Deep Learning Project

Ben Rickard

March 2025

1 Part 1

The paper I have chosen to write about is number 2: *Zhang S, Lu J, Zhao H. On Enhancing Expressive Power via Compositions of Single Fixed-Size ReLU Network. International Conference on Machine Learning 2023 Jul 3.*

1.1 Task 1 - Summarise the background to the paper

There has been lots of recent development in increasing the size and power of deep neural networks. However, this raises challenges in training these enormous models due to the large computation complexity. One of the ways this problem can be countered is by parameter sharing neural network models. This technique reduces the total number of parameters which require tuning, furthermore reducing the computation requirements. There are three prominent forms of parameter sharing. The first involves sharing parameters in the same layer, the second requires sharing parameters across many different layers, and the last is by sharing parameters across different neural networks or models. This paper focusses on a parameter sharing model of the second type.

The parameter sharing model explored in this paper is constructed through repeated compositions of a single fixed-size ReLU network. Note that a ReLU network is a neural network where each neuron uses a ReLU activation function. This paper focusses on neural network models of the form $\mathcal{L}_2 \circ g^{\circ r} \circ \mathcal{L}_1$, where g is a fixed-size ReLU network, $\mathcal{L}_1$ and $\mathcal{L}_2$ are affine linear maps, and $g^{\circ r}$ denotes r compositions of the network g. We call such neural networks constructed of repeated compositions of sub-neural networks RCNets.

The main contribution of this paper is the assertion that such ReLU RCNets can approximate continuous functions $f \in C([0,1]^d)$ surprisingly well for lower computational costs than a large deep neural network. To analyse the order of approximation error for the ReLU RCNets, we first must define the modulus of continuity for any function $f \in C([0,1]^d)$ and for all $t \geq 0$ by,

$$\omega_f(t) := \sup\{|f(x) - f(y)| : \|x - y\|_2 \leq t \text{ and } x, y \in [0,1]^d\}.$$

The paper then proves that a target function $f \in C([0,1]^d)$ can be approximated by a neural network model of form $\mathcal{L}_2 \circ g^{\circ r} \circ \mathcal{L}_1$, with approximation error of order $\mathcal{O}(\omega_f(r'^{-\frac{1}{d}}))$, where $r' \in \mathbb{Z}^+$ is the same order as the number of times the fixed-size ReLU network g is repeatedly composed with itself, r, and d is the same order as the size of the sub-neural network which realises g. The details of these parameters will be given later. If the target function f is

1-Lipschitz, this gives that the approximation is order $\mathcal{O}(\sqrt{d}r'^{-\frac{1}{d}})$.

One of the other main areas of the paper is the discussion on the close relation between neural networks and dynamical systems.The authors frame the iterative application of g as a discrete-time dynamical system, where each composition step corresponds to a state transition. This perspective reveals that the approximation power of RCNets arises not from increasing the network's width or introducing time-dependent dynamics, but from the depth of composition. By interpreting $g^{\circ r}$ as the flow map of a dynamical system, the analysis bridges deep learning theory with dynamical systems theory, offering new insights into the expressive capabilities of compact, reusable architectures.

Some immediate avenues for further research could include exploring RCNets built from other activation functions. For example the sigmoid or Heaviside function. Other possibilities for further research could include considering different neural network architectures. In this paper, the only architecture analysed is fully connected feed-forward network. Hence, we could consider for example convolutional or recurrent neural networks, or perhaps sparser networks with disconnected neurons. This could be done by using dropout methods.

1.2 Task 2 - New real-world application of the proposed method

The main purpose behind the newly proposed method of ReLU RCNets, is that they reduce the amount of parameters required optimising. Hence, there is a reduction in computational cost and complexity, whilst maintaining good approximation power. The numerical examples in this paper displayed evidence that the theoretical results can indeed be applied to real data.

One area in which the advantages of this method could be exploited is in medial image segmentation for real-time diagnosis. For example, MRI scans for tumour detection. The key here is to reduce the number of trainable parameters P, without sacrificing the segmentation error. Reducing P is highly beneficial because it lowers computational cost, making the model feasible for deployment on devices like portable ultrasound machines or low-power MRI scanners. Hence, the scans become much quicker and easier to deliver. At the same time, maintaining a high accuracy ensures clinical reliability since this is a necessity for correct diagnoses. Traditional segmentation networks require millions of parameters, but by using repeated composition of ReLU-based blocks, we can achieve comparable accuracy with far fewer parameters. The theoretical justification comes from the paper's finding that composed ReLU networks achieve exponential approximation efficiency, meaning the error decreases rapidly with respect to the number of parameters P.

The real-world impact of this approach could be substantial. It enables immediate tumour detection during scans, improving diagnostic speed. It also reduces reliance on cloud computing due to reducing computational requirements, making AI-assisted diagnostics viable in resource-limited settings. This also keeps sensitive medical data on-device, minimizing security risks. By leveraging repeated ReLU composition, we can create compact yet accurate segmentation models, making AI-powered medical imaging more accessible, efficient, and privacy-conscious. This aligns perfectly with the growing demand for portable, low-cost diagnostic tools in healthcare.

1.3 Task 3 - Find a use case for Theorem 1.1

Heuristically, this theorem states that a fixed-size ReLU network can be repeatedly composed and sandwiched by linear affine maps to approximate any continuous target function. In order to formally state the theorem, first we denote $\mathcal{NN}\{N, L; \mathbb{R}^{d_1} \to \mathbb{R}^{d_2}\}$ to be the set of functions $\varphi : \mathbb{R}^{d_1} \to \mathbb{R}^{d_2}$ which can be realised by ReLU networks with width N, and depth L. Also recall the definition of the modulus of continuity of a function $f \in C([0,1]^d)$ by

$$\omega_f(t) := \sup\{|f(x) - f(y)| : \|x - y\|_2 \leq t \text{ and } x, y \in [0,1]^d\}.$$

which measures the maximum possible change in the function f when inputs are within a small distance t from each other. This helps quantify how "smooth" the function is, which directly impacts how well the ReLU network can approximate it. We now have all the notation required to state Theorem 1.1.

Theorem 1.1. Given any $f \in C([0,1]^d)$, $r \in \mathbb{N}^+$, and $p \in [1, \infty)$, there exist

- A ReLU network $g \in \text{NN}\{69d + 48, 5; \mathbb{R}^{5d+5} \to \mathbb{R}^{5d+5}\}$,
- Two affine linear maps $L_1 : \mathbb{R}^d \to \mathbb{R}^{5d+5}$ and $L_2 : \mathbb{R}^{5d+5} \to \mathbb{R}$,

such that the composed function approximates f with error bound:

$$\|L_2 \circ g^{\circ(3r+1)} \circ L_1 - f\|_{L^p([0,1]^d)} \leq 6\sqrt{d}\,\omega_f(r^{-1/d}).$$

This theorem is particularly impactful because it highlights that ReLU RESNets can approximate continuous functions well without requiring an excessively large number of parameters. The network architecture uses parameter sharing, which reduces the number of parameters that need to be trained. As a result, this approach not only improves computational efficiency but also shows how a relatively simple network structure (with fixed width and depth) can achieve effective function approximation by iteratively applying compositions and affine maps. In fact, the theorem asserts that increasing the number of repeated compositions of g decreases the approximation error. We will now consider how many parameters are required in a ReLU network and ReLU RESNet to achieve the same level of accuracy when approximating a $1-$Lipschitz function.

Recall that for a ReLU network with n parameters, we can achieve optimal approximation error of order $\mathcal{O}(n^{-\frac{2}{d}})$ for a $1-$Lipschitz function on $[0,1]^d$. Consider a ReLU RCNet of form $\mathcal{L}_2 \circ g^{\circ(3r+1)} \circ \mathcal{L}_1$, where g is a fully-connected ReLU network with width $69d + 48$ and depth 5. We will assume the input and output from g is has dimension $5d + 5$, and that all 3 hidden layers have $69d + 48$ neurons. Hence, the number of parameters in such a RCNet is $2\left[d(d+5) + (d+5)(69d+48) + (69d+48)^2\right]$. Results from the paper tell us that such a network can approximate $1-$Lipschitz functions on $[0,1]^d$ with error order $\mathcal{O}(\sqrt{d}r^{-\frac{1}{d}})$. Using these two results, Table 1 compares the number of parameters needed to achieve an approximation error of order 10^{-4} in ReLU networks and ReLU RCNets, for $1-$Lipschitz functions of different dimensions.
It is clear, that as the dimension of the target function goes beyond three dimensions, a ReLU RCNet requires less parameters than to achieve the same approximation error as a standard ReLU network. This is because ReLU networks suffer from the curse of dimensionality, where exponentially more parameters are required as the dimension increases. RCNets circumnavigate this by the repeated composition of the sub-network. This means the number of required parameters increases much slower with dimension. This is what makes this new method specifically of use, as this dramatically decreases the computational requirements.

Dimension	ReLU	RCNet
d=1	$\approx 10^2$	$28,794 \approx 10^5$
d=2	$\approx 10^4$	$71,824 \approx 10^5$
d=3	$\approx 10^6$	$134,178 \approx 10^6$
d=4	$\approx 10^8$	$215,856 \approx 10^6$
d=5	$\approx 10^{10}$	$316,858 \approx 10^6$

Table 1: Comparison of number of parameters required for RELU networks and ReLU RCNets to achieve an approximation error of order 10^{-4}

1.4 Task 4 - Specify Why the Following Theorems/Figures are of Interest

Theorem 1.3
This theorem is quite similar in spirit to Theorem 1.1, which was discussed in detail in Task 3. In particular, it states that ReLU RCNets can approximate a continuous target function f up to a specific accuracy, which depends on the continuity and dimension of f, the size of the sub-neural network, and the number of repeated compositions. In particular, more repeated compositions, increases the accuracy of the RCNet.

One key difference is that, in this theorem, the closeness of the target function and the RCNet is measured using the modulus of the difference rather than the L^p norm. This means that the approximation is pointwise, meaning it holds for all $x \in [0,1]^d$, rather than just on average. Since pointwise approximation is a stronger requirement than L^p-norm approximation, it is intuitive that a larger sub-network is required to achieve this level of accuracy.

More specifically, the network in Theorem 1.3 requires a size of

$$\mathcal{NN}\{4^{d+5}, 3+2d; \mathbb{R}^{\tilde{d}} \to \mathbb{R}^{\tilde{d}}\}$$

where $\tilde{d} = 3^d(5d+4)-1$, which grows exponentially with d. In contrast, Theorem 1.1 requires a network of size

$$\mathcal{NN}\{69d+48, 5; \mathbb{R}^{5d+5} \to \mathbb{R}^{5d+5}\},$$

which scales linearly in d. This highlights that achieving pointwise approximation rather than an L^p-norm approximation demands a much larger network.

Another difference is in the iteration count of the composition. Theorem 1.1 requires g to be iterated $3r+1$ times, whereas Theorem 1.3 requires $3r+2d-1$ iterations. The additional $2d$ iterations reflect the increased complexity needed to achieve uniform pointwise control across the entire domain.

This theorem in particular is useful as it provides an explicit construction for a neural network that guarantees uniform accuracy at every point x. This is crucial in applications where local precision matters, such as medical imaging, where a single bad approximation at an important location could lead to incorrect diagnoses. So whilst the increased network size might seem like a drawback, the benefit is that we now have a structured way to design deep neural networks that achieve precise pointwise accuracy, rather than just average performance as in Theorem

1.1.

Figure 1
Figure 1 helps visualize the main ideas behind the proofs of Theorems 1.1 and 1.3. The core approach is to construct a piecewise constant function that approximates the target continuous function. However, the continuity of ReLU networks presents a challenge when trying to uniformly approximate such piecewise continuous functions. To address this, a ReLU network is defined to approximate piecewise constant functions outside a sufficiently small region, where it can achieve good approximation. The remaining small region is handled separately to ensure a uniform approximation overall.

More specifically, we divide the domain $[0,1]^d$ into small cubes $\{Q_\beta\}_{\beta\in\{0,1,\dots,K-1\}^d}$ and a small region Ω, where K is a positive integer. Each cube Q_β has a representative point $x_\beta \in Q_\beta$. The goal is to construct the desired network to approximate the target function on $[0,1]^d \setminus \Omega$. The construction process proceeds as follows:

1. First, a sub-network is designed to realize a vector function Φ_1 that maps the cube Q_β to its index β.

2. Next, another sub-network is designed to realize a function φ_2 that maps β to $f(x_\beta)$, so that $\varphi_2(\beta) \approx f(x_\beta)$.

3. Finally, we define the composite function $\varphi = \varphi_2 \circ \Phi_1$. Hence, for any $x \in Q_\beta$, we have

$$\varphi(x) = \varphi_2(\Phi_1(x)) = \varphi_2(\beta) \approx f(x_\beta) \approx f(x).$$

Informally, this process allows us to approximate the target function on $[0,1]^d \setminus \Omega$, as captured in Figure 1. The figure illustrates this construction, highlighting how the network approximates the target function on the larger domain outside Ω. Note, to complete the proof, it remains to show that the approximation holds on Ω, and that the resulting function $\varphi = \mathcal{L}_2 \circ g^{\circ r} \circ \mathcal{L}_1$ is in the desired form of a ReLU RCNet.

Figure 3
Figure 3 is very important to the paper since deep learning is inherently based on applications, and this figure displays the results from applying the ReLU RESNet method to a real function approximation task. The table provides the mean square and maximum loss values, for varying numbers of repeated composition and size sub-neural network. Note that these are calculated over the average of the last 100 epochs of test losses. This is important as it gives numerical evidence that our theoretical results can be applied to real data.

In particular, we see that increasing the number of repeated compositions from one to four decreased the MSE by a factor of 10^3, and as the size of the network grows from $n = 100$ to $n = 200$, the MSE decreases by approximately a factor of 10. This backs up the assertions that repeated compositions and larger sub-networks do decrease the approximation error.

Figure 3 also includes plots showing how the log-loss decreases across epochs, again for different numbers of repeated compositions and size of the sub-network. This is important as the training across the epochs seems regular, and relatively smooth. This gives confidence in the method, that it can be applied to real datasets.

Hence, Figure 3 helps us validate that parameter sharing via repeated compositions enables accurate approximations, and that the RCNet architecture achieves stable optimization, despite its compositional depth. Note that the stability of the optimisation is often a concern in training highly-parameterised models. However, Figure 3 provides us with some numerical evidence that this should not be a serious issue when using these ReLU RESNet models.

2 Part 2

2.1 Classification with Prediction

First of all, I split the full dataset into training, validation and test datasets by $70\%, 15\%, 15\%$ respectively. This meant the already limited number of samples became even smaller to train any models on. However, this is still vital in order to check for overfitting and accuracy of any classifiation model. Then in order to use the data as in computer practicals, I converted each 28×28 sample of intensity values into a JPEG image.

Now, since we are dealing with image data, I first fit a simple convolutional neural network, as in the computer practicals. There was evidence of overfitting in this simple model, which negatively affected the test accuracy which was 50%, far below what I would hope for. This poor power could also be a result of the small training sample. Hence, I improved this model by introducing a dropout probability and data augmentation in the style of slightly deforming the training images. The deformations were less extreme than in the computer practicals, as otherwise the model performed very poorly at around 20% accuracy on the test dataset, hardly better than randomly guessing. However, after refining the augmentation, the model achieved a test accuracy of 70%, with the validation accuracy conveying much less overfitting.

Despite this significance increase in test accuracy, I still endeavoured to improve the model. I tried to use transfer learning techniques, by utilising a large pre-trained CNN. I used the VGG16 model available in Keras which was trained on the large ImageNet database of images of animals and household objects. First I ran this convolutional base on the training dataset, which was then used as an input to a separate densely connected classifier. This process is cheap computationally to run, however, doesn't allow data augmentation. Perhaps due to the risk of overfitting associated with no augmentation, this model performed surprisingly poorly, with an accuracy over the test dataset of 63%.

Hence, in order to improve this transfer learning model, I extended the convolutional base by adding dense layers on top, and running the whole model on the input data. This allows us to now use our data augmentation again. This is more computationally expensive, but performed better with accuracy of 75% on the test dataset. To improve this model further, I undertook some fine-tuning of the model. This means I again added a custom network on top of the convolutional base, then froze the weights of the base before training the part I added. Finally, I unfroze some of the latter layers of the base network and jointly trained these with the layers I added. This means the pre-trained model becomes more adapted to this instance, without losing the large representations previously learnt by the base. This model gave the best accuracy on the test dataset at 80%, and so this is the model I used to make the classification predictions on the new data.

2.2 Generation

First, I attempted to use a generative adversial network in order to generate $3,000$ new samples. This was performed as in the computer practicals, but with a much longer training process with an increased number of epochs. Firstly, the intensity values were needed to be scaled to be in $[0, 1]$, and the data reshaped to appropriate format, noting that they are 'grayscale' instead of 'rgb'. When generating the new images, the model performed very poorly. Random inspections of some of the new images showed that they were mostly noise with limited identifiable structure. This suggests issues with convergence, mode collapsing, or a need for more sophisticated architecture. There was also a relatively small training set for the GAN to be trained on since I only used the original 500 images, probably effecting performance. I considered using the other dataset which I classified, but due to the extremely poor performance and time-consuming training process, I decided I would be better off trying a new type of model.

So, then I attempted instead to use a variational autoencoder. Again, this was very similarly implemented as in the computer practical, but this time I accelerated dimension reduction to mitigate overfitting. Also, after the poor performance of the GAN, I decided to fit a different VAE model for each label. Moreover I had six independently trained VAE models. Again I randomly inspected some of the generated images of each class, and these models had performed much better. The images were clear and it was obvious what classification they had come from. I thought I could improve on these models further by incorporating the images which I had classified. However, when including these, the images I inspected were not as accurately generated for their label. Hence, I settled on using the smaller dataset. I would assume that this decrease in performance was due to images I incorrectly identified corrupting the VAE model. Hence, the images submitted were generated by these six independent VAE models, trained just on the original dataset.