Neural Network report

A journey into hyper-complex sequential neural networks

Emanuele Giacomini

October 2020

Contents

1 Introduction **2**

2 Quaternion Algebra **3**
2.1 Tessarine Algebra 3

3 RNN **4**
3.1 LSTM 4
3.2 GRU 5

4 Experiments **6**
4.1 Synthetic Memory Copy-Task 6
4.2 Human Activity Recognition through inertial data 6
4.3 Indoor User Movement Prediction 7

5 Results **8**

6 Conclusion **11**

1 Introduction

In the world of machine learning, many problems do not benefit from the property of identical independence and distribution (i.i.d.), often exploited by networks themselves to learn latent patterns among the data supply. These types of problems, in fact, concern data that have little, if any, meaning when extracted individually, but on the other hand they take on great significance when contextualized within a sequence. There are several examples where being able to analyze the sequential nature of the data is essential to correctly solve a problem: imagine having to teach a model to predict the next word within a text, or detect certain patterns in the world of signal processing, weather forecasting, etc..
Precisely to analyze these problems, specific models have been developed to treat the data in the form of sequences, exploiting the correlation between the individual measurements.
In this report, results obtained by my study in the world of hyper-complex sequential neural networks will be showed. In the first section a mathematical background is given about the algebra of quaternions necessary to fully argue the structural choices shown in the following sections, related instead to the implementation of these networks, and the experiments made on the latter. In addition, the next section will show recurring neural networks of interest, such as LSTM and GRU. The QLSTM and QGRU counterparts are not explicitly described as their construction characteristics are the same as the real networks. The only implementation difference concerns the products with the weights matrices, replaced in the second case with the Hamilton product, described in the next section.

2 Quaternion Algebra

We define a quaternion as an extension of a complex number:

$$Q = a1 + bi + cj + dk \tag{1}$$

where a, b, c, d are real numbers, while $1, i, j, k$ define a basis of quaternonic space. Quaternion algebra defines the following relationships between the elements of the base:

$$ij = k,\ jk = i,\ ki = j \tag{2}$$

$$i^2 = j^2 = k^2 = ijk = -1 \tag{3}$$

Through these properties, it is possible to define the product of two quaternions Q_1 and Q_2 through the operator $\otimes$ called Hamilton Product:

$$\begin{aligned} Q_1 \otimes Q_2 =& (a_1a_2 - b_1b_2 - c_1c_2 - d_1d_2) + \\ & (a_1b_2 + b_1a_2 + c_1d_2 - d_1c_2)i + \\ & (a_1c_2 - b_1d_2 + c_1a_2 + d_1b_2)j + \\ & (a_1d_2 + b_1c_2 - c_1b_2 + d_1a_2)k \end{aligned} \tag{4}$$

This operator is used to transform, scale or represent rotations on vector elements representing quaternions. In the case of the sequential networks analyzed in the next sections, the Hamiltonian product allows to reinterpret a fully-connected layer under a quaternion key: all inputs are quaternions that are multiplied by a matrix of weights considered quaternions.

2.1 Tessarine Algebra

Tessarine algebra is very similar from the syntactic point of view to quaternion algebra, given its hyper-complex nature in 4 dimensions. In any case the two algebras differ in terms of properties, in fact given the tessarine number $t = a1 + bi + cj + dk$ in tessarine algebra, the following properties are known:

$$ij = k,\ jk = i,\ ki = -j \tag{5}$$

$$i^2 = k^2 = -1,\ j^2 = 1 \tag{6}$$

Moreover the product of two tessarines $t1, t2$ is defined as:

$$\begin{aligned} t_1 \otimes t_2 =& (a_1a_2 - b_1b_2 + c_1c_2 - d_1d_2) + \\ & (a_1b_2 + b_1a_2 + c_1d_2 + d_1c_2)i + \\ & (a_1c_2 - b_1d_2 + c_1a_2 - d_1b_2)j + \\ & (a_1d_2 + b_1c_2 + c_1b_2 + d_1a_2)k \end{aligned} \tag{7}$$

Regarding the sum operation, both quaternion and tessarines have the same qualities.

3 RNN

An RNN is a specific neural network model able to manage the type of data previously described, i.e. sequential series. Obviously to be able to make inference on sequential data, the model uses a latent space in which it can save the information it deems important. To every temporal step in fact, the latent space, or hidden state, is updated in the following way:

$$h_t = f(x_t, h_{t-1}) \tag{8}$$

where f is the function to calculate the latent state in the current time-step. More in detail, let's assume that X_t is a mini-batch of features related to time t and that $H_t \in \mathbb{R}^{n \times h}$ is the latent state at time t. We define W_{xh} as the matrix of weights used to update H_t through the current input X_t, while W_{hh} as the matrix of weights used to transfer information from H_{t-1} to H_t. The calculation of H_t is composed of the following equation:

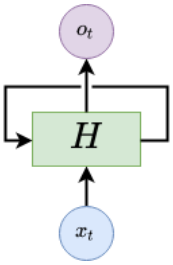

Figure 1: RNN structure

$$H_t = \phi(X_t W_{xh} + H_{t-1} W_{hh} + b_h) \tag{9}$$

where ϕ is the activation function and b_h is the bias term. Recurring units, or RU, are conceptually an appropriate solution to solve problems of a basic nature, but their performance drops dramatically into modern sequential problems, where very long time correlations are present. As shown in [1] a trade-off between the ability to maintain relevant information longer and gradient-based learning efficiency exists. To overcome these problems, multiple solutions have been proposed, some of which will be shown in this section.

3.1 LSTM

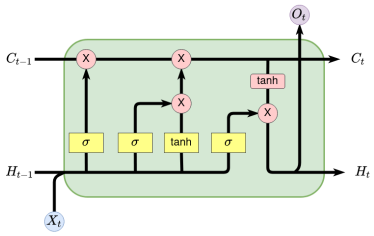

Figure 2: LSTM network structure.

Introduced in [2], Long-Short Term Memory networks are a more complex type of network that aims to solve the problem of time dependencies between inputs and outputs. The structure of the network is visible in Figure 2. The

principle behind LSTMs is to add a memory cell C where it is possible to save long-term information. To access the memory cell, three gates have been designed, namely the *Forget gate*, the *Input gate* and the *Output gate*. The forget gate is in charge of deciding which information should be retained and which should be removed based on the previous instant's latent space h_{t-1}, and the current input X_t. The gate returns a vector of weights f_c that is multiplied with the C_{t-1} memory, altering the values in the latter. The equation to get f_c uses the internal gate weights for the input W_{fx} and the latent space W_{fh}: The second gate, the Input gate is in charge of electing a new candidate for the memory cell.
The candidate $\widetilde{C}$ is produced by an internal recurring unit and the gate's output i_t that weighs the various components of $\widetilde{C}$. Once the memory cell is updated, the last gate takes care of producing the latent memory for the current H_t iteration. The latter is produced by the gate that produces a vector of o_t weights multiplied successively by a C compression performed by the tanh activation function. Overall, the following equations produce the stages of the LSTM:

$$f_t = \sigma(W_{fx}X_t + W_{fh}H_{t-1} + b_f) \tag{10}$$

$$\hat{C} = C_{t-1} \times f_t \tag{11}$$

$$i_t = \sigma(W_{ix}X_t + W_{ih}H_{t-1} + b_i) \tag{12}$$

$$\widetilde{C} = \tanh([H_{t-1}, X_t]) \times i_t \tag{13}$$

$$C_t = \hat{C} + \widetilde{C} \tag{14}$$

$$o_t = \sigma(W_{ox}X_t + W_{oh}H_{t-1} + b_o) \tag{15}$$

$$H_t = \tanh(C_t) \times o_t \tag{16}$$

3.2 GRU

Proposed by [3] in 2014, Gated Recurrent Unit or GRU is another approach to solve long term dependencies issues with the RNNs. GRU follows a philosophy very similar to that of LSTM, but with a simplified structure. To operate on the hidden state in fact, the GRU contains two gates called *reset* and *update* gates. The *reset* gate produces a new candidate for the latent state $\widetilde{H_t}$, producing the weight vector R_t. Next, the update gate takes care of merging the new $\widetilde{H_t}$ candidate and finally generate the new H_t latent state. In this regard, the gate produces the Z_t weight vector used in a mutually exclusive form on H_{t-1} and $\widetilde{H_t}$ to get a combination of the two. This means that values close to 0 of Z_t bring the new state closer to $\widetilde{H_t}$ while the opposite propagates the old latent state into the new one.

$$R_t = \sigma(W_{rx}X_t + W_{rh}H_{t-1} + b_r) \tag{17}$$

$$Z_t = \sigma(W_{zx}X_t + W_{zh}H_{t-1} + b_z) \tag{18}$$

$$\widetilde{H}_t = \tanh(W_{hx}X_t + W_{hh}H_{t-1} + b_h) \tag{19}$$

$$H_t = Z_t \odot H_{t-1} + (1 - Z_t) \odot \widetilde{H}_t \tag{20}$$

4 Experiments

To validate the hyper-complex models presented in section 3, three experiments were prepared. In the following section we will introduce each experiment individually. Finally the results will be discussed in section 5.

4.1 Synthetic Memory Copy-Task

The first experiment, introduced in [2], is proposed as a sanity check to demonstrate the ability of RNNs to maintain long term dependencies. The experiment consists in exposing the RNN to a long sequence L, composed by S different symbols followed by a sequence of time-lags called blanks of length T. The sequence terminator (identified by another symbol) announces the beginning of the copying operation, immediately after which the network should progressively reconstruct the initial sequence provided with length L. This experiment is introduced as a sanity check, as it is intended to demonstrate that the introduction of quaternions within the LSTM model does not alter its ability to maintain information with long time dependencies. As in [4] it is also evaluated the performance of the models at the variation of $T = 10, 50, 100$. More in detail, the experiment is performed in 2000 epochs, consisting of 32 task per epoch. Each task contains a sequence of 16 symbols from an alphabet of 8 elements, each 8-dimensional. Moreover, the same Adam optimizer with initial learning rate of $5 \cdot 10^{-3}$ is used. The LSTM network uses a 40-dimensional latent space (8.2k parameters), whereas the QLSTM uses a 20-dimensional one (8k parameters). GRU has 44 dimensional space (8.6k parameters) and QGRU has 96 (8.6k parameters).

4.2 Human Activity Recognition through inertial data

Human Activity Recognition or HAR provides relevant information about the physical activity that the user is doing in a certain amount of time. There are many applications for HAR, particularly in surveillance, healthcare and human robot. interactions. The interoceptive data associated with this experiment [5] are about 10k inertial measurements. Each measurement was performed by a smartphone equipped with an IMU that provides the triaxial acceleration vector and the triaxial angular velocity vector, sampled at a constant rate of 50 Hz. Each measurement is paired with a label denoting what activity the user was doing during that particular instance between the following: (*WALKING, WALKING UPSTAIRS, WALKING DOWNSTAIRS, SITTING, STANDING, LAYING*). In this context, models with different configurations have been evaluated to understand in more detail how these differences affect their performance. The experiment involves the following models: LSTM with 80, 32 and 16 nodes respectively to represent the latent memory, QLSTM with 80, 32 and 16 quaternionic dimensions, followed by a TLSTM with 16 tessarine dimensions, a 16 node GRU and finally a 16 quaternionic node QGRU. During the tests, a tendency of the networks to overfit after a number of epochs was noted. In

this regard the optimizers for these networks, regularize the latter through the weight decay technique introduced in [6] with a value of 5×10^{-3} for QLSTM and TLSTM, 1×10^{-3} for LSTM, GRU and QGRU.

4.3 Indoor User Movement Prediction

The dataset proposed in [7] contains a binary prediction task on the movements of a user moving within an office. The data was generated through a wireless sensor network (WSN), which consists of a sensor mounted on the user and four sensors anchored inside the office that collect the time series of radio signal strength (RSS) towards the mobile sensor. Each time series has an associated label that represents the type of movement inherent in the data. The dataset includes movements within the same spatial context and movements that lead to change in the latter. The data come from several experiments that include two rooms separated by a corridor in which the anchors' pairs are placed at the corners of the rooms. The peculiarity of this experiment concerns the number of features equal to 4 and their strong correlation. Given the ability of quaternion networks to extrapolate more complex patterns within the data, a higher performance is expected, or a faster convergence to optimal. Again the comparison occurred between an LSTM with 1361 parameters against a QLSTM with 1313 parameters. To take advantage of the mini-batch training, each sequence has been partitioned into smaller sub-sequences of 19 elements in order to fully include even the smallest sequences. The optimizer used in the training is the same as in previous experiments, and 500 eras have been completed with a batch size equivalent to 64.

5 Results

In the first experiment it is possible to observe how the QLSTM was able to converge to an optimal solution in a shorter time than its LSTM counterpart. According to [4], progressively increasing the value of T, it is possible to reach the constructive limit of the LSTM in which the latter is no longer able to successfully complete all the tasks, while the QLSTM should perform much better. In any case the results of this experiment in particular, is shown in figure 3 In

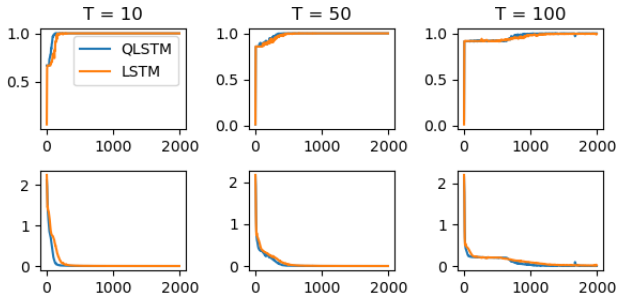

Figure 3: Copy-task comparison between a QLSTM and LSTM

addition, in the interest of experimenting further models, the experiment with $T = 100$ was reproduced using both GRU and QGRU, but also a Tessarine LSTM or TLSTM, to evaluate their performance against QLSTM. In the first case, shown in figure 4, we can see that the GRU has entered into convergence faster than the LSTM pair, and in addition the GRU has had a faster convergence than the QGRU. Finally, in figure 5, we can see a comparison between the

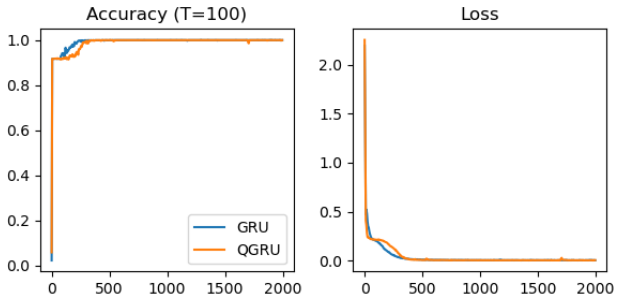

Figure 4: Copy-task comparison between GRU and QGRU

three special models. Contrary to what I expected, the Tessarine LSTM had a very similar behavior to that of the QGRU. The second experiment concerning HAR, produced some interesting results. The second experiment concerning HAR produced interesting results, visible in table 1. In the first variation of the

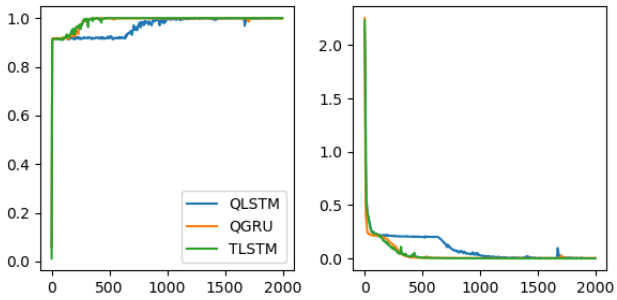

Figure 5: Copy-task comparison between QLSTM, TLSTM and QGRU

experiment, we notice acceptable results for both networks, where the LSTM model performs slightly better, given the great latent state provided to both networks. The situation changes gradually moving towards the third test for this experiment, in fact, in the second test configuration, there is a faster convergence for QLSTM towards the optimal solution. One can notice a loss of performance by the LSTM, given by the reduction of the latent space, of which the QLSTM is not particularly affected, probably due to the ability of quaternions to find more intricate patterns in the input features. Finally, in the third variation of the experiment, we can notice a slower convergence speed than the two previous variations. Also in this case, both networks converge, with a priority towards the QLSTM that manages faster to converge to an optimal solution.

	train_acc	train_loss	val_acc	val_loss
LSTM80	0.957589	0.110651	**0.931184**	**0.212662**
QLSTM160	0.928591	0.193394	0.877327	0.479539
LSTM32	0.957123	0.113523	**0.92254**	**0.249206**
QLSTM64	0.912403	0.243061	0.849623	0.446617
LSTM16	0.956036	0.114685	**0.916223**	**0.257117**
QLSTM32	0.95394	0.116884	**0.906582**	**0.256825**
GRU16	**0.961258**	**0.102602**	0.913564	0.272451
QGRU32	0.956599	0.110729	0.899934	0.289363
TLSTM32	0.957803	0.11285	0.907247	0.30543

Table 1: HAR Experiment results

To conclude, the results of the third experiment are shown. As described in section 4.3, having exactly four features provides hyper-complex networks with an advantage in terms of convergence. Figure 6 in fact shows the results in terms of validation accuracy and loss during the first 150 training periods. Taking as reference the LSTM16 model, we see how the three hyper complex networks QLSTM32, QLSTM16 and QGRU32 are able to obtain a better result

in terms of loss. At the end of the training, the comparable networks obtain a comparable result, in line with the expectation after the experiments seen previously An interesting aspect emerged from this experiment concerns not

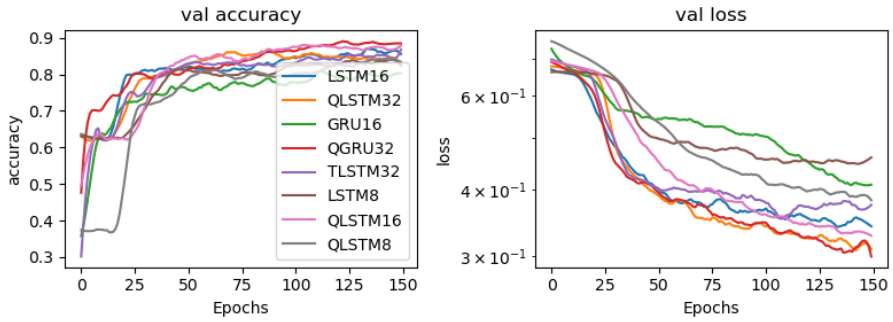

Figure 6: First 150 epochs of RSS training.

only the speed of convergence to an optimal solution, but also the rapidity with which overfit occurs on these networks. Another interesting aspect that emerged from this experiment concerns the rapidity with which overfit on these networks occurs. Once surpassed the 70 epochs in the matter, the hyper-complex models tend to overfit, and this can be verified by looking at how the validation loss tends to increase in time, therefore increasing the model's variance. The

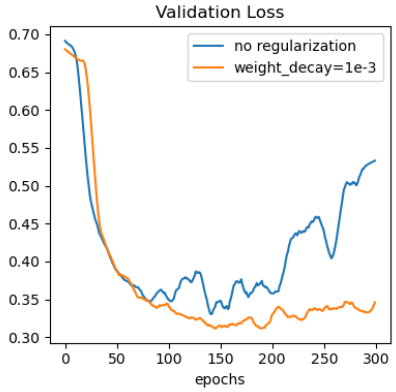

Figure 7: Regularization on QLSTM32 model.

problem of high variance is solved by inserting the L2 norm of the weights within the loss function. The PyTorch framework automatically manages the procedure of weight decay through its optimizer class, which uses the technique introduced in [6]. The result are shown in figure 7. To conclude, the experiments were useful in understanding the strengths of hyper-complex networks.

In particular, it must be noted that with the same parameters, these last ones maintain values of accuracy and loss matched with their counterparts. In any case, the major strength can be seen in more restrictive environments in terms of memory. Hyper-complex networks generally behave much better than their counterparts when the number of parameters allowed is limited (in Figure 6, see the comparison between LSTM8 and QLSTM8), as also demonstrated in [4].

	train_acc	train_loss	val_acc	val_loss
LSTM16	0.958984	0.134852	0.915901	0.265533
QLSTM32	0.957031	0.144966	0.876838	0.294742
GRU16	0.919922	0.224064	0.904105	0.278038
QGRU32	0.972656	0.0919188	0.911918	0.28064
TLSTM32	0.955078	0.135408	0.902267	0.333404
LSTM8	0.871094	0.323735	0.859222	0.433258
QLSTM16	0.927734	0.212317	0.898284	0.296014
QLSTM8	0.904297	0.259297	0.865043	0.338071

Table 2: RSS Experiment results

6 Conclusion

This report shows the performance of some hyper-complex networks in three experiments. Each network has shown its peculiarities when placed in the right contexts, but in general it is possible to mention how they are able to generalize more intricate patterns with fewer parameters than the real counterpart. In this context two hyper-complex algebra have been experimented, that is quaternion algebra and tessarine algebra. The quaternionic networks have generally proved to be more performing than the real counterpart, while the tessarine ones are more variable compared to the context in which they are applied. Two main types of architectures have been used, the LSTM or Long Short Term Memory networks proposed by [2] in 1997, and the GRU or Gated Recurrent Unit proposed in 2014. The first experiment shows the functionality of long-term memory of the models, while the next two experiments show the performance of networks in two areas based on time series extracted from sensors in realistic environments.

References

[1] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994.

[2] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, pp. 1735–80, 12 1997.

[3] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," 2014.

[4] T. Parcollet, M. Morchid, G. Linarès, and R. D. Mori, "Bidirectional quaternion long-short term memory recurrent neural networks for speech recognition," 2018.

[5] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz, *A Public Domain Dataset for Human Activity Recognition Using Smartphones*. 21th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, ESANN 2013. Bruges, Belgium 24-26, Apr. 2013.

[6] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," 2019.

[7] D. Bacciu, P. Barsocchi, S. Chessa, C. Gallicchio, and A. Micheli, "An experimental characterization of reservoir computing in ambient assisted living applications," *Neural Computing and Applications, Springer-Verlag*, vol. 24, no. 6, pp. 1451–1464, 2014.