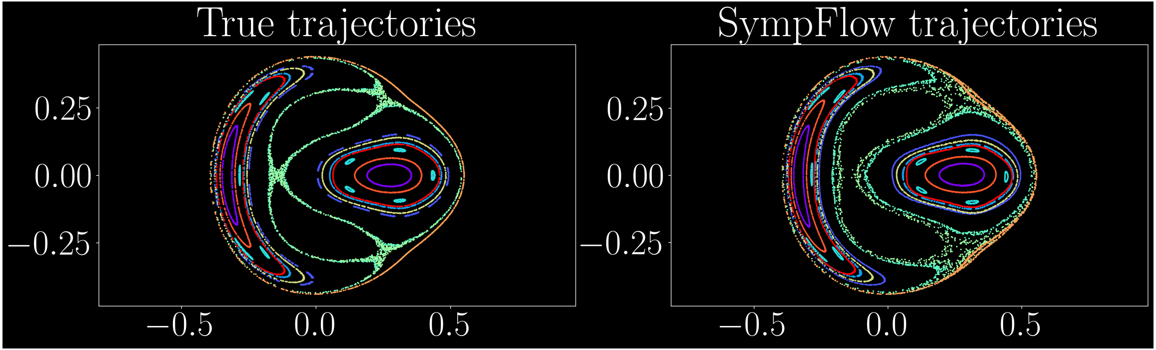
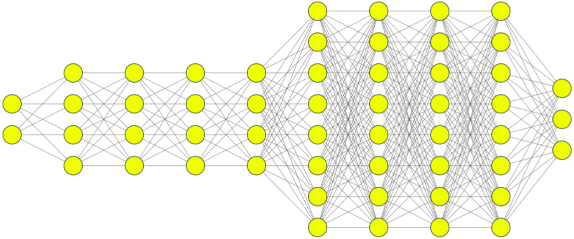
Neural networks and their connections with differential equations



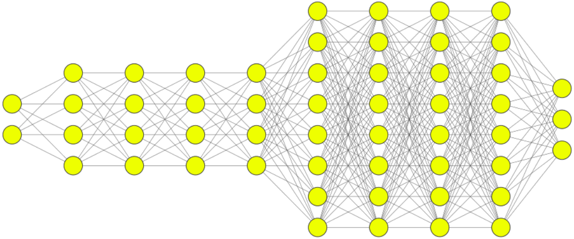
Neural Networks (NNs)

- Neural networks are typically visualised as something like this



Neural Networks (NNs)

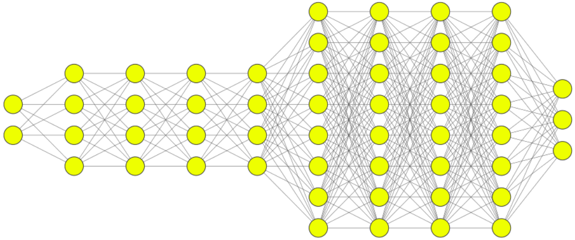
- Neural networks are typically visualised as something like this



- Mathematically, a neural network is just a **parametric map** $\mathcal{N}_\theta : \mathbb{R}^c \to \mathbb{R}^d$, which is usually defined by composing L functions, called **layers**, as $\mathcal{N}_\theta = F_{\theta_L} \circ \ldots \circ F_{\theta_1}$, $F_{\theta_i} : \mathbb{R}^{c_i} \to \mathbb{R}^{c_{i+1}}$, $c_1 = c$, $c_{L+1} = d$.

Neural Networks (NNs)

- Neural networks are typically visualised as something like this



- Mathematically, a neural network is just a **parametric map** $\mathcal{N}_\theta : \mathbb{R}^c \to \mathbb{R}^d$, which is usually defined by composing L functions, called **layers**, as $\mathcal{N}_\theta = F_{\theta_L} \circ \ldots \circ F_{\theta_1}$, $F_{\theta_i} : \mathbb{R}^{c_i} \to \mathbb{R}^{c_{i+1}}$, $c_1 = c$, $c_{L+1} = d$.
- The parametrisation strategy behind $\mathcal{N}_\theta$ is defined by the so-called **neural network architecture**.

Examples of layers

- It is common practice to define layers by alternating linear maps with non-linear functions applied entrywise:

$$F_{\theta_i}(\mathbf{x}) = \Sigma \circ L_i(\mathbf{x}), \;\; \Sigma(\mathbf{x}) := \begin{bmatrix} \sigma(x_1) \\ \vdots \\ \sigma(x_{c_i}) \end{bmatrix}.$$

With a slight abuse of notation, from now on we will use σ both for the scalar function and for the vector function.

Examples of layers

- It is common practice to define layers by alternating linear maps with non-linear functions applied entrywise:

$$F_{\theta_i}(\mathbf{x}) = \Sigma \circ L_i(\mathbf{x}), \;\; \Sigma(\mathbf{x}) := \begin{bmatrix} \sigma(x_1) \\ \vdots \\ \sigma(x_{c_i}) \end{bmatrix}.$$

  With a slight abuse of notation, from now on we will use σ both for the scalar function and for the vector function.
- σ is called **activation function**. Common examples are $\sigma(x) = \mathrm{ReLU}(x) = \max\{0, x\}$, $\sigma(x) = \tanh(x)$, $\sigma(x) = \frac{1}{1+e^{-x}}$.

Examples of layers

- It is common practice to define layers by alternating linear maps with non-linear functions applied entrywise:

$$F_{\theta_i}(\mathbf{x}) = \Sigma \circ L_i(\mathbf{x}), \ \Sigma(\mathbf{x}) := \begin{bmatrix} \sigma(x_1) \\ \vdots \\ \sigma(x_{c_i}) \end{bmatrix}.$$

  With a slight abuse of notation, from now on we will use σ both for the scalar function and for the vector function.
- σ is called **activation function**. Common examples are $\sigma(x) = \mathrm{ReLU}(x) = \max\{0, x\}$, $\sigma(x) = \tanh(x)$, $\sigma(x) = \frac{1}{1+e^{-x}}$.
- Choosing $L_i(\mathbf{x}) = A_i\mathbf{x} + \mathbf{b}_i$, we recover the layer $F_{\theta_i}(\mathbf{x}) = \sigma(A_i\mathbf{x} + \mathbf{b}_i)$, typical of the so-called **fully-connected neural networks**.

Examples of layers

- It is common practice to define layers by alternating linear maps with non-linear functions applied entrywise:

$$F_{\theta_i}(\mathbf{x}) = \Sigma \circ L_i(\mathbf{x}), \;\; \Sigma(\mathbf{x}) := \begin{bmatrix} \sigma(x_1) \\ \vdots \\ \sigma(x_{c_i}) \end{bmatrix}.$$

  With a slight abuse of notation, from now on we will use σ both for the scalar function and for the vector function.
- σ is called **activation function**. Common examples are $\sigma(x) = \mathrm{ReLU}(x) = \max\{0, x\}$, $\sigma(x) = \tanh(x)$, $\sigma(x) = \frac{1}{1+e^{-x}}$.
- Choosing $L_i(\mathbf{x}) = A_i\mathbf{x} + \mathbf{b}_i$, we recover the layer $F_{\theta_i}(\mathbf{x}) = \sigma(A_i\mathbf{x} + \mathbf{b}_i)$, typical of the so-called **fully-connected neural networks**.
- We can also choose $L_i(\mathbf{x}) = k_i * \mathbf{x} + \mathbf{b}_i$, so realise the linear layer by convolution, and get a map that shows up in **convolutional neural networks**

Finding the weights of a NN

- The weights θ of the neural network $\mathcal{N}_\theta$ are usually found by approximately solving a suitable optimisation problem. This optimisation process is called **network training**.

Finding the weights of a NN

- The weights θ of the neural network $\mathcal{N}_\theta$ are usually found by approximately solving a suitable optimisation problem. This optimisation process is called **network training**.
- The cost function which is minimised, called **loss function** in machine learning, is defined thanks to the data one has available, or thanks to properties we would like the approximation to satisfy.

Finding the weights of a NN

- The weights θ of the neural network $\mathcal{N}_\theta$ are usually found by approximately solving a suitable optimisation problem. This optimisation process is called **network training**.
- The cost function which is minimised, called **loss function** in machine learning, is defined thanks to the data one has available, or thanks to properties we would like the approximation to satisfy.
- One of the simplest loss functions we can work with is the **mean-squared error**. Say that we want to approximate the function $F : \Omega \to \mathbb{R}^d$, $\Omega \subset \mathbb{R}^c$, and we have the dataset $\{(\mathbf{x}_i, \mathbf{y}_i = F(\mathbf{x}_i))\}_{i=1}^N$, $\mathbf{x}_i \in \Omega$, then we can work with the loss function

$$\mathcal{L}(\theta) = \frac{1}{N}\sum_{i=1}^N \|\mathcal{N}_\theta(\mathbf{x}_i) - \mathbf{y}_i\|_2^2 .$$

Finding the weights of a NN

- The weights θ of the neural network $\mathcal{N}_\theta$ are usually found by approximately solving a suitable optimisation problem. This optimisation process is called **network training**.
- The cost function which is minimised, called **loss function** in machine learning, is defined thanks to the data one has available, or thanks to properties we would like the approximation to satisfy.
- One of the simplest loss functions we can work with is the **mean-squared error**. Say that we want to approximate the function $F : \Omega \to \mathbb{R}^d$, $\Omega \subset \mathbb{R}^c$, and we have the dataset $\{(\mathbf{x}_i, \mathbf{y}_i = F(\mathbf{x}_i))\}_{i=1}^N$, $\mathbf{x}_i \in \Omega$, then we can work with the loss function

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \|\mathcal{N}_\theta(\mathbf{x}_i) - \mathbf{y}_i\|_2^2 .$$

- After minimising the loss function, we hopefully have a good set of parameters θ^* and we can use $\mathcal{N}_{\theta^*}$ to make new predictions, for unseen inputs.

Universal approximation theorems[1]

Theorem

Let $\Omega \subset \mathbb{R}^c$ be a compact set and assume $\sigma : \mathbb{R} \to \mathbb{R}$ is not a polynomial. For any continuous function $F : \Omega \to \mathbb{R}$ and for any $\varepsilon > 0$ there is a single-layer neural network

$$\mathcal{N}_\theta(x) := \mathbf{w}^\top \sigma(A\mathbf{x} + \mathbf{b}), \; A \in \mathbb{R}^{h \times d}, \mathbf{b}, \mathbf{w} \in \mathbb{R}^h,$$

with $h \in \mathbb{N}$ large enough, such that

$$\max_{\mathbf{x} \in \Omega} |F(\mathbf{x}) - \mathcal{N}_\theta(\mathbf{x})| \leq \varepsilon.$$

[1] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators". In: *Neural networks* 2.5 (1989), pp. 359–366.

Universal approximation theorems[1]

Theorem

Let $\Omega \subset \mathbb{R}^c$ be a compact set and assume $\sigma : \mathbb{R} \to \mathbb{R}$ is not a polynomial. For any continuous function $F : \Omega \to \mathbb{R}$ and for any $\varepsilon > 0$ there is a single-layer neural network

$$\mathcal{N}_\theta(x) := \mathbf{w}^\top \sigma(A\mathbf{x} + \mathbf{b}), \; A \in \mathbb{R}^{h \times d}, \mathbf{b}, \mathbf{w} \in \mathbb{R}^h,$$

with $h \in \mathbb{N}$ large enough, such that

$$\max_{\mathbf{x} \in \Omega} |F(\mathbf{x}) - \mathcal{N}_\theta(\mathbf{x})| \leq \varepsilon.$$

This theorem extends to vector-valued functions, and similar results exist also for deeper networks.

[1] Hornik, Stinchcombe, and White, "Multilayer feedforward networks are universal approximators".

Residual Neural Networks (ResNets)

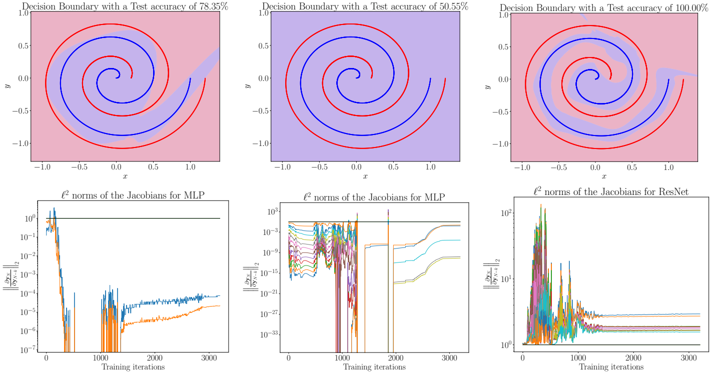
- A particularly interesting network architecture is the one of ResNets. The layers of these networks are of the from
$$F_{\theta_i}(\mathbf{x}) = \mathbf{x} + \mathcal{F}_{\theta_i}(\mathbf{x}),$$
where an example could be $\mathcal{F}_{\theta_i}(\mathbf{x}) = B_i^\top \sigma(A_i \mathbf{x} + \mathbf{b}_i)$, $A_i, B_i \in \mathbb{R}^{h \times c_i}$, $\mathbf{b}_i \in \mathbb{R}^h$.
- The reason why they were introduced is because they are much easier to train when the network has a high number of layers.

Why ResNets?

Recall that to minimise the loss function $\mathcal{L}(\theta)$ we have to use some numerical method, like gradient descent

$$\theta_{k+1} = \theta_k - \tau \nabla L(\theta_k).$$

If $\|\nabla L(\theta_k)\|_2$ is very large or very small, we will struggle to find a meaningful set of weights.



ResNets as dynamical systems

- Residual Neural Networks (ResNets) are networks of the form $\mathcal{N}_\theta = F_{\theta_L} \circ \ldots \circ F_{\theta_1}$ with

$$F_{\theta_i}(\mathbf{x}) = \mathbf{x} + B_i^\top \sigma\left(A_i \mathbf{x} + \boldsymbol{b}_i\right) \in \mathbb{R}^d, \ \mathbf{x} \in \mathbb{R}^d,$$
$$A_i, B_i \in \mathbb{R}^{h\times d}, \ \boldsymbol{b}_i \in \mathbb{R}^h, \ \theta_i = \{A_i, B_i, \boldsymbol{b}_i\}.$$

ResNets as dynamical systems

- Residual Neural Networks (ResNets) are networks of the form $\mathcal{N}_\theta = F_{\theta_L} \circ \ldots \circ F_{\theta_1}$ with

$$F_{\theta_i}(\mathbf{x}) = \mathbf{x} + B_i^\top \sigma\left(A_i \mathbf{x} + \boldsymbol{b}_i\right) \in \mathbb{R}^d,\ \ \mathbf{x} \in \mathbb{R}^d,$$
$$A_i, B_i \in \mathbb{R}^{h\times d},\ \ \boldsymbol{b}_i \in \mathbb{R}^h,\ \ \theta_i = \{A_i, B_i, \boldsymbol{b}_i\}\,.$$

- The layer

$$F_{\theta_i}(\mathbf{x}) = \mathbf{x} + B_i^\top \sigma\left(A_i \mathbf{x} + \boldsymbol{b}_i\right) = \mathbf{x} + \mathcal{F}_{\theta_i}(\mathbf{x}) \in \mathbb{R}^d$$

is an explicit Euler step of size 1 for the initial value problem

$$\begin{cases} \dot{\mathbf{y}}(t) = B_i^\top \sigma(A_i \mathbf{y}(t) + \boldsymbol{b}_i) = \mathcal{F}_{\theta_i}(\mathbf{y}(t)), \\ \mathbf{y}(0) = \mathbf{x} \end{cases} .$$

ResNet-like archtectures

- We can define ResNet-like neural networks by choosing a family of parametric functions $\mathcal{S}_\Theta = \left\{\mathcal{F}_\theta : \mathbb{R}^d \to \mathbb{R}^d : \theta \in \Theta\right\}$ and a numerical method $\varphi_{\mathcal{F}}^h$, like explicit Euler defined as $\varphi_{\mathcal{F}}^h(\mathbf{x}) = \mathbf{x} + h\mathcal{F}(\mathbf{x})$, and set

$$\mathcal{N}_\theta(\mathbf{x}) = \varphi_{\mathcal{F}_{\theta_L}}^{h_L} \circ \cdots \circ \varphi_{\mathcal{F}_{\theta_1}}^{h_1}(\mathbf{x}), \ \mathcal{F}_{\theta_1}, ..., \mathcal{F}_{\theta_L} \in \mathcal{S}_\Theta.$$

- We could also combine these residual blocks with lifting and projection layers, as for usual neural networks.

Example

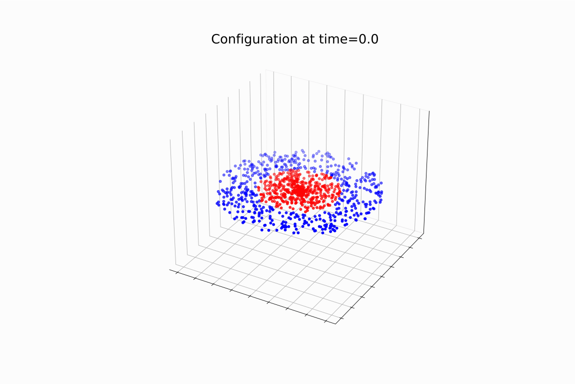


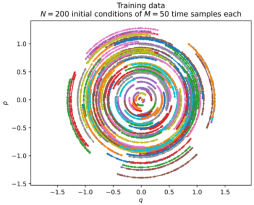
Figure 1: Action of a ResNet based on dynamical systems of the form $B_i^\top \sigma(A_i \mathbf{x} + \mathbf{b}_i)$ trained to distinguish the red from the blue points.

Neural networks for dynamical systems discovery

- Apart from using dynamical systems and numerical analysis to study neural networks, we can also use neural networks to solve and discover differential equations.

Neural networks for dynamical systems discovery

- Apart from using dynamical systems and numerical analysis to study neural networks, we can also use neural networks to solve and discover differential equations.
- The task of dynamical systems discovery can be summarised as follows:



$$\Longrightarrow \begin{bmatrix} \dot{q} \\ \dot{p} \end{bmatrix} = \begin{bmatrix} p \\ -q \end{bmatrix}.$$

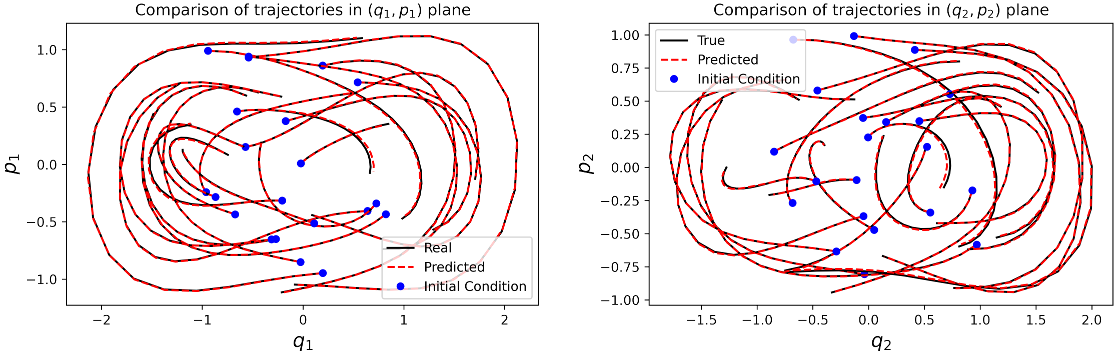
- To train the overall model $\mathcal{N}_\theta$ we can minimise the loss function

$$\mathcal{L}(\theta) = \frac{1}{N}\sum_{n=1}^{N} \left\| \varphi^h_{\mathcal{N}_\theta}(\mathbf{x}_0^n) - \mathbf{x}_1^n \right\|_2^2,$$

where $\mathbf{x}_0^n \in \Omega \subset \mathbb{R}^d$, and $\mathbf{x}_1^n \approx \phi^h(\mathbf{x}_0^n)$.

Example with Hamiltonian system

$$H(q,p) = \frac{1}{2}\begin{bmatrix} p_1 & p_2 \end{bmatrix}^\top \begin{bmatrix} 5 & -1 \\ -1 & 5 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} + \frac{q_1^4 + q_2^4}{4} + \frac{q_1^2 + q_2^2}{2}.$$



Neural networks solving differential equations

- We can also use neural networks to solve differential equations on a certain time interval $[0, T]$, and for initial conditions in $\Omega \subset \mathbb{R}^d$.

Neural networks solving differential equations

- We can also use neural networks to solve differential equations on a certain time interval $[0, T]$, and for initial conditions in $\Omega \subset \mathbb{R}^d$.
- We can define a network $\mathcal{N}_\theta : [0, T] \times \mathbb{R}^d \to \mathbb{R}^d$. We can also enforce the initial condition, so that $\mathcal{N}_\theta(0, \mathbf{x}_0) = \mathbf{x}_0$ for every $\mathbf{x}_0 \in \mathbb{R}^d$. This can be done for example by defining

$$\mathcal{N}_\theta(t, \mathbf{x}) = \mathbf{x} + \widetilde{\mathcal{N}}_\theta(t, \mathbf{x}) - \widetilde{\mathcal{N}}_\theta(0, \mathbf{x}),$$

for an arbitrary network $\widetilde{\mathcal{N}}_\theta : [0, T] \times \Omega \to \mathbb{R}^d$.

Neural networks solving differential equations

- We can also use neural networks to solve differential equations on a certain time interval $[0, T]$, and for initial conditions in $\Omega \subset \mathbb{R}^d$.
- We can define a network $\mathcal{N}_\theta : [0, T] \times \mathbb{R}^d \to \mathbb{R}^d$. We can also enforce the initial condition, so that $\mathcal{N}_\theta(0, \mathbf{x}_0) = \mathbf{x}_0$ for every $\mathbf{x}_0 \in \mathbb{R}^d$. This can be done for example by defining

$$\mathcal{N}_\theta(t, \mathbf{x}) = \mathbf{x} + \widetilde{\mathcal{N}}_\theta(t, \mathbf{x}) - \widetilde{\mathcal{N}}_\theta(0, \mathbf{x}),$$

  for an arbitrary network $\widetilde{\mathcal{N}}_\theta : [0, T] \times \Omega \to \mathbb{R}^d$.
- To train $\mathcal{N}_\theta$ we can minimise the loss function

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{n=1}^{N} \left\| \frac{d}{dt} \mathcal{N}_\theta \left(t, \mathbf{x}_0^n \right) \Big|_{t=t_n} - \mathcal{F}\left(\mathcal{N}_\theta \left(t_n, \mathbf{x}_0^n \right) \right) \right\|_2^2$$

  at sufficiently many collocation points $t_n \in [0, T]$ and $\mathbf{x}_0^n \in \Omega \subset \mathbb{R}^d$.

Example: Hénon–Heiles

Equations of motion

$$\dot{q}_1 = p_1, \ \dot{q}_2 = p_2, \ \dot{p}_1 = -q_1 - 2q_1q_2, \ \dot{p}_2 = -q_2 - (q_1^2 - q_2^2).$$