVMs. Note the different colormap ranges (e.g. higher lowest accuracy for initialization).

Unterthiner et al.[1] generated a dataset of models trained on the same well known datasets (Google dataset) as described in Eilertsen et al. [2]. However, in this case a fixed network of size of 4970 trainable parameters each were used, which is considerable lower in size in the dimensional space.

In order to predict the test accuracy of a given model, Unterthiner et al.[1] have compared three approaches: gradient boosting machine using regression trees (GBM), logit-linear model (Log-linear) and fully-connected deep neural network (DNN). The authors showed a wide research on how to apply these predictors to this domain and best results are achieved when computing basic statistics (as done in [2], but only on the weights and layer-wise) and applying the aforementioned predictors on it. GBM demonstrate its capacity to capture mean and almost all variance on models' accuracy test predictions achieving between 0.986 R^2 score (models trained on SVHN-GS) and 0.993 R^2 score (models trained on MNIST).

The authors further show that their accuracy predictors generalize from MNIST to other datasets, e.g. CIFAR10. Results show good adaption on domain shift tasks. Training models on different datasets in the same task share some properties that are able to be captured in the accuracy prediction task. It can be concluded then that at least, models that are designed for one kind of task are related somehow, which let us to speculate about the potential to be exploited for other tasks. This assumption is not new. Other approaches in the field of transfer learning relay on this idea.

Both works demonstrated that it is possible to predict the performance of deep neural network using its weights, at least under certain circumstances. They also show that the use of simple statistics can be very helpful in the hyper-parameter and accuracy prediction. However, we should be aware that as another work pinpointed [3], this could also be due to the collapsed weight's matrix rank, where few eigenvalues retain almost all variance. This could be an explanation of why it is straightforward to predict accuracy from the weights.

2.2 Attention architectures

In different areas the sequence to sequence structure (seq2seq) was used for a variety of fields. However, it was mainly used in the field of natural language processing (NLP). Architectures based on LSTM and more recently GRU [15] have been used, but the main problem is the inability to maintain long term information. Attention mechanisms arrive to solve this problem being able to access to all the input sequence.

In recent years one of the most widely used architectures in this area has been the well-known architecture called Transformer [6]. Although at first it was developed for language tasks, it has expanded to many other fields. Its good functioning has led to many adaptations with promising results and has shown that it performs well. Different works use as a main module the transformer encoder block as a bidirectional encoder. In the different implementations, in the field of the NLP the expected input is a sequence of words. Without going into detail, word embeddings are generated from each of the words that form the fixed-size embedding variable length sequence, and relative position information is added. This process can be done without context information such as word2vec [12] or Glove [13] (it means that the same word, which can have different meanings, will generate the same representation regardless of the context) or they can be learnt representations such as ELMo [14] or BERT [8] (in this case the generated embedding depends on the context). Attention-based modules are applied by capturing relational information between the different embeddings for tasks such as question answering (SQuAD), natural language inference (MNLI), next sentence prediction (NSP) and next word prediction (NWP), text classification, neural machine translation (NMT), named entity-linking (NEL) and others.

To further develop more related tasks, there have been using different modification to the initial structure. There have introduced special tokens into the sentence such as separator, verb tense, unknown word, masked word, or global classification. The global classification token is used to make the model identify that the output of this token in the transformer encoder is used to generate a global sentence representation embedding. In the paper *Bidirectional Encoder Representations from Transformers* (BERT[8]), they added this last token at the beginning of the sentence for classification tasks. Expanding on this idea, in Vision Transformer (ViT) [7], the authors have applied all these concepts to the domain of image classification. They divided the images into different smaller squares, which through a non-linear transformation they build a fixed size embedding and with the help of the classification token the model has been built. In this thesis we use the knowledge developed for NLP tasks adapted to the domain of weights sapce. We introduce

the concept of tokenising model weights to generate a sequence of embeddings. We also incorporate global tokens at the beginning to generate global representations.

2.3 Visualization and dimensionality reduction

In domains where each of the data available has a large number of features, it is usually necessary to perform one or more methods to reduce their size. This process is used in order to eliminate information that is not relevant or does not provide anything new. In many occasions it is necessary to reduce the size of the data to be able to train models or also to be able to visualize the data in 2D representations.

The main techniques of dimensionality reduction can be summarised in the methods based on feature selection, those based on matrix factorization, those based on manifold learning or the auto-encoder methods. Moreover, these can be classified in two main groups, those that make a linear reduction and those that are non-linear.

In order to be able to represent large dimensional data, the principal component analysis (PCA) method and the non-linear tSNE [16] and the uniform manifold approximation and projection for dimension reduction (UMAP) [17] methods have become very popular. In general, PCA is used for fast reduction and elimination of information that is not relevant. In contrast, tSNE and PCA are used for large compression up to two or three dimensions. They are mainly used to visualise large dimensional data.

tSNE methodology is based on t-distributed stochastic neighbour embedding with non-linear scaling to represent changes at different levels, it preserves local structure in the data. On the other hand, UMAP claims to preserve both local and most of the global structure in the data. Unlike PCA which is deterministic, these technique does not expect the relationship to be linear.

Both algorithms are highly stochastic and very much dependent on choice of hyper-parameters (t-SNE more than UMAP) and can yield very different results in different runs, so the plots might obfuscate an information in the data that a subsequent run might reveal.

3 Methodology

In this thesis I have focused on the representation learning from the weights space. I have tackled different tasks such as accuracy prediction, hyper-parameter classification, weights' compression and noise robustness using in all cases attention mechanisms. In this chapter, we present the auto-encoder architecture attention auto-encoder I (AttnAE I) and its evolution (AttnAE II) in order to be able to deal with variable input sizes. I also present the insights of the Unterthiner et al.[1] Google's dataset and ours tetris-seed and tetri-hyper datasets. I finish off with some definitions of the metrics and data augmentation techniques that have been used.

A model based on neural networks uses a series of mathematical operations to modify its weights during training. This process establishes structure in the weight space. Due to the dimensionality of the environment, the space is going to be sparse. Therefore, it may be difficult to establish relationships between models and to understand the interactions. Attention-based architectures have the potential to learn the relationships that exist among weights. Multi-head attention and positional embeddings both provide information about the relationship between different embeddings. Besides, the transformer encoder architecture does not suffer from long dependency issues and it is suitable for domains with a large amount of training data. Data augmentation techniques such as applying Gaussian noise, erasing input sequences and applying permutations (they do not modify the functioning of the model) have been studied.

3.1 Attention auto-encoder I (AttnAE I)

Attention mechanisms have been used in a wide variety of tasks. In recent years the well-known Transformer architecture has been used and adapted to different fields due to its great capacity to work well. For this reason we decided to use the encoder of the complete model as the main block of our architecture to compute the attention. The main block does not reduce the dimensionality, and it also needs all input embeddings Z_i to be the same size. For this reason, the difficulty to use it lies in the methodology used to adapt or translate the model weight information into a series of embeddings.

In previous works [1] [2], the input was defined as a vector of a concatenation of all the weights of the model. In our case, a criteria has been defined to separate the weights to generate a sequence of embeddings that represents our model. Depending on how the weights have been transformed into these embeddings, they are described as follows:

- **Neuron**: Each weight is passed through a Multi Layer Perceptron (MLP) to obtain an embedding of size d_{model}.
- **NeuronGroup**: All the wheights that compose each neuron are mapped into an embedding of size d_{model}. In case the model is a convolutional neural network (CNN) then all the weights of each kernel are flattened and are used as a group. Biases are included in its corresponding group/kernel weights added at the end.
- **Layer**: All the weights of all neuron for each layer are mapped into an embedding of size d_{model}. In case the model is a convolutional neural network (CNN), then all the weights of all kernels for each layer are used as a group.

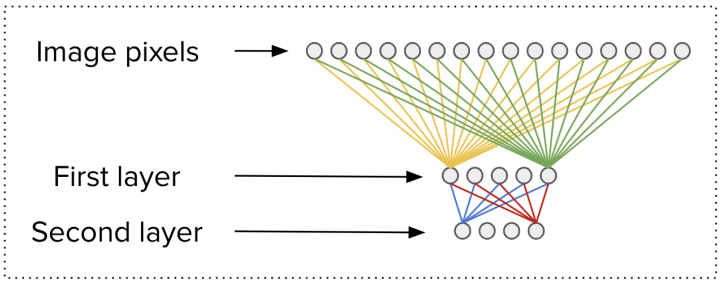


Figure 4: Model architecture used in the tetris dataset. The weights of this model at different checkpoints is considered one sample of the tetris-seed dataset. The colours shown refer to the NeuronGroup's embedding system (an embedding will be generated from all the weights corresponding to each of the neurons). Colors refer to the embeddings in figure 5.

Each group of weights for a model is encoded with a different MLP to a fixed size d_{model} (see Figure 5). After the transformer encoder, all the d_{model} embeddings are concatenated. The function $f_{seq2neck}$ is defined as a MLP of one layer from the size $length\_sequence * d\_model$ to the size of the *bottleneck*. With this approach we have to be aware of how it scales as the length of the sequence and size of the embeddings increases. The reason is that with this structure the dimension of all the embeddings must be reduced to the size of the bottleneck. For large models this could be an issue becasue the dimensionality of the function $f_{seq2neck}$ should be increased. What it would be doing then is using a usual auto-ecoder MLP, where attention mechanism has been used to generate such a vector. In short, attention would be simply added before performing a regular dimensionality reduction. During all the N blocks applied in the transformer encoder it does neither reduce the dimensionality nor telling the model to summarize the key features of each embedding. The best results were obtained by forcing the bottleneck to have values between -1 and 1 by applying tanh function on it.

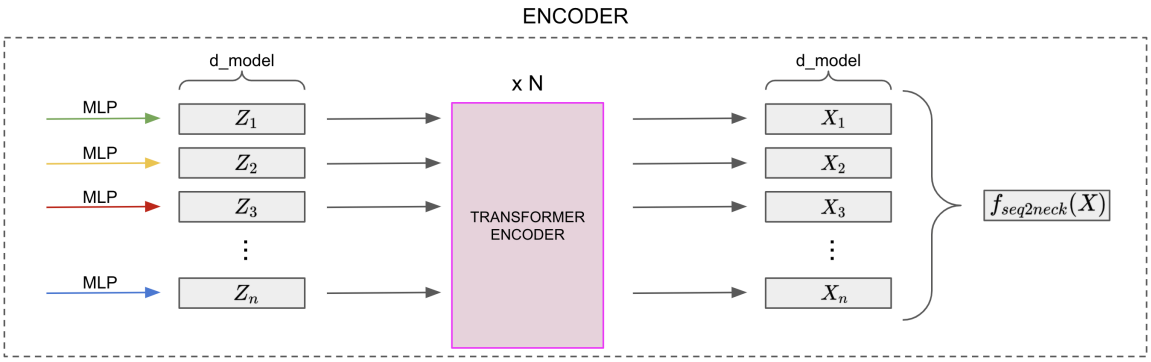


Figure 5: Attention encoder used to map weight's vector to a smaller latent space embedding. The number of input embedding into the transformer encoder remains exactly the same at the output of it.

In order to recover again the input vector from the bottleneck, a different MLP is defined from the neck size to d_{model} size. We have then n (where n is the length input embedding sequence) linear layers to generate the input embeddings to the decoder transformer encoder block. Afterwards, each of the embedding $Z_i^{'}$ is mapped to the correspondent group of weights that it belongs to.

For example, if neuronGroup is being used, each embedding $Z_i^{'}$ would contain the necessary information to be able to recover the weights corresponding to that embedding. For each of the embeddings a MLP is used to transform it back to the weights. For example, the $Z_1^{'}$ would be converted back to the first 16 values. With this approach the model is forced to learn attention in order to be able to capture information of the other neurons in the reconstructed vector.

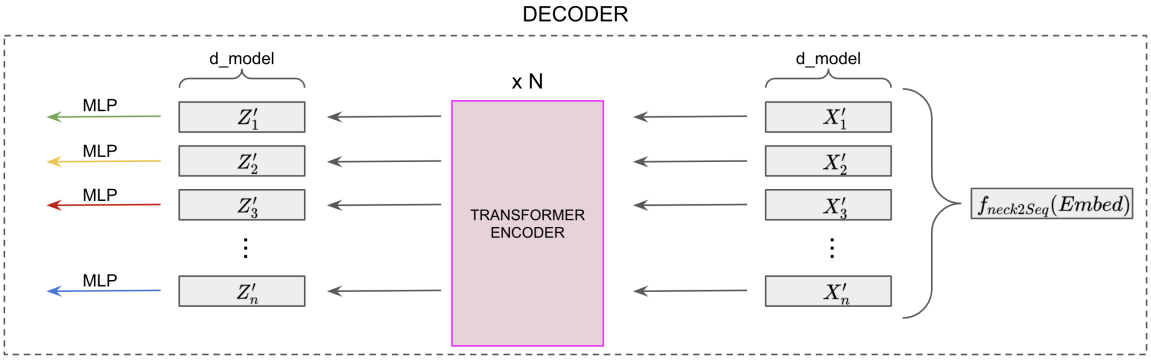


Figure 6: Attention decoder used to map bottleneck to the reconstructed weight's vector. This architecture forces the model to use attention in order to capture information of other groups.

3.2 Attention auto-encoder II (AttnAE II)

In the case that the datasets contain architectures of different lengths, the previous architecture cannot be used, because it would be necessary to train a specific embedder for those new weights. In addition, if the models are very large, the function $f_{seq2neck}$ will be very complex, and the difficulty will be focused on a standard auto-encoder. So the previous architecture does not scale and is not flexible enough to deal with variable input size models. Expanding the idea from vision transformer (ViT) [7] and BERT [8] (where special token is applied to represent the meaning of the entire model), we have introduced some changes in the input embedder and the way we compute the bottleneck.

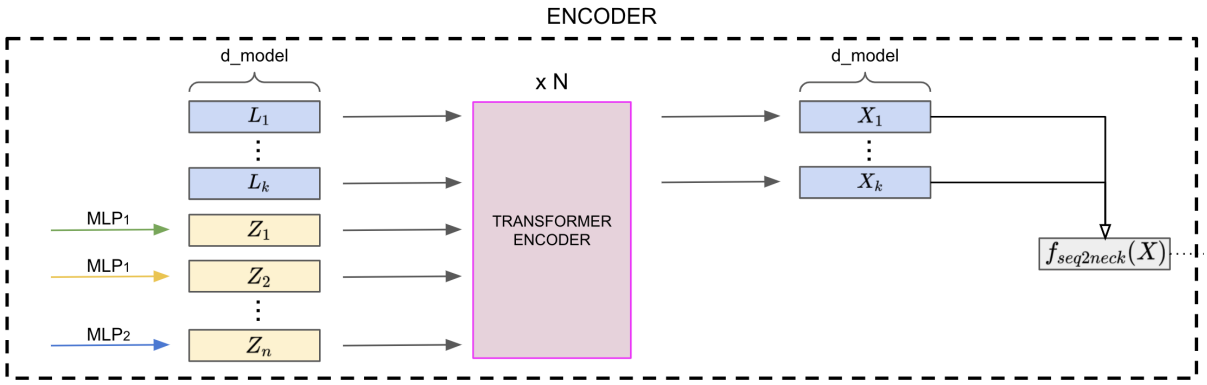


Figure 7: Attention encoder II. It uses unique embedders for each group of weights. It implements sine/cosine position encoding as well as L_k tokens to generate global embeddings of the models.

Depending on the Encoder type, the embedders will end up slightly different. In the case of the NeuronGroup, a single MLP is defiend for each type of neuron. In the tetris-seed dataset (described in section 4.2)there are two types, the first layer (where each neuron has 16 weights) and the second layer (where each neuron has 5 weights). In this particular case, two linear mappings will be learnt from sizes 16 and 5 to d_{model} embedding size, called MLP_1 and MLP_2.

In order to make this step as scalable as possible being able to use different model sizes, sine/cosine position embedding have been used. Taking into account that in an auto-encoder setup the size of the bottleneck needs to be reduced with respect to the input vector, with a transformer encoder the size is not actually being reduced, because the input and the output are exactly the same sequence size. Therefore, the interpretation of the output of the autoencoder has been changed:

- L_k learneable embeddings have been defined which are concatenated at the input sequence. We cannot take any other embedding from the input sequence because its output is the token model's representation. We add a token which has no other purpose than being a model-level representation. After N blocks of the transformer encoder, we are going to take as output only the corresponding embeddings to the ones we introduced. In essence, we can understand these learnebale embeddings as a way of telling the model that those embeddings are not actually the information of the input model, but to encode the entire model during each block of the transformer.
- In order to generate the bottleneck, the function seq2neck is a linear layer from the concatenated X_k embeddings to the size of the neck.

This setup is able to deal with different layer sizes without changing the architecture. The size of the compression can be changed in the transformer encoder by varying the number of L_k tokens. Due to the domain field, it will be very possible to have to deal with very large sequences. In this case the implementation of Reformer [9] for fast attention computation can be done. They replace dot-product attention by one that uses locality-sensitive hashing, changing its complexity from $O(L^2)$ to $O(LlogL)$, where L is the length of the sequence. In case it is needed to increase the number of transformer encoder blocks N, reversible residual layers are used instead of the standard residuals, which allows storing activations only once in the training process instead of N times. We have not implemented it in the proposed methodology.

For the decoder, a similar approach may apply, a MLP may be mapped from the bottleneck to embedding of d_model size plus sine/cosine position embedding. In this case the learneable embeddings have not been used due to the fact that the information is not being compressed. The length of the sequence is exactly the same used to encode the model. From the output of the transformer encoder block a MLP is used for each type of neuron that is going to be reconstructed. In our example, each $Z'n$ is directly the compressed information representation embedding to reconstruct each GroupNeuron. Afterwards, all the output vectors are concatenated to reconstruct the input vector.

3.3 Data augmentation

Generally to be able to train models with many parameters, we usually need very large datasets. One way to be able to get new samples is by using data augmentation techniques applied at the existing dataset.

In the domain of neural networks, one method of being able to generate new samples without modifying their behaviour is by performing permutations among the neurons within a same layer.

With L_i the number of neurons at each i of N layers $N \in \mathbb{N}$ in a deep fully connected network, it can be generated up to $\left(\prod_{i=1}^{N-1} L_i!\right) - 1, for N > 1$ permutations without affecting its operation. Same procedure will be applied when using convolutional neural networks (CNN) on the order of the kernels. When applying permutations layer-wise, the order of the weights of each neuron of the following layer must change accordingly to the permutation applied before.

3.4 Manifold Generation and Visualization

UMAP vs. t-SNE The main difference between t-SNE and UMAP is the interpretation of the distance among clusters. t-SNE preserves local structure in the data. UMAP claims to preserve both local and most of the global structure in the data.

In t-SNE the distance does not mean anything, close proximity is highly informative, distant proximity is not very interesting and cannot rationalise distances, or add in more data.

On the other hand UMAP is faster to compute than tSNE. It can preserve more global structure than tSNE, it can run on raw data without PCA preprocessing, an allow new data to be added to an existing projection. Instead of the single perplexity value in tSNE, UMAP defines nearest neighbours as the number of expected nearest neighbours and minimum distance as how tightly UMAP packs points which are close together.

For these reasons, we used UMAP as it can cope with non-linear scaling and can reduce to 2D well. Due to the dimensionality of the domain the difference in time of the two methods is of great importance. In all the representations of the models in 2D visualizations they correspond in applying UMAP reduction up to two components. Where the first component is the x-axis and the second component is the y-axis.

4 Experiments and results

Training neural networks is generally assumed to impress a pattern in the parameter space, determined by dataset, task and training regime. The main goal is to understand the relationships that exists among neurons. From this knowledge we expect that it helps in a wide variety of downstream tasks. To investigate the weight space for patterns, we apply representation learning on three datasets (tetris-seed, tetris-hyper, and google). On each dataset, we present results of weight reconstruction, accuracy prediction and hyper-parameter classification. On the tetris-dataset we exapand the downstream tasks to be computed on the compressed embeddings generated by the auto-encoder setup. The experiments and results obtained in the different sections will be presented subsequently. They have been separated into two main groups, the experiments performed in the tetris datasets and the experiments performed in the Google dataset.

4.1 Metrics

4.1.1 Signal-to-noise ratio (SNR)

Signal-to-noise ratio is defined as the ratio of the power of the input vector to the power of the distortion injected noise. Logarithmic decibel scale has been used due to very wide dynamic range of the input.

$$SNR_{dB} = 10\log_{10}\left(\frac{P_{input}}{P_{noise}}\right) \quad ; \quad P_x = E[X^2] = \frac{1}{N}\sum_{i=0}^{N} x_i^2 \tag{1}$$

In order to perturb different signals in our setup, we decided to use Gaussian noise $X_{noise} \sim \mathcal{N}(0, \sigma^2)$. Taking it into account and using the expression described in 1, we can define sigma as follows:

$$P_{X_{noise}} = E[X_{noise}^2] = \sigma_{noise}^2 \qquad where \qquad \sigma_{noise}(P_{input}, snr) = \sqrt{P_{input} \cdot 10^{-\frac{snr_{dB}}{10}}} \tag{2}$$

Where σ_{noise} is computed for a given input power and the desired signal to noise ratio to be applied.

4.1.2 Coefficient of determination score (R^2)

R^2 is a statistic that provides information about the goodness of fit of a model. In regression, the R^2 coefficient of determination is a statistical measure of how well the regression predictions approximate the real data points. Non-positive values indicate that we are not doing better than fitting a constant predictor (horizontal hyperplane), and values close to 1 indicate that the regression predictions perfectly fit the data. This implies that 100% of the variability of the dependent variable has been accounted for.

$$R^2 = 1 - \frac{MSE_{res}}{MSE_{tot}} = 1 - \frac{\sum_i (y_i - \widehat{y}_i)^2}{\sum_i (y_i - \bar{y})^2} \tag{3}$$

R^2 score is scale invariant and multiplying the outputs by a constant will not change the metric. In essence, the coefficient of determination is a relative measure which compares the MSE of the Model to the MSE of a constant prediction.

4.2 Dataset exploration

In this chapter, we will start off with the three datasets used in the thesis (Google , tetris-seed and tetris-hyper datasets), continue with its explanation and describing its properties. Public datasets already exists for neural networks trained on popular vision datasets as described in Chapter 2, although, it is for its complexity that only the google dataset [1] has been picked and we have worked on a smaller custom generated datasets, either to validate code as well as to better understand the results due to the reduction in size.

Figure 8: Tetris pieces used to generate tetris-seed and tetris-hyper datasets on the classification task. This representation has been adapted into a color space images. The pictures used in training are composed of one channel images where the background is white and the pieces are black.

The custom tetris dataset is designed to be much more tractable in terms of complexity and size. It is composed of models of fixed size (two feed forward layers, five and four neurons each, respectively). The images on which these models have been trained were also generated. It is composed of four tetris pieces in a 4 by 4 black-and-white pixels setting. The models were trained to classify which piece each image had. The models have been trained for 75 epochs storing each checkpoint of a vectorized version of size 100 (model's weights have been flattened).

Dataset	Model type	Samples	Parameters	Datsets trained	Model diferences
Google [1]	CNN	∼1,1M	4970	Mnist Fashion_mnist Cifar10 Svhn	Optimizer Init method Activation
Tetris-seed	FCN	∼75K	100	Tetris shapes	Init seed
Tetris-hyper	FCN	∼105K	100	Tetris shapes	Init method Activation

Table 1: Dataset properties. Comparison between Google and tetris datasets.

The google dataset consists of a 120k models trained on Mnist, Fashion-mnist, cifar-10 and svhn equally distributed. For each model trained there have been stored 9 checkpoints at epochs 0, 1, 2, 3, 4, 20, 40, 60, 80 and 86. The difference between trainings are the initialization function (Random Normal, Truncated Normal, glorot_normal, he_normal, orthogonal) and the activation function (relu, tanh). Same seed have been used for all the trainings.

In the following model representations Uniform Manifold Approximation and Projection for Dimension Reduction (UMAP) has been used to generate two-dimensinal plots. In all representations, the x-axis and y-axis refers to the first and second components. In the following graphs I do not differentiate between different trainings, so all models and their different checkpoints will be shown as points. All plots have been generated using the same hyperparameters with minimum distance set to 0.1, number of neighbours set to 50 and using euclidean distance function.

4.2.1 Tetris-Seed dataset

In order to generate the representations, the same methodology used in previous studies has been used, where all the weights are concatenated in a single vector. Each model is composed by a single vector of size 100. In addition, a statistical vector has been generated (as [1] used), computed on the original vector, obtaining a new vector of dimension 7 (mean, variance and percentiles 1,25,50,75,99). In all the representations each dot corresponds to a model at specific epoch, meaning that each model trained over 75 epochs will be represented as 75 points in the space.

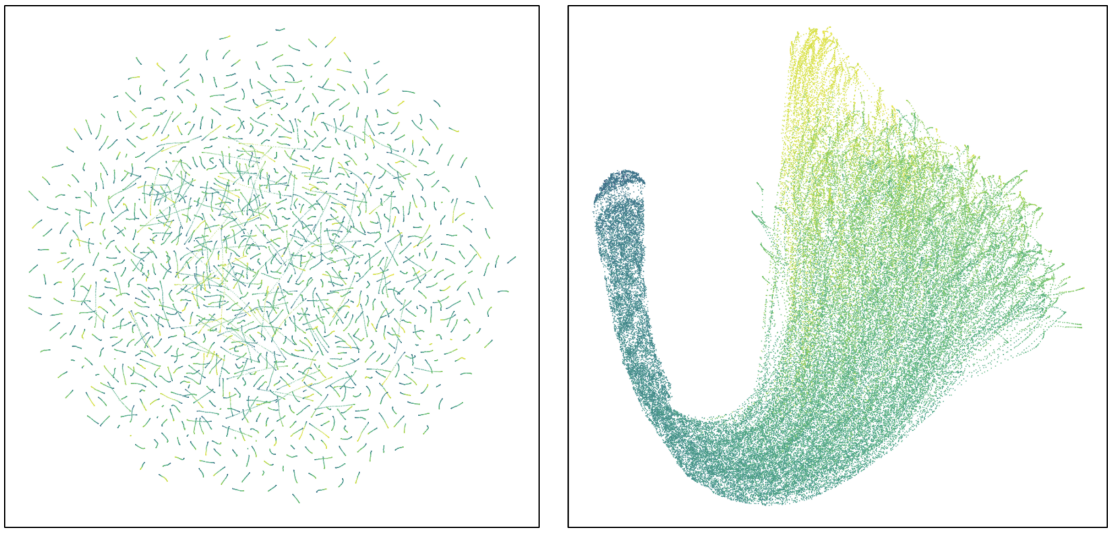


Figure 9: UMAP Representations of the tetris dataset. Color heatmap ranges from blue color for samples which have low test accuracy to yellow color for the models that achieved high performance. Left image is the projection from the weights (size 100 values) into two dimensions. Right image is the projection from the concatenated statistics of dimension 7 (mean, variance, and percentiles 1,25,50,75,99) into two dimensions.

UMAP has been applied to the dataset represented by the vectors containing all the weights of dimension 100, and to the dataset represented by vectors composed of the statistics of dimension seven. In the first case, short trajectories (little worms) for each model have appeared. These worms are not interconnected among them. However, when the UMAP representation of the pre-proceces weights (basics statistics) are projected, a clear trajectory of the models can be seen, which shows a clear correlation between the statistics and the model performance. Statistics of the models' weights capture enough information in order to predict the test accuracy. This behaviour could explain why other works [1] are able to predict accuracy up to $0.99 R^2$ score from the model's statistics.

The plots showed are UMAP mappings from hundred to two dimension which could introduce some undesirable results. For this reason all the plots have been computed several times. Although correlations do not imply causality between two variables, Pearson (assesses linear relationships) and Spearman (it assesses monotonic relationships whether linear or not) correlation between accuracy and each of the statistics of the input vector were computed (see Figure 2). In addition, a linear predictor has been used for each of the values generated by each statistician to predict accuracy. Using linear regression the best result achieves up to 0.86 R^2 score when using all the statistics values. Using only the percentile 25 it can be obtained 0.82 R^2 score (see Table 2). In general using each of the basic statistics alone are high correlated with the accuracy. Although two correlation methods have been used, the results are similar in both cases.

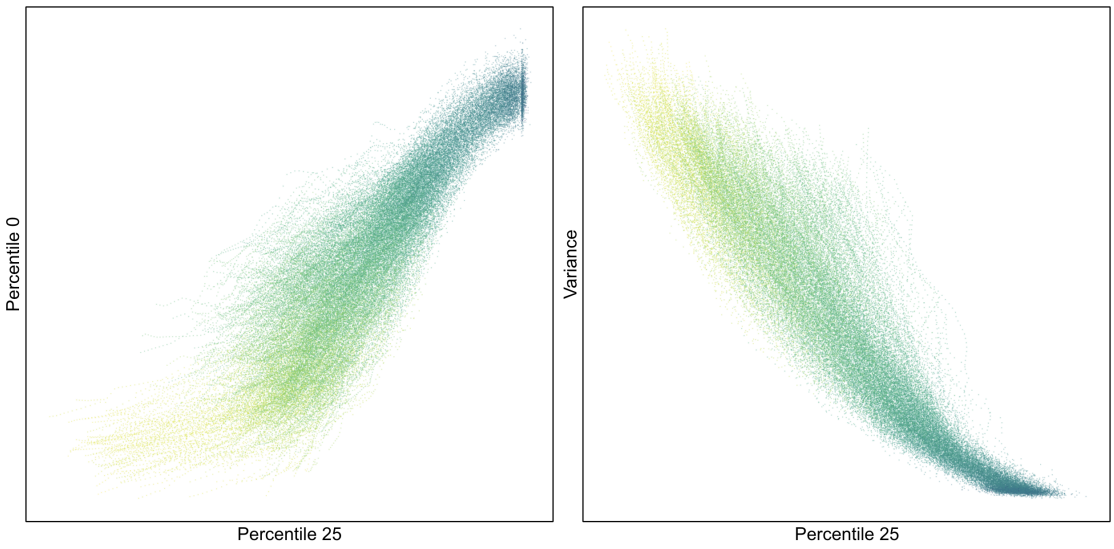


Figure 10: Representations of basic statistics tetris dataset. Color heatmap ranges from blue color for samples which have low test accuracy to yellow color for the models that achieved high performance.

The statistics that have obtained a higher correlation for each of the two metrics used have been selected. The results show that the models with a smaller 0 and 25 percentile and a higher variance have a better performance (see figure 10). We need to be cautious in order to avoid learning this statistics to predict accuracy (this could be a shortcut) if our goal is to learn the weights' structure that there is behind. This could lead us that our architectures learn this statistics instead of learning the structure. In short, out intention is to understand the reason why this correlation is produced and what causes such dependence. Because although the best models are characterized by a high variance, this does not imply that this is the cause of their good performance.

	Correlation		Linear_regression	
	Pearson	Spearman	MSE	R^2
Mean	-0.274	-0.2	5.43e-3	0.066
Variance	0.86	0.88	1.5e-3	0.74
Percentile 0	-0.86	-0.85	1.5e-3	0.74
Percentile 25	-0.91	**-0.9**	1.1e-3	**0.82**
Percentile 50	0.28	0.22	5.3e-3	0.082
Percentile 75	0.83	0.86	1.8e-3	0.69
Percentile 100	0.71	0.72	2.9e-3	0.51
Stack All	-	-	0.8e-4	**0.86**

Table 2: Pearson and Spearman correlation between the statistics of the model weights and the accuracy prediction of the model on the tetris-seed dataset. Linear regression applied for each of the statistics and also stacking all as seven ordered variables.

UMAP representation of neuronLayer and neuronGroup approaches have been generated, Figure 11. In this case the basic statistics have been computed for each embedder type and concatenate all the values in a single vector. In the first case the vector has been composed by the concatenation of seven statisticians for each one of the layers $2x7 = 14$, and in the second case computed by each one of the weights that conform each neuron $9x7 = 63$. In both cases the shapes does not show a clear pattern for accuracy prediction.

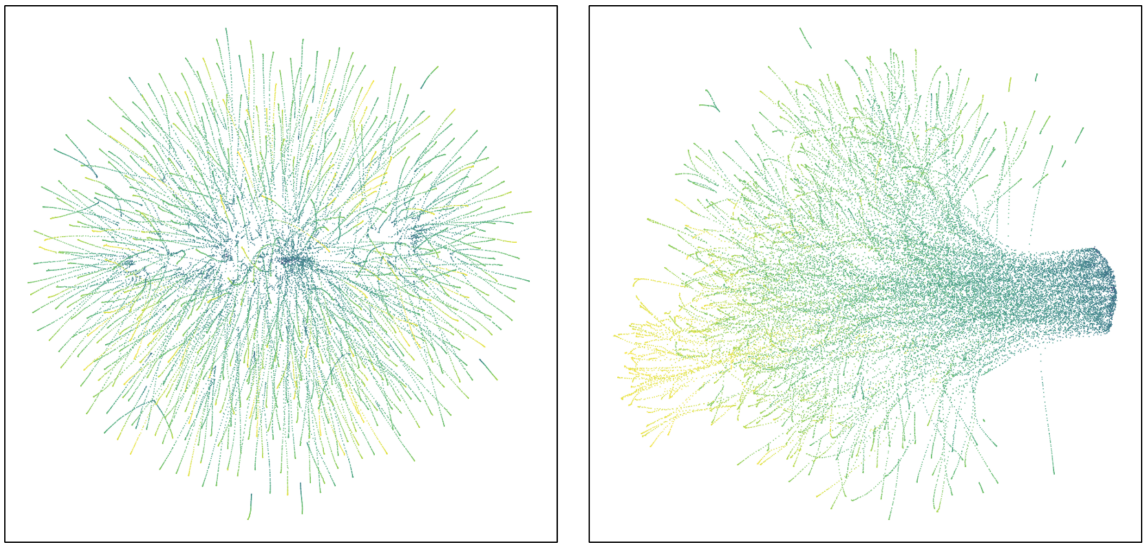


Figure 11: UMAP reduction of dataset tetris-seed using basic statistics applied at each neuronGroup (left image) and applied at each layer (right image). Heatmap ranges from blue color for samples which have low test accuracy to yellow color for the models that achieved high performance.

4.2.2 Tetris-hyper dataset

Same scheme described in Tetris-Seed has been used to present the following results. UMAP representations have been generated on the complete weight vector and on the statistics vector. This dataset was generated using two types of activation functions and six different initialization procedures, for this reason we also present the results of accuracy and its properties.

The UMAP representations of the model samples directly from the weights show similar results as described in the Tetris-Seed dataset. In this case, as different architecture and training procedures have been used, I also have plotted the models which have comformed by relu or tanh as an activation functions. The statistical vectors present a better separability of the classes both according to the accuracy of the models and to differentiate the activation function used.

As seen in Figure 12, by looking directly to the model's weights (top row) it is barely impossible to define clear regions where the models differ. However, when the vectors are used with the statistics the UMAP projection (bottom row) presents a more defined pattern. In this case both models start at a similar point in the space and statistics of the converged models changes due to updates during training depending on the usage of relu or tanh. This allows them to derive into new and different shapes, which is easy to classify them even by human-eye. Figure 12 shows how the weights are modified differently depending on the hyper-parameters and structure used. The trained models have properties in the space of the weights that are characteristic of their characteristics and that have been changed during their training. Depending on the nature of each one, its impact will be more noticeable at the beginning (like the function used to initialise the weights) or when the model has already converged (in general this will happen after a few iterations).

Deep neural networks are sensitive to initialitzation [2] and slightly differences on some hyper-parameters completely change the final optimized weights. The complexity of the space where this models belongs could not be defined by just some statistics, slightly different weights could end up with a model with completely different performance. For a given distribution, if we take samples of it in order to generate models, only some of them will be able to still predict the tetris pieces.

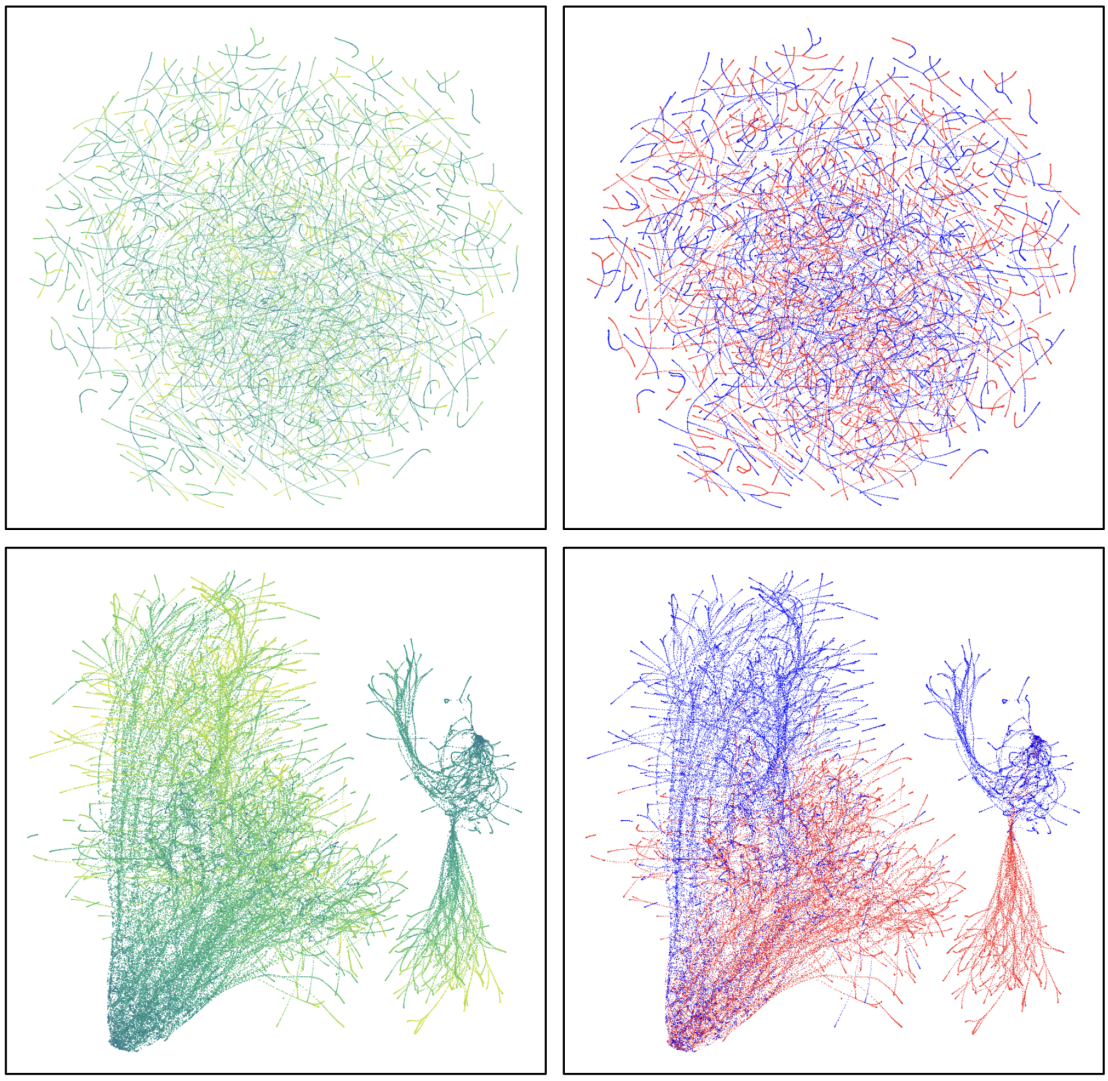


Figure 12: Top images are UMAP mappings directly from input vector of dimension hundred to two dimensions. Bottom images are UMAP mappings from basic statistics of dimension seven to dimension two. Color map of left images show accuracy score from blue to yellow (low to high). Color map of right images show the activation function, red to tanh and blue to relu.

4.2.3 Google dataset

The results showed in this chapter have been separated in four main groups depending on the dataset in which the models were trained on. Although the MNIST group is going to be discussed in detail, the same conclusions can be applied to the rest of the splits (FASHION-MNIST, SVHN, CIFAR-10). The UMAP mappings are computed directly from the input vector of dimension 4970 to two dimensions and from the concatenation of the basic statistics described previously. Unlike the tetris-Seed and tetris-Hyper datasets, the google dataset uses convolutions at the beginning of the network (the properties could be different than fully connected layers).

Unlike tetris-seed and tetris-Hyper datasets, the UMAP embeddings applied directly to the input weight vectors show clear patterns of the models' trainings (see Figure 13). It can be seen kind of individual leaves which at the beginning achieves low accuracy, but as the models are being trained, the accuracy grows. As expected, Figure 13 show how different distributions in a different position in the space achieve similar results in terms of accuracy. This result could be understood as models that have converged in different relative minimums but maintain the same performance. In case all trainings achieved the absolute minima (considering there is only one possible absolute minima) all the models should then converged to the exact same weights being mapped closely in the UMAP transformation, which in reality this is not going to happen.

In the activation and initialization plots (Figure 13), each of the leaves belongs to a different properties of the architecture. Depending on them the weights are modified accordingly. There are some models that do not flourish which are spread out in the space. Similarly, the models at the beginning of the training with the same initialization procedure are close (different samples of the same distribution are mapped close while different distributions are mapped far away). As the training starts, each of the properties changes the way the updates in the back propagation affects the final distribution, creating this regions where it can be easily seen properties at human-eye level.

Using the statistics vector of the model's dataset (Figure 14), the interpretability has changed. Each model started in a different position at the beginning, but as is being trained, the models start to join. Therefore the results shows that is easier to predict the accuracy of the models from the statistics. However, it seems more complicated to distinguish the properties of the model or how they were trained.

	Correlation		Linear_regression	
	Pearson	Spearman	MSE	R^2
Mean	-0.26	-0.20	0.11	0.07
Variance	0.18	0.72	0.11	0.03
Percentile 0	**-0.40**	**-0.73**	**0.099**	**0.16**
Percentile 25	0.36	**-0.56**	0.10	**0.13**
Percentile 50	-0.05	-0.09	0.12	0.002
Percentile 75	0.30	0.52	0.11	0.09
Percentile 100	**0.40**	**0.73**	0.099	0.156
Stack All	-	-	**0.08**	**0.30**

Table 3: Pearson and Spearman correlation between the statistics of the model weights and the accuracy prediction of the model on the google dataset. Linear regression applied for each of the statistics and also stacking all as seven ordered variables.

As explained in the tetris datasets, Pearson and Spearman correlation have been applied between basic statistics and accuracy (see Table 3). Correlation values between those are really high. Same pattern can be seen in previous smaller datasets. In this dataset the model trained has a difference on the architecture (in this case is a convolutional neural network). Nonetheless, we are able to predict accuracy up to $0.3R^2$ score with linear regression from the concatenation of the statistics. Comparing results with the tetris-dataset, the correlation values between statsitics are different, meaning that I cannot generalize which statistics better define the overall accuracy score.

Overall, these results suggests that having information of the distribution of the model's weights give us a lot of information of the model's behaviour. This shortcut can lead us to fail when defining and designing architectures. If computing mean and variance is enough for achieving good performance, the model will end up learning this transformation instead of learning the relation between weights. Besides, permutations as a data augmentation will not help because this data augmentation is invariant to the mean and variance transformations.

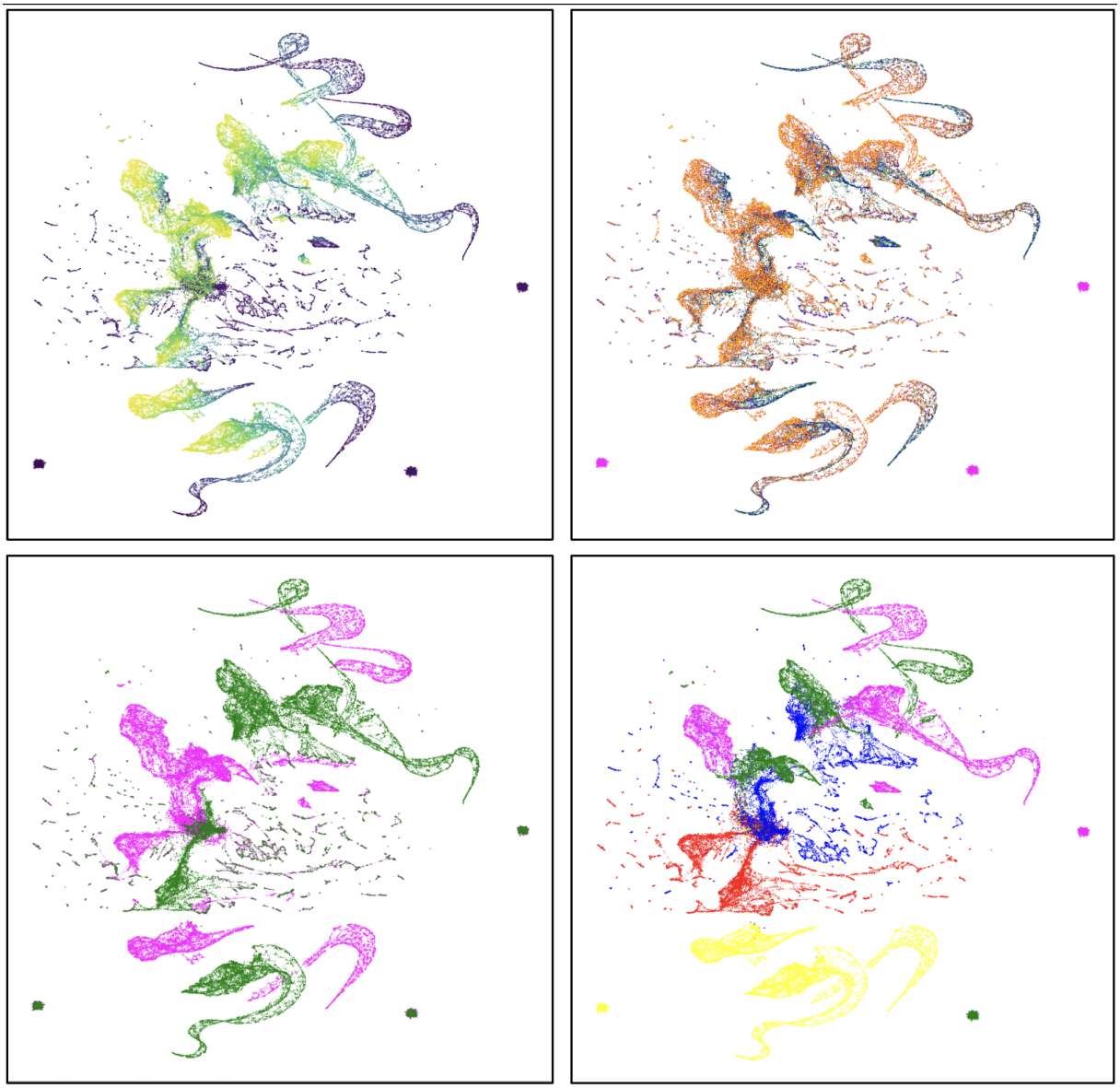


Figure 13: UMAP embedding applied directly from the input vector of dimension 4970 to two dimensions. Top left image show the accuracy of each sample (low accuracy in blue, high accuracy in yellow). Top right show the checkpoint during training (checkpoints: 0-magenta, 1,2-green, 3-blue, 40,60,80-red, 86-yellow). Bottom left image show the activation function (relu-magenta, tanh-green). Bottom right image show the inisialitation function (RandomNormal-red, TruncatedNormal-blue, glorotNormal-green, heNormal-magenta, orthogonal-yellow)

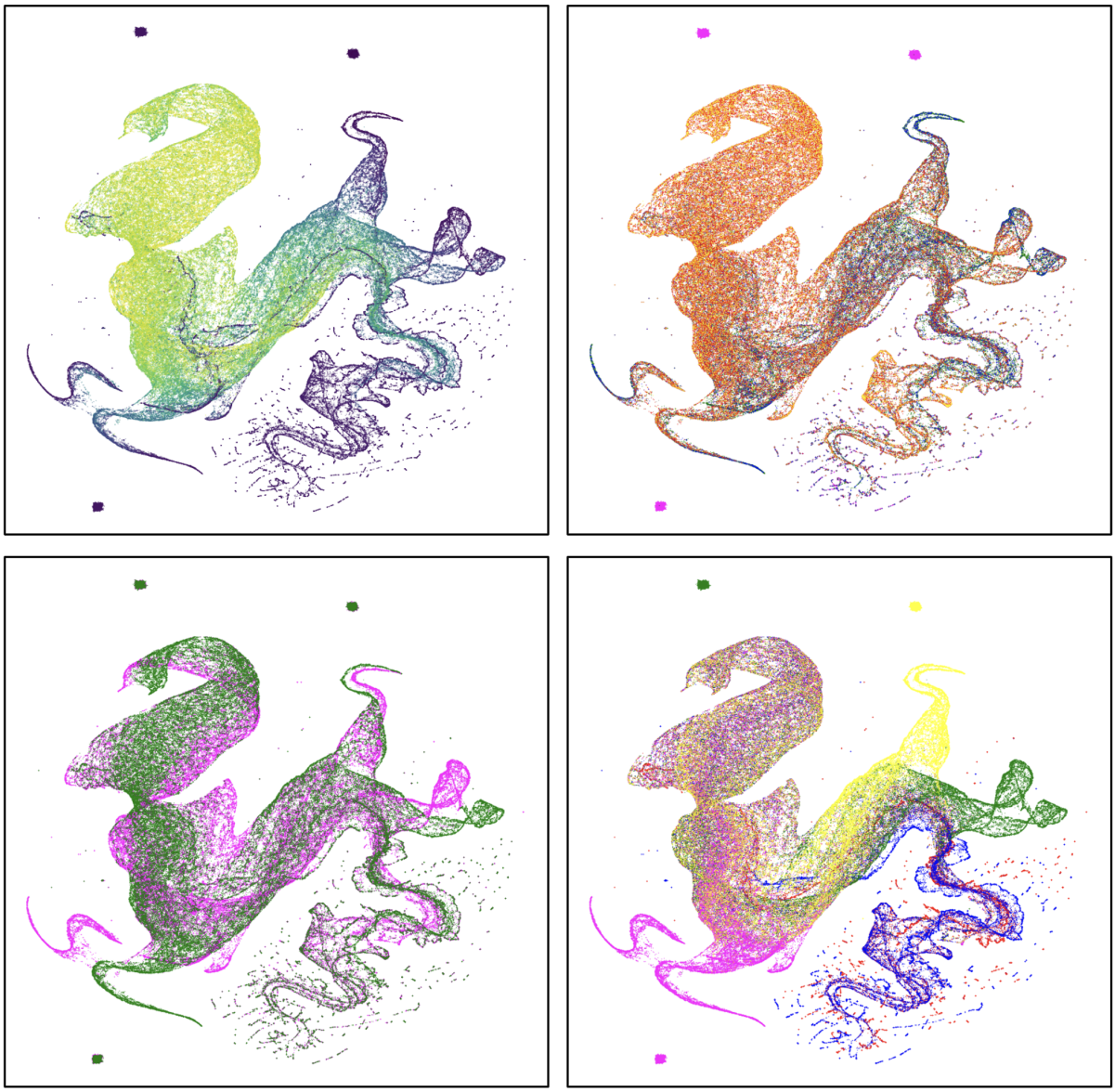


Figure 14: UMAP embedding of statistics from the input vector of dimension 7 to two dimensions. Top left image show the accuracy of each sample (low accuracy in blue, high accuracy in yellow). Top right show the checkpoint during training (checkpoints: 0-magenta, 1,2-green, 3-blue, 40,60,80-red, 86-yellow). Bottom left image show the activation function (relu-magenta, tanh-green). Bottom right image show the inisialitation function (RandomNormal-red, TruncatedNormal-blue, glorotNormal-green, heNormal-magenta, orthogonal-yellow)

4.3 Tetris-seed and tetris-hyper datasets experiments

We begin our experiments on the tetris-seed and tetris-hyper datasetes, as they are the smallest dataset. On both datasets, we have performed compression and accuracy prediction tasks with AttnAE I and AttnAE II architectures as well as vanilla DNN and PCA based auto-encoders. In a further step, we also present information regarding the interpretability of the attention score maps as well as the robustness to noise when performing compression tasks.

To establish the existence of patterns and in an attempt to learn useful embeddings, we learn incomplete auto-encoders with a compression ratio of 5:1 in all experiments (from input vector of size 100 to a vector size of 20). The dataset was divided into two groups of 50% of the samples for each group. I have also ensured that each model with all its 75 checkpoints are to the same corresponding split. All the values presented are computed on the test split. The number of models for each checkpoint are equally distributed on both splits.

We investigate whether auto-encoder for weight reconstruction yields embedding spaces, that are useful for general downstream tasks and preserve information on the samples embedded. To evaluate the amount of information that is preserved, accuracy prediction was performed using linear regression and vanilla multi-layer perceptron (MLP).

4.3.1 Model weights reconstrucction

For the AE reconstruction task, we compare several architectures. As baseline, we consider PCA and a MLP-based Auto-encoder. Further, we apply the transformer-based architectures we propose in the previous section, the first version (AttnAE I) and the second update (AttnAE II) using the general context tokens concepts used in Bert and ViT. To evaluate the quality of the reconstruction we have used the metrics R^2 between the input vector and the reconstructed vector of all the samples of the test set.

Each sample of the dataset is represented by a vector where all the weights are concatenated. The vanilla auto-encoder uses this vector directly. However, AttnAE I/II are defined to work with embeddings which represents the models. The neuronGroup approach (where all the weights associated with each neuron are used to generate an embedding) was used to encode the input data. This decision was taken due to its better performance. It was considered a reasonable form to spread the information without generating too many embeddings (an embedding for each weight) or losing information among different neurons (generating one embedding for all the weights for each layer).

Deep neural networks are able to modulate nonlinear functions, therefore they should obtain at least the same performance as PCA, which relies on a linear auto-encoder. All experiments were trained for 8000 epochs (except PCA). The experiments where data augmentation was applied only permutations on the weights of each model were used as permutation. As a reminder, this augmentation generates equivalent samples without changing the performance. 120 possible permutations were generated and applied randomly during training.

Architecture	Embed Size	Parameters (N, d_{model})	Model Size	Data Augmentation	R^2 Test Reconstruction
PCA	20	-	-	No	0,352
10 layer MLP	20	-	∼90k	Permutations	0,756
AttnAE I	20	1, 20	∼16k	No	0,643
AttnAE I	20	1, 128	∼80k	Permutations	0,830
AttnAE I	20	2, 128	∼180k	Permutations	0,843
AttnAE II	20	4, 128	∼100k	Permutations	**0,865**

Table 4: Model performance in weight reconstruction task. PCA computed with a linear kernel.

The results of our experiments are presented in Table 3. All non-linear models have achieved better performance than linear auto-encoder (Principal Component Analysis for reconstruction in a setup 100-20-100). Comparatively, the two attention-based versions (AttnAE I/II) achieved better scores than regular multi-layer perceptron auto-encoder (10 layer MLP for the encoder and symmetric decoder). The AttnAE II model achieved the best reconstruction score. In both cases it was necessary to use permutations as data augmentation techniques in order to improve the results, otherwise the model suffers from huge over-fitting and performance is considerably lower. In the case of MLP, without permutations, values around 0 R^2 were obtained. Without data augmentation the variance could not be predicted and only the mean value was forecasted.

The results shown in Table 4 clearly show how transformer-based systems achieve superior results in reconstruction. It has also been proven that they have a higher convergence speed than the regular vanilla MLP auto-encoder as seen in Figure 15. In the case of AttnAE II, it was able to work with models that had not seen before with a different number of weights. However, in this comparison the whole dataset is composed of a fixed architecture. The fact that AttnAE II uses the same encoder for each type of neuron greatly reduces the size of the model and scales better as the input has more neurons. For the model to converge it is necessary to increase the number of transformer encoder blocks so that the model-level representation token is rich enough to perform the compression. This behaviour may be due to the reduction in the number of trainable parameters used in the embedder to generate the tokens that describe the input models. That is the reason for increasing the complexity in the transformer encoder block where the compression is generated. This combination enables to reduce the complexity of the encoding of the input with the disadvantage of increasing the number of transformer encoder blocks needed to make the model converge.

The main hypothesis is that as the models are being trained, their weights will get structured depending on different parameters, such as which data have been used for training, which architecture has been chosen or which hyper-parameters have been selected. Therefore, based on the good functioning of AttnAE I/II architecture that uses attention, the hypothesis is that it is possible to better identify and capture the relationship between the embeddings. In these experiements, we used neuronGroup to generate an embedding from the weights that composes each neuron. We expect that the reconstruction performance increases as the weights gain more structure.

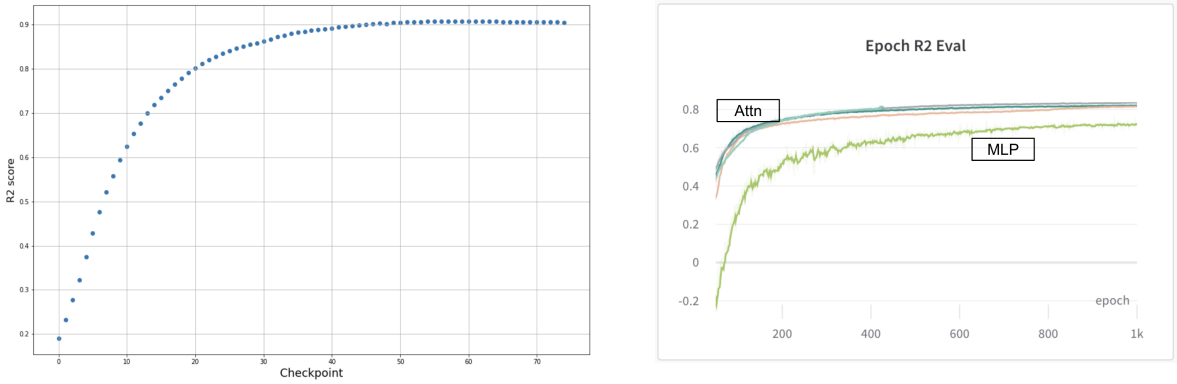


Figure 15: Left figure: Reconstruction R^2 score performance over epochs. Each dot represents the score computed only for models that belongs to the selected checkpoint in the test split. The model has been trained with models from all epochs without any distinction. Right Figure: Model reconstruction R^2 score performance on the first 1k epochs training. Results given in the test split.

To get a better understanding where structure exist, we evaluate the reconstruction performance over the epoch of the samples (see Figure 15). The AttnAE II model with 100K parameters was chosen to evaluate the reconstruction R^2 score. The test split consists of models belonging to different epochs. The results presented in Table 4 were computed on all the samples without differentiating in which group they belonged. The model used was trained with samples belonging to all groups without distinction.

The reconstruction R^2 in models belonging to the firsts epochs is considerably lower than the models that have already converged (the models are expected to be converged on the last checkpoints). The weights of the models in the first epochs are samples from a known distribution (they have not gotten structures from their properties). This implies that we are able to predict the mean of the population, but it is difficult to predict the exact realisation from the given input. Even knowing the distribution we are sampling from, the best it an be done is to sample from the same distribution so the reconstruction predictions will match the mean and will have the same variance.

This behaviour can be attributed to the difference in structure that exists between the models with random weights and those that have acquired structure during their training. This results validate the idea that at the beginning there is no structure among weights (just samples from distribution) that is why we cannot understand and capture the relation because there have not been place any back-propagation update yet. However, as the training progresses, the weights are updated in a particular manner depending on wide

variety of parameters (such us the dataset to train it or the hyper parameters used) giving them structure. Our hypotesis is that more structure in the weights space allows the attention mechanism to capture this interaction and therefore it is able to reconstruct the weights (because these values are not random samples from a given distribution).

4.3.2 Downstream tasks: accuracy and hyper-parameters prediction

We want to explode the interaction between neurons to better define properties of the models and be meaningful in downstream tasks. In this section, we explore the accuracy prediction and hyperparameters classification (activation function and initialization method) on the tetris-seed and tetris-hyper datasets. To evaluate what properties are encoded in the embeddings learned for reconstruction (for each input vector of size 100 an embedding of size 20 from the bottleneck is generated), we apply linear probing and vanilla MLP from the embeddings to accuracy. Performance is measured using the R^2 score metric for the entire test set.

Mainly two approaches were used to construct the samples in the two datasets. On the one hand, a linear regressor was used to predict the accuracy in two modalities: Directly from the weights' vector and on a vector generated with the concatenatenation of basic statistics. In the latter group, the statistics are compared by applying them directly to all the weights in each layer and to the neuronGroup approaches. On the other hand, the AttnEnc architecture was used (adapted from the AttnAE I/II) and the three different methods of grouping the weights have been compared to generate the embeddings (layer-wise, neuron and neuronGroup).

In order to tackle accuracy prediction and hyperparameters classification tasks, the architecture AttnAE was adapted. To adapt the architecture the encoder part was used (a neural network that transforms the embeddings that describe the model into a reduced fixed size embedding). In order to be able to make predictions, we have adapted the last layer from the bottleneck to fit the requirements:

- **Accuracy prediction:** From the bottleneck vector (generated by the encoder block from AttnAE) a linear layer and sigmoid function is applied. The model generates for each sample a prediction in the range $[0, 1]$ as accuracy is defined in this range.

- **Hyper-parameters classification:** From the bottleneck vector (generated by the encoder block from AttnAE) a linear layer of size the number of possible classes to be predicted for each group is used. Afterwards, a softmax function is applied to force the output to be a probability distribution. The class chosen is the one that has achieved greater output value or probability.

Architecture	Input type	Data Augmentation	Accuracy R^2 tetris-seed	Accuracy R^2 tetris-hyper
Linear regression	weights	No	0.48	0,23
Linear regression	weights statistics	No	0.86	0,67
Linear regression	weights statistics layer-wise	No	0,87	0,74
Linear regression	weights statsitics neuronGroup	No	0,89	0,74
MLP (4 layers)	weights	No	0,87	0,71
Attn Enc	weights layer-wise	No	0,87	0,72
Attn Enc	weights neuronGroup	No	0,92	0,87
Attn Enc	weights neuronGroup	Permutations	**0,95**	**0,89**

Table 5: Accuracy prediction on tetris-seed and tetris-hyper datasets. Comparison of models using directly the weights and using the basic statistics transformation.

As shown in Table 4, applying a linear regressor on the weights' vector to predict accuracy has obtained 0,48 R^2 and 0.23 R^2 for tetris-seed and tetris-hyper datasets. The usage of the statistical transformation with neuronGroup encoding has increased the results up to 0,89 R^2 and 0,74 R^2 respectively. Results reported on previous works concluded that pre-processing the weights by computing its statistics layer-wise helps on predicting the accuracy. In our datasets, computing the statistics layer-wise did not give any advantage with respect to the global statistics.

We extend these results by applying our attention encoder architecture, which outperforms the linear regression results in all experiments. This results were expected due to the fact that our model is able to map non linear relations and we suppose the interactions among neurons are highly non linear. In our approach we expect the model to learn in an unsupervised manner the interconnections by learning from the neuron's relations. We have not used neuron encoder because of the poor scalability that this approach offers when dealing with bigger models. We need some criteria to reduce the size of the weight vector. From Table 5 we can state that treating all the weight for each neuron gives us the best performance. Using permutations in training helped to make the model more accurate and gain in terms of generalization.

Architecture	Input type	Data Augmentation	Activation (F1 score)	Initialization (F1 score)
Linear regression	weights	No	0,82	0,32
Linear regression	weights statistics	No	0.80	0,42
Linear regression	weights statistics layer-wise	No	0,80	**0,47**
Linear regression	weights statsitics neuronGroup	No	0,87	0,45
MLP (4 layers)	weights	No	0,90	0,39
Attn Enc	weights neuronGroup	No	0,91	0,38
Attn Enc	weights NeuronGroup	Permutations	**0,97**	0,46

Table 6: Hyper-paramaters prediction on the tetris-hyper dataset. Comparison of models using directly the weights and using the basic statistics vectors transformation. Performance is evaluated using F1 score.

The tetris-hyper dataset adds additional complexity and makes both reconstruction and property prediction harder. It is composed of models trained with two different activation functions and six different ways of initializing the weights. By using a log linear predictor it has been possible to predict in a very notable degree the activation used. In the case of the initialization the best combination has been achieved using the vector of layer-wise computed statistics. The attention-based model has obtained the best classification using neuronGroup encoding (the other two methods of coding the models have obtained a lower value and have been omitted) surpassing the vanilla MLP. Both models are formed by approximately 100K parameters.

The fact that using a linear model on basic statistics of the weights has been the approach that best predict initialization could be due to its nature. Statistics are a useful method of determining the density functions used to generate the weights. Using transformers and permutations such as data augmentation has achieved a slightly lower result. We compare these findings again to *Classifying the classifier: dissecting the weight space of neural networks* [2], who present that in the classification section, better results are obtained for the activation functions than for the initialisation (different datasets and different classes for each of the categories).

In the chapter 4.2.2 the architecture AttnAE II is used so that in a self supervised manner the initial vector is reconstructed. The thinnest layer called bottleneck is formed by a vector of length 20. The model trained for reconstruction has been used to transform the tetris-seed dataset into the respective latent space vectors from the bottleneck. Structure and information is generally lost when the weights that compose a model are compressed. However, depending on the task, it is not necessary to have the full initial available information and there may be irrelevant or highly correlated variables.

A linear regressor and a vanilla MLP were trained to predict accuracy from the AE embeddings. The AE embeddings of tetris-seed dataset are formed by exactly the same number of samples but instead of being stored in a 100 length vector they have been compressed with a 5:1 ratio. The same split train-test has been used to train the accuracy predictors. Only data augmentation techniques have been used in the training of the reconstruction model (which has then been used to compress the dataset). The accuracy prediction task has not been trained end-to-end with the reconstruction architecture. The metric used to measure the prediction is the R^2 relative score.

Architecture	Input type	Data Augmentation	Accuracy R^2 tetris-seed
Linear Regression	bottleneck (size 20)	Used only when training the auto-encoder architecture to generate the bottlenecks	0,54
MLP (3 layers)	bottleneck (size 20)	Used only when training the auto-encoder architecture to generate the bottlenecks	0,85

Table 7: Accuracy prediction score from the bottlenecks obtained on the task of reconstruction. The model used to generate this bottlenecks is the AttnAE II achieving an R^2 score reconstruction of 0,86

Using a linear regressor for accuracy prediction applied directly on all weights provided a result of $0.48R^2$ score. However, if it is applied to dimension 20 compressed vectors it has been possible to predict with a performance of $0.54R^2$. On the other hand, using a non-linear method, better results have been obtained. In the latter case, the value obtained is similar to those obtained with Attn Enc and a vanilla MLP using all the available information. This means that by using aggressive compression (ratio 5:1), the information needed to predict accuracy is only slightly lower than in the full space, at least for this ratio compression.

4.3.3 Attention score maps interpretability

This section presents the attention score maps generated in the AttnAE I and AttnAE II models for the entire range of models that composes the tetris-seed dataset test partition. In addition, the correlation between them is shown, those that obtain the best score and those that obtain the least score. Lastly, the noise robustness of the model used for reconstruction applied in the weight vector and applied in the bottleneck is investigated.

One benefit of using attention mechanisms in a model is that the attention maps can be investigated to gain insights in the model's mapping. The attention score maps can be understood as the amount of information needed to obtain from the corresponding embedding. An advantage of using this methodology is that it is possible to investigate existing relationships by interpreting the score maps. Models with lower accuracy tend to have less structured weights. Since the AttnAE model uses the information among neurons to perform different tasks, it should present different patterns in the attention score maps depending on the accuracy obtained by each model. For this reason, the expected result is being able to establish a correlation between connections of the models that belong to the same range of accuracy prediction performance.

First of all, attention-based auto-encoders (trained for the task of reconstruction) were used to reconstruct all the models of the test split. The three methods (neuron, neuronGroup and layer) described in section 3.1 were used to encode each of the models, but the neuronGroup was used for two reasons. Firstly because better results have been obtained for the reconstruction task and secondly because it provides adequate amount of data to be computed (using each neuron to generate an embedding is not suitable for computing correlation with the 30,000 samples available in the test set).

Secondly, for each of the models, attention score maps corresponding to the four encoder heads were stored. For the AttnAE I model, each of the attention score maps has a 9x9 size that represents the attention between the neurons. The first five correspond to the first layer, and the following four to the second layer. In the case of the AttnAE II model the maps have a measure of 10x10 where the first corresponds to the global context token.

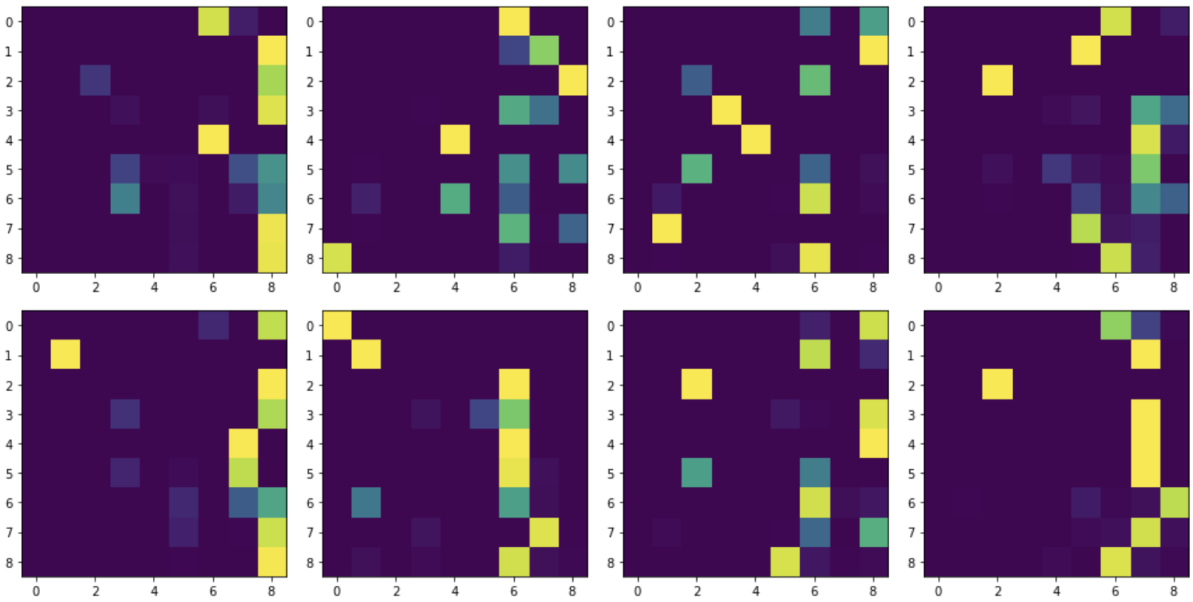


Figure 16: Attention score maps row-wise. Each column represents each of the four heads used. First row represents the scores from a sample with high accuracy. Bottom row represents the scores from a sample with low accuracy. X and Y axis have nine values because it is the number of embeddings used in the transformer encoder (five neuron groups for the first layer and four neuron groups for the second layer). The heat-map represents a gradient color between blue and yellow between 0 and 1 values

Different approaches of encoding the networks have been used but the results are similar in all of them. The results presented correspond to the neuronGroup encoding. The attention score maps in language tasks are composed of a diagonal with higher values and depending on the connection with the rest of the words they are interconnected. However, in this case the patterns do not coincide. Figure 16 shows that the attention in general is looking at the last neurons, which seem to be more relevant to reconstruct the entire vector. Although we were expecting to get human interpretation, we could not make any conclusion. Using four heads rather than just one achieved better performance, however it does not translate it to better interpretation, at least for the human-eye.

Using the AttnAE II model, no different patterns can be distinguished from the AttnAE I model either. The only difference is that the values of the scores are not so concentrated, obtaining more blurred values, but neither is there any visual similarity.

Attention score maps do not present a clear pattern identifying their direct relationship between accuracy, structure and precision. For this reason we have computed Pearson's linear correlation between all the attention score maps. First, the score maps of all the samples from the tetris-seed test partition dataset have been computed. The score maps have been flattened to compute 1D Pearson correlation among all the samples. Afterwards, the symmetric correlation matrix has been sorted in ascending order (0 index for the model that obtained the lowest accuracy value, 0.32 R^2). Finally, the mean over neighbourhood regions has been computed to decrease its size.

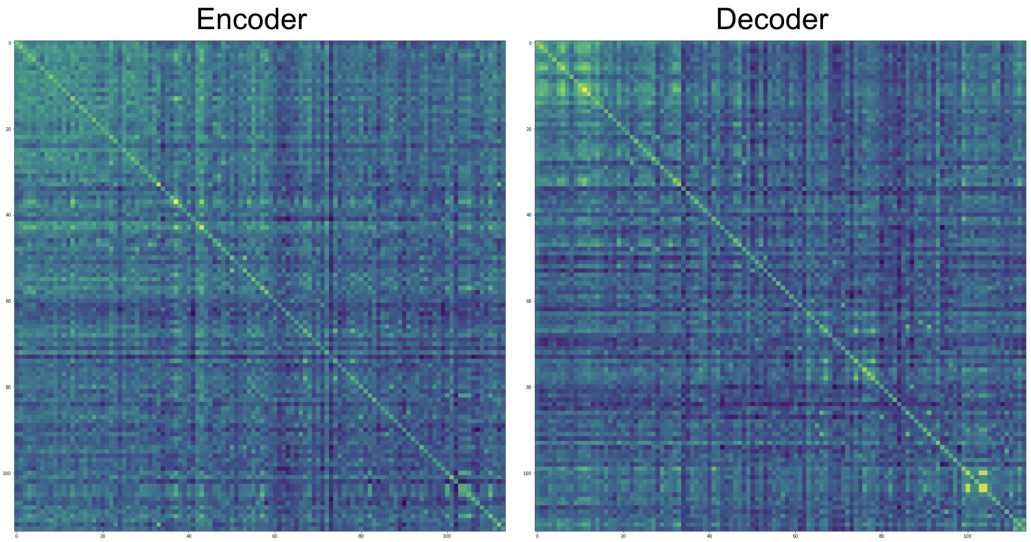


Figure 17: Correlation matrix of attention score maps of the first heads. The samples used are from the test split.

The results presented correspond to the first head of the four used. In all heads the results are very similar. Mainly, a diagonal corresponding to the autocorrelation of value 1 is observed. However, there is not a clear correlation between the models that obtain better prediction accuracy (lower right corner of the correlation matrix) and respectively with those of lower value (upper left corner of the correlation matrix). Certain correlation dynamics can be appreciated, where certain proximate groups obtain similar correlation values. In both plots it can be seen that the models that obtain a lower performance are more correlated to each other, however there is no clear interpretation. The same conclusions can be extracted for the case of the decoder's attention block. In overall terms, no clear correlation has been found that identifies that the model is attending differently according to the structure of the weights.

In situations where model weights can be affected by distortion, for example in communications, it can be interesting to know how robust the model is to compress and decompress these values in a noisy environment. It is also a form of understanding how it affects the fact that there is structure in the weights, where the quality of the reconstruction can differ depending on the checkpoint from where the model comes from (different weight convergence). In order to further investigate the properties of transformer-based attention auto-encoder and how it behaves under certain noise conditions, a comparison has been made. Three options are proposed for evaluation based on the AttnAE model:

- Model trained only using permutations as a data augmentation technique in training time.
- Model trained using permutations and applying a Gaussian noise of 10dB signal to noise ratio (SNR) on the input vectors that build the model weights.
- Model trained using permutations and applying a Gaussian noise of 10dB signal to noise ratio (SNR) in the latent space (reduced dimension vector, bottleneck)

In these three cases, two experiments were conducted. First, the R^2 score of the reconstruction was measured on the test partition by applying noise to the input vector. Secondly, the same approach was performed but in this case applying the noise in the latent space vector (bottleneck of size 20). Finally a noise of 10 dB SNR has been selected (applied in the latent space in the test partition) and the behaviour has been evaluated of the three models mentioned for each group of chekpoints (a total of 74 checkpoints' groups).

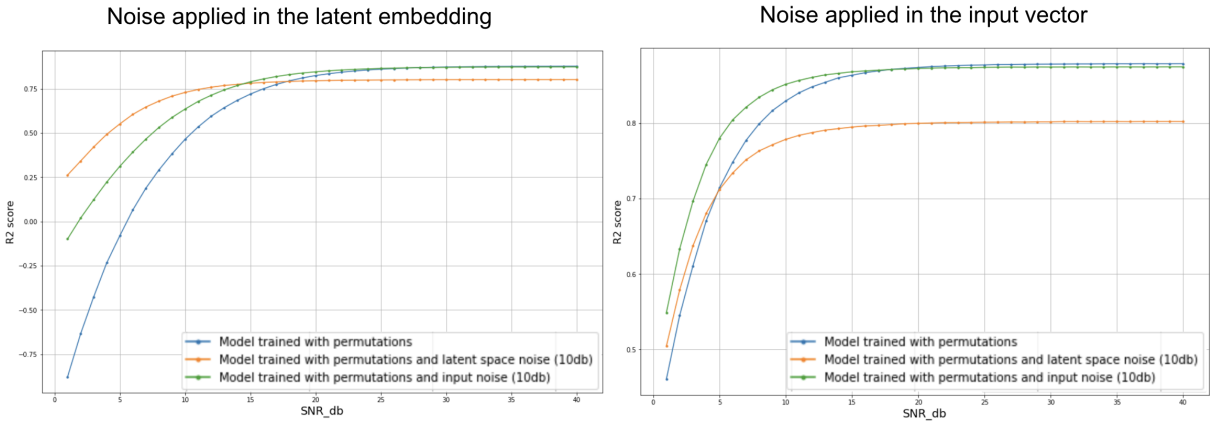


Figure 18: Comparison of attention auto-encoder models for weight reconstruction. Reconstruction R^2 score over different SNR when applying Gaussian noise on the latent embedding (left image) and on the input vector (right image) in test time.

When noise is applied to the latent sapace (see Figure 18), two main regions can be differentiated. The first one is found for low SNR values (the noise is high compared to the signal) where the model that has not been trained with noise in any of the presented methods obtains worse scores. The best model in this aspect is the one that has been trained specifically for this task. Secondly, when the SNR is high (signal is considerably higher than noise) the model that is able to obtain better results is the one that has not been trained with noise.

On the other hand, when noise is applied directly to the vector of weights that define the models, a slightly different behaviour is observed. In this case the model able to obtain better results for low SNR values is again the one that has been entered with noise in the input vector. However, when the SNR is high this last model obtains results practically similar to the model that has not used noise in the training.

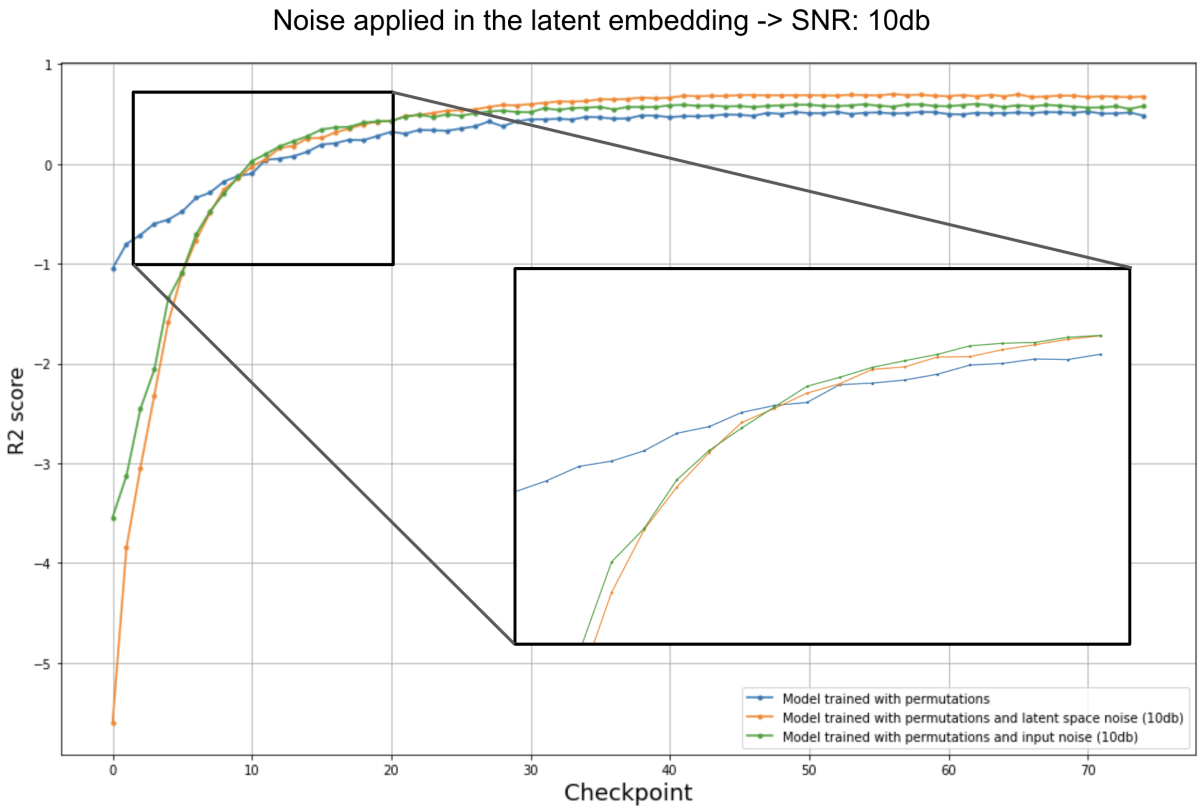


Figure 19: Comparison of attention auto-encoder models for weight reconstruction. Left figure: Reconstruction R^2 score performance over different Signal to noise ratio (SNR) when injecting Gaussian noise on the latent embedding in test time. Right figure: Reconstruction R^2 score performance over different Signal to noise ratio (SNR) when injecting Gaussian noise on the input vector in test time.

The results show the reconstruction model is very sensitive to noise. Even the models that have been trained to take noise into account in their multiple configurations have a low robustness to it. However, the model that was trained applying noise in the input vector presents a middle point, improving when there is more interference and obtaining practically the same results when there is no noise. The main conclusions are that the weights are very sensitive for reconstruction tasks, where the AttnAE model is not able to correctly capture the interactions.

The results presented previously refer to all the samples from all the checkpoints. However, as the values for each of the groups are computed (Figure 20) it is possible to see a difference in the results. The model that has not been trained with noise obtains a better robustness in those models that obtained a low accruacy prediction (in general they correspond to the groups of the initial checkpoints where statistically the weights have not yet converged). However, when the model is evaluated for the groups that in general have converged, the best model is the one that was trained to deal with noise in the latent space, followed by the one that was trained with noise applied in the input vector.

In conclusion, the models that were trained with noise obtain a lower score for reconstruction mainly due to a significantly poor performance in the lower groups. However, if only the converged models are considered (only the converged dataset samples, corresponding to the highest checkpoints) and therefore have a higher structure, the noise-trained model applied in the latent space is the one with the best capacity to capture the relationships among neurons.

4.4 Google dataset experiments

As the last and most complex dataset, we apply my architecture on the published dataset from Google [1] in their four splits of models trained on mnist, fashion-mnist, cifar10 and svhn. Results presented in the tetris-seed and tetris-hyper datasets showed the capability of the attention architectures to successfully work in this domain. The training times have been significantly increased by working with a significantly larger and a more complex dataset. In their work they did not neither present tasks of reconstruction nor hyperparameter prediction. In our case, we have tackled all these downstream tasks as well as the accuracy prediction.

4.4.1 Model exploration and code validation

We begin with reproducing the results from the paper *Predicting neural network accuracy from weights* [1] from the google paper described in the state of the art. I decided to implement their approach to validate that the metrics and setup is working properly. In [1] two technologies were used (deep neural networks and gradient boosting machines) to predict accuracy. They also use two methodologies to perform the computation, using the vector with all weights of each model concatenated and using a basic statistical vector. The results presented by the authors show that both technologies work better using the statistics.

In our case a vanilla MLP was implemented similar to the DNN presented in the paper in order to predict accuracy of the model. In addition, we implemented different well known basic architectures to see their behaviour in comparison. Since in our work we have focused on the understanding of weights, we have only considered to use directly the weights' vector of the models. Except for my vanilla DNN, the other models (have been designed to be able to deal with embeddings that represents the entire model) have used chunks of the the weights' vector. In this first approach the weights' vector were splited into equal size chunks. The results presented is using four chunks in total to represent each sample. In all the models the accuracy prediction was achieved using one last neuron plus a sigmoid function. All trainings used MSE as a cost function:

- **DNN**: Vanilla MLP of ten layers. Dropout set to 0.1 and no norm layers.
- **GRU+Attention**: Vanilla bidirectional GRU sequence to sequence. From the resulting sequence attention layer has been used to generate a 128 size embedding.

- **Transformer Encoder + MLP**: After the transformer encoder seq2seq layer, all the embeddings have been concatenated and a vanilla MLP has been applied.
- **Tranformer Encoder* + MLP**: Same approach used in the regular Transformer encoder but in this case using sine/cosine position embeddings. For each of the chunks a different MLP has been used to generate embeddings to feed the transformer encoder.
- **MLPLayer + SUM**: Different MLP have been used to generate each embedding. Afterwards all the embeddings are equally added.

	Architecture	Model Size	Input Type	MNIST R^2	SVHN R^2
Google Paper [1]	Baseline GBM	-	weights	0,988	0,971
	Baseline GBM	-	weights statistics	**0,993**	**0,986**
	Baseline DNN	∼1.5-3.6M	weights	0,980	0,931
Our implementation	DNN	∼1.8M	weights	0,983	0,946
	GRU+Attn	∼2M	weights	0,985	0,948
	Transformer Encoder	∼2M	weights	0,989	0,920
	Transformer Encoder*	∼54M	weights	0,993	0,975
	MlpLayer + Sum	∼800K	weights	0,990	0,975

Table 8: Comparsion between results from Google Paper and our vanilla implementation in the MNIST and SVHN splits. We have either not use cross validation nor different realisations. Just for validation purposes.

The results in Table 7 indicate that, the reproduced DNN architecture obtained similar results to those reported by the authors. Our custom vanilla DNN is in the range of model size with respect to the baseline. Without entering into too many details, the different architectures used obtain a very similar result. However, their functioning is different compared to the vanilla DNN. The main property is that they work with tokens that describe the models. This allows us to develop models capable of working with variable inputs in size.

Table 8 it can be seen that regardless of the number of parameters of the models, similar results are achieved. We cannot state that those models work the best for this domain

because we did not extensively fine-tune them. This scores make us think that with this dataset (google dataset) it is really easy to predict the accuracy.

Taking into account that there is a wide variety of architectures, it is necessary an approach that is able to work with variable input size. Moreover, since according to the first results in Table 8 the models based on attention mechanisms work well in this field, we have decided to explore into this direction. Since models can be very large, we have discouraged the use of LSTM or GRU based systems due to the issue with vanishing gradients.

4.4.2 Model weights reconstrucction

In the dataset tetris-seed the reconstruction task has been performed using neuronGroup to generate the tokens that describe each of the models. For this reason, it was decided to use the same approach for the Google dataset. PCA (with linear kernel) was used as the linear method and AttnAE II as the non-linear method. Permutations were also used in the training split as data augmentation technique. The results were evaluated by computing the R^2 score between the reconstructed weights and the initial weights for all the samples coming from all the checkpoints in the test set.

For the transformer-based model, it was necessary to modify the training method. Its size was increased to 2.5M parameters, using 4 blocks $N = 4$, and four heads for each one $h = 4$. The learning rate was decreased $lr = 1e^{-5}$ and it was reduced every 150 epochs with a ratio 10:1. It was trained for 600 epochs. The size of the latent space was set at 128.

Using the linear system for a bottleneck measurement of 128, a R^2 of 0.225 was achieved in test on the MNIST partition. In the case of the non-linear model a score of 0.214 R^2 has been achieved. The use of permutations has been crucial to make the model converge, however the results have not been as expected. Different configurations of the hyperparameters, modification of the bottleneck size and different schedulers were used to modify the learning rate, however in no case it has been possible to exceed the baseline imposed by the PCA. The results are similar for all the dataset partitions (Fashion-Mnist, svhn, cifar10 and mnist). In addition, all models present a great instability between train and test partitions being very sensitive to small modifications. Due to the limited time available to complete the thesis, it was decided not to continue in this direction.

4.4.3 Downstream tasks: accuracy and hyper-parameter prediction

In [1] they predicted the accuracy from the weights and from the basic statistics. In order to tackle this task, an adaptation of the AttnAE II model was used to predict accuracy. The same approach described in the tetris-seed dataset to predict the accuracy of the models (the encoder part of the attention auto-encoder with neuronGroup embeddings) was used. As explained in the tetris-seed downstream section, a last layer plus sigmoid function has been applied from the bottleneck vector to predict accuracy values in the range of $[0, 1]$. The metric used is R^2 score computed between the predictions and the real value of the accuracys for all the samples from the test set.

In this dataset the first layers of each sample in the collection of models are convolutional layers. For this reason, the neuronGroup embedding was applied using all the weights that compose each kernel for each of the dimensions. In this case, the weights were flattened and 2-dimensional position information has not been included. In addition, the value of the bias corresponding to each kernel was added at the end of each of these vectors. The number of embeddings necessary to represent the model is composed by 58 tokens (3x16 corresponding to the three initial convolutional layers and 1x10 for the last dense layer). Finally, the model was composed of a two global information context tokens plus the 58 representative tokens of the model.

Architecture	Input type	MNIST	SVHN	CIFAR10	FASHION
DNN Google Paper [1]	weights	0,980	0,931	0,954	0,980
AttnPred neuronGroup (ours)	weights	0,992	0,972	0,975	0,990
GBM Google Paper [1]	weights statistics	**0,993**	**0,986**	**0,984**	**0,993**

Table 9: Comparsion accuracy prediction results between our approach (AttnPred) and the best experiments obtained in the google paper for their respective input type. Values are expressed in R^2 score values. DNN from [1] is composed of 1.5-3.6M parameters. AttnPred is composed of 600K parameters.

The results show in Table 9 better accuracy prediction performance in all four splits of the dataset compared to the models that were trained using the weights vector. However, the performance is not as good as the best approach the authors performed, using GBM applied on the statistics vector layer-wise. However, the main characteristic of our model

with respect to the proposed ones is that it is designed to be able to predict models of variable length and it has 600K prameters instead of 1.5-2.6M of the Gools DNN [1].

The structure generated in the weights space during training tends to be highly correlated with accuracy. The basic statistics show good behaviour in capturing the information needed for prediction. It is possible that the attention-based model is learning to generate these statistics for prediction rather than modelling the connections between the weights. This could explain the similarity of the results without surpassing them.

This correlation between weights structure and accuracy may be due to the fact that the generated models have been trained only on one dataset. Furthermore, we are assuming that the partitions split between train and test come from the same source. Once the model is trained, the accuracy obtained will also depend on the source of the image used in the test. The same model with defined weights will have a different accuracy depending on the test dataset. The results presented show that the statistics work well in predicting the performance of the model using the same dataset, however, not understanding the behaviour of the model is probably not enough to predict its performance on other datasets.

In [1] there is no data regarding the predictions made on the hyper-parameters that identify each of the models, however in the dataset the authors provide all the information concerning their training and architecture. For each sample they provide the type of activation, initialization and optimizer function used as well as the epoch where the model becomes. In particular they provide models in just nine different groups (0, 2, 3, 4, 20, 40, 60, 80, 86).

To perform the hyperparameter prediction task it was approached as a classification problem. However, in the epoch prediction task, we decided to approach it as a regression because there is a difference between classifying an epoch closer to or further away from the true value. Two predictors were used, a log-linear classifier (using the statistic's vector) and a non-linear architecture (using directly the weights). The same architecture used for accuracy prediction was adapted to obtain the number of neurons corresponding to the number of classes to predict. We apply softmax on the logits and assign labels on the highest prediction value. To evaluate the predictor the F1 score was used (to measure the accuracy and the recall in the same metric). For each of the tasks a specific classifier was trained, minimising the binary cross entropy function as a cost function.

Classification Labels	Architecture	MNIST	SVHN	CIFAR10	FASHION
Activation (2 classes)	Linear (weights_s)	0,822	0,823	0,807	0,839
	AttnPred (neuronGroup)	**0,903**	**0,924**	**0,894**	**0,912**
Init method (5 classes)	Linear (weights_s)	0,735	0,749	0,709	0,773
	AttnPred (neuronGroup)	**0,946**	**0,929**	**0,941**	**0,957**
Optimizer (3 classes)	Linear (weights_s)	0,674	0,700	0,679	0,696
	AttnPred (neuronGroup)	**0,786**	**0,740**	**0,742**	**0,806**
Epoch (9 classes)	Linear (weights_s)	0,280	0,290	0,268	0,287
	AttnPred (neuronGroup)	**0,324**	**0,319**	**0,313**	**0,312**

Table 10: Hyper-parameters classification results using an adaption of AttnAE II named AttnPred. The metric used to evaluate the performance is F1 score. The results presented are the average of five different splits between train and test (scores obtained from the test split with checkpoints group balanced). The variance in all fields is less than 0.001.

Using attention-based neural networks it is possible to identify the different properties that compose each of the models only from the information of the weights as shown in Table 9. In all cases the classes are equally balanced. In all the hyper-parameters under study, scores higher than random guessing were obtained. We found the model with $N = 4$ and $heads = 4$ (190k parameters) to be the best fit to obtain the best performance found on the classification task. In all hyper-parameters and all datasets the non-linear attention-based model has obtained better results than the linear baseline. However, we consider that the difference in performance is not too large considering the computational complexity realised by the DNN-based model. For epoch prediction the result obtained by the log-linear prediction was entered for classification since as a regressor the results were equal to random guessing. In the case of the AttnPred the results presented were trained as a regressor to obtain better results than as a classification.

For each of the predictors we have evaluated their performance by plotting the recall score for the predictions of the initialisation method and the activation and optimiser functions for each of the groups composed by different epochs. The figure 21 shows the

recall obtained for each of the classes during training. For the epoch prediction evaluation, the confusion matrix including the distribution of predictions for each group of models coming from the same epoch is shown in Figure 22.

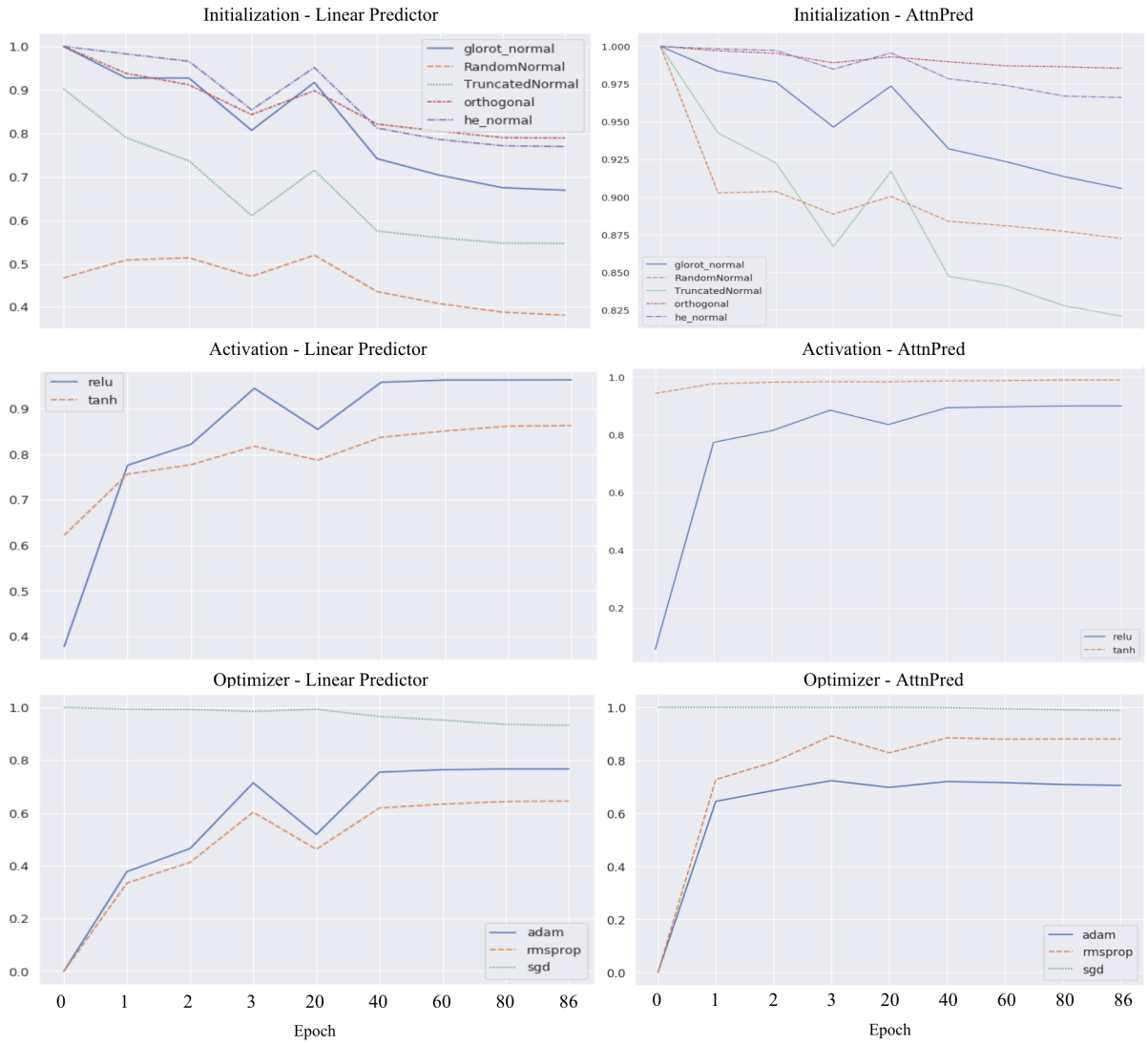


Figure 20: Recall curves for hyper-parameter classification on the Google dataset. Right column are results for the log-linear predictor. Left column are results for the attention based architecture predictor, AttnPred.

The Google dataset [1] is composed of models initialised by five different methodologies using the same seed. For this reason, the group corresponding to epoch zero has only five possible configurations of weights. From this point on, depending on the configuration used for the hyperparameters of each model, it progresses in a different manner. Each of these configurations share a set of rules that correlate the weights with its properties

of the models that are configured the same way. Figure 20 shows how the non-linear model is able to perfectly classify the first group. However, the linear regression-based model does not manage to group them correctly. In the two predictors the two worst predicted initialisation functions coincide, being truncatedNormal and random Normal. This may make sense given that these distributions are very similar to each other. In general, the linear predictor performs worse in the epochs 20, 40, 60 where there is more variety of models that are at stages further away from convergence and some that are already converged (see Figure 21). The non-linear predictor predict better the classes in all hyper-parameter classification tasks.

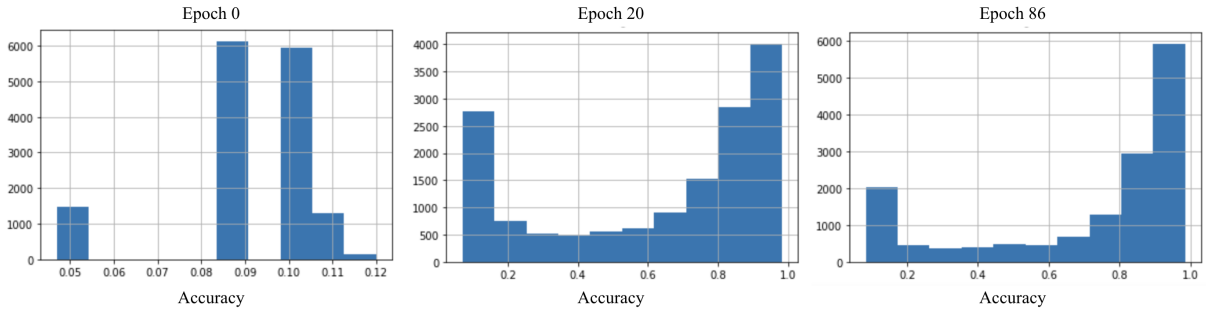


Figure 21: Accuracy distribution at different epoch groups of the google dataset [1]

In the case of epoch prediction, the regression-based implementation for the AttnPred model was chosen instead of classification because of its better performance. In general the results compared to the other hyper-parameters classification tasks are much worse. It is more difficult to predict in which epoch the model comes from just by looking at the weights space. The amount of update weights may not be proportional to the convergence of such models. One model may obtain the same solution trained for only two epochs while another model may use a longer path requiring more updates to arrive at the same point. Conversely, a model can be trained indefinitely and never converge to a good solution. At the distribution of predictions (see figure 22) for each epoch it can be seen that the model tends to predict values close to the correct ones. The variance of the predictions is higher in the central range, coinciding in the region with more diversity of converged and non-converged models.

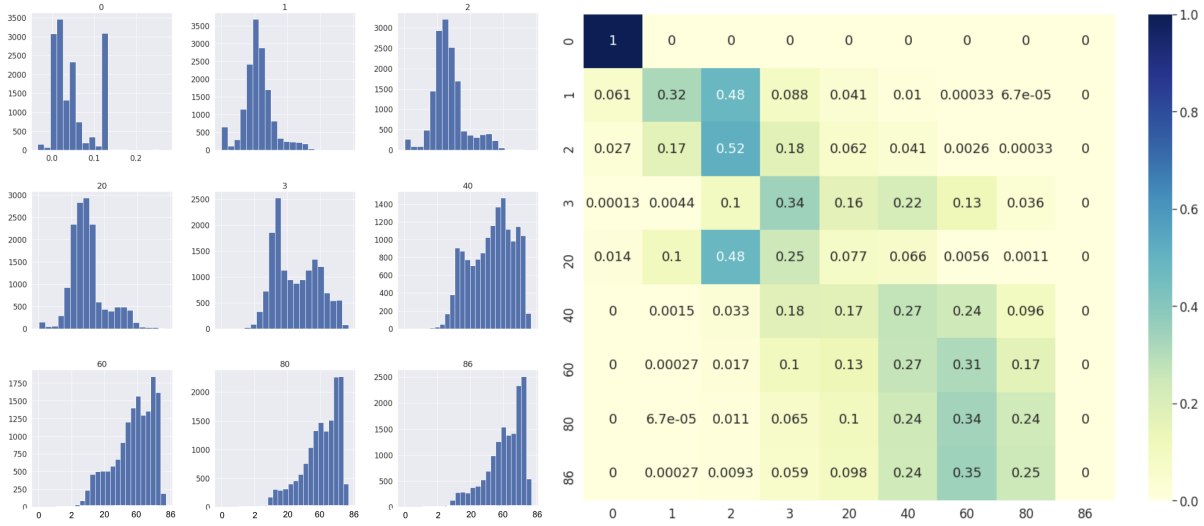


Figure 22: Attention based architecture for epoch prediction on the Google dataset. Right images are the epoch prediction distribution for each of the epoch groups. Left image is the confusion matrix.

5 Environment impact

In recent years the use of neural networks has increased exponentially, mainly due to the improvement of the hardware that makes possible the large number of necessary calculations on this kind of processes. In addition, a large number of data centers have also been built and expanded. Predictably, in the coming years we will continue using these technologies due to its great impact in many areas and its good performance.

A distinction must be made between energy consumption and the method of production. The main source of energy used is electrical energy, which will be produced differently depending on the country where the computer centers are located. There are countries where renewable energy production sources are used and other places that are more dependent on non-renewable sources. This thesis have been developed in Switzerland. Although most of the energy consumed in the country comes from non-renewable source of energy [11], 25% is from electricity production in the country. This electricity is mainly produced by renewable source of energy, around 75% in 2020.

In this work we have used deep neural networks for development and understanding of this technology itself. Due to the nature of neural networks, with the technology we have today, it can be considered a high energy expenditure. In order to reduce the total energy consumption it is necessary to intervene and develop a variety of aspects (from process optimisation to hardware energy efficiency), which together will help to reduce energy consumption. For this reason, although neuronal networks have been used in this thesis, this work is focused on the line of explainable artificial intelligence, so deciphering and knowing better its operation can be very beneficial in the future. Knowing how the models are trained and how the solutions are achieved can help in the future to develop more efficient methods that require less training than in the present. In addition, to make a complete study on the amount of energy used, we should also take into account what is the impact that this development produces and and how these improvements will change energy consumption in the future.

6 Conclusions

In this thesis we have gained insight into the dataset exploration as well as the performance of attention mechanisms on representation learning. Depending on the task to be performed, the statistics synthesize the information achieving the best results in accuracy prediction. There is a high correlation between the global statistics and their performance. For hyper-parameter classification on predicting the activation function the statistic's transformation does not help for its proper interpretation. On the other hand, for the initialization function prediction best results have been obtained. The hyper-parameters used during training define a subspace where the weights acquire structure during training. Therefore, at the beginning the neural networks are composed of weights obtained from a certain distribution. As the model converges, the weights are modified by the rules that are characteristic of the hyper-parameters as well as the architecture and the training data.

The results obtained show that it is possible to predict the characteristics of the neuronal networks and their performance from their weights. It has been seen how reconstruction tasks are also possible with a high level of compression, however it was not been possible to establish an improvement over a linear approach in the google dataset. In addition, the AE embeddings keeps significant amount of information so that they can perform downstream tasks on it with a similar performance to using directly the weights. To obtain better results, weights and neurons permutations were used as data augmentation technique, which was found to be a fundamental component in allowing some models to converge.

Deep neural networks based on attention mechanisms have shown good behaviour. AttnAE architecture uses the structure in the weights to perform tasks of reconstruction, accuracy prediction and hyper-parameter classification. Using the neuronGroup encoder (generating an embedding with all the weights that constitute each of the neurons in a model) and the use of transformer encoder blocks allowed to find a solution that scales much more efficiently than a traditional multi-layer perceptron (MLP) and with a higher convergence speed. It was demonstrated that this approach outperforms the DNN baseline as presented in [1] and it is competitive against GBM. The field of neural networks is characterised by a great variety of structures with different parameters. Tokenizing these models allow working with variable input sizes which the proposed architecture is designed for it.

6.1 Future work

During the development of this thesis we have mainly encountered three unsolved questions. On the one hand, it was not possible to establish a direct relationship between the attention score maps and the performance of the models. Unlike other tasks such as in the field of NLP, no clear patterns have been identified that define a characteristic behaviour understandable by humans.

On the other hand, an intrinsic feature of neural networks is their ability to offer multiple different structures and very different dimensions. Although in this work a more suitable solution than vanilla MLP was proposed, it is still a direction to be explored. It would be very interesting to develop an elastic and global form that allows to properly encode the information of the models. It could also be explored an efficient method of capturing relations that exists in the different parts of a model and how to model the interaction among weights.

Finally, no satisfactory results have been achieved in the task of reconstruction in the google dataset. We think that a possible line of development could be based on improving the current model to improve the reconstruction. In particular, we think that the decoder part should be further developed. Furthermore, the interaction between the models in an unsupervised manner can be investigated. By using permutations of the same models we could think of using similarity learning which could be a way of not inducing bias towards the modelling of statistics. Using the model described in this thesis we could continue in the direction of predicting the characteristics of the models. This could allow the prediction of their behaviour in early steps during training. Developing a system that is able to quantify the probabilities that such a model becomes a good solution could reduce the number of trainings to be performed.

References

[1] Thomas Unterthiner, Daniel Keysers, Sylvain Gelly, Olivier Bousquet, and Ilya Tolstikhin. Predicting neural network accuracy from weights. *arXiv preprint arXiv:2002.11448*, 2020.

[2] Eilertsen. et al. Classifying the classifier: dissecting the weight space of neural networks. *arXiv preprint arXiv:2002.05688*, 2020.

[3] Charles H Martin and Michael W Mahoney. Traditional and heavy-tailed self regularization in neural network models. *arXiv preprint arXiv:1901.08276*, 2019.

[4] Surafel Melaku Lakew, Mauro Cettolo, and Marcello Federico. A comparison of transformer and recurrent neural networks on multilingual neural machine translation. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 641–652, Santa Fe, New Mexico, USA, August 2018. Association for Computational Linguistics.

[5] Albert Zeyer, Parnia Bahar, Kazuki Irie, Ralf Schluter, and Hermann Ney. A comparison of transformer and lstm encoder decoder models for asr. pages 8–15, 12 2019.

[6] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[7] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[9] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020.

[10] Ronald M Parra-Hernández, Jorge I Posada-Quintero, Orlando Acevedo-Charry, and Hugo F Posada-Quintero. Uniform manifold approximation and projection for clustering taxa through vocalizations in a neotropical passerine (rough-legged tyrannulet, phyllomyias burmeisteri). *Animals*, 10(8):1406, 2020.

[11] The Federal Council. Energy – Facts and Figures. `https://www.eda.admin.ch/aboutswitzerland/en/home/wirtschaft/energie.html`, 2019. [Online; accessed 14-December-2020].

[12] KENNETH WARD CHURCH. Word2Vec. *Natural Language Engineering*, 23(1):155–162, 2017. Publisher: Cambridge University Press.

[13] Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, October 2014. Association for Computational Linguistics.

[14] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *arXiv:1802.05365 [cs]*, March 2018. arXiv: 1802.05365.

[15] Rahul Dey and Fathi M. Salem. Gate-variants of gated recurrent unit (GRU) neural networks. *CoRR*, abs/1701.05923, 2017.

[16] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008.

[17] Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

[18] A. Adadi and M. Berrada. Peeking inside the black-box: A survey on explainable artificial intelligence (xai). *IEEE Access*, 6:52138–52160, 2018.

[19] David Gunning. Explainable artificial intelligence (xai). *Defense Advanced Research Projects Agency (DARPA), nd Web*, 2(2), 2017.