From ResNets to Runge-Kutta Networks: A Review of Generalized Residual Connections in Deep Learning

Tom Feldhausen
Department of Mathematics
Ruhr-Universität Bochum
44801 Bochum, Germany
tom.feldhausen@ruhr-uni-bochum.de

Sebastian Vallbo
Department of Data Science
Göteborgs universitet
405 30 Gothenburg, Sweden
sebastianvallbo@gmail.com

Abstract

As the field of deep learning excels in numerous areas, this paper aims to review one of the more recent disruptive breakthroughs. A major problem in deep learning is that adding more layers to a neural network can result in a worse performance, thus preventing networks from learning highly complex functions as encountered in fields such as image recognition. The introduction of the Residual Neural Network [He et al., 2016] improved performance for deep networks significantly, an approach that is also used in state-of-the-art transformer models [Vaswani, 2017]. It was then recently discovered that these ResNets are part of an even bigger class of networks, the Runge-Kutta Networks [Benning et al., 2019], building on an early paper [LeCun et al., 1988] that discovered a connection between the learning problem and ordinary differential equations, which can be numerically solved by Runge-Kutta methods. In this work, we will summarize these architectures and experimentally compare their respective accuracies, providing an implementation of Runge-Kutta networks for any explicit method and determining the best way to do residual connections.

1 Introduction

We begin by introducing the notation for general fully connected neural networks. For a comprehensive introduction we refer to [Higham and Higham, 2019].

Starting with the input $y^{[0]} = x \in \mathbb{R}^n$, let $y^{[i]} \in \mathbb{R}^{n^{[i]}}$ denote the input to the $(i+1)$-th layer, whose components we call neurons. The recursive formula

$$y^{[i+1]} = \sigma(W^{[i]}y^{[i]} + \beta^{[i]}) = f(y^{[i]}, u^{[i]})$$

defines the forward propagation with component-wise application of the activation function σ, typically the sigmoid or relu function. The transformation applied by the i-th layer is summarized as $f : \mathbb{R}^{n^{[i]}} \to \mathbb{R}^{n^{[i+1]}}$. We finally get the vector $y^{[N]} \in \mathbb{R}^{n^{[N]}}$ of the output layer, which in case of binary classification is then mapped to a probability $C(W^{[N]}y^{[N]} + \beta^{[N]}) \in [0,1]$ by the hypothesis function C, typically using sigmoid for binary classification or softmax for multi-class problems.

We write $u = (u^{[0]}, ..., u^{[N]})$ with weights and biases $u^{[i]} = (W^{[i]}, \beta^{[i]})$ and define the mean-squared loss function

$$J(u) = \sum_{j=1}^{m} |C(W^{[N]} y_j^{[N]} + \beta^{[N]}) - c_j|,$$

where we sum over the m samples of the training data and c_j denotes the label of the j-th sample. As we aim to minimize the loss, we face an optimization problem that can be solved iteratively by the method of gradient descent, a process referred to as training or learning.

Notice that neural networks can be seen as a combination of ensemble methods for one-neuron classifiers. Neurons are organized horizontally within a layer, and layers are stacked vertically, with the output of one layer serving as the input to the next.

However, especially for deep models - architectures involving a lot of layers - the learning process faces many challenges which can lead to the situation that deeper models perform worse than shallower ones. This is counterintuitive: a deeper network should at least be capable of learning the identity function, preserving the output of the smaller network and performing no worse.

2 Related work

This inspired the introduction of the Residual Neural Network (ResNet) [He et al., 2016], winner of ILSVRC 2015 in image classification, detection, and localization as well as winner of MS COCO 2015 detection, and segmentation [Tsang, 2018]. Let $F^*(x)$ be the optimal transformation which the neural network aims to learn and let $F(x)$ be the actually learned transformation, e.g. for two layers we have

$$F(x) = \sigma(W^{[1]}\sigma(W^{[0]}x + \beta^{[0]}) + \beta^{[1]}).$$

For an optimal classical network we have $F(x) = F^*(x)$. The ResNet however aims to learn the residual $F(x) = F^*(x) - x$ by changing the forward propagation formula to

$$y^{[i+1]} = y^{[i]} + f(y^{[i]}, u^{[i]}),$$

so called residual or shortcut connections. These connections simplify the learning process by making it easier for the network to approximate the identity function when necessary. This approach intuitively resolves the problem of deeper networks performing worse, as the network can now focus on learning incremental refinements to the input, rather than relearning the entire transformation.

The introduction of the Runge-Kutta networks generalize this approach by exploiting a connection to the theory of ordinary differential equations [Benning et al., 2019]. A first order ODE with intial condition $y(0) = y_0$ can be written as

$$y'(t) = f(t, y(t)),$$

that is the derivative of a function depends on the variable and the function itself. An easy example for a fixed $\lambda \in \mathbb{R}$ is

$$y'(t) = \lambda y(t),$$

which is solved by $y(t) = e^{\lambda t}$. In the general case analytical solutions can often not be found, leaving us with methods to approximate solutions. Numerical methods for solving ODEs are well-studied, the most famous methods being one-step Runge-Kutta methods and multi-step methods. The easiest example for the former is the explicit Euler method.

First, set a grid $0 = t^{[0]} < t^{[1]} = \Delta t < ... < t^{[K]} = K\Delta t$ with the goal to approximate the exact solution $y(t^{[i]}) \approx y^{[i]}$. The easiest way to guess the derivative is the first-order Taylor approximation

$$y(t + \Delta t) \approx y(t) + \Delta t y'(t),$$

which is exact if the average slope between t and $t+\Delta t$ equals $y'(t)$. As we only have that information it is a reasonable estimate. This motivates the iterative method

$$y^{[i+1]} = y^{[i]} + \Delta t f(t^{[i]}, y^{[i]})$$

for solving the ODE. To give an intuitive summary: We know the starting value and derivative. In order to determine the next value, we estimate the slope to be the slope in our starting point and get a new value. Obviously there are more advanced ways to determine the slope, yielding the family of Runge-Kutta methods

$$y^{[i+1]} = y^{[i]} + \Delta t \Psi(t^{[i]}, y^{[i]}, y^{[i+1]}).$$

Remember the ResNet's formula for forward propagation is

$$y^{[i+1]} = y^{[i]} + f(y^{[i]}, u^{[i]}),$$

which we can now identify as Euler's method with the residual parameter $\Delta t = 1$. The idea is now to substitute this by the general Runge-Kutta method

$$y^{[i+1]} = y^{[i]} + \Delta t \Psi(u^{[i]}, y^{[i]}, y^{[i+1]}),$$

providing a more general framework for residual connections.

Notice that this architecture relies on keeping the dimension of each layer identical. However, there are straightforward extensions to variable dimensions for layers, see [He et al., 2016].

3 Proposal

In this paper, we aim to achieve the following objectives:

1. **Demonstrate the failure of standard deep neural networks:** We will analyze the limitations of traditional deep networks, showing that, as depth increases, their performance tends to degrade.
2. **Show that ResNets overcome this issue:** We will explore how ResNets mitigate the problems of deep networks by introducing residual connections, which allow for easier training and improved performance in deeper architectures.
3. **Extend this analysis to Runge-Kutta Networks:** Building upon ResNets, we will show that Runge-Kutta Networks also solve the issue of performance degradation in deep networks.
4. **Determine the optimal method for residual connections:** By comparing standard residual connections, as used in ResNets, with those in Runge-Kutta Networks, we will investigate which type of residual connection yields the best performance in deep neural networks.
5. **Analyze the validity of the analogy to ODEs:** Finally, we will critically examine how well the analogy between deep learning models and ODE solvers holds, and assess its implications for understanding and designing neural network architectures.

Although the analogy to ODEs theoretically extends to implicit solvers and multi-step methods, our work will focus on explicit one-step Runge-Kutta methods. The reason for this choice is twofold: implicit methods involve the solving of non-linear equation systems, which is computationally expensive, and multi-step methods complicate backpropagation, making them less practical for use in deep learning.

To the best of our knowledge, this paper presents the first implementation of Runge-Kutta Networks that works for arbitrary explicit Runge-Kutta methods. However, unlike the approach proposed by [Benning et al., 2019], which connects the optimization problem to optimal control theory for gradient computation, we will rely on standard backpropagation and automatic differentiation for computing gradients.

4 Experiments and results

To facilitate replication of the experiments and further analysis, the Python code used in this study can be downloaded from GitHub [Feldhausen, 2024].

We implemented three classes of neural networks: the standard feed forward neural networks, the residual networks and the Runge-Kutta networks (RKN).

For all networks we used batch-normalization after each layer. All hidden layers were of the same dimension, see fig. 1 and the number of parameters for each network architecture was approximately the same for equivalent depth.

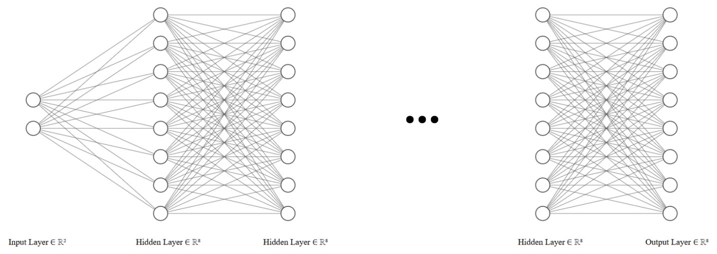

Figure 1: neural network with N hidden layers of dimension 8

The networks were implemented using the tensorflow library, thus using standard backpropagation and automatic differentiation for gradient computation, which differs from the approach of [Benning et al., 2019] that is based on optimal control theory, but sped up training due to built-in parallelization.

For the first experiment we used the spiral dataset, see fig. 2. As a training algorithm we used mini-batch ADAM with batch-size 32 and 200 epochs, while choosing sparse cross-entropy as the loss function.

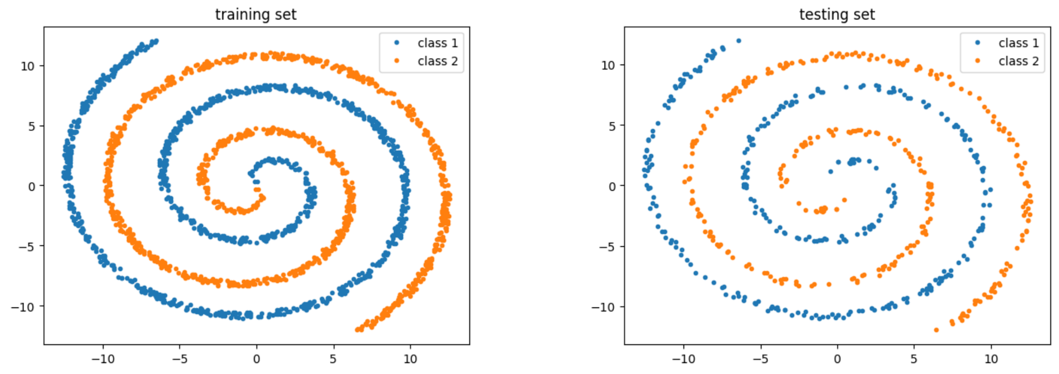

Figure 2: Spirals 2D

For a brief introduction to Runge-Kutta methods, including definitions of key terms such as order of convergence, consistency, and stages, see Appendix A.1. We focused on methods with residual parameter $\Delta t = 1$, for RKN this means all methods considered were consistent. For the RKN we reviewed the following methods: Euler (order 1), Heun (order 2), RK4 (order 4) and RK8 (order 8). Intuitively it holds the higher the order, the more accurate the method. In that sense, all methods described are as accurate as they can be with respect to their number of stages (i.e., their complexity). If the analogy to ODEs holds, we would generally expect higher-order methods to perform better.

For all the described network classes, we trained the models for $N = 5, 10, ..., 50$ layers. We recommend viewing the results in this Tableau dashbord [Feldhausen, 2024].

First of all, we were able to reproduce the results from [He et al., 2016]. For deeper networks, standard networks do converge only slowly, if they do at all. The residual network however reliably converges even for deep networks. The same holds for Runge-Kutta networks, even though the observed convergence is slower for all considered models, see fig. 3. This concludes the objectives 1, 2 and 3.

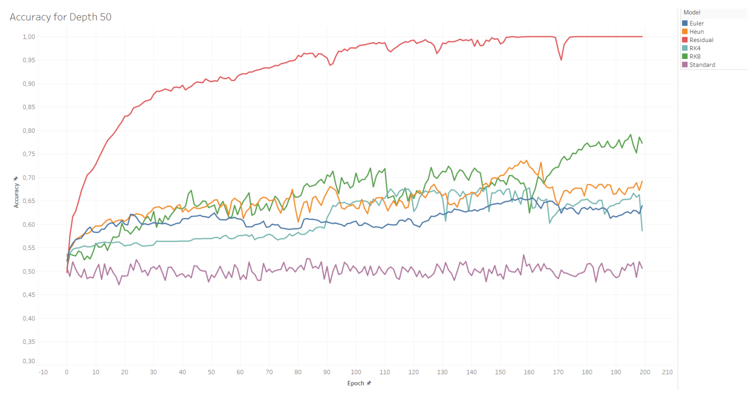

Figure 3: accuracy of depth 50 networks. ResNet (red), Euler (blue), Heun (yellow), RK4 (light blue), RK8 (green) and Standard (purple).

There is no evidence that higher-order methods perform better, especially as the residual network itself is induced by Euler's method. This seems to suggest the analogy to ODEs (objective 5) is not very well suited. Expectedly, methods with more stages take longer to train, see fig. 4.

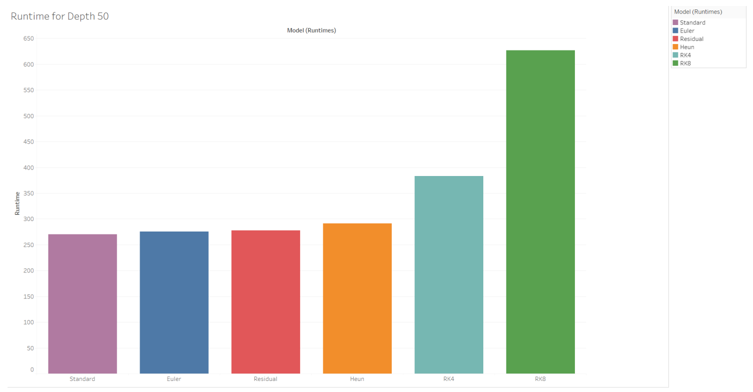

Figure 4: runtime of depth 50 networks

In a second experiment, we also considered different choices of the residual parameter $\Delta t \in \{0.1, 0.5, 0.8, 1, 1.5\}$. As is illustrated in fig. 5, the parameter choice $\Delta t = 1$ does indeed yield the best results (objective 4). This does intuitively make sense because it corresponds to learning the residual or from the perspective of adding layers to an already existing network, it does switch the task of learning an equally good model from learning the identity to learning the easier-to-learn 0 map.

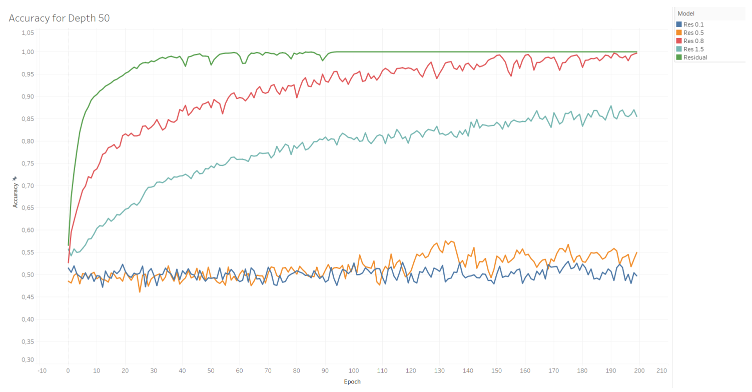

Figure 5: accuracy of depth 50 residual networks. 0.1 (blue), 0.5 (yellow), 0.8 (red), 1 (green) and 1.5 (light blue).

5 Conclusion

We reviewed the breakthrough paper [He et al., 2016] introducing residual connections and significantly improving performance of deep models. We could replicate the results and show that traditional neural networks face major challenges in the case of deep architectures, even for low-complexity datasets. We also showed that the residual network as well as the Runge-Kutta networks, inspired by an analogy between the resiudal network to ODEs, still learn efficiently also for deep networks.

However, we also showed that the residual network has better performance than the Runge-Kutta networks and that the order of the Runge-Kutta method inducing the model architecture does not correlate to model performance. Therefore, we conclude that the analogy even though mathematically elegant, is limited in its practical relevance.

Finally we experimentally verified that the residual parameter $\Delta t = 1$ seems to be the best choice for residual connections.

References

Kendall E. Atkinson. *Numerical Solution of Ordinary Differential Equations*. John Wiley & Sons, Hoboken, NJ, 2008. ISBN 978-0-470-18352-6.

Martin Benning, Elena Celledoni, Matthias J Ehrhardt, Brynjulf Owren, and Carola-Bibiane Schönlieb. Deep learning as optimal control problems: Models and numerical methods. *arXiv preprint arXiv:1904.05657*, 2019. URL `https://doi.org/10.48550/arXiv.1904.05657`.

Tom Feldhausen. Generalized residual connections, 2024. URL `https://github.com/tom22-5/Generalized-Residual-Connections`. Accessed: 2024-12-27.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2016. URL `https://openaccess.thecvf.com/content_cvpr_2016/papers/He_Deep_Residual_Learning_CVPR_2016_paper.pdf`.

Catherine F Higham and Desmond J Higham. Deep learning: An introduction for applied mathematicians. *Siam review*, 61(4):860–891, 2019.

Yann LeCun, D Touresky, G Hinton, and T Sejnowski. A theoretical framework for back-propagation, 1988.

Sik-Ho Tsang. Review: Resnet — winner of ilsvrc 2015 (image classification, localization, detection). *Towards Data Science*, 2018. URL `https://towardsdatascience.com/review-resnet-winner-of-ilsvrc-2015-image-classification-localization-detection-e39402bfa5d8`.

A Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

A Appendix

A.1 Runge-Kutta methods

For a comprehensive introduction to Runge-Kutta methods we recommend [Atkinson, 2008]. Remember our definition of ODEs:

$$y'(t) = f(t, y(t)).$$

We define a m-stage Runge-Kutta method with parameters $b \in \mathbb{R}^m$, $c \in \mathbb{R}^m$ and $A \in \mathbb{R}^{m \times m}$ as

$$y^{[i+1]} = y^{[i]} + \Delta t \sum_{l=1}^{m} b_l \eta_l^{[i]},\ \eta_l^{[i]} = f(t^{[i]} + \Delta t c_l, y^{[i]} + \Delta t \sum_{j=1}^{m} A_{lj} \eta_j^{[i]}).$$

and shorthand write

$$y^{[i+1]} = y^{[i]} + \Delta t \Phi(t^{[i]}, y^{[i]}, y^{[i+1]}, \Delta t),$$

where Φ estimates the slope. If A is lower-triangular, the resulting non-linear system of equations can be solved recursively, and this family of methods is called explicit. We can illustrate the methods in a butcher tableau, see fig. 6.

$$\begin{array}{c|c} c & A \\ \hline & b \end{array}$$

Figure 6: Butcher tableau

A RK-method is convergent of order $p \geq 0$ if for sufficiently smooth functions f the approximation

$$\max_{i=0,\dots,K} ||y(t^{[i]}) - y^{[i]}||_\infty = O(\Delta t^p).$$

holds, where $y(t^{[i]})$ denotes the solution and $y^{[i]}$ the method's approximation. Consistency can be seen as a local version of convergence and is equivalent to $\sum_{l=1}^{m} b_l = 1$. Intuitively this translates to the total stepsize of the method being equal to Δt, a natural condition. Our models are induced by the following consistent methods:

$$\begin{array}{c|c} 0 & 0 \\ \hline & 1 \end{array}$$

Euler (Order 1)

$$\begin{array}{c|cc} 0 & & \\ 1 & 1 & \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}$$

Heun (Order 2)

$$\begin{array}{c|cccc} 0 & & & & \\ \frac{1}{2} & \frac{1}{2} & & & \\ \frac{1}{2} & 0 & \frac{1}{2} & & \\ 1 & 0 & 0 & 1 & \\ \hline & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{array}$$

RK4 (Order 4)

Figure 7: Runge-Kutta methods: Euler, Heun and RK4 (from left to right).