

Copyright Notice

© 2026 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

The published version is available at: <https://doi.org/10.1109/MCS.2026.3715420>

Deep learning from a control vantage point

FEEDFORWARD, RESIDUAL, TRANSFORMERS, LLMS, AND SELECTIVE STATE-SPACE MODELS

Á. Rodríguez Abella[□], J. P. Silvestre[□], B. Ghahserifard[□], and P. Tabuada[□]

[□]Comillas Pontifical University, [□]Queens University, [□]University of California at Los Angeles

Deep neural networks, in their different shapes and flavors, are having a remarkable impact in almost every area of science and engineering. While they are also influencing the field of control, the purpose of this article is to discuss recent efforts in using ideas from control to better understand and design neural networks.

The connections between neural networks and control are both old and new. It was already observed in the '80s that backpropagation, the main technique to train neural networks, is closely related to optimal control. A different connection emerged more recently, motivated by a class of neural networks termed *residual neural networks*. It was noticed that their update equations are similar to those resulting from the forward Euler discretization of differential equations. This insight motivated the use of differential equation models, and control systems in particular, to describe deep neural networks. Consequently, several research efforts were initiated to tackle a range of problems in deep learning by leveraging the interplay between control and deep neural networks.

It is from this control vantage point that this article offers a gentle introduction to deep learning with the objective of making its key ideas more accessible to a control audience while emphasizing how control theory has played, and continues to play, a fundamental role in deep learning. The choice of contents is based on the last author's plenary talk at the 2024 IEEE CDC and it reflects the authors' idiosyncratic view of the area. Any omission of relevant research efforts is simply a consequence of the authors' incomplete view of the rapidly moving interface

between control theory and deep learning. Other articles adopting a control theory perspective include [1], [2] and complement this article.)

Notwithstanding the topic of this article, the authors emphatically state that no AI tools were used to generate its contents.

Summary

This article offers an overview of recent developments in deep learning from a control vantage point. It reviews several classes of models including feedforward and residual neural networks, transformers, large-language models, and selective state space models while explaining how control theory plays an important role in understanding their training, properties, or architecture design. The article explains how residual neural networks can be interpreted as the discretization of a continuous-time control system thereby enabling the use of control theoretic techniques for its analysis. To illustrate the power of this bridge between control and deep learning, two technical results are discussed: 1) the ability of a residual neural network to approximate arbitrary continuous functions is a non-standard controllability problem that can be solved using geometric nonlinear control and 2) transformers, the key architectural component of most large language-models, can be seen the discretization of a consensus algorithm and thus large language-models exhibit consensus. Another point of contact between control theory and deep learning is the connection between backpropagation, the algorithm used to train neural net-

Notwithstanding the topic of this article, the authors emphatically state that no AI tools were used to generate its contents.

works, and optimal control. The article addresses how the older connection between these problems, established in the 80's using discrete-time models can be reconciled with the more recent approach using continuous-time models. Several open problems and directions for future research are discussed, including selective state-space models, a recent class of models inspired by the models used in control.

A CONCISE HISTORICAL ACCOUNT OF ARTIFICIAL NEURAL NETWORKS

Despite the recent rise into prominence of neural networks (NNs), their origins date back to the '40s. This section provides a brief overview of the historical context of NNs. Readers not familiar with NNs are encouraged to first read the sidebar "What are feedforward neural networks?".

NNs were originally introduced as simple models of biological neurons and synapses. They were composed of two main components: neurons and the connections between them. An influential early model was introduced by McCulloch and Pitts in 1943 [3], who described neurons as Boolean threshold elements. Building on this line of work, Rosenblatt introduced the perceptron in 1958 [4], usually regarded as one of the first trainable artificial NN architectures. Mathematically, each neuron can be described by a function transforming the input $z_{in} \in \mathbb{R}^n$ into the output

$$z_{out} = \sigma(w z_{in} + b) \in \mathbb{R}, \quad (0.1)$$

where $w \in \mathbb{R}^{1 \times n}$ denotes the *weights*, $b \in \mathbb{R}$ denotes the *bias*, and $\sigma : \mathbb{R} \rightarrow \mathbb{R}$ is the *activation function*

$$\sigma(z) = \begin{cases} 1, & z \geq 0, \\ 0, & z < 0. \end{cases} \quad (0.2)$$

The perceptron was a trainable network of such artificial neurons with supervised learning used to adjust its parameters.

This network could, in theory, solve any classification task whose classes are linearly separable, see "Classification vs regression tasks" sidebar. However, as noted in the book *Perceptrons* (1969) [5], it could not solve simple nonlinear problems such as classifying the inputs of the XOR function into the class $\{(0,0), (1,1)\}$, mapped by the XOR function to 0, and the class $\{(0,1), (1,0)\}$, mapped to 1. This limitation arises from the architectural limitations of single layer perceptron networks, and the lack, at the time, of an effective method for training networks with

multiple layers. The book [5] is often cited as a contributing factor to the so-called "AI Winter", a period where artificial intelligence (AI) research stagnated dramatically. For an overview of how AI relates to NNs and deep learning see the sidebar "AI, machine learning, or deep learning?".

The main breakthrough that pushed the field forward was the introduction of backpropagation. This approach introduced differentiable activation functions, such as sigmoids, along with a practical method for training multilayer NNs [6], [7]. Other derivations of backpropagation appeared earlier in different contexts such as [8]. Although these developments still shared some similarities with biological models, such as the sigmoid function used in the Wilson-Cowan model [9], the connection between artificial NNs and neuroscience gradually weakened. By the late '80s, the two fields had largely diverged, with NNs increasingly driven by empirical results rather than biological realism. Nevertheless, some later models such as spiking NNs [10] retained closer ties to neuroscience, drawing inspiration from classical models including the Hodgkin-Huxley equations [11].

As the field progressed, many other NN architectures were introduced. Among the most prominent were recurrent neural networks (RNNs) [12], which were constructed as sequence-to-sequence models, that is, as systems transforming input signals into output signals. RNNs possess, in theory, the ability to retain information over arbitrarily long time horizons, allowing current outputs to be related to past inputs. In practice, this capability is limited by several issues, most notably the vanishing gradient problem (the use of gradients in the training of NNs is discussed in *Optimal control and backpropagation*) [13]. Despite these limitations, RNNs were widely used.

Among the many different architectures that appeared in the literature, the following three have deep connections to control: residual neural networks, transformers, and selective state-space models. They are described in more detail in the sidebars "What are feedforward neural networks?", "What are residual neural networks?", "What are transformers?", and "What are recurrent neural networks and selective state-space models?"

DIFFERENTIAL EQUATIONS, CONTROL SYSTEMS, AND NEURAL NETWORKS

As discussed in *A concise historical account of artificial neural networks*, differential equations were already used in the

CLASSIFICATION VS REGRESSION TASKS

Classification and regression are two common machine learning tasks. Both tasks can be seen as the learning of a map $\Phi : X \rightarrow Y$. The goal of *classification* is to separate data into finitely many different classes, hence, Y is a finite set.

The classes are said to be linearly separable when $X = \mathbb{R}^m$, for some $m \in \mathbb{N}$, and they can be separated by an affine subspace of X .

In a *regression* task, data is to be mapped to real-valued outputs and thus Y is a subset of $\mathbb{R}^n$, for some $n \in \mathbb{N}$.

AI, MACHINE LEARNING, OR DEEP LEARNING?

AI is the acronym for Artificial Intelligence, a term that encompasses a multitude of techniques aiming at creating machine intelligence. One such technique is machine learning but AI also includes other techniques such as logical and

symbolic reasoning. Machine learning focuses on algorithms that learn from data while deep learning is a narrower class of such algorithms utilizing deep neural networks, that is, neural networks with a large number of layers.

Consequently, (0.19) can be regarded as a control system and the problem of training a NN becomes an open-loop control design problem.

'50s to model *biological* NNs. Surprisingly, it was recently observed that differential equations can also be used to describe *artificial* NNs. This recent connection was motivated by an architecture called residual networks (ResNets), see sidebar "What are residual neural networks?" for a more detailed description of ResNets.

What distinguishes ResNets from other architectures is how the state at layer $k \in \{1, \dots, \ell - 1\}$, denoted by $z(k) \in \mathbb{R}^n$, gives rise to the state at layer $k + 1$

$$z(k + 1) = z(k) + w(k)\Sigma(W(k)z(k)). \quad (0.18)$$

The function $\Sigma : \mathbb{R}^n \rightarrow \mathbb{R}^n$ describes the action of an activation function $\sigma : \mathbb{R} \rightarrow \mathbb{R}$ on each coordinate

$$\Sigma(v) = (\sigma(v_1), \sigma(v_2), \dots, \sigma(v_n)).$$

The argument of Σ in (0.18) is the linear function of $z(k)$ defined by the weights $W(k) \in \mathbb{R}^{n \times n}$. Note also the presence of the scalar weight $w(k) \in \mathbb{R}$. Although the original formulation in [18] allows for w to be a matrix, it is pedagogical to focus on the scalar case.

It was observed in [30], [31], [32], [33] that (0.18) has a striking similarity with the numerical solution of the differential equation

$$\dot{z}_c(t) = \Sigma(W_c(t)z_c(t)) \quad (0.19)$$

given by the forward Euler method

$$z(t_k + \tau) = z(t_k) + \tau \Sigma(W_c(t_k)z(t_k)),$$

where $\tau \in \mathbb{R}^+$ is the step size and $t_k = k\tau$. Hence, if the weights $w(k)$ are small, it is conceivable that ResNets can be regarded as the discretization of (0.19). Before elaborating further on the possible equivalence between ResNets and differential equations, it matters to delve

into the control perspective afforded by this connection. In (0.19), W_c is a map $W_c : [0, 1] \rightarrow \mathbb{R}^{n \times n}$ transforming time $t \in [0, 1]$ into a matrix of weights. For convenience, the time domain was normalized to $[0, 1]$. The map W_c is to be learned from data, that is, it is to be designed. This suggests that W_c can be interpreted as a control input. Consequently, (0.19) can be regarded as a control system and the problem of training a NN becomes an open-loop control design problem. This perspective brings into clear view how control theory can play a fundamental role in deep learning. Examples of how this perspective brought solutions to new and existing problems include an alternative connection between backpropagation and control (an earlier connection was identified in the '80s), establishing deep neural networks as universal function approximators, and explaining the attention mechanism in transformers through consensus.

Since the interplay between control systems and ResNets is the unifying theme of this article, a detailed discussion of when ResNets can be seen as the discretization of differential equations, or a differential equation can be seen as the continuous limit of a ResNet, is in order. Although outside the scope of this article, there is also a parallel line of work relating NNs and partial differential equations [34], [35], [36].

Fixed inputs

Consider first the case where the control input $W_c : [0, 1] \rightarrow \mathbb{R}^{n \times n}$ is known and fixed where we normalized time to the interval $[0, 1]$ for simplicity. This represents situations where analysis or design was

WHAT ARE FEEDFORWARD NEURAL NETWORKS?

A neural network (NN) is a family of functions $\Phi_W : \mathbb{R}^{n_1} \rightarrow \mathbb{R}^o$ parameterized by a set of weights W . The function Φ_W is built by composing simpler functions

$$\Phi_W = h \circ g_{\ell-1} \circ g_{\ell-2} \circ \dots \circ g_1, \quad (\text{S3})$$

where each $g_k : \mathbb{R}^{n_k} \rightarrow \mathbb{R}^{n_{k+1}}$ takes a different form depending on the architecture and $h : \mathbb{R}^{n_\ell} \rightarrow \mathbb{R}^o$ is chosen according to the task to be performed. The length or depth of the NN is denoted by $\ell \in \mathbb{N}$. A NN is said to be a *deep neural network* when ℓ is large. The domains and co-domains of the functions g_i are often interpreted as layers, especially when a NN is represented as a graph, see Figure S1. Under such interpretation, the state $z(k) \in \mathbb{R}^{n_k}$ describes the contents of each of the n_k neurons in layer k and the function g_k transforms the state $z(k)$ of layer k to the state $z(k+1) \in \mathbb{R}^{n_{k+1}}$ of layer $k+1$. In general, different layers have different dimensions. For notational simplicity, fix $n \stackrel{\text{def}}{=} \max_k n_k$, and lift each function $g_k : \mathbb{R}^{n_k} \rightarrow \mathbb{R}^{n_{k+1}}$ to the function $\tilde{g}_k : \mathbb{R}^n \rightarrow \mathbb{R}^n$

$$\tilde{g}_k(z) = (g_{k,1}(z_1, \dots, z_{n_k}), \dots, g_{k,n_{k+1}}(z_1, \dots, z_{n_k}), \underbrace{0, \dots, 0}_{n-n_{k+1} \text{ zeros}})$$

that has the same domain and codomain. In this article g_k is used to directly denote its lifted version.

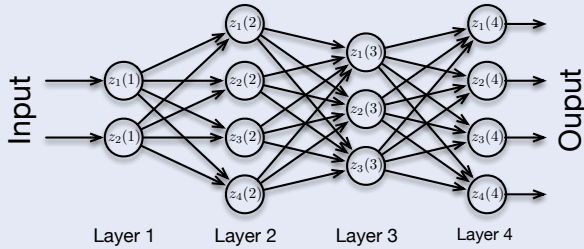


FIGURE S1 Neural network depicted as a graph. The vertically stacked circles represent neurons and form a layer. The arrows from layer k to layer $k+1$ represent the different entries of the weight matrix $W(k)$.

Feedforward Networks

Feedforward networks are among the earliest NN architectures and still widely used. In a feedforward NN, data moves only in one direction: it enters at the first or input layer as $z(1)$, is successively transformed by the sequence of functions g_k producing $z(k+1)$, and results in the output $h(z(\ell))$. In feedforward NNs the maps g_k take the form

$$g_k(z) = \Sigma(W(k)z), \quad (\text{S4})$$

where the function $\Sigma : \mathbb{R}^n \rightarrow \mathbb{R}^n$ operates element-wise, that is

$$\Sigma(v) = (\sigma(v_1), \sigma(v_2), \dots, \sigma(v_n))$$

with $\sigma : \mathbb{R} \rightarrow \mathbb{R}$ an activation function. The argument of Σ is the linear function of z defined by the weights $W(k) \in \mathbb{R}^{n \times n_k}$. In the literature, the argument of Σ is often an affine function $W(k)z + b(k)$. To simplify the presentation, a linear representation is used; it can be obtained by writing $W(k)z + b(k)$ in the linear form

$$\begin{bmatrix} W(k) & b(k) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} z \\ 1 \end{bmatrix}.$$

In this article W and z are used to directly denote their lifted variants.

Classical feedforward NNs used the logistic sigmoid $\sigma(v_i) = (1 + e^{-v_i})^{-1}$ or the hyperbolic tangent $\sigma(v_i) = \tanh(v_i)$ as activation functions. While smooth, these functions can saturate and attenuate gradients thus making training more challenging. To facilitate deeper architectures, modern networks predominantly employ the Rectified Linear Unit (ReLU), $\sigma(v_i) = \max\{0, v_i\}$ [14], and its variants. More recently, smooth gating-style activations like the Gaussian Error Linear Unit (GELU) [15], defined as $\sigma(v_i) = \frac{v_i}{2} \left(1 + \text{erf}\left(\frac{v_i}{\sqrt{2}}\right)\right)$, $\text{erf}(v_i) = \frac{2}{\sqrt{\pi}} \int_0^{v_i} e^{-t^2} dt$, have become standard in transformer-based models.

Task Dependence

The choice of the function h in (S3) must be made according to the task being solved. Two common cases are:

Regression: In a regression problem the output of a NN is an element of $\mathbb{R}^o$ and it is common to choose h to be affine

$$h(z(\ell)) = Az(\ell) + c, \quad (\text{S5})$$

where $A \in \mathbb{R}^{o \times n}$ and $c \in \mathbb{R}^o$ are weights learned from data.

Classification: In a classification problem, the output is the class to which the input belongs. Rather than directly producing the class, NNs produce a vector that is interpreted as the probability of the input belonging to each class. Hence, if there are $o \in \mathbb{N}$ classes, the output is a vector in

$$\Delta_o \stackrel{\text{def}}{=} \left\{ p \in \mathbb{R}^o : p_i \geq 0, \sum_{i=1}^o p_i = 1 \right\}.$$

The output map h in (S3) is the composition of two maps. The first is an affine map acting on $z(\ell)$ and producing *logits* $q \in \mathbb{R}^o$

$$q = Az(\ell) + c,$$

and the second converts the logits into an element $p \in \Delta_o$ using the softmax function

$$p_i = \text{softmax}_i(q) \stackrel{\text{def}}{=} \frac{e^{q_i}}{\sum_{j=1}^o e^{q_j}}, \quad i \in \{1, \dots, o\}. \quad (\text{S6})$$

Each p_i is then interpreted as the probability of the given input $z(1)$ belonging to class i .

A more detailed introduction to this class of networks can be found in [16, Chapter 6].

WHAT ARE RESIDUAL NEURAL NETWORKS?

In recent years, the surge in computational power and large-scale datasets has enabled the training of substantially deeper networks, beginning with milestones such as AlexNet [17], which established convolutional neural networks as the standard for visual recognition. However, as depth increases, the training of such networks becomes notoriously difficult. Deep residual networks (ResNets) [18] addressed this by introducing residual (skip) connections, which allow the gradient to bypass intermediate layers, see Figure S1.

Formally, the functions g_k (see (S3)) in a ResNet are of the form

$$g_k(z) = z + W_2(k)\Sigma(W_1(k)z), \quad (S7)$$

where $W_1(k), W_2(k) \in \mathbb{R}^{n \times n}$ are weight matrices. It has been

empirically shown that this architectural change, with respect to feedforward NNs, effectively smoothens the training loss landscape and mitigates the vanishing gradient problem [19] (the use of gradients in the training of NNs is discussed in *Optimal control and backpropagation*) thereby enabling the reliable training of ResNets with hundreds of layers.

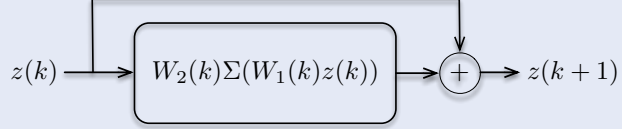


FIGURE S1 Graphical description of a skip connection in a ResNet.

conducted using the model (0.19) and the resulting conclusions are to be carried over, or implemented, on a ResNet. One example is the use of (0.19) to establish existence of inputs guaranteeing universal approximation, as discussed in *Universal approximation*, and then seeking to determine if ResNets have the same ability. Using the Euclidean norm, denoted by $\|\cdot\|$, questions of this type can be distilled to:

Given a desired error tolerance $\varepsilon \in \mathbb{R}^+$, is there a number of layers $\ell \geq 2$ and a choice of weights w so that the mismatch between the solution $z_c : [0, 1] \rightarrow \mathbb{R}^n$ of $\dot{z}_c(t) = \Sigma(W_c(t)z_c(t))$ and the solution $z : \{1, 2, \dots, \ell\} \rightarrow \mathbb{R}^n$ of the recursion

$$z(1) = z_c(0),$$

$$z(k+1) = z(k) + w(k)\Sigma\left(W_c\left(\frac{k-1}{\ell-1}\right)z(k)\right), \quad (0.20)$$

satisfies

$$\left\|z_c\left(\frac{k-1}{\ell-1}\right) - z(k)\right\| \leq \varepsilon \quad (0.21)$$

for every $k \in \{1, \dots, \ell\}$?

Note how the control input $W_c : [0, 1] \rightarrow \mathbb{R}^{n \times n}$ directly defines the weights $W : \{1, 2, \dots, \ell\} \rightarrow \mathbb{R}^{n \times n}$ of the ResNet

$$W(k) = W_c\left(\frac{k-1}{\ell-1}\right)$$

and the only degrees of freedom are the number of layers ℓ and the weights w that play the role of step size. This is a classical problem in the numerical analysis of differential equations and the answer is affirmative under suitable technical conditions, see, for example [37]. A particular choice of such conditions is continuous differentiability of Σ , boundedness of W_c , and the assumption that (0.21) only needs to hold for solutions with $z_c(0)$ belonging to a compact set. Note that implicit in the problem formulation is that solutions of (0.19) exist in the time domain $[0, 1]$. In general, lower values of ε will require larger ℓ , that is, deeper ResNets. Since ε in (0.21) can be rendered as small as desired, the differential equation model (0.19) can be

seen as the limit of a ResNet when $\ell \rightarrow \infty$ for a suitable choice of weights w such as $w(k) = 1/\ell$.

Learned inputs

Whereas the case of known and fixed control inputs can be resolved using numerical analysis, there are situations where the control inputs are not known beforehand. One such example occurs when the model (0.19) is used to design an algorithm that learns $W_c : [0, 1] \rightarrow \mathbb{R}^{n \times n}$ from data. Denote such algorithm by a map

$$\phi_c : (\mathbb{R}^n \times \mathbb{R}^n)^s \rightarrow (\mathbb{R}^{n \times n})^{[0,1]}$$

sending a data set $\mathcal{D} \in (\mathbb{R}^n \times \mathbb{R}^n)^s$ into $W_c = \phi_c(\mathcal{D})$. The notation $(\mathbb{R}^{n \times n})^{[0,1]}$ denotes the set of all maps from $[0, 1]$ to $\mathbb{R}^{n \times n}$ and a data set is a collection of pairs $\mathcal{D} = ((x_1, y_1), (x_2, y_2), \dots, (x_s, y_s))$ where y_i is the evaluation of some unknown function on the data point x_i . When using supervised learning, the objective of training a NN is to learn the values of the weights so that the NN approximates the unknown function. Training of NNs is described in more detail in *Optimal control and backpropagation*. Typically, ϕ_c models the construction of W_c through the minimization of a loss functional based on the training data

$$W_c = \operatorname{argmin}_{\widehat{W}_c} \int_0^1 L_c(\mathcal{D}, \widehat{W}_c(t)) dt. \quad (0.22)$$

In order to implement such algorithm, (0.19) is discretized to obtain a ResNet and ϕ_c is discretized to obtain a family of maps

$$\phi_\ell : (\mathbb{R}^n \times \mathbb{R}^n)^s \rightarrow (\mathbb{R}^{n \times n})^{\{1, 2, \dots, \ell\}}$$

parameterized by the length ℓ of the network. When ϕ_c is defined through (0.22), ϕ_ℓ would also be defined by the minimization of a suitably discretized loss

$$W = \operatorname{argmin}_{\widehat{W}} \sum_{k=1}^{\ell-1} L_\ell(\mathcal{D}, \widehat{W}(k)). \quad (0.23)$$

WHAT ARE TRANSFORMERS?

A common requirement in modern applications of NNs is to approximate functions mapping signals to signals, also known as sequence-to-sequence mappings. One example is the translation of text from one language to another. In such settings, the input is a sequence

$$(s_1, \dots, s_m), \quad s_i \in \mathbb{R}^n \quad (\text{S8})$$

and the output can be a sequence or a classification label. It has been empirically shown that the transformer architecture [20] outperforms feedforward NNs in learning sequence-to-sequence maps. Transformers handle sequences by building intermediate token representations and repeatedly mixing information across different elements of the sequence using an *attention* mechanism. Attention mechanisms were popularized in neural machine translation [21] and later instantiated as *self-attention* in transformers [20]. This sidebar provides a brief introduction to these concepts.

The transformer architecture can still be described by a map similar to (S3)

$$\Phi_W = h \circ g_{\ell-1} \circ g_{\ell-2} \circ \dots \circ g_1 \circ f. \quad (\text{S9})$$

The function $f : \mathbb{R}^{n \times m} \rightarrow \mathbb{R}^{d \times m}$ is called the embedding map and sends the matrix representation of the sequence (S8)

$$S = [s_1 | s_2 | \dots | s_m] \in \mathbb{R}^{n \times m} \quad (\text{S10})$$

into a token matrix in $\mathbb{R}^{d \times m}$. The maps $g_k : \mathbb{R}^{d \times m} \rightarrow \mathbb{R}^{d \times m}$ describe the transformer blocks, and the map h is the output map. The domain of h is $\mathbb{R}^{d \times m}$ and its co-domain depends on the task.

Embedding and positional information: The embedding map f is typically of the form

$$f(S) = [E(s_1, 1) | E(s_2, 2) | \dots | E(s_m, m)]$$

for a function $E : \mathbb{R}^n \times \mathbb{N} \rightarrow \mathbb{R}^d$ whose form depends on the specific application and whose parameters are learned from data. For the sake of brevity, we also call each element

$$z_i(1) = E(s_i, i) \quad (\text{S11})$$

a token although the term token is often reserved for the elements s_j . Collecting tokens column-wise leads to the token matrix representation

$$Z(1) \stackrel{\text{def}}{=} \begin{bmatrix} z_1(1) & \dots & z_m(1) \end{bmatrix} \in \mathbb{R}^{d \times m} \quad (\text{S12})$$

that will be used in the next sections.

Self-attention and the transformer block

At each layer $k \in \{1, \dots, \ell - 1\}$, the transformer updates $Z(k) \in \mathbb{R}^{d \times m}$ using multi-head self-attention followed by a token-wise feedforward NN layer, each combined with a residual connection and a normalization function [20].

Denote by $a \in \mathbb{N}$ the total number of heads in a transformer. A head is simply a parallel path in the architecture along which an attention mechanism is implemented. The output of all the heads is then combined into a single output. Define the key $K_j(k) \in \mathbb{R}^{e \times m}$, query $Q_j(k) \in \mathbb{R}^{e \times m}$, and value $V_j(k) \in \mathbb{R}^{e' \times m}$ matrices for the j -th head as

$$Q_j(k) = W_{Q_j}(k)Z(k),$$

$$K_j(k) = W_{K_j}(k)Z(k),$$

$$V_j(k) = W_{V_j}(k)Z(k),$$

where $W_{Q_j}(k), W_{K_j}(k) \in \mathbb{R}^{e \times d}$ and $W_{V_j}(k) \in \mathbb{R}^{e' \times d}$ are weight matrices. The output of the attention mechanism is given by summing over all the attention heads

$$A(Z(k)) = \sum_{j=1}^a W_{V_j}(k) \text{softmax} \left(\frac{K_j^\top(k) Q_j(k)}{\sqrt{e}} \right), \quad (\text{S13})$$

where softmax is applied column-wise and $W_{V_j} \in \mathbb{R}^{d \times e'}$ is another weight matrix. The matrix $K_j^\top(k) Q_j(k)$ contains the inner products between the columns of Z when using the inner product defined by the weight matrix $W_{K_j}^\top(k) W_{Q_j}(k)$. In particular, the entry (i_1, i_2) of $K_j^\top(k) Q_j(k)$ measures how much token i_1 is aligned with token i_2 . The division by $\sqrt{e}$ prevents the inner products in $K_j^\top(k) Q_j(k)$ from growing with the dimension. The standard transformer block update, that is, the map g_k in (S9), can then be written as the composition of two maps:

$$\tilde{Z}(k) = N(Z(k) + A(Z(k))), \quad (\text{S14})$$

$$Z(k+1) = N(\tilde{Z}(k) + R_k(\tilde{Z}(k))), \quad (\text{S15})$$

where $N : \mathbb{R}^{d \times m} \rightarrow \mathbb{R}^{d \times m}$ is a normalization map applied token-wise (column-wise) and $R_k(Z(k)) = (r_k(z_1), \dots, r_k(z_m))$, with r_k defined by

$$r_k(z_i) = W_2(k) \Sigma(W_1(k) z_i),$$

as depicted in Figure S1.

In the original transformer architecture, there are two types of transformer blocks: encoders and decoders. In the encoder blocks, attention is typically computed over all token positions $1, \dots, m$. In the decoder blocks, attention is made *causal*, that is, (S13) is modified so that attention for token i only depends on the tokens that precede it, that is, tokens 1 through $i - 1$ [20].

Output and text generation: The transformer output is task-dependent. For classification, the output map h is typically the composition of two maps, similarly to what is done in feedforward NNs, see sidebar “What are residual neural networks?”. The first is an affine map acting on a designated token (often the final token or a special classification token) to obtain logits. The second map is the entry-wise softmax to produce a distribution over classes.

For text generation, the output map is similarly defined but the class set is the vocabulary of the language being used. The output map sends a token matrix to logits over all the vocabulary words, and the softmax converts these logits into a probability distribution for the next token. The next token is then selected by a decoding rule, passed through the embedding map f and appended to the end of the token matrix $Z(1)$. The process is then restarted by processing the enlarged token matrix starting with the map g_1 . This text generation process ends when a special *end of sentence* or *end of text* token is generated.

In practice, decoding often modifies the softmax distribution before sampling. Two common rules are: (i) *top-k* sampling, which restricts sampling to the k most probable tokens; and (ii) *top-p* sampling, which restricts sampling to the smallest set of tokens whose cumulative probability is at least p , see [22].

Readers seeking additional background on large language models and transformers may consult [23, Chapter 7-8].

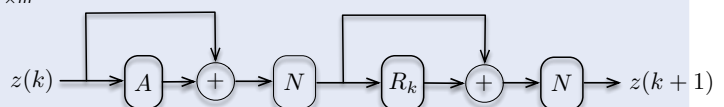


FIGURE S1 Diagrammatic representation of a transformer block architecture described by (S14) and (S15).

WHAT ARE RECURRENT NEURAL NETWORKS AND SELECTIVE STATE-SPACE MODELS?

Recurrent Neural Networks

An early architecture for sequence-to-sequence modeling is the recurrent neural network (RNN) [12]. All the NN architectures so far described were formulated as composition of maps that update the state from one layer to the next. These maps were *memoryless* whereas RNNs employ *memoryfull* maps defined by a nonlinear control system. The memory of such maps is denoted by $c(k, i) \in \mathbb{R}^d$ with k denoting the dependence on the layer and i denoting the dependence on the position (in the input sequence) of the token being processed. As illustrated in Figure S1, a RNN can be interpreted as a two-dimensional architecture in which information moves horizontally, from layer 1 to layer ℓ , and also vertically, from position 1 to position m . The following update equations explain how the memory and state are updated

$$\begin{aligned} c(k, i) &= \Sigma(A(k)c(k, i-1) + B(k)z(k, i)), \\ z(k+1, i) &= \Sigma(C(k)c(k, i)), \\ c(k, 0) &= 0, \end{aligned} \quad (\text{S16})$$

for matrices $A(k) \in \mathbb{R}^{d \times d}$, $B(k) \in \mathbb{R}^{d \times n}$ and $C(k) \in \mathbb{R}^{n \times d}$. In practice, RNNs can be difficult to train due to the well known vanishing/exploding gradient problem. Some variants such as Long Short-Term Memory (LSTM) networks [24] can mitigate this problem, but the training process remains inherently sequential in i thus limiting parallelization.

A good introduction to this class of models can be found in [16, Chapter 10].

Selective State-Space Models

More recently, *selective state-space models* (SSMs) have emerged as an alternative approach to transformers [25], [26]. Among all the different models, which are explained in depth in [27], the most prominent one is the Mamba model [28] whose defining feature is the use of input dependent weight matrices.

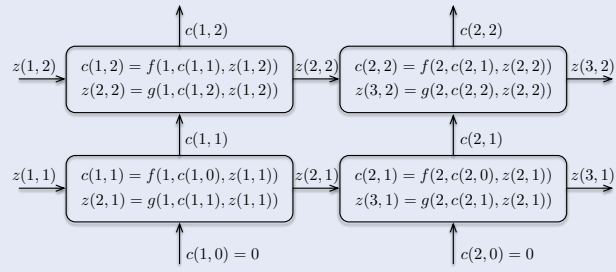


FIGURE S1 Graphical description of the information flow in an SSM. The update equations (S17) were simplified as follows $f(k, c(k, i-1), z(k, i)) = A(k, z(k, i)) c(k, i-1) + B(k, z(k, i)) z(k, i)$ and $g(k, c(k, i), z(k, i)) = C(k, z(k, i)) c(k, i)$. Note that $c(1, 0)$ and $c(2, 0)$ are only used for the purpose of initialization since position $i = 0$ in the input string does not exist.

Its update equations are

$$\begin{aligned} c(k, i) &= A(k, z(k, i)) c(k, i-1) + B(k, z(k, i)) z(k, i), \\ z(k+1, i) &= C(k, z(k, i)) c(k, i), \\ c(k, 0) &= 0, \end{aligned} \quad (\text{S17})$$

where $A : \{1, \dots, \ell\} \times \mathbb{R}^n \rightarrow \mathbb{R}^{d \times d}$, $B : \{1, \dots, \ell\} \times \mathbb{R}^n \rightarrow \mathbb{R}^{d \times n}$, and $C : \{1, \dots, \ell\} \times \mathbb{R}^n \rightarrow \mathbb{R}^{n \times d}$ are nonlinear maps whose parameters are learned from data. The dependence on the layer state (which is the input for layer 1) allows the model to adapt the weights to the current input thereby changing how the state propagates. Consequently, more relevant inputs can have more impact on the state whereas less relevant input information can be discarded [28].

Empirically, selective SSMs such as the Mamba can have a comparable performance to transformers [28]. Moreover, under certain choices for the matrices $A(k, z(k, i))$, $B(k, z(k, i))$, and $C(k, z(k, i))$ the computational complexity of training these models is lower than transformers. Finally, it is possible to interleave SSM blocks with attention blocks to form hybrid architectures such as the Jamba model [29].

It is natural to question if the mismatch between these models decreases as ℓ increases. In particular, to question if the weights obtained by minimizing (0.23) converge to the weights minimizing (0.22) as $\ell \rightarrow \infty$. Note, however, that in this case the control input W for each ResNet does not come from the discretization of a continuous control input W_c , which was the setting discussed in the previous section.

Earlier solutions to this problem were investigated in the literature on numerical methods for optimal control problems, see for example [38], [39], [40]. A more detailed discussion of how optimal control relates to the learning of the weights W , appears in *Optimal control and backpropagation*. For now, it suffices to observe that the minimization

problems (0.22) and (0.23) can be interpreted as optimal control problems. The results in [40] establish that the weights W obtained by solving (0.23) and the evolution of the state of a ResNet when using such weights converge to the control input obtained by solving (0.22) and to the solution of (0.19) when using such control input. These convergence results are predicated on the assumption that the control input obtained by solving (0.22) is Lipschitz continuous (or on other assumptions implying Lipschitz continuity). The recent paper [41] revisits this problem using more recent mathematical tools and forgoes the Lipschitz continuity assumption. Instead, it describes how to regularize the continuous (0.22) and the discrete loss (0.23) to ensure convergence. Intuitively, the regularizer penal-

izes weights that change too fast across layers which can be interpreted as penalizing the discretization of the derivative of the continuous inputs. Another interesting point of departure in the paper [41] is that it does not rely on Pontryagin's optimality conditions and, instead, it employs the notion of Γ -convergence enabling the direct comparison between the minimizers of the regularized losses (0.22) and (0.23) as $\ell \rightarrow \infty$.

Novel architectures based on different discretization schemes

The link between ResNets and the differential equation (0.19) led several researchers to investigate if discretization schemes beyond the forward Euler method could be used to produce new NN architectures with better properties. Some of the new architectures produced by this process include PolyNets [42], based on a polynomial approximation of the backward Euler method, a Hamiltonian inspired architecture [31], based on the semi-implicit Euler method that is known to be symplectic, the momentum ResNet [43], based on the Verlet integration scheme, and several others [32], [44], [45], [46] based on generalizations of the differential model (0.19) that include second order derivatives.

Rather than starting with a differential equation and discretizing it to arrive at a NN architecture, the differential equation can be directly taken as the architecture. This is the key idea behind Neural ODEs [33], in which the NN directly parameterizes the right hand side of a differential equation. The evaluation of a Neural ODE on a given input consists in numerically solving the corresponding differential equation using the input as initial condition. The objective of training is to find the differential equation whose solutions best match the training data. Similarly to evaluation, the training of Neural ODEs requires the use of a numerical solver for differential equations. Neural ODEs are especially useful when working with data that is known to be produced by a differential equation but the use of numerical solvers makes them less scalable than existing alternatives.

OPTIMAL CONTROL AND BACKPROPAGATION

The classical connection

The classical connection between control and backpropagation is known since the '80s [6], [47]. To elucidate it, recall the setting in "What are feedforward neural networks?" for a NN with $\ell \in \mathbb{N}$ layers, all of which have n neurons. The state at layer $k+1 \in \{2, \dots, \ell\}$ is computed from the state at layer k using

$$z(k+1) = \Sigma(W(k)z(k)) \quad (0.24)$$

and a NN can be seen as a function $\Phi_W : \mathbb{R}^n \rightarrow \mathbb{R}^n$ mapping $z(1)$ to $\Phi_W(z(1)) = z(\ell)$, where $z(\ell)$ is recursively computed using (0.24). NNs are trained, that is, their

weights W are computed so that Φ_W approximates an unknown function for which *samples* are available. A collection of s samples $((x_1, y_1), (x_2, y_2), \dots, (x_s, y_s))$ consists of pairs $(x_i, y_i) \in \mathbb{R}^n \times \mathbb{R}^n$ where y_i represents the evaluation of the unknown function at x_i . Supervised training of a neural network entails learning the weights W so that Φ_W maps x_i to y_i .

The learning objective can be formalized as the minimization of the loss functional

$$\sum_{i=1}^s L(y_i, \Phi_W(x_i))$$

subject to the constraints described by the equalities (0.24) and where $L : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}_0^+$ is a differentiable function measuring the discrepancy between y_i and $\Phi_W(x_i)$, such as $\|y_i - \Phi_W(x_i)\|^2$. Incorporating the constraints (0.24) into the loss leads to the expanded loss functional

$$\begin{aligned} \mathcal{J}(\lambda, z, W) = & \sum_{i=1}^s L(y_i, \Phi(x_i)) \\ & + \sum_{i=1}^s \sum_{k=2}^{\ell} \lambda_i^T(k) [z_i(k) - \Sigma(W(k-1)z_i(k-1))] \end{aligned}$$

with $\lambda_i(k) \in \mathbb{R}^n$ being Lagrange multipliers (adjoint variables). This is now an unconstrained minimization problem for which necessary conditions for optimality can be written.

The first necessary condition for optimality arises by differentiating the cost with respect to the Lagrange multipliers

$$\frac{\partial \mathcal{J}}{\partial \lambda_i(k)} = 0 \implies z_i(k) = \Sigma(W(k-1)z_i(k-1)),$$

and recovers (0.24) as expected. The second arises by differentiating the cost with respect to the state

$$\frac{\partial L}{\partial z_i(k)} = 0. \quad (0.25)$$

In particular, at the final layer $k = \ell$, it provides

$$\frac{\partial \mathcal{J}}{\partial z_i(\ell)} = \frac{\partial L}{\partial z_i(\ell)} + \lambda_i(\ell) = 0,$$

defining the value of λ_i at the last layer

$$\lambda_i(\ell) = -\frac{\partial L}{\partial z_i(\ell)}. \quad (0.26)$$

To obtain λ_i at the intermediate layers, condition (0.25) is used for $k = 2, \dots, \ell - 1$

$$\frac{\partial \mathcal{J}}{\partial z_i(k)} = \lambda_i(k) - \frac{\partial \Sigma}{\partial v} W(k) \lambda_i(k+1) = 0,$$

giving rise to the recursion

$$\lambda_i(k) = \frac{\partial \Sigma}{\partial v} W(k) \lambda_i(k+1). \quad (0.27)$$

Note that it is possible that $\lambda_i(k+1)$ is in the kernel of $W(k)$, leading to vanishing of the adjoint along the NN. In what follows, as it is commonly done in the literature, the weight matrix is assumed not to be rank deficient.

To compute the last necessary condition, note that the derivative of $\Sigma(v)$ with respect to v is the diagonal matrix whose entry in row and column j is $\frac{d\sigma}{dv_j}$, $j \in \{1, \dots, n\}$. Next, define

$$v_i(k+1) \stackrel{\text{def}}{=} W(k)z_i(k) \quad (0.28)$$

for $i \in \{1, \dots, s\}$ and note that the j -th component of $v_i(k+1)$ is

$$v_{i,j}(k+1) = \sum_{m=1}^n W(k)_{jm} z_{i,m}(k).$$

Consequently

$$\frac{\partial v_{i,j}(k+1)}{\partial W(k)_{ab}} = \delta_{ja} z_{i,b}(k), \quad (0.29)$$

$a, b \in \{1, \dots, n\}$, and where δ_{ja} is Kronecker's delta. Using this equality and the definition of $\mathcal{J}$, the derivative of $\mathcal{J}$ with respect to the entry in row a and column b of W can be computed

$$\begin{aligned} \frac{\partial \mathcal{J}}{\partial W(k)_{ab}} &= \frac{\partial}{\partial W(k)_{ab}} \left(\sum_{i=1}^s -\lambda_i(k+1)^T \Sigma(v_i(k+1)) \right) \\ &= - \sum_{i=1}^s \sum_{j=1}^n \lambda_{i,j}(k+1) \frac{d\sigma}{dv_{i,j}(k+1)} \frac{\partial v_{i,j}(k+1)}{\partial W(k)_{ab}} \\ &= - \sum_{i=1}^s \lambda_{i,a}(k+1) \frac{d\sigma}{dv_{i,a}(k+1)} z_{i,b}(k). \end{aligned} \quad (0.30)$$

This expression can be written in the simpler form

$$\frac{\partial \mathcal{J}}{\partial W(k)_{ab}} = - \sum_{i=1}^s r_{i,a}(k+1) z_{i,b}(k),$$

where

$$r_i(k) = \text{diag} \left(\frac{d\sigma}{dv_{i,1}(k)}, \dots, \frac{d\sigma}{dv_{i,n}(k)} \right) \lambda_i(k).$$

From the derivative with respect to each entry of $W(k)$, the derivative with respect to the weight matrix $W(k)$ can be obtained

$$\frac{\partial \mathcal{J}}{\partial W(k)} = - \sum_{i=1}^s r_i(k+1) z_i(k)^T = 0, \quad (0.31)$$

where $k \in \{2, \dots, \ell-1\}$. In principle, solving the stationarity conditions (0.31) jointly with the state (0.24) and adjoint (0.27) equations would yield stationary points. However, because $z_i(k)$ and $\lambda_i(k)$, and hence $r_i(k)$, depend nonlinearly and recursively on $\{W(k')\}_{k' \leq k}$ through the forward and backward passes, the resulting coupled nonlinear system is analytically intractable. Instead, it is customary to find the weights W that enforce (0.31) via gradient descent. This is done by viewing the loss functional as defining, for each fixed collection of weights $W = (W(1), \dots, W(\ell-1))$, a scalar functional

$$J(W) \stackrel{\text{def}}{=} \mathcal{J}(z(W), \lambda(W), W),$$

where $z(W)$ and $\lambda(W)$ satisfy the forward (0.24) and backward (0.27) recursions. The gradient descent update

for minimizing J then takes the form

$$\begin{aligned} W^{(q+1)}(k) &= W^{(q)}(k) - \eta_q \frac{\partial \mathcal{J}}{\partial W(k)} \Big|_{W=W^{(q)}} \\ &= W^{(q)}(k) + \eta_q \sum_{i=1}^s r_i^{(q)}(k+1) z_i^{(q)}(k)^T, \end{aligned} \quad (0.32)$$

where $k \in \{1, \dots, \ell-1\}$, the superscript q denotes the q -th iterate of the gradient update, $r_i^{(q)}(k)$ and $z_i^{(q)}(k)$ are obtained from the backward and forward passes using the current weights $W^{(q)}$, and η_q is the step size also known as the learning rate.

Combining the necessary conditions for optimality with the gradient descent iterates results in the backpropagation algorithm that can be summarized as the iteration of the following steps:

- 1) **Forward pass:** propagate the input $z_i(1)$ to the output $z_i(\ell)$ using (0.24).
- 2) **Backward pass:** initialize the adjoint $\lambda_i(\ell)$ at layer ℓ with (0.26) and backpropagate it to $\lambda_i(1)$ using (0.27).
- 3) **Gradient descent:** update the weights W using (0.32).

The modern connection

The connection between optimal control and backpropagation presented in *The classical connection* consists in formulating the training of NNs as an optimal control problem in discrete-time, deriving necessary conditions for optimality, and observing that those conditions essentially define backpropagation. As discussed in *Differential equations, control systems, and neural networks*, for certain NN architectures, such as the ResNet, the state update equation can be regarded as the discretization of a differential equation. Similarly, the optimal control problem addressed in *The classical connection* can be seen as the discretization of a continuous-time optimal control problem. This observation exposes an alternative path towards the training of NNs: formulate necessary conditions for optimality using an optimal control formulation in continuous-time and then discretize them to obtain a training algorithm. It is shown in this section that both approaches lead to the same solution, provided that suitable discretization techniques are employed.

Redoing the analysis in *The classical connection* for a ResNet requires changing the expanded loss functional $\mathcal{J}(z, W, \lambda)$ to

$$\begin{aligned} \sum_{i=1}^s L(y_i, \Phi(x_i)) + \sum_{i=1}^s \sum_{k=1}^{\ell} \lambda_i^T(k) [z_i(k) - z_i(k-1) \\ - w(k-1) \Sigma(W(k-1)z_i(k-1))] \end{aligned}$$

in order to account for how the state is updated across layers in a ResNet, see (0.18). The corresponding necessary conditions for optimality become

$$\begin{aligned}
z_i(k) &= z_i(k-1) + w(k-1)\Sigma(W(k-1)z_i(k-1)) \\
\lambda_i(k) &= \lambda_i(k+1) + w(k)\frac{\partial \Sigma}{\partial v}W(k)\lambda_i(k+1) \quad (0.33) \\
0 &= -\sum_{i=1}^s r_i(k+1)z_i(k)^\top. \quad (0.34)
\end{aligned}$$

The very same conditions can be obtained starting from the continuous-time optimal control problem defined by the minimization of

$$\sum_{i=1}^s L(y_i, z_{c,i}(1)), \quad (0.35)$$

where $z_{c,i}(1)$ denotes the solution of (0.19) at time 1 when starting from initial condition x_i . Since multiple solutions are used in the loss (0.35) an ensemble control system consisting of s identical copies of (0.19) running in parallel and controlled by the same input W_c is considered. The necessary conditions for optimality are given by Pontryagin's maximum principle [48], [45]

$$\dot{z}_{c,i}(t) = \frac{\partial H}{\partial \lambda_{c,i}} = \Sigma(W_c(t)z_{c,i}(t)) \quad (0.36)$$

$$\dot{\lambda}_{c,i}(t) = -\frac{\partial H}{\partial z_{c,i}} = -\frac{\partial \Sigma}{\partial v}W_c(t)\lambda_{c,i}(t) \quad (0.37)$$

$$0 = \frac{\partial H}{\partial W_c},$$

where the Hamiltonian H is

$$H(\lambda_{c,i}(t), z_{c,i}(t), W_c(t)) = \sum_{i=1}^s \lambda_{c,i}^\top(t) \Sigma(W_c(t)z_{c,i}(t)).$$

A naive discretization of (0.37), using the forward Euler method, *does not* lead to (0.33). In order to recover (0.33), the symplectic version of the forward Euler method, also known as the semi-implicit Euler method [49, Eq. (1.9)], must be employed. It consists of the forward Euler method applied to (0.36) but requires discretizing (0.37) using the implicit rule

$$\lambda_i(k+1) = \lambda_i(k) - w(k)\frac{\partial \Sigma}{\partial v}W(k)\lambda_i(k+1),$$

which is simply (0.33).

Although the interplay between backpropagation and optimal control in discrete-time already allows for the migration of ideas from control to deep learning, the linkage with continuous-time optimal control opens new perspectives. Continuous-time control theory is substantially more developed than its discrete-time counterpart. Moreover, continuous-time results are often more intuitive thus making it easier to adapt/generalize to the setting of deep learning. Some examples of this migration of ideas are discussed in the next section.

New training techniques

The previous sections shed light into the interplay between optimal control and the training of NNs. However, the use of gradient descent is but one possible path to find the

weights enforcing equality (0.34). When the weights are real valued, [45] discusses why gradient descent may be preferable to directly solving (0.34). The work in [50] uses the extra generality afforded by the maximum principle to derive training algorithms for NNs whose weights belong to finite sets, in which case gradient descent is not an option. Extending the optimal control perspective from one to two players brings us to the setting of games which proved useful to develop more efficient adversarial training algorithms in [51], see also [52] for a rigorous convergence analysis. The modern continuous-time perspective was exploited in [48] to obtain training algorithms where the discretization time step is regarded as one additional control input used for more efficient training.

Rather than linkages to optimal control, other advances in training techniques leveraged the view of NNs as discretizations of differential equations. The important property of reversibility, addressed in [46], is intimately tied to training, since backpropagation requires both forward and backward passes. In particular, it shows how reversibility can be exploited for more efficient training and proposes new reversible architectures. Another important property is stability [31], [46], understood as Lipschitz continuity of the map sending the state of the first layer to the state of the last layer. Stability guarantees robustness of the network against noisy inputs or perturbations of the data. As discussed in [53], the Lipschitz constant depends on the activation function σ and the weights for the continuous-time model (0.19). For the discrete-time model (0.18), it also depends on the number of layers. The work in [53] discusses several architectures inspired by ideas from dynamical systems, such as Hamiltonian dynamics, that ensure stability by construction rather than relying on training.

UNIVERSAL APPROXIMATION

One of the reasons underlying the wide applicability of NNs is their perceived ability to approximate arbitrary functions. This is known as the power of universal approximation. To make this notion precise, recall that a NN is a family of functions $\Phi_W : \mathbb{R}^{n_1} \rightarrow \mathbb{R}^o$ parameterized by the weights W , see "What are feedforward neural networks?" sidebar. It can then be asked if, given a function $\psi : K \rightarrow \mathbb{R}^o$ defined on a domain of interest $K \subseteq \mathbb{R}^{n_1}$, there is a choice of weights W so that Φ_W approximates ψ arbitrarily well with respect to a chosen topology (uniform, L^p , Sobolev, etc.).

A concise historical account of universal approximation in neural networks

Classical universal approximation results established in late '80s and early '90s focused on proving that feedforward NNs with small depth, that is, a small number of layers, but unrestricted width, that is, an arbitrary

number of neurons per layer, can approximate continuous functions on compact sets, under mild assumptions on the activation functions. In particular, universality of sigmoidal activation functions was proved in [54], [55]. In a series of follow-up work, including [56], [57], it was shown that for broad classes of NNs, *non-polynomial* activations are essentially necessary and sufficient for approximating continuous functions on compact sets. The mathematical machinery behind these results is functional analysis' density arguments. A different argument was employed in [58] to establish that the approximation error decays at the rate $1/n$, where n is the number of sigmoids in the hidden layer, independently of the dimension of the approximated function's domain.

The modern deep learning era revived these questions due to the evolution in NN architectures. Deep NNs have large depth, that is, a large number of layers, but bounded width, that is, a bounded number of neurons per layer. This mismatch between architectures prevents a direct application of the classical universal approximation results. Establishing universal approximation for deep neural networks relies on understanding composition of maps, which, due to nonlinearities of the activation functions, is challenging.

The literature on this subject is quite recent. One of the early results, [59], focused on the class of ReLU activation functions to establish that one neuron per layer suffices to approximate an arbitrary Lebesgue integrable function from $\mathbb{R}$ to $\mathbb{R}$ with respect to the L^1 norm. The results that followed addressed more general functions from $\mathbb{R}^{n_1}$ to $\mathbb{R}^o$. For instance, when using the ReLU activation function and $o = 1$, it was shown in [59] that universal approximation with respect to the L^1 topology can be achieved with a minimum width between $n_1 + 1$ and $n_1 + 4$. The theory of universal approximation for deep NNs substantially advanced in early 2020, with notable discoveries in [60], and particularly in [61], where universal approximation was established with respect to the *infinity norm* for non-residual deep NNs, allowing for general classes of activation functions. More precisely, it is shown in [61] that under very mild conditions on the activation functions, any continuous function $\psi : K \rightarrow \mathbb{R}^o$, where $K \subset \mathbb{R}^{n_1}$ is compact, can be approximated in the infinity norm using a deep NN of width $n_1 + o + 2$.

The control theory vantage point

Inline with the objectives of this article, this section provides the control-theoretic viewpoint on universal approximation developed in [62], [63], [64], and also in [65]. The starting point is the continuous limit of a ResNet, as explained in *Differential equations, control systems, and neural networks*, and written as

$$\dot{z}_c(t) = w_c(t)\Sigma(W_c(t)z_c(t)). \quad (0.38)$$

Note that (0.38) differs from (0.19) by the presence of $w_c(t)$. This is achieved by writing the weight $w(k)$ in (0.18) as the product $w(k) = \tau w_c(k)$ and letting $\tau \rightarrow 0$ as $\ell \rightarrow \infty$ so that $w_c(t)$ is the continuous limit of $w_c(k)$. For notational convenience, denote by $\gamma : \mathbb{R}^n \rightarrow \mathbb{R}^n$ the *flow* of (0.38), that is, the map sending an initial condition $z_c(0)$ to the point reached by the solution of (0.38) at time $t = 1$ under the input (w_c, W_c) .

The objective of approximating $\psi : K \rightarrow \mathbb{R}^o$ with γ requires the existence of control inputs (w_c, W_c) for which the corresponding flow γ satisfies $\gamma(x_i) = \psi(x_i)$ for every sample point $x_i \in K \subseteq \mathbb{R}^{n_1}$ or, at least, that $\gamma(x_i)$ is close to $\psi(x_i)$. This can be understood as a controllability problem for which the initial conditions $x_1, \dots, x_s$ have to be steered to $y_1 = \psi(x_1), \dots, y_s = \psi(x_s)$ at time $t = 1$ by the same open-loop control input (w_c, W_c) .

At this point, it is worth pointing out the sharp difference between this setting and earlier work on controllability of differential equation models for neural networks, for example, [66], where the weights were assumed to be constant and an exogenous control signal was fed into the neurons.

The simultaneous control of multiple initial conditions is naturally related to the classical settings of ensemble control theory, for example, [67], [68]. However, in ensemble control, the motion of each member of the ensemble is governed by a different, even if very similar, control system. In the setting of universal approximation, the motion from each initial condition is governed by the same control system under the same input. This setting is most closely related to [69], where controllability results for ensembles of infinitely many copies of the same control system are considered. With this perspective in mind, there are two key problems that can be considered.

Memorization Problem: Let $\psi : \mathbb{R}^{n_1} \rightarrow \mathbb{R}^o$ be a continuous function, $K_{\text{samples}} \subset \mathbb{R}^{n_1}$ be a finite set, and $\varepsilon \in \mathbb{R}^+$ be the approximation accuracy. Given any desired ε and $p \in [1, +\infty]$, does there exist an initialization map $f : \mathbb{R}^{n_1} \rightarrow \mathbb{R}^n$, a finalization map $h : \mathbb{R}^n \rightarrow \mathbb{R}^o$, and an input $(w_c, W_c) : [0, 1] \rightarrow \mathbb{R} \times \mathbb{R}^{n \times n}$ so that the flow $\gamma : \mathbb{R}^n \rightarrow \mathbb{R}^n$ defined by the solution of (0.18) under the said input satisfies

$$\|\psi - h \circ \gamma \circ f\|_{L^p(K_{\text{samples}})} \leq \varepsilon?$$

The notation $\|\cdot\|_{L^p(K_{\text{samples}})}$ denotes the L^p norm on the set K_{samples} , in other words, the approximation guarantee only needs to hold for elements in the finite set K_{samples} . The purpose of the initialization function f is to map $\mathbb{R}^{n_1}$ to $\mathbb{R}^n$ since the continuous model (0.38) was developed assuming all the layers have the same number n of neurons. Similarly, the finalization map converts $\mathbb{R}^n$ to $\mathbb{R}^o$.

We note that the memorization problem above is a simultaneous controllability problem which is difficult to solve since the *same* differential equation (0.38) governs the evolution of the state from each initial condition $z_c(0)$ and

under the *same* open-loop control input.

The next problem considers the more challenging case of universal approximation on compact sets.

Universal Approximation Problem: Let $\psi : \mathbb{R}^{n_1} \rightarrow \mathbb{R}^o$ be a continuous function, $K \subset \mathbb{R}^{n_1}$ be a compact set, and $\varepsilon \in \mathbb{R}^+$ be the approximation accuracy. Given any desired ε and $p \in [1, +\infty]$, does there exist a continuous initialization map $f : \mathbb{R}^{n_1} \rightarrow \mathbb{R}^n$, a continuous finalization map $g : \mathbb{R}^n \rightarrow \mathbb{R}^o$, and an input $(w_c, W_c) : [0, 1] \rightarrow \mathbb{R} \times \mathbb{R}^{n \times n}$ such that the flow $\gamma : \mathbb{R}^n \rightarrow \mathbb{R}^n$ defined by the solution of (0.38) under the said input satisfies

$$\|\psi - h \circ \gamma \circ f\|_{L^p(K)} \leq \varepsilon?$$

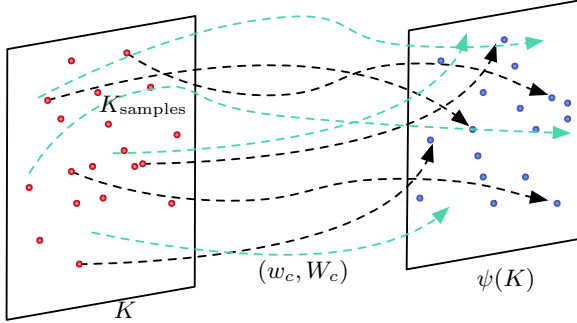


FIGURE 5 A schematic picture of the two problems under study. The objective in the memorization problem is to find a control input (w_c, W_c) that takes all the sample points in K_{samples} (represented by red dots) to their corresponding points in $\psi(K_{\text{samples}})$ (represented by blue dots); some of these assignments are shown by black dashed arrows. In the universal approximation problem, the points outside K_{samples} also need to be carefully mapped to guarantee approximation in the uniform sense; some of these assignments are represented by dashed green arrows.

Contrary to the memorization problem, the universal approximation problem requires the approximation guarantees to hold for every element in K , which is no longer a finite set. Figure 5 provides a schematic picture of memorization and universal approximation. Naturally, geometric control theory plays a key role in the study of the memorization problem when framed as a controllability problem. In particular, a keen reader would realize that the classical Lie algebraic conditions for accessibility require the derivatives of the activation function to appear in the analysis of this problem. As shown in [63], a sufficient condition for controllability is provided by asking the activation function σ , or its derivatives, to satisfy an ordinary differential equation with *quadratic righthand side*; interestingly, a large class of widely used activations functions satisfy this condition.

The key result in [63] provides answers to both memorization and universal approximation problems. In particular, given a continuous function $\psi : \mathbb{R}^{n_1} \rightarrow \mathbb{R}^o$, and assuming that the activation function satisfies a differential equation with quadratic right-hand side, the following was established:

- » **Memorization:** The memorization problem has a solution for any $\varepsilon \in \mathbb{R}^+$.
- » **Universal Approximation:** The problem has a solution for any $\varepsilon \in \mathbb{R}^+$ when n is given by $2 \max\{n_1, o\} + 1$.

The results in [63] are, in fact, sharper in the sense that, under suitable technical assumptions, the memorization problem can be solved with $\varepsilon = 0$ for almost all data sets and universal approximation can be achieved using a smaller number of neurons for a more restricted class of functions. Notably, in the case where $n_1 = o$, the universal approximation statement above improves upon the one in [61], which requires $n_1 + o + 2$ neurons.

The problem of universal approximation led to a fruitful bridge between control theory and deep learning, see for instance [70], [71], [72], [73], [74]. In particular, the composition of flows, a concept with deep roots in control theory, has become instrumental for the study of Neural ODEs. This perspective has spurred substantial activity in the past few years at this intersection, including [75], [76], [77], [78], [79], [80], [81].

We close this section by discussing a few intriguing directions related to the approximation power of Neural ODEs for density estimation by analyzing controllability properties of the corresponding continuity equation, where the driving vector field is parameterized by a Neural ODE. For instance, results in [82], [83] establish approximate controllability of the controlled continuity equation on the space of probability measures, with connections to optimal transportation theory: given initial and target distributions, it is shown that the Neural ODE's weights can be chosen so that the resulting terminal distribution is arbitrarily close to the target in an appropriate measure. The literature on this subject is fast growing [84], [85], [86], and the techniques have also recently been employed for continuity equations corresponding to the self-attention dynamics in transformers [87], [88].

LARGE LANGUAGE MODELS, TRANSFORMERS, AND CONSENSUS

Large language models (LLMs) is now used as a generic term for NN architectures that generate textual responses to user's queries in natural language. It was formed by pre-pending "large", referring to model and training data size, to "language models", referring to the models used in natural language processing. Transformer-based LLMs are a good example of how ideas from control can be used to better understand deep learning models. First, by passing to the continuous limit, as explained in *Differential equations, control systems, and neural networks*, the transformer architecture can be described by a differential equation. Second, the differential equation model can be interpreted as a generalization of the consensus models studied in control theory. This section explains these connections and

how ideas from consensus on manifolds reveal the role of the attention mechanism in transformer-based LLMs.

An LLM first converts the query or prompt into a sequence of tokens $z_1(1), \dots, z_m(1)$ using the map f appearing in (S9). Then, a sequence-to-sequence NN is employed to process the sequence of tokens and produce the output sequence $z_1(\ell), \dots, z_m(\ell)$. Although there are several choices for this NN, the most widely used is the transformer described in the sidebar “What are transformers?”. Other possibilities are the RWKV model [89], which is based on a recurrent neural network (RNN), and the Mamba model, which is based on a selective state-space-model (SSM). See sidebar “What are recurrent neural networks and selective state-space models?” for an introduction to RNNs and SSMs. The output of the sequence-to-sequence model, $z_1(\ell), \dots, z_m(\ell)$, is processed by the output map h , see (S9), to predict the next token $z_{m+1}(1)$ which is appended to the end of the input token sequence to generate the longer sequence $z_1(1), \dots, z_m(1), z_{m+1}(1)$. This longer sequence is then processed by the sequence-to-sequence NN to predict the next token, analogously to a feedback loop. The process is iterated until the reply sentence or sentences are completed.

Self-attention dynamics

Denote by

$$Z(k) = [z_1(k)|z_2(k)|\dots|z_m(k)] \in \mathbb{R}^{d \times m}$$

the matrix representation of the tokens at layer k of a transformer NN. The transformer block updates each token according to the recurrence formula

$$\tilde{z}_i(k) = \mathbf{n}(z_i(k) + A_i(Z(k))), \quad (0.39)$$

$$z_i(k+1) = \mathbf{n}(\tilde{z}_i(k) + r_k(\tilde{z}_i(k))), \quad (0.40)$$

where $\mathbf{n} : \mathbb{R}^d \rightarrow \mathbb{R}^d$ is the normalization map, $r_k(\tilde{z}_i(k)) = W_2(k)\Sigma(W_1(k)\tilde{z}_i(k))$, and

$$A = [A_1|A_2|\dots|A_m] : \mathbb{R}^{d \times m} \rightarrow \mathbb{R}^{d \times m}$$

is the self-attention mechanism (see sidebar “What are transformers?” for a detailed description of these objects).

Focusing on the self-attention mechanism by removing the skip-connection (0.40), the transformer block (0.39), (0.40) simplifies to

$$z_i(k+1) = \mathbf{n}(z_i(k) + A_i(Z(k))). \quad (0.41)$$

It was first noted in [90] and further developed in [91] that the attention block (0.41) resembles the dynamics of a multi-particle system and, following the ideas presented in *Differential equations, control systems, and neural networks*, it can be modeled by a system of differential equations.

In the differential equation model, the matrix representation of tokens is parameterized by $t \in \mathbb{R}_0^+$:

$$Z_c(t) = [z_{c,1}(t)|z_{c,2}(t)|\dots|z_{c,m}(t)] \in \mathbb{R}^{d \times m}.$$

Note the deviation from the description in *Differential equations, control systems, and neural networks*, where the time domain was normalized to $[0, 1]$. This section discusses the asymptotic behavior of transformers thus suggesting $\mathbb{R}_0^+$ as the natural time domain. As it will be seen, existence of solutions is not a concern since the state space will be a product of spheres, hence compact. Similarly to the token matrix, the weight matrices are functions of time $W_{Q_j}(t), W_{K_j}(t) \in \mathbb{R}^{e \times d}$ and $W_{V_j}(t) \in \mathbb{R}^{e' \times d}$. Hence, the attention mechanism can be written as

$$A_c(t, Z_c) = [A_{c,1}(t, Z_c)|A_{c,2}(t, Z_c)|\dots|A_{c,m}(t, Z_c)] \\ = \sum_{j=1}^a W_{U_j}(t) Z_c \text{softmax}\left(\frac{Z_c^\top W_{P_j}(t) Z_c}{\sqrt{e}}\right), \quad (0.42)$$

where a is the number of heads, the softmax is applied column-wise, and the matrices

$$W_{U_j}(t) = W_{O_j} W_{V_j}(t) \in \mathbb{R}^{d \times d},$$

$$W_{P_j}(t) = W_{K_j}^\top(t) W_{Q_j}(t) \in \mathbb{R}^{m \times m}$$

were used for brevity. When there is only one head ($a = 1$), the subscript $j = 1$ is omitted for simplicity, that is, $W_{U_1}(t)$ simplifies to $W_U(t)$ and $W_{P_1}(t)$ simplifies to $W_P(t)$.

The self-attention mechanism (0.41) can now be written

$$\dot{z}_{c,i} = T_{z_{c,i}} \mathbf{n} \cdot A_{c,i}(t, Z_c(t)), \quad (0.43)$$

where $T_z \mathbf{n} : \mathbb{R}^d \rightarrow \mathbb{R}^d$ denotes the tangent map of $\mathbf{n}$ at $z \in \mathbb{R}^d$ using the identification $T_z \mathbb{R}^d \simeq \mathbb{R}^d$.

For the purpose of analyzing the asymptotic behavior of $z_{c,i}$, the normalization map is typically chosen to be the projection onto the $(d-1)$ -dimensional sphere $\mathbb{S}^{d-1} = \{z \in \mathbb{R}^d | z^\top z = 1\}$. In other words,

$$\mathbf{n}(z) = \frac{z}{\|z\|}, \quad (0.44)$$

where $\|z\|$ denotes the Euclidean norm of $z \in \mathbb{R}^d \setminus \{0\}$ thus forcing tokens to have unit norm. Therefore, the configuration space of both the discrete (0.41) and the continuous (0.43) dynamical systems governing the evolution of attention is the product of m copies of the $(d-1)$ -sphere

$$(\mathbb{S}^{d-1})^m = \underbrace{\mathbb{S}^{d-1} \times \dots \times \mathbb{S}^{d-1}}_{m\text{-times}}.$$

Recall that the tangent space of the sphere at each point $z \in \mathbb{S}^{d-1}$ consists of all vectors that are orthogonal to z , that is

$$T_z \mathbb{S}^{d-1} = \{X \in \mathbb{R}^d | z^\top X = 0\}.$$

The tangent map of the projection thus reads

$$T_z \mathbf{n} \cdot X = (\mathbb{I}_d - zz^\top) \cdot X$$

for each $X \in \mathbb{R}^d$.

In real transformers, however, the standard layer normalization,

$$\mathbf{n}(z) = \frac{1}{\sigma(z)}(z - \mu(z)\mathbf{1}) \star \gamma + \beta,$$

is the most common choice. In the previous expression, $\mathbf{1}$ denotes the vector $(1, \dots, 1) \in \mathbb{R}^d$, $\star$ denotes the element-wise product of vectors, and $\gamma, \beta \in \mathbb{R}^d$ are the *learned scale* and *shift*, respectively. Similarly, $\mu(z)$ and $\sigma(z)$ denote the mean and standard deviation of z , respectively. An easy check shows that, if $\gamma = \gamma_0 \mathbf{1}$ for some fixed $\gamma_0 \neq 0$, then the tokens lie on the $(d-1)$ -sphere of center β and radius $d\gamma_0^2$. Therefore, the theoretical results with the normalization to the sphere (0.44) also apply to the standard layer normalization. For arbitrary scale parameters, it is conjectured that the tokens also lie on a certain hypersurface of $\mathbb{R}^d$ diffeomorphic to the $(d-1)$ -sphere, making the theoretical results still valid.

Consensus dynamics

Many empirical works starting with [92] observed that tokens tend to converge to consensus as the depth of the transformer increases, that is, the model collapses as all tokens converge to a single point. The first theoretical explanation for this observed behavior was provided in [93], [94]. The continuous attention dynamics (0.43) was regarded as a mean-field interacting particle system and it was shown that, for almost every initial condition, all tokens converge to a fixed position as time goes to infinity. The analysis was restricted to a simplified model with only one head, $a = 1$, and weight matrices assumed to be constant and proportional to the identity, $W_U(t) = \mathbb{I}_d$ and $W_P(t) = \beta \mathbb{I}_m$, where $\beta > 0$ is known as the *coldness* or *inverse temperature*.

Under different assumptions on d, m and β , asymptotic convergence to consensus was proven using techniques from partial differential equations and ergodic theory. In subsequent papers [95], [96], [97], it was also shown that metastability holds under certain conditions on the parameters and the initial conditions, that is, tokens first form several clusters and remain there for some time before eventually collapsing to a single one. The aforementioned papers address the full self-attention mechanism, but analogous results have been proven for the *causal* (auto-regressive) attention [97]. The causal attention mechanism is a modification of (0.42) where the attention for the i -th token only depends on the previous ones.

Ideas from consensus dynamics and control theory have also demonstrated usefulness in the asymptotic analysis of the transformer dynamics [98], [99], [100], enabling the analysis of (0.43) for more general weight matrices. To better understand the relation between (0.43) and the consensus models used in control, note that (0.43) can also be written in the following form when there is a single head ($a = 1$):

$$\dot{z}_{c,i} = T_{z_{c,i}} n \cdot W_U(t) \sum_{j=1}^m a_{ij}(t, Z_c) z_{c,j},$$

where the coefficients a_{ij} appearing in the above linear

combination are strictly positive and sum to 1, that is, $\sum_{j=1}^m a_{ij} = 1$. Differently from most consensus models investigated in control, the coefficients a_{ij} are nonlinear functions of the state Z_c . The presence of the matrix $W_U(t)$ is another difference with respect to the consensus models in the control literature.

For causal self-attention, an input-to-state stability argument shows that the behavior of the system is determined by the evolution of the first token. Causality results in a triangular structure on the dynamics suggestive of control systems in strict feedback form [101]. Moreover, the dynamics of token i is input-to-state stable with respect to distance between tokens $1, \dots, i-1$ and the consensus set, that is, the set defined by $z_{c,1} = z_{c,2} = \dots = z_{c,i-1}$. Hence, asymptotic stability towards the consensus set is established using input-to-state stability of cascade interconnections.

The results proven using these techniques point in the same direction as previous contributions to this problem: the model collapses as all tokens converge to consensus as time increases. The main advantage of using control theoretic techniques compared to the mean-field approach is that they allow for addressing non-identity and time-varying weight matrices. This situation is significantly more realistic as weights depend on the layer in real transformers. Namely, the results in [98] for the causal self-attention case may be summarized in three cases:

- 1) $a \geq 1$ and $W_{U_j}(t) = \mathbb{I}_d$ for all heads. Then the first token is fixed and, thus, all tokens converge to the initial position of the first token. The domain of attraction of this consensus is co-null, that is, its complement has zero measure.
- 2) $a = 1$ and $W_U(t) = W_U$ constant and symmetric. Then the first token converges to the principal eigenvector of W_U , that is, the eigenvector (up to sign) corresponding to maximum eigenvalue. The sign is determined by the initial position of the first token. More recently, [100] extended this result to the full self-attention. They also identified new types of equilibria showing that all of them are unstable.
- 3) $a = 1$ and $W_U(t)$ bounded and symmetric for all times. Then the principal eigenvalue of $W_U(t)$ also depends on time. As a result, tokens do not converge to a fixed position, but they keep following the principal eigenvector instead. Specifically, they converge to a ball around the principal eigenvector forming a moving cluster. The size of the ball depends on how fast the eigenvector moves.

Even though a fully general asymptotic analysis of the model (0.43) is still missing, all the results to date suggest model collapse.

Empirical validation of theoretical analysis

The collapse of the attention mechanism, explained by the theoretical analysis in [98], is supported by empirical results on LLMs that include the skip-connection (0.40) and thus go beyond the simplifying assumption made when deriving the theoretical differential equation model.

Several tests on the GPT-2 XL and the GPT-Neo 2.7B models are reported in [99]. Since the theoretical results are asymptotic, the depth was increased by looping the transformers, that is, by feeding the tokens at the final layer as inputs to the first layer.

The error function

$$E(k) = 1 - \frac{1}{m} \sum_{i=1}^m \frac{z_1^\top(k) z_i(k)}{|z_1(k)| |z_i(k)|} \quad (0.45)$$

was used to measure how far the tokens are from the first one. Note that $E(k) \geq 0$ and it vanishes only when all tokens are aligned, $z_1(k) = z_2(k) \dots = z_m(k)$, that is, when consensus is reached. The average of (0.45) was computed across 100 random prompts generated by uniformly sampling 200 tokens from the GPT-2 tokenizer's vocabulary.

Two types of experiments comparing the GPT-2 XL and the GPT-Neo 2.7B models were performed (all figures are extracted from [99]):

- 1) *Evolution of tokens using pre-trained LLMs.* Using the pre-trained weights provided by the Hugging Face library [102]. Looping the transformers yields weight matrices that are time-varying but periodic. Figure 6 shows the evolution of E when the skip-connection (0.40) is absent and thus there is agreement between the theoretical model and the LLMs. Figure 7 shows the same experiment with the full model, that is, (0.39) and (0.40).
- 2) *Evolution of tokens using random weight matrices.* As for the previous case, the experiments were carried out for the self-attention only, as well as for the full model. In Figure 8, the random weights were fixed, that is, they were the same for each pass of the model, providing time-varying, periodic weights. In Figure 9, new random weights were generated before each pass of the model, providing time-varying, non periodic weight matrices.

We can observe that in all the above described cases the function E converges to zero thus showing that all the tokens converge to the same value. The experiments with randomly generated matrices suggest that consensus is a structural property of the architecture and does not depend on the choice of weights or prompts. It can be observed that the rate of convergence depends on the skip-connections: for the GPT-2 model it increases, whereas for the GPT-Neo model it decreases. Nevertheless, consensus still takes place when the skip-connection layers are considered.

As an illustration of consensus, we took the first

sentence of this article, Deep neural networks, in their different shapes and flavors, are having a remarkable impact in almost every area of science and engineering., and passed it through GPT-2-XL. The decoded output is presented in

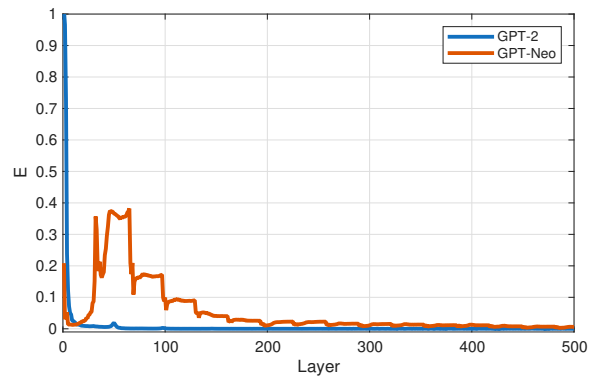


FIGURE 6 Comparison between the GPT-2 XL and GPT-Neo 2.7B architectures with feedforward layers removed; evaluation of the average of (0.45) across all the random prompts.

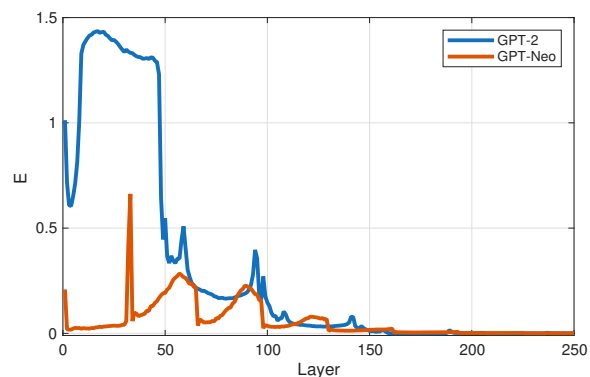


FIGURE 7 Comparison between the GPT-2 XL and GPT-Neo 2.7B architectures with the full model; evaluation of the average of (0.45) across all the random prompts.

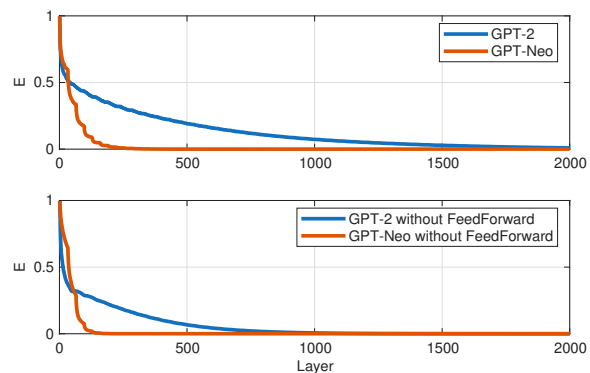
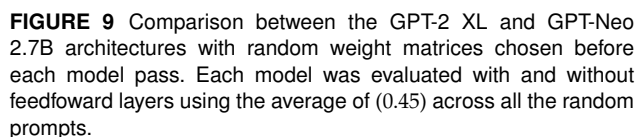


FIGURE 8 Comparison between the GPT-2 XL and GPT-Neo 2.7B architectures with fixed and randomly chosen weight matrices. Each model was evaluated with and without feedforward layers using the average of (0.45) across all the random prompts.

# passes	Output
1	ly networks are which particular simplest forms and sizes, are the a moment impact on many every field of science and technology. that automated them is used.' produce They
2	, neural for network network the the andy the the same good good, The a the,,, a,
4	TheTheTheTheTheTheTheThe TheTheTheTheTheTheTheThe TheTheTheTheTheTheTheThe TheTheTheTheTheTheThe "
10	TheTheTheTheTheTheTheThe TheTheTheTheTheTheTheThe TheTheTheTheTheTheTheThe TheTheTheTheTheTheTheThe

RNNs and SSMs are the architectures that most closely resemble the models studied in control theory as they are directly given as discrete-time nonlinear control systems, see “What are recurrent neural networks and selective



As the field progresses, however, control-theoretic techniques have yet to be used with SSMs. Given their structure, which naturally resembles nonlinear control systems, it remains an open and promising direction to investigate whether insights from control theory can be leveraged to

analyze the limitations of SSMs and potentially improve their design and training.

Safety and Alignment

In recent years, the alignment of LLMs and SSMs with human intent has become a central challenge in deep learning. From a control-theoretic perspective, alignment can be formulated as a safety enforcement problem or as a trajectory tracking problem on a NN regarded as the plant.

Several efforts have adopted a feedback control perspective, even if this is not explicitly stated. One of the most popular techniques for alignment [107] uses human feedback to modify the underlying weights of a trained NN via reinforcement learning. This approach introduces a separate neural network, known as a reward model, which is trained to emulate human preferences. During the alignment phase, the language model acts as the plant, and the reward model provides the feedback signal used by a reinforcement learning algorithm to update the plant's internal parameters. This effectively closes the loop between model generation and human intent, steering the plant towards a preferred behavior.

Performing alignment during training is expensive since it delays the training process which is already quite expensive. A convenient alternative is *test time alignment*, that is, algorithms that act as external controllers to regulate the NN behavior during inference without modifying the underlying weights. This is analogous to adding a controller to steer a given system to a reference.

Control barrier functions (CBFs) have been used for test time alignment purposes. One framework [108] designs an add-on safety filter that intervenes in the token prediction process. The filter acts by filtering out undesirable words thus ensuring the generated text will not violate specified safety constraints, such as toxicity thresholds. A different approach is taken in [109] by adding a one token prefix to every prompt. Control is exerted by steering this prefix so as to prevent toxic outputs. Toxicity is measured by a classifier that maps the state z of an LLM into a numerical toxicity score for the text encoded by the tokens. The key result in [109] is that, under mild technical assumptions, a toxicity classifier defines a CBF for almost every choice of weights of an LLM with the added one token prefix as an input.

In [110], an optimal control approach is adopted to prevent toxicity. Text generation is framed as a trajectory in the space of tokens. The alignment is formulated as the problem of minimally changing this trajectory, subject to the token dynamics (0.24), so as to steer the output of the activation functions away from regions that generate toxic outputs.

Often, preventing toxicity is but one of many objectives. Multi-objective decoding (MOD) [111] addresses the challenge of satisfying multiple competing alignment goals

simultaneously. By formulating the problem as maximizing a linear combination of a measure of goal satisfaction during the decoding step (the map h in (S3) or (S9)), this approach allows for the dynamic adjustment of these objectives at runtime, functioning like a variable-gain controller that can be tuned on the fly.

Controllers for architectural and training design

Feedback controllers have been used to improve the behavior of certain architectures. A prime example is the PIDformer [112] that inserts PID controllers in transformer blocks to create a new architecture. The objective of the PID controller is to prevent consensus, as discussed in *Large language models, transformers, and consensus*, and improve robustness to input perturbations.

In addition to altering architectures, feedback controllers have been designed to improve the training of NNs. Research into PID-based optimizers has shown that applying PID control to adaptive gradient methods can dampen the oscillations seen in standard stochastic gradient descent or Adam method, leading to faster convergence [113]. Other approaches [114] treat learning rate scheduling as an online control problem, adjusting the optimizer's step size based on real-time feedback from the training loss via an incremental PID controller. This line of work suggests that training algorithms can be seen as plants controlled by a feedback controller designed to drive the NN towards a minimum with desirable properties. Until now, only relatively simple control techniques have been used and more advanced control ideas are likely to have greater benefits.

Contraction and implicit neural networks

Contraction theory [115] has found multiple applications within control theory. Recently, its use has been expanded to also include the analysis of deep equilibrium networks [116]. This is a class of NNs where state $z(k+1)$ of layer $k+1$ is computed from the state of layer k using an implicit equation such as

$$z(k+1) = \Sigma(W_1(k)z(k+1) + W_2(k)z(k)). \quad (0.46)$$

One of the key challenges in this class of NNs is to ensure the weight matrices $W_1(k)$ and $W_2(k)$ are such that (0.46) admits a solution for $z(k+1)$. Since a solution can be interpreted as a fixed-point, the use of contraction ideas is natural and was employed in [117] to establish existence of solutions as well as other properties. Recurrent versions of deep equilibrium networks have also been developed and, once again, contraction theory played a key role in their analysis [118]. The authors believe that contraction theory, and its generalizations such as k -contraction [119], can also play an important role in the analysis and design of other classes of NNs as well as their training algorithms.

ACKNOWLEDGMENT

The authors acknowledge Matteo Marchi, Rahal Nanayakkara, and Valen Yamamoto for valuable suggestions on how to improve this article. The 1st author was partially supported by grant PID2024-156578NB-I00 funded by MICIU/AEI/10.13039/501100011033/FEDER, EU. The work of the 2nd and 4th authors was partially supported by the NSF award 2502536. The work of the 3rd author is supported by the Natural Sciences and Engineering Research Council of Canada.

AUTHOR INFORMATION

Álvaro Rodríguez Abella (arabella@comillas.edu) received the PhD degree in Mathematics from Complutense University of Madrid in Spain. He is currently an Assistant Professor of Applied Mathematics at Comillas Pontifical University.

João Pedro Silvestre (joaosilvestre@ucla.edu) received the Masters degree in Electrical and Computer Engineering from Instituto Superior Técnico in Lisbon, Portugal. He is currently pursuing his PhD on Electrical and Computer Engineering at UCLA. He is a Student Member of IEEE.

Bahman Gharesifard (bahman.gharesifard@queensu.ca) received the PhD degree in Mathematics from Queen's University where he is currently a Professor of Mathematics. He is a Senior Member of IEEE.

Paulo Tabuada (tabuada@ee.ucla.edu) received the PhD degree in Electrical and Computer Engineering from Instituto Superior Técnico in Lisbon, Portugal. He is the Vijay K. Dhir Professor of Engineering at UCLA's Electrical and Computer Engineering Department. He is an IEEE Fellow.

REFERENCES

- [1] E. Zuazua, "Machine learning and control: Foundations, advances, and perspectives," 2025. <https://arxiv.org/abs/2510.03303>.
- [2] E. Weinan, J. Han, and Q. Li, "Dynamical systems and optimal control approach to deep learning," *Mathematical Aspects of Deep Learning*, pp. 422–438, 2022.
- [3] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.
- [4] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain," *Psychological review*, vol. 65, no. 6, p. 386, 1958.
- [5] M. Minsky and S. A. Papert, *Perceptrons: An Introduction to Computational Geometry*. The MIT Press, 09 2017.
- [6] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.
- [7] Y. Le Cun, "Learning process in an asymmetric threshold network," in *Disordered systems and biological organization*, pp. 233–240, Springer, 1986.
- [8] P. Werbos, "Beyond regression: New tools for prediction and analysis in the behavioral sciences," *PhD thesis, Committee on Applied Mathematics, Harvard University, Cambridge, MA*, 1974.
- [9] H. R. Wilson and J. D. Cowan, "Excitatory and inhibitory interactions in localized populations of model neurons," *Biophysical journal*, vol. 12, no. 1, pp. 1–24, 1972.
- [10] W. Maass, "Networks of spiking neurons: the third generation of neural network models," *Neural networks*, vol. 10, no. 9, pp. 1659–1671, 1997.
- [11] A. L. Hodgkin and A. F. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve," *The Journal of physiology*, vol. 117, no. 4, p. 500, 1952.
- [12] J. L. Elman, "Finding structure in time," *Cognitive science*, vol. 14, no. 2, pp. 179–211, 1990.
- [13] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber, "Gradient flow in recurrent nets: The difficulty of learning long-term dependencies," in *A Field Guide to Dynamical Recurrent Networks* (J. F. Kolen and S. C. Kremer, eds.), pp. 237–243, IEEE Press, 2001. Available at: <https://ieeexplore.ieee.org/document/5264952>.
- [14] V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814, 2010.
- [15] D. Hendrycks and K. Gimpel, "Gaussian error linear units (Gelus)," *arXiv preprint arXiv:1606.08415*, 2016.
- [16] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. The MIT Press, 2016.
- [17] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, vol. 25, pp. 1097–1105, Curran Associates, Inc., 2012.
- [18] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.
- [19] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein, "Visualizing the loss landscape of neural nets," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems, NIPS'18*, (Red Hook, NY, USA), p. 6391–6401, Curran Associates Inc., 2018.
- [20] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS'17*, (Red Hook, NY, USA), p. 6000–6010, Curran Associates Inc., 2017.
- [21] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2015. Available at: <http://arxiv.org/abs/1409.0473>.
- [22] A. Holtzman, J. Buys, L. Du, M. Forbes, and Y. Choi, "The curious case of neural text degeneration," in *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, OpenReview.net, 2020. Available at: <https://openreview.net/forum?id=rygGQyrFvH>.
- [23] D. Jurafsky and J. H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition, with Language Models*. 3rd ed., 2026. Online manuscript released January 6, 2026.
- [24] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [25] A. Gu, K. Goel, and C. Ré, "Efficiently modeling long sequences with structured state spaces," in *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*, OpenReview.net, 2022. Available at: <https://openreview.net/forum?id=uYLFoz1vlAC>.
- [26] J. T. H. Smith, A. Warrington, and S. W. Linderman, "Simplified state space layers for sequence modeling," 2023.
- [27] C. A. Alonso, J. Sieber, and M. N. Zeilinger, "State space models as foundation models: A control theoretic overview," in *2025 American Control Conference (ACC)*, pp. 146–153, IEEE, 2025.
- [28] A. Gu and T. Dao, "Mamba: Linear-time sequence modeling with selective state spaces," in *First conference on language modeling*, p. 1–32, 2024.
- [29] B. L. et al., "Jamba: Hybrid transformer-mamba language models," in *Proceedings of the Thirteenth International Conference on Learning Representations*, 2025. Available at: <https://openreview.net/forum?id=JFPaD7lpBD>.
- [30] W. E., "A proposal on machine learning via dynamical systems," *Communications in Mathematics and Statistics*, vol. 5, no. 1, pp. 1–11, 2017.
- [31] E. Haber and L. Ruthotto, "Stable architectures for deep neural networks," *Inverse Problems*, vol. 34, p. 014004, dec 2017.
- [32] Y. Lu, A. Zhong, Q. Li, and B. Dong, "Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations," in *Proceedings of the 35th International Conference on Machine Learning, ICML*

- 2018, *Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018* (J. G. Dy and A. Krause, eds.), vol. 80 of *Proceedings of Machine Learning Research*, pp. 3282–3291, PMLR, 2018.
- [33] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud, “Neural ordinary differential equations,” in *Proceedings of the 32nd International Conference on Neural Information Processing Systems, NIPS’18*, (Red Hook, NY, USA), p. 6572–6583, Curran Associates Inc., 2018.
- [34] Z. Li, N. B. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. M. Stuart, and A. Anandkumar, “Fourier neural operator for parametric partial differential equations,” in *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*, OpenReview.net, 2021. Available at: <https://openreview.net/forum?id=c8P9NQVtmnO>.
- [35] N. Kovachki, Z. Li, B. Liu, K. Azizzadenesheli, K. Bhattacharya, A. Stuart, and A. Anandkumar, “Neural operator: Learning maps between function spaces with applications to PDEs,” *Journal of Machine Learning Research*, vol. 24, no. 89, pp. 1–97, 2023.
- [36] M. Krstic, “Machine learning: Bane or boon for control?: 2023 Bode prize lecture,” *IEEE Control Systems*, vol. 44, no. 4, pp. 24–37, 2024.
- [37] A. M. Stuart and A. R. Humphries, *Dynamical Systems and Numerical Analysis*. Cambridge University Press, 1996.
- [38] A. L. Dontchev and W. W. Hager, “The Euler approximation in state constrained optimal control,” *Mathematics of Computation*, vol. 70, no. 233, pp. 173–203, 2001.
- [39] W. Hager, “Runge-Kutta methods in optimal control and the transformed adjoint system,” *Numerische Mathematik*, vol. 87, no. 2, p. 247–282, 2000.
- [40] J. F. Bonnans and A. Festa, “Error estimates for the Euler discretization of an optimal control problem with first-order state constraints,” *SIAM Journal on Numerical Analysis*, vol. 55, no. 2, pp. 445–471, 2017.
- [41] M. Thorpe and Y. van Gennip, “Deep limits of residual neural networks,” *Research in the Mathematical Sciences*, vol. 10, 2023.
- [42] X. Zhang, Z. Li, C. C. Loy, and D. Lin, “PolyNet: A pursuit of structural diversity in very deep networks,” *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3900–3908, 2017.
- [43] M. E. Sander, P. Ablin, M. Blondel, and G. Peyré, “Momentum residual neural networks,” in *Proceedings of the 38th International Conference on Machine Learning* (M. Meila and T. Zhang, eds.), vol. 139 of *Proceedings of Machine Learning Research*, pp. 9276–9287, PMLR, 18–24 Jul 2021.
- [44] A. N. Gómez, M. Ren, R. Urtasun, and R. B. Grosse, “The reversible residual network: backpropagation without storing activations,” in *Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS’17*, (Red Hook, NY, USA), p. 2211–2221, Curran Associates Inc., 2017.
- [45] Q. Li, L. Chen, C. Tai, et al., “Maximum principle based algorithms for deep learning,” *Journal of Machine Learning Research*, vol. 18, no. 165, pp. 1–29, 2018.
- [46] B. Chang, L. Meng, E. Haber, L. Ruthotto, D. Begert, and E. Holtham, “Reversible architectures for arbitrarily deep residual neural networks,” in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence, AAAI’18/IAAI’18/EAAI’18*, pp. Pages 2811 – 2818, AAAI Press, 2018.
- [47] Y. L. Cun, “A theoretical framework for back-propagation,” in *Proceedings of the 1988 Connectionist Models Summer School* (D. S. Touretzky, G. E. Hinton, and T. J. Sejnowski, eds.), pp. 21–28, Pittsburgh, Pennsylvania: Morgan Kaufmann, 1988.
- [48] M. Benning, E. Celledoni, M. J. Ehrhardt, B. Owren, and C.-B. Schönlieb, “Deep learning as optimal control problems: Models and numerical methods,” *Journal of Computational Dynamics*, vol. 6, no. 2, pp. 171–198, 2019.
- [49] E. Hairer, C. Lubich, and G. Wanner, *Geometric numerical integration. Structure-preserving algorithms for ordinary differential equations*, vol. 31 of *Springer Series in Computational Mathematics*. Springer, 2nd ed., 2006.
- [50] Q. Li and S. Hao, “An optimal control approach to deep learning and applications to discrete-weight neural networks,” in *Proceedings of the 35th International Conference on Machine Learning* (J. Dy and A. Krause, eds.), vol. 80 of *Proceedings of Machine Learning Research*, pp. 2985–2994, PMLR, 10–15 Jul 2018.
- [51] D. Zhang, T. Zhang, Y. Lu, Z. Zhu, and B. Dong, “You only propagate once: Accelerating adversarial training via maximal principle,” in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada* (H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. B. Fox, and R. Garnett, eds.), pp. 227–238, 2019.
- [52] J. H. Seidman, M. Fazlyab, V. M. Preciado, and G. J. Pappas, “Robust deep learning as optimal control: Insights and convergence guarantees,” in *Proceedings of the 2nd Conference on Learning for Dynamics and Control* (A. M. Bayen, A. Jadbabaie, G. Pappas, P. A. Parrilo, B. Recht, C. Tomlin, and M. Zeilinger, eds.), vol. 120 of *Proceedings of Machine Learning Research*, pp. 884–893, PMLR, 10–11 Jun 2020.
- [53] E. Celledoni, M. J. Ehrhardt, C. Etmann, R. I. McLachlan, B. Owren, C.-B. Schönlieb, and F. Sherry, “Structure-preserving deep learning,” *European journal of applied mathematics*, vol. 32, no. 5, pp. 888–936, 2021.
- [54] G. Cybenko, “Approximation by superpositions of a sigmoidal function,” *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989.
- [55] K. Hornik, M. Stinchcombe, and H. White, “Multilayer feedforward networks are universal approximators,” *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.
- [56] K. Hornik, “Approximation capabilities of multilayer feedforward networks,” *Neural networks*, vol. 4, no. 2, pp. 251–257, 1991.
- [57] A. Pinkus, “Approximation theory of the MLP model in neural networks,” *Acta Numerica*, vol. 8, pp. 143–195, 1999.
- [58] A. Barron, “Universal approximation bounds for superpositions of a sigmoidal function,” *IEEE Transactions on Information Theory*, vol. 39, no. 3, pp. 930–945, 1993.
- [59] H. Lin and S. Jegelka, “ResNet with one-neuron hidden layers is a universal approximator,” in *Proceedings of the 32nd International Conference on Neural Information Processing Systems, NIPS’18*, (Red Hook, NY, USA), pp. 6172–6181, Curran Associates Inc., 2018.
- [60] S. Park, C. Yun, J. Lee, and J. Shin, “Minimum width for universal approximation,” in *International Conference on Learning Representations*, 2021. Available at: <https://openreview.net/forum?id=O-XJwyoIF-k>.
- [61] P. Kidger and T. Lyons, “Universal Approximation with Deep Narrow Networks,” in *Proceedings of Thirty Third Conference on Learning Theory* (J. Abernethy and S. Agarwal, eds.), vol. 125 of *Proceedings of Machine Learning Research*, pp. 2306–2327, PMLR, 09–12 Jul 2020.
- [62] P. Tabuada and B. Ghahsifard, “Universal approximation power of deep residual neural networks via nonlinear control theory,” in *The 2021 International Conference on Learning Representations (ICLR)*, 2021. Available at: <https://openreview.net/forum?id=1XhmY16R3M>.
- [63] P. Tabuada and B. Ghahsifard, “Universal approximation power of deep residual neural networks through the lens of control,” *IEEE Transactions on Automatic Control*, vol. 68, no. 5, pp. 2715–2728, 2023.
- [64] P. Tabuada and B. Ghahsifard, “Corrigendum to “universal approximation power of deep residual neural networks through the lens of control”,” *IEEE Transactions on Automatic Control*, vol. 69, no. 7, pp. 4891–4892, 2024.
- [65] A. Agrachev and A. Sarychev, “Control on the manifolds of mappings with a view to the deep learning,” *Journal of Dynamical and Control Systems*, vol. 28, no. 4, pp. 989–1008, 2022.
- [66] E. D. Sontag and Y. Qiao, “Further results on controllability of recurrent neural networks,” *Systems & control letters*, vol. 36, no. 2, pp. 121–129, 1999.
- [67] J.-S. Li and N. Khaneja, “Control of inhomogeneous quantum ensembles,” *Physical Review A*, vol. 73, no. 3, p. 030302, 2006.
- [68] R. W. Brockett, “Optimal control of the liouville equation,” in *Proceedings of the International Conference on Complex Geometry and Related Fields*, vol. 39 of *AMS/IP Studies in Advanced Mathematics*, pp. 23–35, Providence, RI: American Mathematical Society, 2007.
- [69] A. Agrachev and A. Sarychev, “Control in the spaces of ensembles of points,” *SIAM Journal on Control and Optimization*, vol. 58, no. 3, pp. 1579–1596, 2020.
- [70] M. D. Kvalheim and E. D. Sontag, “Autoencoding dynamics: Topological limitations and capabilities,” 2025.
- [71] E. Bayram, M.-A. Belabbas, and T. Başar, “Geometric foundations of tuning without forgetting in neural ODEs,” *arXiv preprint arXiv:2509.03474*, 2025.
- [72] T. Veeravalli and M. Raginsky, “Nonlinear controllability and function representation by neural stochastic differential equations,” in *Learning for Dynamics and Control Conference*, pp. 838–850, PMLR, 2023.

- [73] J. Hanson and M. Raginsky, "Fitting an immersed submanifold to data via Sussmann's orbit theorem," *Mathematics of Control, Signals, and Systems*, vol. 37, no. 3, pp. 479–506, 2025.
- [74] J. Hanson and M. Raginsky, "Expressivity of quadratic neural ODEs," *arXiv preprint arXiv:2504.09385*, 2025.
- [75] A. Agrachev and C. Letrouit, "Generic controllability of equivariant systems and applications to particle systems and neural networks," *Annales de l'Institut Henri Poincaré C*, vol. 43, no. 3, pp. 639–668, 2025.
- [76] A. Scagliotti, "Deep learning approximation of diffeomorphisms via linear-control systems," *Mathematical Control and Related Fields*, vol. 13, no. 3, pp. 1226–1257, 2023.
- [77] A. Scagliotti and S. Farinelli, "Normalizing flows as approximations of optimal transport maps via linear-control neural ODEs," *Nonlinear Analysis*, vol. 257, p. 113811, 2025.
- [78] J. Cheng, Q. Li, T. Lin, and Z. Shen, "Interpolation, approximation, and controllability of deep neural networks," *SIAM Journal on Control and Optimization*, vol. 63, no. 1, pp. 625–649, 2025.
- [79] Y. Duan and Y. Cai, "A minimal control family of dynamical systems for universal approximation," *IEEE Transactions on Automatic Control*, vol. 70, no. 11, pp. 7233–7244, 2025.
- [80] A. Álvarez-López, A. H. Slimane, and E. Zuazua, "Interplay between depth and width for interpolation in neural ODEs," *Neural Networks*, vol. 180, p. 106640, 2024.
- [81] C. Prieur, M. Lazar, and B. Robu, "A unified representation of neural networks architectures," 2026.
- [82] D. Ruiz-Balet and E. Zuazua, "Neural ODE control for classification, approximation, and transport," *SIAM Review*, vol. 65, no. 3, pp. 735–773, 2023.
- [83] K. Elamvazhuthi, B. Ghahesifard, A. L. Bertozzi, and S. Osher, "Neural ODE control for trajectory approximation of continuity equation," *IEEE Control Systems Letters*, vol. 6, pp. 3152–3157, 2022.
- [84] K. Elamvazhuthi, X. Zhang, S. Oymak, and F. Pasqualetti, "Learning on manifolds: Universal approximations properties using geometric controllability conditions for neural ODEs," in *Learning for Dynamics and Control Conference*, pp. 1–11, PMLR, 2023.
- [85] D. Ruiz-Balet and E. Zuazua, "Control of neural transport for normalising flows," *Journal de Mathématiques Pures et Appliquées*, vol. 181, pp. 58–90, 2024.
- [86] B. Geshkovski and E. Zuazua, "Turnpike in optimal control of PDEs, ResNets, and beyond," *Acta Numerica*, vol. 31, pp. 135–263, 2022.
- [87] B. Geshkovski, P. Rigollet, and D. Ruiz-Balet, "Measure-to-measure interpolation using transformers," *arXiv preprint arXiv:2411.04551*, 2024.
- [88] D. O. Adu and B. Ghahesifard, "Approximate controllability of continuity equation of transformers," *IEEE Control Systems Letters*, vol. 8, pp. 964–969, 2024.
- [89] Z. Li, T. Xia, Y. Chang, and Y. Wu, "A survey of RWKV," *Neurocomputing*, vol. 649, p. 130711, 2025.
- [90] Y. Lu, Z. Li, D. He, Z. Sun, B. Dong, T. Qin, L. Wang, and T.-Y. Liu, "Understanding and improving transformer from a multi-particle dynamic system point of view," in *Proceedings of the Second Workshop on Machine Learning and the Physical Sciences (NeurIPS 2019)*, 2019. Available at: <https://arxiv.org/abs/1906.02762>.
- [91] S. Dutta, T. Gautam, S. Chakrabarti, and T. Chakraborty, "Redesigning the transformer architecture with insights from multi-particle dynamical systems," in *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual* (M. Ranzato, A. Beygelzimer, Y. N. Dauphin, P. Liang, and J. W. Vaughan, eds.), pp. 5531–5544, 2021.
- [92] Y. Dong, J.-B. Cordonnier, and A. Loukas, "Attention is not all you need: Pure attention loses rank doubly exponentially with depth," in *International Conference on Machine Learning*, pp. 2793–2803, PMLR, 2021.
- [93] B. Geshkovski, C. Letrouit, Y. Polyanskiy, and P. Rigollet, "A mathematical perspective on transformers," *Bulletin of the American Mathematical Society*, vol. 62, no. 3, pp. 427–479, 2025.
- [94] B. Geshkovski, C. Letrouit, Y. Polyanskiy, and P. Rigollet, "The emergence of clusters in self-attention dynamics," *Advances in Neural Information Processing Systems*, vol. 36, pp. 57026–57037, 2023.
- [95] B. Geshkovski, H. Koubbi, Y. Polyanskiy, and P. Rigollet, "Dynamic metastability in the self-attention model," *arXiv preprint arXiv:2410.06833*, 2024.
- [96] G. Bruno, F. Pasqualotto, and A. Agazzi, "Emergence of meta-stable clustering in mean-field transformer models," in *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*, OpenReview.net, 2025. Available at: <https://openreview.net/forum?id=eBS3dQQ8GV>.
- [97] P. Rigollet, "The mean-field dynamics of transformers," *arXiv preprint arXiv:2512.01868*, 2025.
- [98] Á. R. Abella, J. P. Silvestre, and P. Tabuada, "The asymptotic behavior of attention in transformers," *arXiv preprint arXiv:2412.02682*, 2024.
- [99] Á. Rodríguez Abella, J. Silvestre, and P. Tabuada, "Consensus is all you get: The role of attention in transformers," in *Forty-second International Conference on Machine Learning*, pp. 174 – 184, 2025.
- [100] C. Altafini, "Multistability of self-attention dynamics in transformers," *IEEE Transactions on Automatic Control*, pp. 1–16, 2026.
- [101] R. A. Freeman and P. Kokotović, *Robust Nonlinear Control Design*. Birkhäuser Boston, MA, 2008.
- [102] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, et al., "Transformers: State-of-the-art natural language processing," in *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pp. 38–45, 2020.
- [103] N. Nishikawa and T. Suzuki, "State space models are provably comparable to transformers in dynamic token selection," in *International Conference on Learning Representations*, 2024. Available at: <https://api.semanticscholar.org/CorpusID:270095302>.
- [104] W. Merrill, J. Petty, and A. Sabharwal, "The illusion of state in state-space models," in *Proceedings of the 41st International Conference on Machine Learning* (R. Salakhutdinov, Z. Kolter, K. Heller, A. Weller, N. Oliver, J. Scarlett, and F. Berkenkamp, eds.), vol. 235 of *Proceedings of Machine Learning Research*, pp. 35492–35506, PMLR, 21–27 Jul 2024.
- [105] R. Waleffe et al., "An empirical study of mamba-based language models," *arXiv preprint arXiv:2406.07887*, 2024.
- [106] L. Ren, Y. Liu, Y. Lu, Y. Shen, C. Liang, and W. Chen, "Samba: Simple hybrid state space models for efficient unlimited context language modeling," in *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*, OpenReview.net, 2025. Available at: <https://openreview.net/forum?id=bllnpVM4bc>.
- [107] L. Ouyang et al., "Training language models to follow instructions with human feedback," *Advances in Neural Information Processing Systems*, vol. 35, pp. 27730–27744, 2022.
- [108] Y. Miyaoka and M. Inoue, "Control barrier function for aligning large language models," *IEEE Transactions on Control Systems Technology*, pp. 1–10, 2026.
- [109] J. P. Silvestre, Á. Rodríguez Abella, and P. Tabuada, "Stopping LLMs from going rogue: A control barrier approach to text generation," in *2025 IEEE 64th Conference on Decision and Control (CDC)*, pp. 151–156, 2025.
- [110] E. Cheng and C. A. Alonso, "Linearly controlled language generation with performative guarantees," *arXiv preprint arXiv:2405.15454*, 2024.
- [111] R. Shi, Y. Chen, Y. Hu, A. Liu, H. Hajishirzi, N. A. Smith, and S. S. Du, "Decoding-time language model alignment with multiple objectives," in *Advances in Neural Information Processing Systems* (A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, eds.), vol. 37, pp. 48875–48920, Curran Associates, Inc., 2024.
- [112] T. M. Nguyen, C. A. Uribe, T. M. Nguyen, and R. Baraniuk, "PIDformer: Transformer meets control theory," in *Proceedings of the 41st International Conference on Machine Learning* (R. Salakhutdinov, Z. Kolter, K. Heller, A. Weller, N. Oliver, J. Scarlett, and F. Berkenkamp, eds.), vol. 235 of *Proceedings of Machine Learning Research*, pp. 37776–37797, PMLR, 21–27 Jul 2024.
- [113] S. Chen, J. Liu, P. Wang, C. Xu, S. Cai, and J. Chu, "Accelerated optimization in deep learning with a proportional-integral-derivative controller," *Nature Communications*, vol. 15, no. 1, 2024.
- [114] Z. Wang and J. Zhang, "Incremental PID controller-based learning rate scheduler for stochastic gradient descent," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 35, no. 5, pp. 7060–7071, 2022.
- [115] F. Bullo, *Contraction Theory for Dynamical Systems*. Seattle, WA: Kindle Direct Publishing, 1.3 ed., 2026.

- [116] S. Bai, J. Z. Kolter, and V. Koltun, "Deep equilibrium models," in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada* (H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, eds.), pp. 688–699, 2019.
- [117] S. Jafarpour, A. Davydov, A. V. Proskurnikov, and F. Bullo, "Robust implicit networks via non-euclidean contractions," in *Proceedings of the 35th International Conference on Neural Information Processing Systems, NIPS '21*, (Red Hook, NY, USA), Curran Associates Inc., 2021.
- [118] M. Revay, R. Wang, and I. R. Manchester, "Recurrent equilibrium networks: Flexible dynamic models with guaranteed stability and robustness," *IEEE Transactions on Automatic Control*, vol. 69, no. 5, pp. 2855–2870, 2024.
- [119] C. Wu, I. Kanevskiy, and M. Margaliot, "k-contraction: Theory and applications," *Automatica*, vol. 136, p. 110048, 2022.

