REGRESSION AND CLASSIFICATION USING HIGH ORDER POLYNOMIALS AND TENSOR DECOMPOSITIONS

Raül Pérez i Gonzalo, Michael Rahbek, Kim Reinhardt Jensen, Jørgen Taule

Technical University of Denmark, DTU Compute, Building 324, 2800 Kongens Lyngby

ABSTRACT

In this project, we seek to build machine learning models based on high order polynomials using tensor decompositions to significantly reduce the number of parameters in the model. We have investigated CP, tensor train and tensor ring decompositions and shown that models based on these may be used to approximate continuous functions in a low-dimensional space. All of the algorithms do, however, suffer from significant numerical instabilities that presents itself for data containing a large number of features. We have shown that these instabilities may be alleviated but not eliminated by using a correction factor in the forward pass. The models were tested using the California housing and Make Blobs dataset for regression and classification respectively, where their performance was compared with a neural network and tensor train algorithm using alternating least squares from [1].

Index Terms— Supervised learning, tensor train, tensor ring, CPD, polynomial regression, polynomial classifier

1. INTRODUCTION

Regression and classification are supervised learning models that require a number of observations for them to be able to predict the target value or class of a new observation. As polynomial functions form a set of universal function approximators [2], they may be very useful when creating a model. However, a well performing polynomial regression or classification model requires high order polynomials, which requires a large number of parameters.

Tensor decompositions can be used to reduce the effect of the curse of dimensionality while still retaining much of the information contained in the full tensor. The learning algorithms focuses on the smaller number of parameters in a tensor decomposition. This way the model complexity is lower, and the chance of overfitting is reduced.

There are different tensor decompositions. In this paper, the focus will be on the *tensor train decomposition* (TT), the *tensor ring decomposition* (TR), and the *canonical polyadic decomposition* (CP). All of them have been optimized using gradient descent. The goal is to compare their performance and the number of free parameters of the different models, and compare them to a *feed forward neural network* (FFNN) for classification and the TT optimized using *alternating least squares* (ALS) from the paper [1] for regression.

2. TENSOR DECOMPOSITION

In this paper, we use the following notation. Calligraphic letters for tensors $\mathcal{A}$, uppercase letters for matrices A, bold lowercase letters for vectors **a** and lowercase letters for scalars a. We denote the k-mode product of a tensor by $\times_k$ [1][3].

2.1. Preliminaries

Given a d-dimensional data observation $\mathbf{x} = (x_1, ..., x_d)$ with an o-dimensional target **y**, the so-called Vandermonde vector for each dimension is the power vector of x_k. The order of the power expansion n_k or *poly-order* is a hyperparameter that may be tuned.

$$\mathbf{v}(x_k) = \left(x_k^0, x_k^1, ..., x_k^{n_k}\right) = (1, x_k, ..., x_k^{n_k}) \,. \tag{1}$$

Hence a polynomial function contraction can be written as:

$$\mathbf{f}(\mathbf{x}) = \mathcal{A} \times_1 (\mathbf{v}^{(1)})^T \times_2 (\mathbf{v}^{(2)})^T \times_3 ... \times_d (\mathbf{v}^{(d)})^T. \tag{2}$$

Here $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times ... \times n_k \times o}$ where o is the output dimention is a tensor containing the weights of the polynomial function. This is the tensor we are interested in decomposing such that we can compute the function contraction $f(\mathbf{x})$ without explicitly construct $\mathcal{A}$.

2.2. Tensor train decomposition

The TT decomposes a tensor into d 3-order tensors called TT-cores $\mathcal{G}_k \in \mathbb{R}^{r_{k-1} \times n_k \times r_k}$ [4]. r_k is called the TT-ranks where $r_0 = 1$ and $r_d = o$. The rest of the TT-ranks are hyper parameters. This decomposition gives the tensor elements:

$$\mathcal{A}_{i_1, i_2, ..., i_d} = \mathcal{G}_1(i_1)\mathcal{G}_2(i_2)...\mathcal{G}_d(i_d), \tag{3}$$

where the matrix $\mathcal{G}_k(i_k) \in \mathbb{R}^{r_{k-1} \times r_k}$ is the i_kth lateral slice of tensor $\mathcal{G}_k$. This gives the function contraction:

$$\mathbf{f}(\mathbf{x}) = \left(\mathcal{G}_1 \times_2 (\mathbf{v}^{(1)})^T\right) ... \left(\mathcal{G}_d \times_2 (\mathbf{v}^{(d)})^T\right). \tag{4}$$

2.3. Tensor ring decomposition

The TR decomposition is closely related to the tensor train decomposition with the difference being that all the cores in the tensor ring are of the same order s.t. the first core $\mathcal{G}_1$ and the Nth core $\mathcal{G}_n$ have a shared mode [5]. In relation to regression and classification, the function contraction is shown below with the cores being 3-way arrays.

$$\mathbf{f}(\mathbf{x}) = \mathcal{G}_N \times_3 [\left(\mathcal{G}_1 \times_2 (\mathbf{v}^{(1)})^T\right)\left(\mathcal{G}_2 \times_2 (\mathbf{v}^{(2)})^T\right)\dots \left(\mathcal{G}_{N-1} \times_2 (\mathbf{v}^{(N-1)})^T\right)].$$

Please note, the contraction of $\mathcal{G}_{N-1}$ and $\mathcal{G}_N$, which completes the ring is not shown. The last core, $\mathcal{G}_N$, does not have a corresponding Vandermonde vector as its second mode has the same size as the output o. $N-1$ would then be the number of features. The size of mode-1 and mode-3 of the core tensors may be considered a hyperparameter for the model.

2.4. Canonical polyadic decomposition

CP decomposes a tensor into a d order diagonal tensor $\mathcal{D}$. The dimensions of $\mathcal{D}$ are equal to the rank R of $\mathcal{A}$ and is a hyper parameter. $\mathcal{D}$ is mode multiplied by matrices $\mathcal{B}_k(i) \in \mathbb{R}^{n_k \times R}$ which is the horizontal slice of the tensor $\mathcal{B}_{||} \in \mathbb{R}^{o \times n_k \times R}$.

$$\mathcal{A}(i) = \mathcal{D} \times_1 \mathcal{B}_1(i) \times_2 \mathcal{B}_2(i) \times_3 \dots \times_d \mathcal{B}_d(i). \quad (5)$$

Using the rule: $\mathcal{X} \times_n A \times_n B = \mathcal{X} \times_n (BA)$, given in [6], we get the function contraction:

$$\mathbf{f}(\mathbf{x}) = \mathcal{D} \times_1 \left(\mathcal{B}_1 \times_2 (\mathbf{v}^{(1)})^T\right) \times_2 \dots \times_d \left(\mathcal{B}_d \times_2 (\mathbf{v}^{(d)})^T\right). \quad (6)$$

We now define:

$$M_i = \mathcal{B}_i \times_2 (\mathbf{v}^{(i)})^T. \quad (7)$$

A diagonal tensor mode-multiplied with a vector gives a new diagonal tensor. We define $\odot$ as the element wise vector multiplication, and if we let $\mathcal{D}$ only contain ones in the diagonal, we can write the final function contraction as:

$$U = M_1 \odot M_2 \odot \dots \odot M_d, \;\; \mathbf{f}(\mathbf{x}) = \sum_i^R U_{:,i}\,. \quad (8)$$

3. METHODS

The algorithms made for this project (Github-link) were developed using PyTorch. The numerical experiments were done on the HPC from DTU, with an Nvidia-Volta-100 GPU with 16GB of memory.

The class `torch.nn.module` is used to define the models from the previous section. In the `_init_` the weights are initialized as the decomposed tensor, and in the `forward` pass, the function contractions are performed. After correctness was ensured, there was a focus on speeding up the algorithm by optimizing the forward pass.

The purpose of these algorithms is to test whether they can fit their models to a data set and compare these to neural network models. The models are tested for regression with the California housing dataset [7] and for classification with the Make Blobs data set [8]. With a fixed number of features and three combinations of hyperparameters, different optimizers are tested, and the best performing chosen. Afterwards, cross-validation is used to tune the hyperparameters, subsequently, the optimised models are compared.

3.1. Initialization

As stated in [1], the TT suffers from numerical instability, and consequently the CP and the TR. In order to simplify the following analysis, we assume that all ranks are equal $r := r_1 = \cdots = r_{d-1}$, and similarly for the poly-order $n := n_1 = \cdots = n_d$. To alleviate this issue, the TT-cores $\mathcal{G}_k$ are initialized such that the TT output has a zero expectation and a unit variance, two strategies are proposed:

i. Initialising the elements from a $N(0,\sigma)$, where $\sigma^2 = \frac{1}{n} \cdot r^{-\frac{d-1}{d}}$.

ii. In addition to the initialization strategy, we define the correction factors q_k and $\hat{q}_k$, which are computed in the first forward pass to ensure unit variance over the training set after each operation. Hence, the forward pass is modified such that each 2-mode multiplication is scaled by q_k, $V_k = G_k \times_2 (\mathbf{v}^{(k)})^T \cdot q_k$. Similarly, the tensor contraction $\prod_{k=1}^d V_k$ is modified to $\prod_{k=1}^d V_k \cdot \hat{q}_k$.

See Appendix A.6 for more details. Similar procedures were followed for the other two models, but are not stated in this report.

Lastly, the different models are initialized and trained 10 times to reduce the possibility that the model converges to local minima.

3.2. Forward pass

In both the TT and the TR, the forward pass can be broken down in to a series of independent mode-n products of a tensor $\mathcal{G}_i$ and a vector followed by tensor contractions of the cores. If all the cores have the same dimension, they may instead be performed as a single mode-n product of a tensor and a matrix:

$$\left.\begin{matrix} \mathcal{G}_1 \times_2 \mathbf{v}^1 \\ \vdots \\ \mathcal{G}_d \times_2 \mathbf{v}^d \end{matrix}\right\} \tilde{\mathcal{G}} \times_2 \tilde{\mathbf{v}}$$

where $\mathcal{G}_i \in \mathbb{R}^{r \times n \times r}$, $\tilde{\mathcal{G}} \in \mathbb{R}^{r \times d \times n \times r}$, $\mathbf{v}^i \in \mathbb{R}^n$, $\tilde{\mathbf{v}} \in \mathbb{R}^{n \times d}$ in the case where the cores are 3-way arrays, the 'stacked' cores

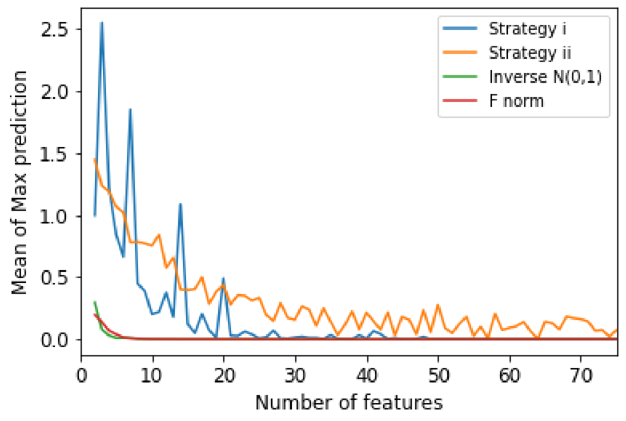

Fig. 1. Mean of the maximum output prediction of the TT after initializing it with several strategies. Results obtained with 1000 samples of artificial data.

$\tilde{\mathcal{G}}$ is then a 4-way array. Implementing the forward pass in this way provides speedup of about factor 3 compared with the naïve implementation, where the n-mode products are performed in a loop and the results are stored in a list. The same method has also been applied to the forward pass of the CP.

4. RESULTS

4.1. Initialization

To test the best initialization strategy, synthetic data X following $x_i \sim (N(1,0), N(0,1), \cdots, N(0,1))$ was worked with, using different numbers of features. Experiments on a classification problem with 5 classes were run. Furthermore, two baseline strategies were added: (1) initializing the TT-cores from a $N(0,1)$, and (2) initializing them as [1], i.e. normalizing each initial core s.t. its Frobenius norm is equal to one.

The numerical instability is prominent for a large number of features, when the TT contraction underflows resulting in a vector of zeros, for which the softmax function cannot predict any class properly. Thus, for each number of features, we computed the mean of the maximum prediction for each instance, to monitor the softmax outcome.

Figure 1 shows clearly how the proposed strategies are better at dealing with a much larger number of features than the baseline ones. Indeed, strategy ii outperforms all the other ones and has the smoothest behaviour, ensuring more stability in the initialization. Therefore, the subsequent results were obtained using strategy ii from section 3.1.

It is also important to remark that initializing the weights following a $N(0,1)$ is the only method that overflows, so Figure 1 contains the inverse of the maximum prediction for this baseline strategy. Hence, we can see that this method overflows faster than the others underflow.

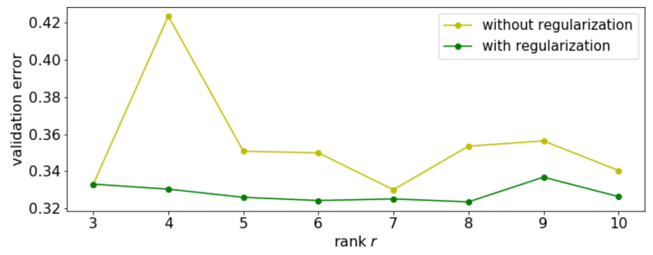

Fig. 2. Validation errors (minimas out of 10 runs) of a TT regression, shown for different ranks, poly order $n = 2$. The curve with regularization uses optimal parameters for L2-regularization (see A.2).

4.2. Regularization

The TT was run with L2-regularization, in order to show how the validation error behaves when using regularization. The validation errors are shown in Figure 2, which shows that it slightly improves the validation error. Weight decay adds, however, another hyperparameter, and testing the influence of this further is not in the scope of this article. The ranks and poly orders are special to the models discussed in this article, and will therefore be focused on in the following.

4.3. Optimizers

The optimizers that were tested to train the model are: LBFGS (`max_iter=20`), LBFGS (`max_iter=1`), SGD (`lr=0.2`), Adam, (`lr=0.1`). The LBFGS optimizers also take the *strong Wolfe conditions*, making sure that each iteration decreases the training error, and preventing divergence [9]. The `max_iter=1` parameter indicates how many iterations the optimizer does per epoch. The difference between the two LBFGS optimizers is therefore essentially just how many epochs is required for the training error to converge. The Adam and SGD optimizers take the learning rate as parameter, which controls how much the model parameters can be changed dependent on the change in the model error.

Figure 3 shows the training error as a function of epochs during training of the TT on the regression data set. The Adam and the LBFGS optimizers converges to approximately the same training error, while the SGD converges at a higher-valued local minimum than the other optimizers. Both LBFGS optimizers remain unchanged in all the runs after 1000 epochs, while the Adam is still non-stable.

The LBFGS using several iterations per epoch converges most quickly. The other optimizer that converged to the same error in Figure 3, Adam, is compared to the LBFGS for two other initializations of n and r (see A.1). The result is the same and therefore the LBFGS using several iterations per epoch will be used in the rest of the article.

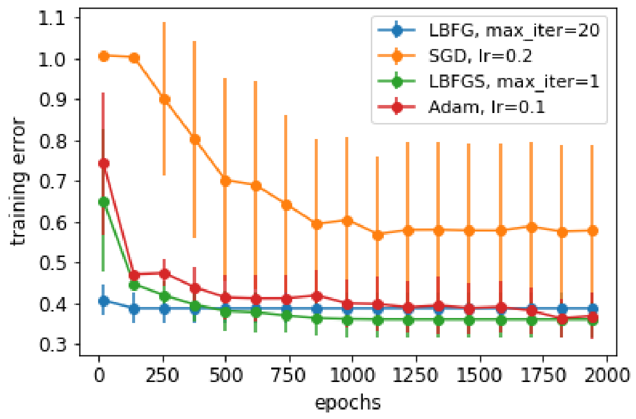

Fig. 3. Training error (MSE) on the regression problem using the TT with poly-order $n = 2$ and rank $r = 2$ obtained for the different optimizers. Each training was run five times, and the mean is plotted along the standard error as bars.

4.4. Function approximation over the training

Specifications of the datasets used can be found in A.3.

The TT was run for different ranks r and poly orders n on the classification problem, being able to determine whether the TT can converge to a zero-error for a big number of parameters. Figure 4 shows that for poly order $n = 2$, the training error decreases as the rank is increased from $r = 1$ to around $r = 5$. For higher ranks, the training error does not improve. For higher poly orders, $n > 2$, the number of parameters increases, and the learning does not converge (see also A.4). This is due to overshooting of the training. An specific tuning of the learning rate and the weight-decay could be enough to prevent this from happening. Even so, all this tuning implies that TT is nearly an unpractical universal model approximator.

4.5. Cross validation and comparison of models

Using a fixed number of instances and features, cross validation is performed over the rank and the poly order to choose the optimal hyper parameters for each model for both regression and classification.

This method is used to find all the optimal hyper parameters for the different models. Performance in regards to the MSE for regression and accuracy for classification along with the number of parameters for each model with optimal set of hyperparameter is shown in Table 1. The regression results imply that all our models outperformes the ALS, which is consistent with statistical testing (see A.5 Table 3). The CP also tends to perform better than TR and TT. For classification CP seems to be the worst performing of all the models, which is confirmed by statistical tests (A.5 Table 4).

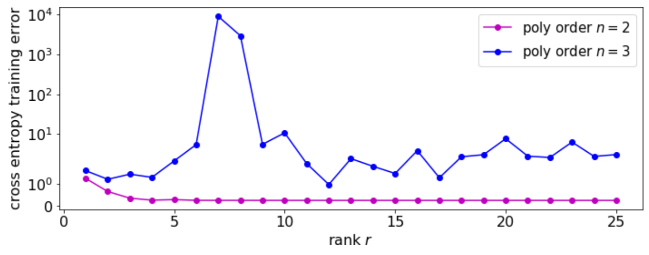

Fig. 4. Cross entropy training error of the TT on the classification problem, shown for different ranks r and poly orders $n = 1$ and $n = 2$.

Dataset	Model	MSE	Acc	Hyp1	Hyp2	Par.
California housing	TT	0.614	-	2	5	320
	TR	0.617	-	2	9	1377
	CP	0.52	-	2	14	224
	ALS	1.29	-	2	2	56
Make blobs	TT	-	88.76	2	10	920
	TR	-	88.81	2	4	272
	CP	-	87.29	2	3	180
	FFNN	-	88.51	3	0.1	9487

Table 1. Test results for optimal hyper parameters. For all models Hyp1 is poly-order and Hyp2 is rank, except for FFNN, where Hyp1 is the number of hidden layers (which 64 hidden units each one) and Hyp2 is the Dropout rate.

5. CONCLUSION

This report presents a framework containing three different tensor decomposition models for regression and classification problems. The regression experiments show competitive test errors, especially for CP. On the other hand, the classification results did not differ significantly, so no definite conclusion can be obtained. Thus, we suggest that these experiments must be repeated for another dataset.

In contrast to [1], our models are trained using gradient descent optimization, suffering from an additional number of hyperparameters that must be tuned. Even so, [1] also suffers from under-/overflowing problems during the training. Consequently, the proposed models are a set of universal function approximators that are impractical for big problems. For instance, in order to train a model with relatively small poly-order, it requires prior fine-tuning of the weight decay and the learning rate - along with a proper initialization.

Hence, we propose that these models should be applied for small datasets or as the last layer of a big ANN model, such as stacking our models at the end of a CNN. This should benefit the CNN, as our model will compute explicitly all the interactions between the weights of the hidden output. Additionally, it is also important to remark that [10] applies the tensor decompositions in a different way which should reduce the numerical instability.

6. REFERENCES

[1] Z. Chen, K. Batselier, J.A.K. Suykens, and N. Wong, "Parallelized tensor train learning of polynomial classifiers," in *IEEE Transactions on Neural Networks and Learning Systems*. IEEE, 2017, vol. 29, pp. 4621 – 4632.

[2] E. W. Weisstein, "Weierstrass approximation theorem.," `https://mathworld.wolfram.com/WeierstrassApproximationTheorem.html`, From MathWorld–A Wolfram Web Resource. Accessed 26-April-2020.

[3] Morten Mørup, "Applications of tensor (multiway array) factorizations and decompositions in data mining," *WIREs Data Mining and Knowledge Discovery*, vol. 1, no. 1, pp. 24–40, 2011.

[4] I. Oseledets, "Tensor-train decomposition," in *SIAM Journal on Scientific Computing*. SIAM, 2011, vol. 33, pp. 2295–2317.

[5] Yuwang Ji, Qiang Wang, Xuan Li, and Jie Liu, "A survey on tensor techniques and applications in machine learning," *IEEE Access*, vol. PP, pp. 1–1, 10 2019.

[6] Tamara G. Kolda and Brett W. Bader, "Tensor decompositions and applications," *SIAM Rev*, pp. 455–500, 2009.

[7] U.S. Census Bureau, "California housing dataset," 1990, data retrieved from Sklearn, `https://scikit-learn.org/stable/modules/generated/sklearn.datasets.fetch_california_housing.html#sklearn.datasets.fetch_california_housing`.

[8] "Make blobs data set," data retrieved from OpenML, `https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_blobs.html`.

[9] Jorge Nocedal and Stephen J. Wright, *Numerical Optimization*, pp. 33–36, Springer, New York, NY, second edition, 2006.

[10] Grigorios Chrysos, Stylianos Moschoglou, Yannis Panagakis, and Stefanos Zafeiriou, "Polygan: High-order polynomial generators," *ICLR 2020 Conference Withdrawn Submission*, 2019.

A. APPENDIX

A.1. Optimizers

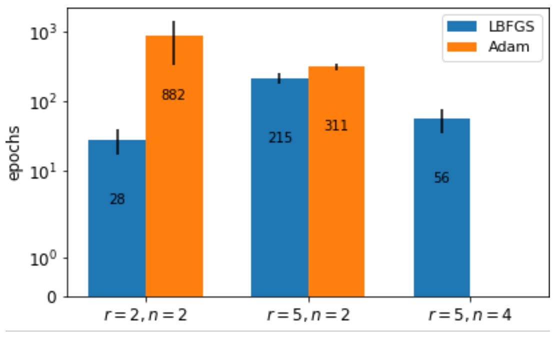

Fig. 5. Mean number of epochs required for the training error (mean squared error) to converge on the TT regression problem. This is used to determine the fastest optimizer across a number of settings (in this case, rather arbitrarily chosen). As seen, LBFGS requires fewer epochs in all the settings tested. Sometimes LBFGS reaches lower errors than the Adam (not shown in the figure). In addition, the Adam optimizer tends to diverge to a large training error after converging to a low value for several epochs. The bars show the standard error made from several runs. The Adam optimizer for $r = 5$ and $n = 4$ did not converge, even after 30000 epochs, and is therefore omitted here.

A.2. L2 regularization

The regularization parameters used in Figure 2 are shown in Table 2.

Rank	λ
1	1e-2
2	0
3	0
4	1e-7
5	1e-4
6	1e-3
7	1e-4
8	1e-4
9	1e-4
10	1e-10

Table 2. The optimal regularization parameters λ for L2-regularization used in Figure 2.

A.3. Dataset

The classification problem consists of 6 features, 20640 observations, and 5 classes (5 outputs), while the regression problem consists of 8 features, 20640 observations, and 1 output. Both sets are split into the following:

1. Training set: 14912
2. Test set: 3096 (15 % of the dataset)
3. Validation set: 2623 (15 % of the dataset after first split)

A.4. Function approximator

A corresponding plot as Figure 4 is provided in Figure 6, only with more poly order values and using the TR.

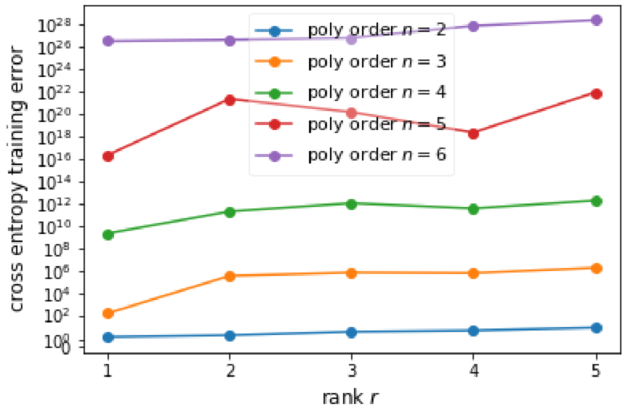

Fig. 6. Cross entropy training error of the TR on the classification problem, shown for different ranks r and poly orders $n=2$ to $n=6$.

A.5. Model comparison

Tables containing statistical tests on the results from Table 1.

Comparison	Confidence interval	p-value
TT - TR	$[-0.33, 0.28]$	0.44
TT - CP	$[0.12, 0.21]$	3.11e-13
TT - ALS	$[-1.91, -0.28]$	4e-3
TR - CP	$[-0.10, 0.49]$	0.10
TR - ALS	$[-1.59, -0.55]$	2.66e-5
CP - ALS	$[-2.07, -0.45]$	1e-3

Table 3. Regression statistical comparison. Made with Setup I, MSE pairwise t-test.

Comparison	Confidence interval	p-value
TT - TR	[-2.1e-3, 1e-3]	0.76
TT - CP	[1.1e-2, 1.8e-2]	3.1e-8
TT - FFNN	[-2e-4, 5.1e-3]	0.23
TR - CP	[1.2e-2, 1.9e-2]	2.3e-9
TR - FFNN	[4e-4, 5.5e-3]	0.13
CP - FFNN	[-1.6e-2, -8.7e-3]	2.2e-6

Table 4. Classification statistical comparison. Made with Setup I, accuracy Mcnemar test.

A.6. Initialization

After computing the vandermonde vectors $(\mathbf{v}^{(k)})^T$, all their components are standardized except the first one. Thus, we ensure an input $(\mathbf{v}^{(k)})^T \sim (N(1,0), N(0,1), \cdots, N(0,1))$. Moreover, the components are assumed to be independent from each other to simplify the analysis, though clearly there is some dependence as they are different powers of feature k.

Theorem 1 *Let Y be the random distribution of the output through the TT forward prediction. Given the TT-cores following $\mathcal{G}_k \sim N(0,\sigma)$ with $\sigma^2 = \frac{1}{n}\cdot r^{-\frac{d-1}{d}}$, then $E(Y)=0$ and $Var(Y)=1$.*

Proof. For the following derivation, several probability distribution properties are used. Given two independent probability distribution X and Y with $\mathrm{E}(X)=\mu_x$, $\mathrm{Var}(X)=\sigma_x^2$ and $\mathrm{E}(y)=\mu_y$, $\mathrm{Var}(y)=\sigma_y^2$, then:

1. $\mathrm{E}(X+Y)=\mu_x+\mu_y$.
2. $\mathrm{E}(XY)=\mu_x\mu_y$.
3. $\mathrm{Var}(X+Y)=\sigma_x^2+\sigma_y^2$.
4. $\mathrm{Var}(XY)=(\sigma_x^2+\mu_x^2)\cdot(\sigma_y^2+\mu_y^2)-\mu_x^2\mu_y^2$.

The TT forward is defined by

$$y = \prod_{k=1}^{d} \mathcal{G}_k \times_2 (\mathbf{v}^{(k)})^T = \prod_{k=1}^{d}\sum_{i=1}^{n} v_i^{(k)}\mathcal{G}_k(i),$$

where $\mathcal{G}_k(i)$ is a slice matrix $r_{k-1}\times r_k$.

Thus, let $V_k = \sum_{i=1}^{n} v_i^{(k)}\mathcal{G}_k(i)$, we get that $\mathrm{E}([V_k]_{ab}) = 1\cdot 0 + \sum_{i=2}^{n} 0\cdot 0 = 0$, where a,b define the indices of the matrix V_k . Therefore, $\mathrm{E}(\prod_{k=1}^{d} V_k)=0$.

Regarding the variance,

$$\mathrm{Var}([V_k]_{ab}) = \sum_{i=1}^{n}\frac{1}{n}\cdot r^{-\frac{d-1}{d}} = r^{-\frac{d-1}{d}}.$$

Let

$$\hat{\sigma}^2 = r^{-\frac{d-1}{d}}.$$

Then we have that:

$$
\begin{aligned}
&\text{Var}(\prod_{k=1}^{d} V_k) \\
=&\text{Var}(V_1 \cdot \prod_{k=2}^{d} V_k) \\
=&\sum_{i=1}^{r} \hat{\sigma}^2 \cdot \text{Var}(\prod_{k=2}^{d} V_k) \\
=&r\hat{\sigma}^2 \cdot \text{Var}(V_2 \cdot \prod_{k=3}^{d} V_k) \\
=&r\hat{\sigma}^2 \cdot \left(\sum_{i=1}^{r} \hat{\sigma}^2 \cdot \text{Var}(\prod_{k=4}^{d} V_k)\right) \\
=&r^2(\hat{\sigma}^2)^2 \cdot \left(\text{Var}(V_3 \cdot \prod_{k=4}^{d} V_k)\right) \\
=&\cdots \\
=&r^{d-1}\hat{\sigma}^{2d} = r^{d-1}r^{-(d-1)} = r^0 = 1.
\end{aligned}
$$

□