

Experimental Evaluation of Memory Capacity of Recurrent Neural Networks

Aliaksei KOLESAU, Dmitrij ŠEŠOK, Nikolaj GORANIN,
Mindaugas RYBOKAS

Vilnius Gediminas Technical University, Saulėtekio al. 11, Vilnius, Lithuania

kolesov93@gmail.com, dmitrij.sesok@vgtu.lt,
nikolaj.goranin@vgtu.lt, mindaugas.rybokas@vgtu.lt

Abstract: In this article, the importance of correct representation of input data for recurrent neural network is experimentally analysed on the basis of the task for recognizing handwritten digits and task for incrementing an integer. In order to solve this task, the same information in a different form is provided for the neural network and quality of classification is evaluated. It was received, that a simple permutation of inputs has caused the decrease of quality from several percentage points (for short sequences, e.g. incrementing 32-bit integer in binary) up to 15% for long ones (784 steps). In addition, the phenomena that models examining the depiction of handwritten digits, presented in a horizontal way converge on average faster than analogue models with vertical digit representation.

Keywords: recurrent neural networks, handwriting recognition, training models.

1. Introduction

Neural networks and deep learning methods have demonstrated the impressive results in many areas: speech recognition (Hinton et al., 2012), medical diagnostics (Rajpurkar et al., 2017), pattern recognition (Krizhevsky et al., 2012), machine translation (Vaswani et al., 2017) and many others.

In general, neural network is a special form complex non-linear function $f(x)$, which parameters are being optimized with the help of gradient methods and algorithm of back error propagation (e.g. (LeCun et al., 1998)). Thus, during the training algorithm gets the collection of pairs $S = ((x_1, y_1), \dots, (x_m, y_m))$, $x_i \in X$, $y_i \in Y$, where x_i is a task input (e.g. graphical file with a handwritten digit) and y_i is a corresponding answer (e.g. recognized digit). The task of training algorithm is to get a function f , which insures the best mapping approximation of $X \mapsto Y$ on $x \in X$, that were not included in S .

The majority of neural networks (e.g., feedforward or convolutional (LeCun et al., 1989)) present themselves a composition of functions, i.e. $f = f_n \circ f_{n-1} \circ \dots \circ f_1$, where each of f_i – is a non-linear function, representing matrix multiplication and non-linearity (e.g. sigmoid). In such a case, parameters are usually matrix elements that participate in matrix multiplication.

Below, general application of such a model for recognizing handwritten digits is reviewed. Assume, that each digit is presented by a graphical file of size 100 by 100

pixel. Each pixel is characterized by 3 numbers of R-, G- and B- components intensity in RGB colour representation scheme. In such a case information stored in such a graphical file can be represented in a form of numeral vector of size $3 \cdot 100 \cdot 100 = 30000 - x_i$ (in our MNIST experiments we have grayscale images with size 28×28 , so our vectors have size $1 \cdot 28 \cdot 28 = 784$). The problem solution, i.e. y_i , can be presented as a 10-dimensional vector, in which all elements are equal to zero, except of exactly one element equal to 1, that corresponds to the digit, presented on the picture. Then $f(x_i)$ can be seen as a vector in which each element is a probability, that the corresponding digit is shown from the point of view of a model.

In that case f can be selected as follows:

$$\begin{aligned} n &= 2, \\ f_1(x) &= \sigma(A_1 x + b_1), \\ f_2(x) &= \text{softmax}(A_2 x + b_2), \end{aligned}$$

where $\sigma(x) = \frac{1}{1 + e^{-x}}$, $\text{softmax}(x)_i = \frac{e^{x_i}}{\sum_j e^{x_j}}$, $A_1 \in \mathbb{R}^{128 \times 30000}$, $A_2 \in \mathbb{R}^{10 \times 128}$.

Function uses P for transforming arbitrary numeral vector into probability distribution. The model received is called a neural network with one hidden layer of a size of 128 neurons (Fig.1). The size and the number of hidden layers are selected based on the problem and the size of data used for model training.

However, neural networks of such type are not suitable for all the tasks. Suppose, the size of picture that should be analysed is of a free size and is not limited to 100 by 100 pixels. Then the size of matrix A_1 to be selected is not clear. However, in such a case the majority of parameters are useless. Also for the parameters, that are responsible for the bigger picture size it is rather difficult to find the necessary number of training samples. Of course, such an approach cannot solve a problem, when the initial size is not limited, as in case of translation or speech recognition.

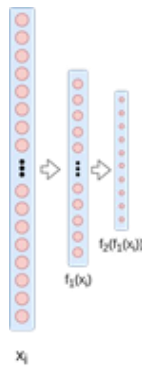


Fig. 1. Outline of a neural network for solving handwritten digit recognition problem.

Another option is to perform initial processing of incoming data. For example, in image recognition problems before applying the model image can be brought to the

canonical size by stretching or compression. Such an approach cannot be applied for the mentioned translation or speech recognition tasks.

For such tasks, recurrent neural networks (e.g. (Jain and Medsker, 1999)) are used. In many cases problems with a variable input can be presented as a sequence of data of the same type. For example, for machine translation input text can be presented as a sequence of words: $x_i = (x_1, x_2, \dots, x_{l_i})$. For speech recognition in many cases the initial audio file is split into segments of a fixed length, e.g. 25 ms (see (Hinton et al., 2012)).

On a high level recurrent neural networks present themselves as a function $g(x_i, s_{i-1}) = (y_i, s_i)$. Thus, this function has two entry points: x_i is another sequence element, s_i – state of a neural network in a previous period. This function returns two outputs: y_i is an answer on the i^{th} period and s_i is a state on a current period. This means that on each period, the same function g is used, and the number of its parameters does not depend on the sequence length (see Fig. 2).

input : $x = (x_1, x_2, \dots, x_l)$	input sequence
output : $y = (y_1, y_2, \dots, y_l)$	output sequence
$s_0 := 0;$	
for i 1 to l do	
$y_i, s_i = g(x_i, s_{i-1})$	
end	
return $y = (y_1, y_2, \dots, y_l)$	

Fig. 2. Recurrent neural network application algorithm

Such a construction appeared to be very powerful, i.e. being able to calculate extremely complex functions: such networks are used for speech recognition (e.g. (Miao et al., 2015)). Moreover, in (Siegelmann and Sontag, 1995) the fact was proved, that for any function that is computable on a Turing machine it is possible to find a recurrent network that can approximate this function with a defined precision.

However, in practice training of recurrent neural networks is so an easy task (refer to (Pascanu et al., 2012) for review of model training difficulties). Moreover, additional problem is based on that even format of input data has a great influence on model training effectiveness, i.e. it is not enough just to send available data to the input, but it is also necessary to structure it in a convenient way.

Article is organized in a following way: in Part 2, a short literature review on the topic is performed, in Part 3 the experiments performed are described and in Part 4 the results achieved and future research topics are discussed.

2. Prior and related works

Recurrent neural networks were developed in 1980s (refer to (Goodfellow et al., 2016) for review). One of the first neural networks were Hopfield networks.

Neural networks are difficult for training. The biggest obstacles for that are usually related to problems, caused by exploding and vanishing gradients. They are mainly related to the fact, that gradient, that carries information on error, passes through the big number of non-linearities (proportional to the depth of the network, and what is more

important, to the length of a sequence). This means that if gradient norm is multiplied by a number more than 1 while passing through the non-linearity, the norm will grow exponentially, causing numerical stability problems.

The main method for dealing with exploding gradients is gradient clipping (please refer to (Pascanu et al., 2012) for a more detailed problem discussion). Vanishing gradients problem is usually solved by a proper architecture selection for the recurrent neural network. The most popular in this case are LSTM (Hochreiter, 1997) and GRU (Cho et al., 2014a) methods. They solve a problem by selecting a special gradient path that does not change its norm. As a result, even if the gradient has vanished on the main path, the additional path will protect against information losses. In (Chung et al., 2014) authors have compared different types of recurrent units and found that gated units (such as LSTM or GRU) are indeed better than traditional units and that GRU is comparable to LSTM. In (Jozefowicz et al., 2015) it was empirically found that careful initialization of bias in forget gate in LSTM closes the gap between LSTM and GRU models in all problems evaluated by the authors. We use LSTM, GRU and simple RNN models in our increment experiments.

General review of neural networks and samples of their successful practical applications can be found in (Karpathy, 2015) and (Goodfellow et al., 2016).

From a model point of view, many facts on the calculative power of recurrent neural networks were proved. The most important result is the proof of their equivalence to the Turing machine (Siegelmann and Sontag, 1995). In (Khrulkov et al., 2017) the theorem, stating that a special kind of recurrent neural networks is exponentially better than one layer convolutional networks, was proved, i.e. that it requires less parameters for achieving the same quality of results. For our digit recognition experiments one-layer recurrent neural networks will be used. For quicker convergence the initialization scheme, proposed in (Le et al., 2017) will be applied.

Regarding MNIST datasets and object recognition in general, convolutional neural networks are better suited for the task. For example, (Ciresan et al., 2012, Jarrett et al., 2009, Ranzato et al., 2006) report several ways to train and build CNN model to get recognition accuracy of 99.5% or higher. The core idea of this paper is not to achieve better accuracy, but to investigate the storage capacity of recurrent models.

3. Experiments

In the following subsections we present two experiments supporting the importance of correct representation of sequential data for recurrent neural networks. We use numerical data in the task of incrementing the integer (subsection 3.1) and images in the handwritten digits recognition problem (subsection 3.2). Both experiments show that the rate of the convergence is very dependent on the chosen data representation despite the same amount of information given to the model.

3.1. Increment task

In this subsection the experiments, related to the increment task of an integer were carried out. Number in a K -base numeral system of N digits with possible leading zeros was considered. The model task is to read the given number and provide a number increased by one (or to show zero if $K^N - 1$ was given on input) as an output.

In the first experiment (let's call it "simple") on entry the number was provided on entry by one digit from lower order to higher. In the second experiment (to be called "complicated") digits were provided in a random, but fixed within the frame of one experiment, order.

In case of a "complicated" experiment model has not enough information to give "the first" digit of an increased number during the first step (since "the first" digit can be arbitrary to the digit in a permuted version, i.e. to be dependent on lower orders, that were not seen by the model on the first step. That is why the initial number was given to input two times and we were expecting to obtain the same orders for first K reports and increased orders for the second K reports. Explanations are provided on Fig. 3 and Fig. 4.

$$\begin{array}{r}
 x = \begin{array}{|c|c|c|c|c|c|c|c|c|c|} \hline 3 & 3 & 1 & 0 & 2 & 3 & 3 & 1 & 0 & 2 \\ \hline \end{array} \\
 y = \begin{array}{|c|c|c|c|c|c|c|c|c|c|} \hline 3 & 3 & 1 & 0 & 2 & 0 & 0 & 2 & 0 & 2 \\ \hline \end{array}
 \end{array}$$

Fig. 3. Sample input data for an increment task. $N = 5$, $K = 4$. Number $x = 201334 = 54310$, repeated two times is provided to input (please note, that number is provided in the order from lower to higher). On output we expect to obtain $2N$ digits: first of all N digits of initial number, then N digits of the increased number (in that exact case $54410 = 202004$)

$$\begin{array}{r}
 x = \begin{array}{|c|c|c|c|c|c|c|c|c|c|} \hline 2 & 3 & 3 & 0 & 1 & 2 & 3 & 3 & 0 & 1 \\ \hline \end{array} \\
 y = \begin{array}{|c|c|c|c|c|c|c|c|c|c|} \hline 2 & 3 & 3 & 0 & 1 & 2 & 0 & 0 & 0 & 2 \\ \hline \end{array}
 \end{array}$$

Fig. 4. The same sample as on Fig. 3 but in "complicated" definition, namely after permutation [4, 0, 1, 3, 2].

Three architectures of recurrent neural networks were used: vanilla rnn (the same as in experiment with MNIST), lstm (Hochreiter, 1997) and gru (Cho et al., 2014b). In each experiment the minibatch of a size of 16, 5 epochs with 10240 minibatches in it, single layer neural network with the size of a recurrent cell equal to 16 elements and optimizer RMSprop with a learning rate of $\eta = 3 \cdot 10^{-4}$ were used. At the end of each minibatch the accuracy of the obtained model was calculated with the help of hold-out set. This set was formed of all possible K^N , or, if the resulting size was more than $3 \cdot 10^5$, of $3 \cdot 10^5$ random numbers of the same distribution, as the training set.

In case if number is selected out of K^N equally likely options, rather simple and "uninteresting" samples will be dominating in a measurable metric, where the lower order has just to be increased by one in order to perform the increment task. Because of that the distribution was modified to select an "interesting" sample with a 0.1 probability in the following way:

- choosing an equally likely integer number T from 0 to N

- returning $(K^T - 1) + R_K(N - T) \cdot K^T$, where $R_K(N - T)$ –randomnumber in a K -base numeral system of $N - T$ digits.

In such an interesting sample there will be at least T transfers from the lower to higher order. As it can be seen from Figures 5, 6, 7 in his experiment the similar behavior as in case with MNIST experiment is observed: in case of a “correct” of data provided for input model converges to optimum much quicker. In case of “complicated” case convergence is visible, but it is always much slower.

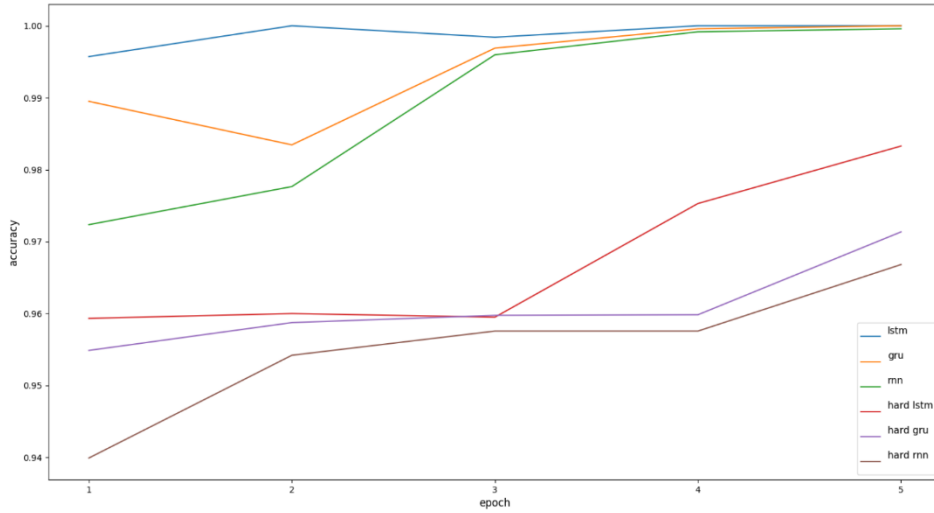


Fig. 5. Accuracy with $K = 2, N = 32$

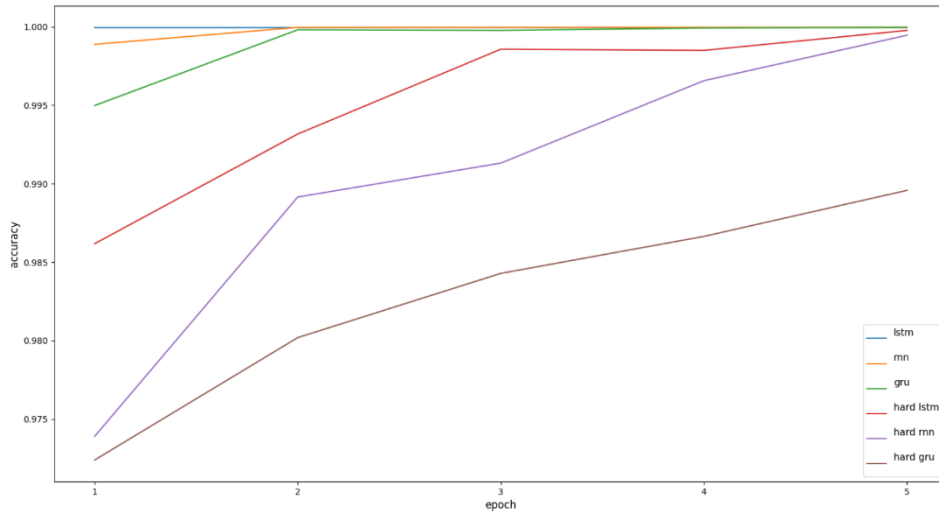


Fig. 6. Accuracy with $K = 10, N = 7$

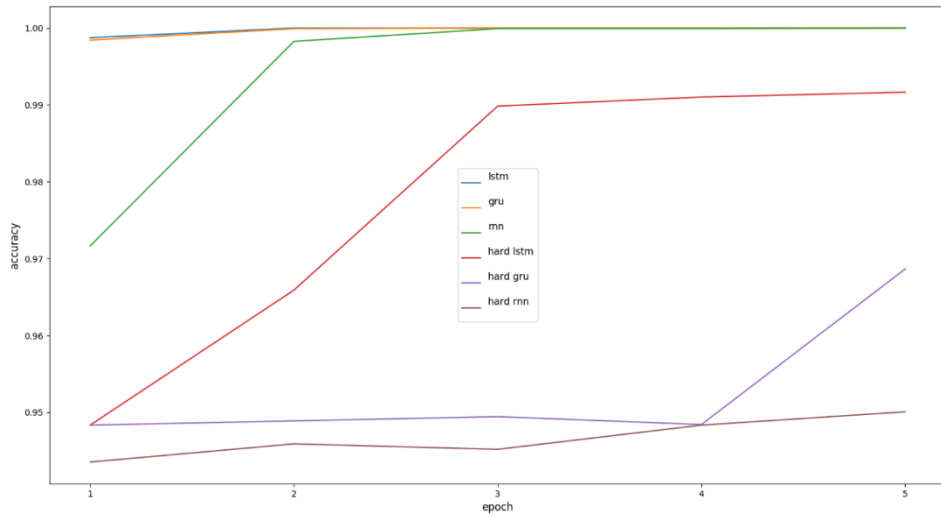


Fig. 7. Accuracy with $K=4$, $N=15$








3.2. Recognizing of handwritten digits

In this subsection we present the experiment carried out with MNIST dataset (LeCun and Cortes, 2010), which is used for recognizing handwritten digits. This dataset contains 60 000 samples for training and 10 000 samples for testing. Each sample is an image of size 28 by 28 pixels. In Fig. 8, several samples from the dataset specified are presented.



Fig. 8. Samples of input data from the MNIST dataset

Table 1. Different representation of the same testing sample information. In each of the cases presented vectors are provided in the upside-down order.

Case	Example
original	
transposed	
original random permute	
transposed random permute	
flattened	
transposed flattened	
flattened random	

Training of several models of the following types was performed:

- original: each picture is a sequence of 28 vectors; each of it has a size of 28 lines of initial image directed upside down;
- transposed: equivalent to the original model, but each vector stores columns of the initial image from left to right;

- original random permute: equivalent to the original model, but the order of vectors is random (not compulsory to be upside down), still it is fixed for the training and testing sets.
- transposed random permute: equivalent to the original random permute, but each sequence element is a column of an image, not a row;
- flattened: each picture is a sequence of 784 single-size vectors, pixels are presented in a left-to-right and upside down order;
- transposed flattened: each picture is a sequence of 784 single-size vectors, pixels are presented in an upside down and left-to-right order;
- flattened random: each picture is a sequence of 784 single-size vectors, pixels are presented in a random order, but this order is fixed for the training and testing sets.

In Table 1, different representation of the same testing image information is provided.

Cases flattened random and flattened are in fact Permuted MNIST and Sequential MNIST from (Le et al., 2015).

It can be noticed, that flattened representation complicates the digit recognition, since adjacent image pixels can be distinct from one another in the representation. So for a neural network, that works with rows, it is necessary to understand, that for example in case of digit 1 white pixels should follow after 28 counts to form a white vertical line. The same is to be said for a neural network working with columns in case of the horizontal line in digit 7.

Even more complicated task is for a model that works with any kind of a random permutation. In these cases, proximity in representation does not mean that there is the proximity in image. Nevertheless, for any function of an orderly sequence a function returns the same answers for a random case. That means that in case of ideal selection of training parameters and big amount of data in both cases the same result will be achieved.

For each representation option we were training a single-layer recurrent network with a size of 128 neurons with the help of RMSProp algorithm (Tieleman and Hinton, 2012), batch size was equal to 16 images and the learning rate was equal to $3 \cdot 10^{-5}$ and 10^{-6} . Two parameter options were selected for more reliable determination of data representation option.

For all random variation, training was performed on three different permutations in order to remove random fluctuations. Training with the help of Keras framework (Chollet, 2015) took two weeks on a 4-core computer. Models were training with the help of a cross entropy function between the last element of the output sequence and the correct answer.

The source code used in experiments can be found at <https://github.com/kolesov93/rnn-memory>.

The best model for each option are presented in Table 2. Table 3 provides a more detailed information on each of training launches.

It was also noticed during model training, that cases, that analyse the picture in a left-to-right order (transposed) converge to a good result almost always faster, than cases, when picture in a vertical direction. We suppose, that this is related to the fact that digits are stretched more to the height, rather than to the width. The model has fewer “interesting” instances that are worth attention. The remaining instances (e.g. a couple of first and last that are almost always black) deserve less attention. For hypothesis proof

Table 2. Average number of counts necessary for the model to stop changing the answer.

model	latency
original	23.3126; 21.7976
transposed	16.7835; 19.5972
original random permute	23.1492; 22.0422; 22.2437; 20.7726; 22.114; 23.1491
transposed random permute	15.757; 15.8633; 16.4686; 17.2456; 15.4135; 15.5195

Table 3. Top scores of each model by accuracy from a testing set. Name of the model is coding the way of data presentation (optional permutation number) @ batch size @ size of a hidden layer @ learning rate.

model	epoch	acc	vacc	loss	vloss
transposed@16@128@3e-05	118	0.988	0.979	0.058	0.135
original@16@128@3e-05	74	0.980	0.978	0.075	0.107
transposed random permute 2@16@128@3e-05	75	0.972	0.964	0.105	0.178
original random permute 0@16@128@3e-05	217	0.964	0.958	0.135	0.184
transposed random permute 1@16@128@3e-05	186	0.968	0.955	0.133	0.247
transposed random permute 0@16@128@3e-05	81	0.969	0.954	0.113	0.214
original random permute 1@16@128@3e-05	104	0.955	0.954	0.177	0.195
transposed@16@128@1e-06	242	0.948	0.949	0.180	0.179
original random permute 2@16@128@3e-05	155	0.942	0.940	0.254	0.246
transposed random permute 2@16@128@1e-06	234	0.934	0.934	0.223	0.225
original@16@128@1e-06	239	0.931	0.932	0.234	0.229
transposed random permute 1@16@128@1e-06	238	0.910	0.912	0.290	0.281
transposed random permute 0@16@128@1e-06	239	0.909	0.911	0.304	0.311
flattened@16@128@1e-06	231	0.901	0.910	0.305	0.291
original random permute 1@16@128@1e-06	242	0.906	0.905	0.311	0.321
flattened@16@128@3e-05	24	0.862	0.891	0.477	0.386
transposed-flattened@16@128@3e-05	116	0.861	0.891	0.499	0.397
original random permute 0@16@128@1e-06	239	0.850	0.854	0.498	0.484
original random permute 2@16@128@1e-06	234	0.851	0.852	0.487	0.488
transposed-flattened@16@128@1e-06	242	0.815	0.827	0.567	0.536
flattened random permute 1@16@128@3e-05	142	0.712	0.764	0.887	0.741
flattened random permute 2@16@128@3e-05	156	0.690	0.747	0.957	0.790
flattened random permute 2@16@128@1e-06	232	0.715	0.744	0.849	0.763
flattened random permute 1@16@128@1e-06	242	0.688	0.722	0.923	0.825
flattened random permute 0@16@128@3e-05	243	0.694	0.720	0.932	0.855
flattened random permute 0@16@128@1e-06	234	0.677	0.705	0.959	0.874

we have calculated a value, that was called latency, that shows how many counts does it take for the model to produce the result coincide with the final result and does not change after that. The average results were received from a separate sampling set. The latency results are provided in Table 4.

As it can be seen, the transposed model makes a decision on 4-5 counts (out of 28!) earlier, compared to the analogical original model and the same tendencies remain for other permute cases.

Table 4. Best models for each option

model	epoch	acc	vacc	loss	vloss
transposed	118	0.988	0.979	0.058	0.135
original	74	0.980	0.978	0.075	0.107
transposed random permute	75	0.972	0.964	0.105	0.178
original random permute	217	0.964	0.958	0.135	0.184
flattened	231	0.901	0.910	0.305	0.291
transposed-flattened	116	0.861	0.891	0.499	0.397
flattened random permute	142	0.712	0.764	0.887	0.741

Thus it can be noticed, that permutation decreases the classification quality by relative 0.7% for transposed case, by 1.4% for the original, by 16% for transposed flattened and by 14% for flattened.

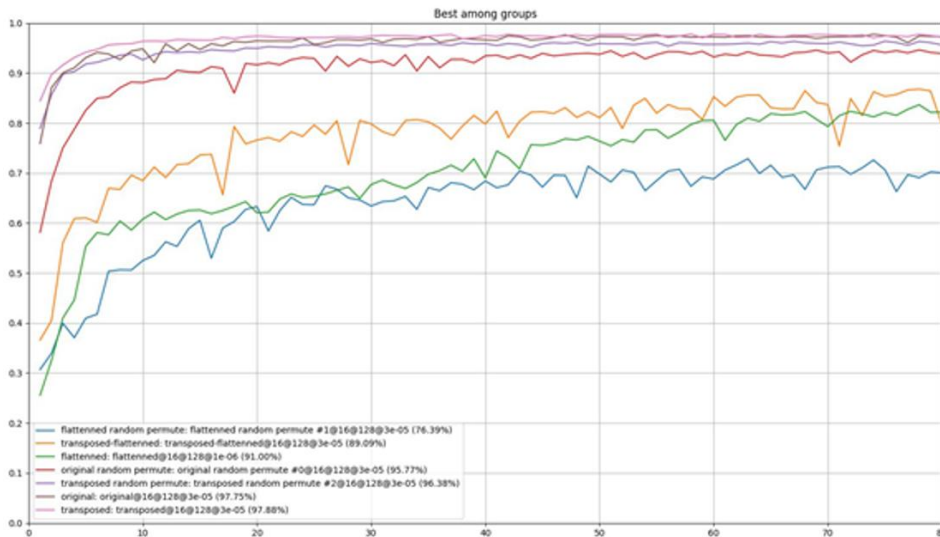


Fig. 9. Training curves for best cases in each of the groups

Image stretching into line also decreases the quality of classification, for example, original has decreased by 7% (0.979 to 0.910), and apart from that number of mistakes has increased 4.2 times. It is necessary to stress, that the quality decrease can be explained by a fact, that in initial model there are more parameters (since matrix multiplication is performed with a 28-column matrix, while in flattened case there is a 1-column matrix).

In Fig. 9, curves representing model by accuracy for a sample testing set in each of the groups are presented.

4. Conclusions and future work

It can be stated, that data representation format highly influences the convergence. It was empirically proven for data types: numerical data (in integer increment problem) and images (handwritten digits recognizing).

However, it can be expected, that neural network will be able to determine dependencies in complicated conditions. For example, on Fig. 9 it can be seen, that best cases provide the result, close to an optimum already during the 10th epoch. It takes longer for weaker cases to grow. However, in practice we cannot expect that any data representation case will lead to success due to a limited amount of data and limited calculation budget. For example, original random permute case was not able to minimize the lag from the best three cases even after 250 epochs.

The phenomena that models examining the depiction of handwritten digits, presented in a horizontal way converge to the optimum on average faster than analogical models with vertical digit representation was noticed. The experiments proving our hypothesis, that such models require less counts for decision making on classification and more computational resources can be spend on classification itself, rather than on memorizing the sequence, were carried out. However, additional research is planned on that topic and should be carried out in future.

In addition, additional topics for future work are defined:

- Bigger depth allows models to produce more sophisticated functions. It is necessary to check, how the depth increase influences the speed of sequence memorization.
- It is necessary to verify the achieved results on other data types like audio or text information.

References

- Cho, K. H., Merriënboer, B., Bahdanau, D., Bengio, Y. (2014). *On the properties of neural machine translation: Encoder-decoder approaches*. CoRR, abs/1409.1259.
- Cho, K., Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y. (2014). *Learning phrase representations using RNN encoder-decoder for statistical machine translation*. CoRR, abs/1406.1078.
- Chollet, F. et al. (2015). Keras. <https://github.com/keras-team/keras>.
- Chung, J., Gulcehre, C., Cho, K., Bengio, Y. (2014). *Empirical evaluation of gated recurrent neural networks on sequence modeling*. CoRR, abs/1412.3555.
- Ciresan, D.C., Meier, U., Schmidhuber, J. (2012). *Multi-column deep neural networks for image classification*. Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR): 3642–3649. IEEE Computer Society.
- Goodfellow, I., Bengio, Y., Courville, A. (2016). *Deep Learning*. MIT Press, Cambridge.

- Hinton, G., Li, D., Dong, Y., Dahl, G., Abdel-Rahman, M., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Kingsbury, B., Sainath, T. (2012). *Deep neural networks for acoustic modeling in speech recognition*. IEEE Signal Processing Magazine, 29:82–97, November 2012.
- Hochreiter S., Schmidhuber J. (1997). *Long short-term memory*. NEURAL COMPUTATION, 9(8):1735–1780.
- Jain, L. C., Medsker, L. R. (1999). *Recurrent Neural Networks: Design and Applications*. 1st ed., CRC Press, Inc., Boca Raton.
- Jarrett, K., Kavukcuoglu, K., Ranzato, MA., LeCun, Y. (2009). *What is the best multi-stage architecture for object recognition?* In 2009 IEEE 12th International Conference on Computer Vision, ICCV 2009: 2146–2153. IEEE Computer Society.
- Jozefowicz, R., Zaremba, W., Sutskever, I. (2015). *An empirical exploration of recurrent network architectures*. Proceedings of the 32nd International Conference on Machine Learning, PMLR 37:2342-2350.
- Karpathy, A. (2015). *The Unreasonable Effectiveness of Recurrent Neural Networks*, available at <http://karpathy.github.io/2015/05/21/rnn-effectiveness>.
- Khrulkov, V., Novikov, A., Oseledets, I. V. (2017). *Expressive power of recurrent neural networks*. CoRR, abs/1711.00811.
- Krizhevsky, A., Sutskever, I., Hinton G. E. (2012). *Imagenet classification with deep convolutional neural networks*. In Advances in neural information processing systems, 1097–1105.
- Le, Q. V., Jaitly, N., Hinton, G. E. (2015). *A simple way to initialize recurrent networks of rectified linear units*. CoRR, abs/1504.00941.
- LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., Jackel, L. D. (1989). *Backpropagation applied to handwritten zip code recognition*. Neural computation, 1(4): 541–551.
- LeCun, Y., Bottou, L., Orr, G. B., Müller, K. R. (1998). *Efficient backprop*. In *Neural Networks: Tricks of the Trade*, This Book is an Outgrowth of a 1996 NIPS Workshop, Springer-Verlag, London, 9–50.
- LeCun, Y., Cortes, C. (2010). MNIST handwritten digit database. AT&T Labs [Online]. Available: <http://yann.lecun.com/exdb/mnist>.
- Miao, Y., Gowayyed, M., Metze, F. (2015). *EESEN: end-to-end speech recognition using deep RNN models and wfst-based decoding*. CoRR, abs/1507.08240.
- Pascanu, R., Mikolov, T., Bengio, Y. (2012). *Understanding the exploding gradient problem*. CoRR, abs/1211.5063.
- Rajpurkar, P., Irvin, J., Zhu, K., Yang, B., Mehta, H., Duan, T., Ding, D., Bagul, A., Langlotz, C., Shpanskaya, K., Lungren, M. P., Ng, A. Y. (2017). *Chexnet: Radiologist-level pneumonia detection on chest x-rays with deep learning*. CoRR, abs/1711.05225.
- Ranzato, MA., Poultney, C.S., Chopra, S., LeCun, Y. (2006). *Efficient learning of sparse representations with an energy-based model*. NIPS'06 Proceedings of the 19th International Conference on Neural Information Processing Systems: 1137-1144. MIT Press.
- Siegelmann, H. T., Sontag, E. D. (1995). *On the computational power of neural nets*. JOURNAL OF COMPUTER AND SYSTEM SCIENCES, 50(1):132–150.
- Tieleman T., Hinton, G. (2012). Lecture 6.5 *RmsProp: Divide the gradient by a running average of its recent magnitude*. COURSERA: Neural Networks for Machine Learning.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., Polosukhin, I. (2017). *Attention is all you need*. Advances in Neural Information Processing Systems. 5998-6008.