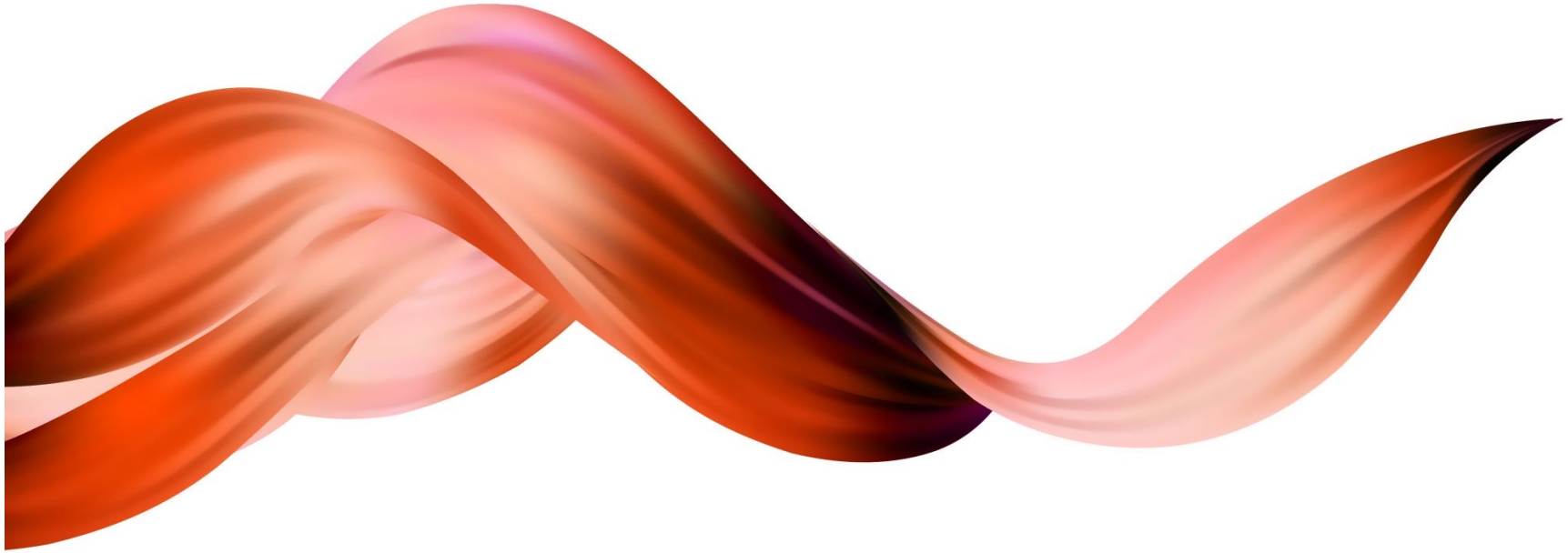
Neural ODEs and Runge Kutta methods

MA8404 – Autumn 2023



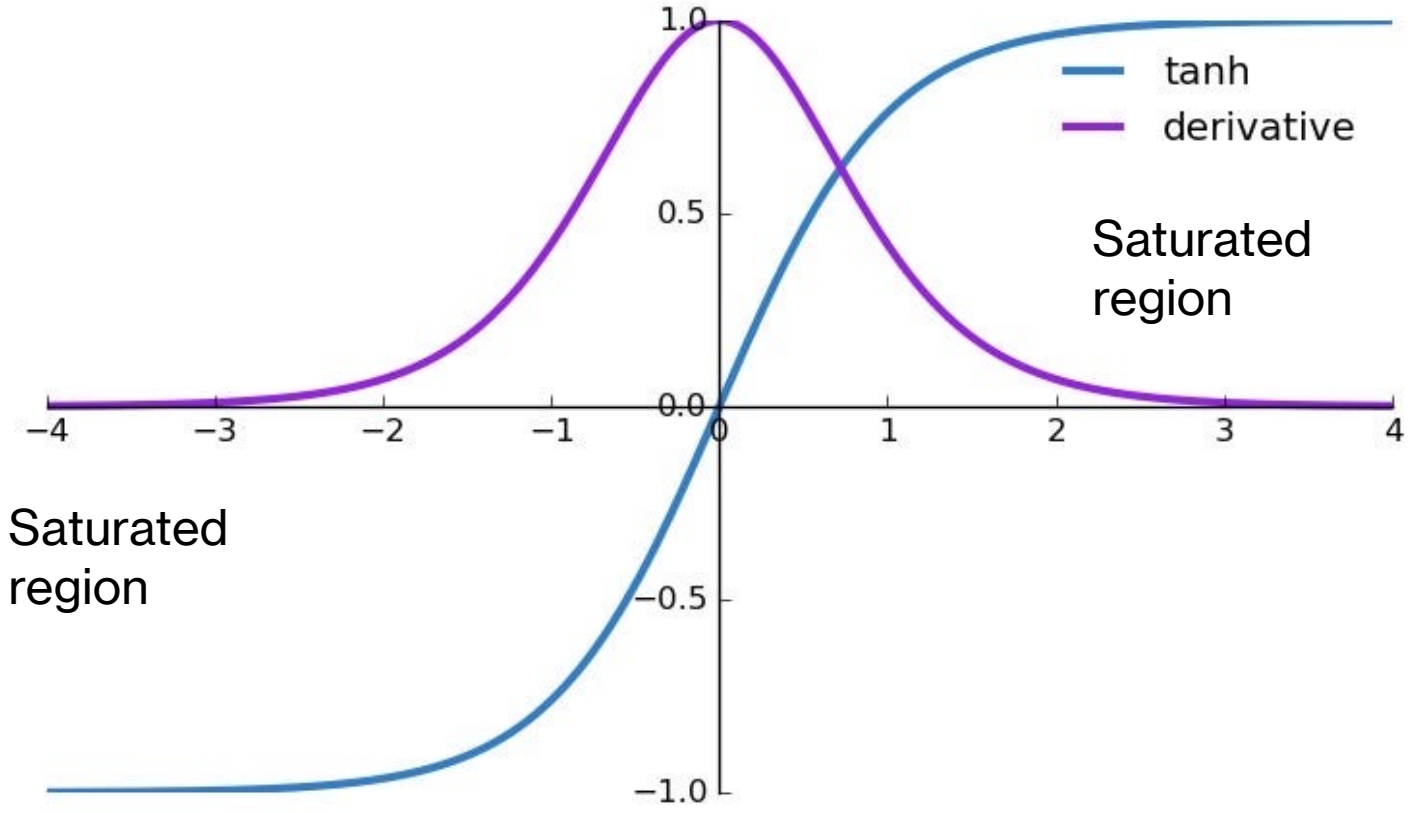
The Vanishing Gradient Problem

- Gradients of the loss function becomes extremely small when backpropagating in the training process -> Slow learning.
- Highly dependent on activation functions, like sigmoid and tanh.
- Stacking -> Problem happens in each layer -> the gradients vanish!
  - Example: n hidden layers –> n derivatives multiplied together –> gradient decreases exponentially as we backpropagate through the neural network.

Example

Tanh squishes large input between -1 and 1, but its derivative only between 0 and 1.

Therefore, the updated weight values are small, and the new weight values are very similar to the old weight values.



Residual Neural Networks (ResNets)

- Solves the vanishing gradient problem
- Skip connections -> Gradients flow easily through network
- The update in each layer looks like discretized Euler with h = 1!

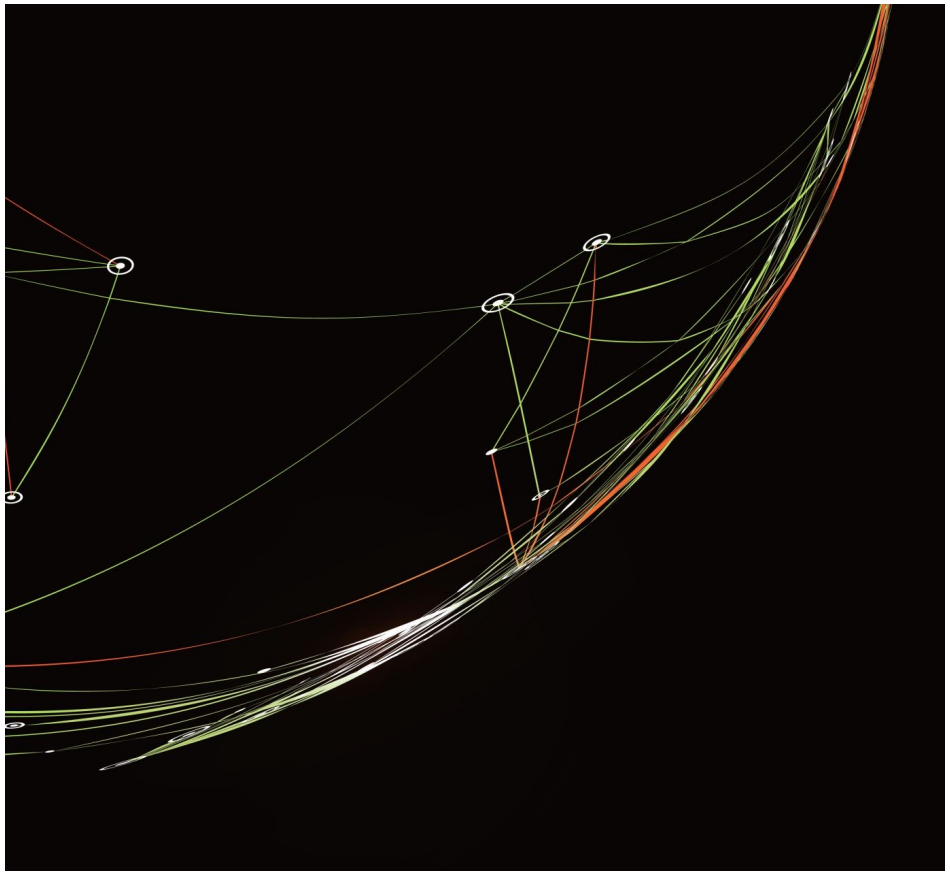
$$\mathbf{h}_{t+1} = \mathbf{h}_t + f(\mathbf{h}_t, \theta_t)$$

$$\frac{d\mathbf{h}(t)}{dt} = f(\mathbf{h}(t), t, \theta)$$

What if we switched it out with a different Explicit Runge Kutta method?

Neural ODEs

- Defines the dynamics of the network continously using ODEs
- Forward pass is an initial value problem
- Can capture continous time behavior and model complex data.
- Models transformations within each layer of a ResNet.

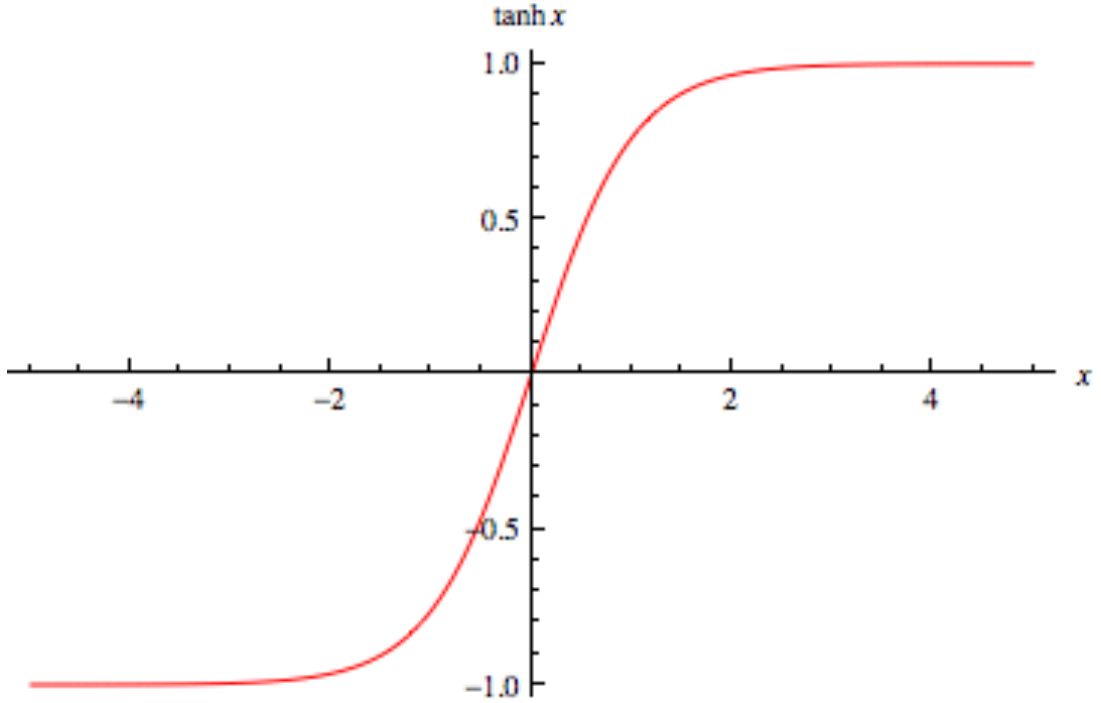


$$\frac{dy}{dt} = \sigma(Wy + b)$$

Where $\sigma(z) = \tanh(z)$.

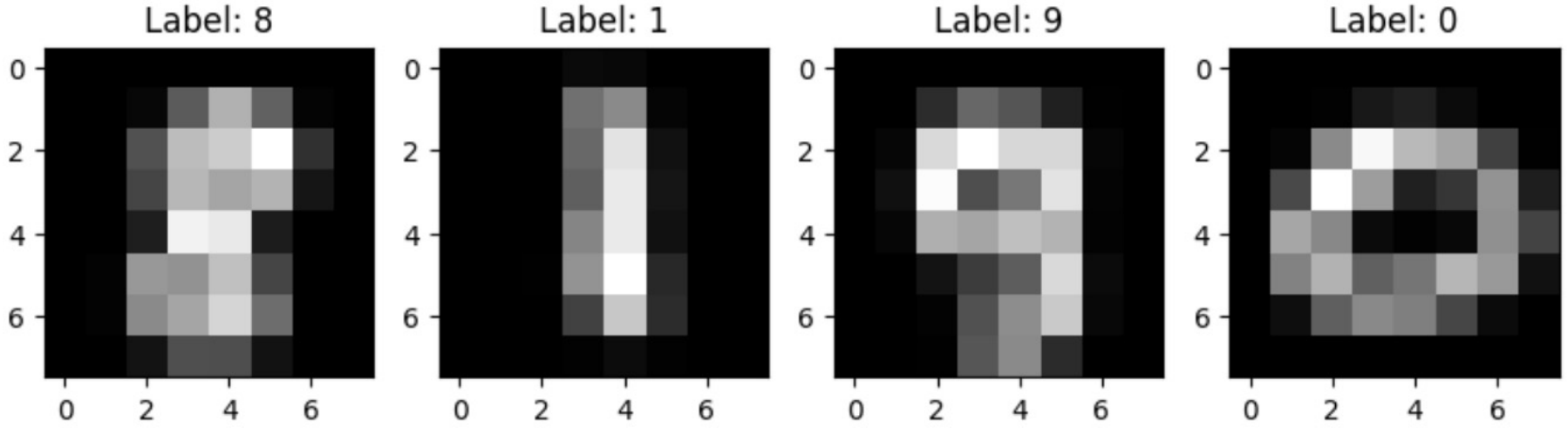
Why tanh?

- Zero-centered (faster convergence, both negative and positive gradients.)
- Smooth and continous -> differentiable, stable and predictable gradients during training.
- Non-Linear –> Enable NNs to learn complex and non-linear relationships in the data.



The Data

- MNIST Dataset (torch library)
- Handwritten digits (with label)
- Only looking at 8x8 pixels. (Reduce computational cost + same size for every datapoint)
- Shuffled -> Avoid overfitting if there is a fixed order



The Runge-Kutta methods

Euler Method

0	0
	1

RK4

0	0	0	0	0
1/2	1/2	0	0	0
1/2	0	1/2	0	0
1	0	0	1	0
	1/6	1/3	1/3	1/6

Dormand-Prince (Dopri5)

0							
1/5	1/5						
3/10	3/40	9/40					
4/5	44/45	–56/15	32/9				
8/9	19372/6561	–25360/2187	64448/6561	–212/729			
1	9017/3168	–355/33	46732/5247	49/176	–5103/18656		
1	35/384	0	500/1113	125/192	–2187/6784	11/84	
	35/384	0	500/1113	125/192	–2187/6784	11/84	0
	5179/57600	0	7571/16695	393/640	–92097/339200	187/2100	1/40

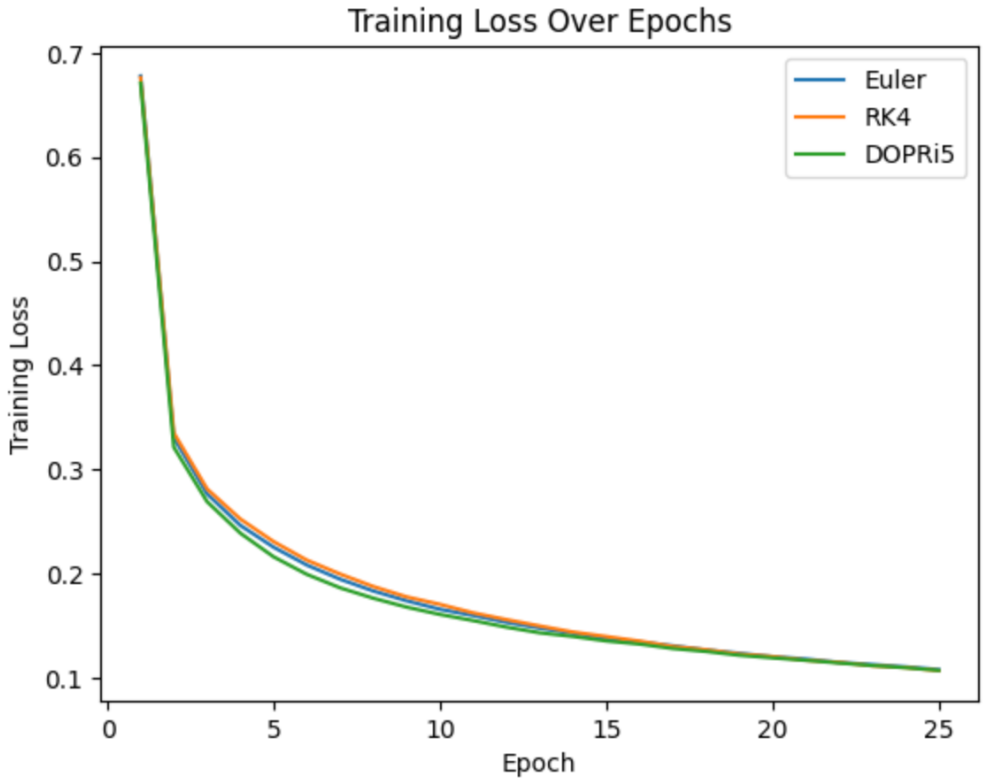
Training Results

TRAINING TIME AND ACCURACY

```
Training time of different methods
Forward Euler: 1.8483364661534627 minutes
RK4 : 3.0864835500717165 minutes
Dopri5 : 18.188071064154308 minutes
```

```
Euler Method Training Accuracy: 96.76%
RK4 Method Training Accuracy: 96.70%
DOPRI5 Training Accuracy: 96.74%
```

LOSS OVER EPOCHS



Testing Results

```
Euler method accuracy on the test set: 96.43%
RK4 method accuracy on the test set: 96.27%
Dopri5 method accuracy on the test set: 96.10%
```

- Very similar accuracy
- Euler method performed the best!

Conclusion

- Higher order Runge-Kutta methods resulted in a significantly longer training time.
- Higher order Runge-Kutta methods does not necessarily produce more accurate predictions.
- However, our data was rather simple. Maybe higher order Runge-Kutta methods are better for more complex data?
- Maybe test neural ODEs for time series?