Apprentissage automatique et systèmes dynamiques : Comment l'IA découvre les lois cachées de la nature

Franck Jeannot

Montréal, Canada, AC851, Décembre 2025

Résumé

Les systèmes dynamiques gouvernent l'évolution temporelle de phénomènes complexes dans la nature, allant des circuits électriques aux réactions chimiques en passant par les signaux biologiques. Traditionnellement, la découverte des lois mathématiques décrivant ces systèmes nécessitait une expertise scientifique considérable et des analyses mathématiques complexes. Dans cet article de vulgarisation basé sur l'article de Moore, Mann & Chen (2025), on présente les concepts fondamentaux de l'intelligence artificielle (IA) et des systèmes dynamiques, puis on explore comment une nouvelle approche, basée sur l'opérateur de Koopman et les réseaux de neurones profonds, permet d'automatiser cette découverte scientifique. Cette méthode transforme des dynamiques non linéaires complexes en représentations linéaires de dimension réduite, facilitant ainsi l'analyse, la prédiction et le contrôle de systèmes auparavant insaisissables. Nous détaillons progressivement les concepts d'apprentissage automatique, de réseaux encodeurs-décodeurs, de fonctions propres de Koopman, et de fonctions de Lyapunov neuronales, offrant ainsi une introduction accessible aux lecteurs débutants tout en préservant la rigueur scientifique nécessaire.

Keywords: Intelligence artificielle, Systèmes dynamiques, Opérateur de Koopman, Réseaux de neurones, Réduction dimensionnelle, Fonctions de Lyapunov, Apprentissage automatique, Autoencodeur

1. Introduction

1.1. Contexte historique et motivation

Depuis la publication des *Philosophiae Naturalis Principia Mathematica* par Isaac Newton en 1687, l'étude des systèmes dynamiques a façonné notre compréhension du monde naturel [1, 2]. Initialement centrée sur les forces entre corps célestes, cette discipline a évolué vers un cadre théorique plus vaste englobant l'analyse de variables d'état variant dans le temps dans des

domaines aussi divers que l'ingénierie mécanique et électrique, la climatologie, les neurosciences, la physiologie et l'écologie [3, 4].

Un système dynamique décrit comment l'état d'un système évolue au fil du temps selon des règles déterminées. Par exemple, le pendule oscillant, les battements du cœur humain, les fluctuations boursières ou encore les prévisions météorologiques sont tous régis par des systèmes dynamiques. Historiquement, les scientifiques ont cherché à dériver des équations mathématiques précises pour décrire ces comportements, une tâche exigeant une expertise considérable et souvent limitée à des systèmes relativement simples [5].

Aujourd'hui, l'intelligence artificielle (IA) offre de nouvelles possibilités révolutionnaires. Plutôt que de formuler manuellement des équations à partir de principes fondamentaux, les chercheurs peuvent désormais utiliser des algorithmes d'apprentissage automatique pour découvrir automatiquement les lois sous-jacentes directement à partir de données observationnelles [5, 2]. Cette approche data-driven (guidée par les données) présente un potentiel immense pour élucider des systèmes complexes où les équations traditionnelles sont soit inconnues, soit trop compliquées à formuler.

1.2. L'article de référence : Moore, Mann et Chen (2025)

L'article scientifique sur lequel se base cette synthèse, publié dans *npj Complexity* en décembre 2025 par Samuel Moore, Brian Mann et Boyuan Chen de l'Université Duke [1], présente un cadre novateur d'intelligence artificielle capable de découvrir des règles simples et compréhensibles régissant des dynamiques hautement complexes. Cette approche s'inspire du travail des grands « dynamiciens » de l'histoire—scientifiques étudiant les systèmes évoluant dans le temps—qui ont découvert de nombreuses lois physiques gouvernant ces comportements [2].

Tout comme Newton, le premier dynamicien, a dérivé les équations reliant force et mouvement, le système d'IA développé par Moore et ses collaborateurs analyse des données sur l'évolution temporelle de systèmes complexes et génère automatiquement des équations qui les décrivent avec précision [1, 5]. Ce qui distingue particulièrement cette approche est sa capacité à traiter une complexité dépassant largement les capacités humaines : l'IA peut transformer des systèmes non linéaires impliquant des centaines voire des milliers de variables interdépendantes en règles simplifiées comportant beaucoup moins de dimensions [2].

La méthode repose sur la théorie de l'opérateur de Koopman, un cadre mathématique qui représente des dynamiques non linéaires en termes d'un opérateur linéaire de dimension infinie agissant sur l'espace de toutes les fonctions de mesure possibles du système [6, 1]. En trouvant des représentations

de dimension finie tractables de cet opérateur, les chercheurs peuvent effectivement linéariser des dynamiques fortement non linéaires, facilitant ainsi considérablement l'analyse et le contrôle [6].

1.3. Objectifs de cet article de vulgarisation

Cet article vise à rendre accessible au lecteur débutant les concepts fondamentaux sous-tendant cette révolution scientifique. Nous structurerons notre présentation en deux volets complémentaires :

- **Concepts d'intelligence artificielle** : Nous introduirons progressivement les fondements de l'apprentissage automatique, les réseaux de neurones artificiels, les architectures encodeur-décodeur (autoencodeurs), les mécanismes d'entraînement par rétropropagation, et les fonctions de perte. Ces éléments constituent la boîte à outils informatique permettant à l'IA d'apprendre à partir de données.
- **Application aux systèmes dynamiques** : Nous explorerons comment ces techniques d'IA sont appliquées spécifiquement aux systèmes dynamiques, en détaillant la théorie de l'opérateur de Koopman, les fonctions propres et valeurs propres, la réduction dimensionnelle, et les fonctions de Lyapunov neuronales pour l'analyse de stabilité.

Notre approche pédagogique suivra une progression logique, chaque section s'appuyant sur les connaissances établies précédemment, afin de construire une compréhension cohérente et complète de cette intersection fascinante entre intelligence artificielle et systèmes dynamiques.

2. Fondements de l'intelligence artificielle

2.1. Qu'est-ce que l'apprentissage automatique ?

L'apprentissage automatique (machine learning en anglais) constitue une branche de l'intelligence artificielle permettant aux ordinateurs d'apprendre à partir de données sans être explicitement programmés pour chaque tâche spécifique [7, 8]. Contrairement à la programmation traditionnelle où un développeur écrit des instructions détaillées pour résoudre un problème, l'apprentissage automatique permet aux algorithmes de découvrir automatiquement des patterns (motifs) et des règles à partir d'exemples.

Il existe trois paradigmes principaux d'apprentissage automatique [7, 9, 10] :

1. **Apprentissage supervisé** : Le modèle est entraîné sur des données étiquetées, où chaque exemple d'entrée est associé à une sortie connue. Par exemple, pour apprendre à reconnaître des images de chats et de chiens, on fournirait au modèle des milliers d'images déjà étiquetées

« chat » ou « chien ». Le modèle apprend alors à généraliser ces patterns pour classifier correctement de nouvelles images jamais vues auparavant [7, 9].

2. **Apprentissage non supervisé** : Le modèle travaille avec des données non étiquetées et doit découvrir lui-même la structure sous-jacente des données. Cette approche est utilisée pour des tâches comme le regroupement (clustering), la détection d'anomalies ou la réduction dimensionnelle [7, 9]. Par exemple, un algorithme pourrait automatiquement grouper des clients en segments selon leurs comportements d'achat sans qu'on lui indique à l'avance quels groupes existent.
3. **Apprentissage par renforcement** : Un agent apprend à prendre des décisions en interagissant avec un environnement, recevant des récompenses ou pénalités selon ses actions. Cette approche est utilisée dans la robotique, les jeux et la navigation autonome [10, 8].

Dans le contexte de l'article de Moore et al. [1], l'approche utilisée s'apparente principalement à l'apprentissage non supervisé, car le système doit découvrir la structure sous-jacente des dynamiques à partir de séquences temporelles de données, sans étiquettes explicites indiquant quelle est la « bonne » représentation simplifiée.

2.2. Réseaux de neurones artificiels : principes de base

Les réseaux de neurones artificiels constituent l'architecture computationnelle fondamentale de l'apprentissage profond (deep learning) moderne. Inspirés vaguement du fonctionnement du cerveau humain, ces réseaux sont composés de couches de neurones artificiels interconnectés [11, 12].

2.2.1. Structure d'un neurone artificiel

Un neurone artificiel est une unité computationnelle simple qui :

1. Reçoit plusieurs entrées numériques $\{x_1, x_2, \ldots, x_n\}$
2. Multiplie chaque entrée par un poids correspondant $\{w_1, w_2, \ldots, w_n\}$
3. Additionne ces produits pondérés et ajoute un biais b : $z = \sum_{i=1}^{n} w_i x_i + b$
4. Applique une fonction d'activation $\sigma(\cdot)$ pour produire la sortie : $a = \sigma(z)$

Les poids w_i et le biais b sont les paramètres ajustables du neurone, qui seront optimisés durant l'entraînement. La fonction d'activation introduit des non-linéarités essentielles permettant au réseau d'apprendre des relations complexes. Les fonctions d'activation courantes incluent la fonction sigmoïde, la tangente hyperbolique (tanh), et la fonction ReLU (Rectified Linear Unit) [11].

2.2.2. Architecture en couches

Un réseau de neurones est organisé en couches successives [12, 13] :

- **Couche d'entrée** : Reçoit les données brutes
- **Couches cachées** : Effectuent des transformations successives des données, extrayant progressivement des caractéristiques de plus en plus abstraites
- **Couche de sortie** : Produit la prédiction finale

Les réseaux comportant plusieurs couches cachées sont appelés réseaux profonds (deep networks), d'où le terme « apprentissage profond » (deep learning). La profondeur permet d'apprendre des représentations hiérarchiques complexes, où chaque couche capture des patterns à différents niveaux d'abstraction [14].

2.3. Entraînement des réseaux : rétropropagation et descente de gradient

2.3.1. Fonction de perte

Pour qu'un réseau de neurones apprenne, il faut quantifier à quel point ses prédictions sont éloignées des valeurs désirées. C'est le rôle de la fonction de perte (loss function), qui attribue un score numérique mesurant l'erreur du modèle [15, 16].

Pour les problèmes de régression (prédiction de valeurs continues), la fonction de perte la plus courante est l'erreur quadratique moyenne (Mean Squared Error, MSE) [15, 17] :

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

où y_i représente la valeur réelle, $\hat{y}_i$ la prédiction du modèle, et n le nombre d'exemples. En élevant les différences au carré, on pénalise davantage les grandes erreurs, et la moyenne normalise l'erreur totale par rapport au nombre d'échantillons [15, 18].

L'objectif de l'entraînement est de minimiser cette fonction de perte en ajustant les poids et biais du réseau. Plus la perte est faible, meilleure est la performance du modèle [16].

2.3.2. Descente de gradient

La descente de gradient est l'algorithme d'optimisation fondamental utilisé pour minimiser la fonction de perte. L'idée est d'ajuster itérativement les paramètres du réseau dans la direction qui réduit le plus la perte [11, 13].

Mathématiquement, pour un paramètre w (poids), la mise à jour s'écrit :

$$w_{\text{nouveau}} = w_{\text{ancien}} - \alpha\frac{\partial L}{\partial w}$$

où L est la fonction de perte, $\frac{\partial L}{\partial w}$ est le gradient (dérivée partielle) de la perte par rapport au poids, et α est le taux d'apprentissage (learning rate), un hyperparamètre contrôlant la taille des pas d'ajustement [13, 19].

Le gradient indique la direction et l'amplitude de l'augmentation de la perte. En soustrayant le gradient multiplié par le taux d'apprentissage, on déplace le poids dans la direction opposée, réduisant ainsi la perte [11].

2.3.3. Rétropropagation (backpropagation)

La rétropropagation est l'algorithme qui calcule efficacement les gradients nécessaires à la descente de gradient dans un réseau de neurones multicouche [11, 12]. C'est une application intelligente de la règle de dérivation en chaîne du calcul différentiel.

Le processus comporte deux phases principales [13, 19] :

1. **Propagation avant (forward pass)** : Les données traversent le réseau couche par couche, de l'entrée vers la sortie, produisant une prédiction.
2. **Propagation arrière (backward pass)** : L'erreur à la sortie est propagée en sens inverse à travers le réseau. Pour chaque poids, on calcule sa contribution à l'erreur totale en utilisant la règle de dérivation en chaîne.

La rétropropagation permet de calculer le gradient de la perte par rapport à chaque poids du réseau en un seul passage arrière, ce qui est beaucoup plus efficace que de calculer chaque dérivée partielle individuellement [20, 12]. Sans rétropropagation, l'entraînement de réseaux profonds serait d'un coût de calcul prohibitif.

2.4. Autoencodeurs : architecture encodeur-décodeur

Les autoencodeurs constituent une architecture de réseau de neurones particulièrement pertinente pour le travail de Moore et al. [1], car ils sont conçus pour apprendre des représentations compressées de données de haute dimension [21, 14].

2.4.1. Principe général

Un autoencodeur est un réseau de neurones entraîné de manière non supervisée pour reconstruire ses propres entrées [22, 23]. Il se compose de deux parties principales :

— **Encodeur** : Compresse les données d'entrée en une représentation de dimension réduite appelée espace latent (latent space) ou goulot d'étranglement (bottleneck).

— **Décodeur** : Reconstruit les données d'entrée originales à partir de cette représentation compressée.

Formellement, si $x \in \mathbb{R}^n$ représente l'entrée de haute dimension, l'encodeur apprend une fonction $f_{\text{enc}} : \mathbb{R}^n \to \mathbb{R}^d$ qui mappe x vers une représentation latente $z \in \mathbb{R}^d$ de dimension inférieure (où $d \ll n$, c'est-à-dire d est très inférieur à n), et le décodeur apprend une fonction $f_{\text{dec}} : \mathbb{R}^d \to \mathbb{R}^n$ qui reconstruit l'entrée [14, 24].

2.4.2. Entraînement et fonction de perte

L'autoencodeur est entraîné en minimisant la différence entre l'entrée originale x et sa reconstruction $\hat{x}$, typiquement mesurée par l'erreur quadratique moyenne [23, 24] :

$$L = \frac{1}{n} \sum_{i=1}^{n} \|x_i - \hat{x}_i\|^2$$

En forçant l'information à passer par un goulot d'étranglement de dimension réduite, l'autoencodeur est obligé d'apprendre les caractéristiques les plus saillantes et essentielles des données, éliminant le bruit et les redondances [21, 14].

2.4.3. Réduction dimensionnelle non linéaire

Un avantage majeur des autoencodeurs par rapport aux méthodes classiques de réduction dimensionnelle comme l'analyse en composantes principales (PCA) est leur capacité à effectuer des réductions non linéaires [21, 14]. Alors que la PCA ne peut capturer que des relations linéaires, les autoencodeurs, grâce à leurs fonctions d'activation non linéaires, peuvent apprendre des *manifolds* (variétés), c'est-à-dire des surfaces généralisées de dimension inférieure immergées dans l'espace de haute dimension où résident effectivement les données.

Cette capacité de réduction dimensionnelle non linéaire est exactement ce dont on a besoin pour les systèmes dynamiques complexes, où les relations entre variables d'état sont souvent fortement non linéaires [1].

2.4.4. Types d'autoencodeurs

Plusieurs variantes d'autoencodeurs existent, chacune adaptée à des applications spécifiques [21, 24] :

— **Autoencodeurs sous-complets** : Le goulot d'étranglement a une dimension strictement inférieure à l'entrée, forçant la compression.

— **Autoencodeurs épars** : Utilisent une régularisation pour encourager la parcimonie (sparsity), où seuls quelques neurones sont actifs simultanément.

— **Autoencodeurs débruiteurs** : Entraînés à reconstruire des entrées propres à partir de versions bruitées, apprenant ainsi des représentations robustes.
— **Autoencodeurs variationnels** : Apprennent des distributions probabilistes dans l'espace latent, permettant la génération de nouvelles données.

3. Systèmes dynamiques : concepts fondamentaux

3.1. Qu'est-ce qu'un système dynamique ?

Un système dynamique est un modèle mathématique décrivant comment l'état d'un système évolue au fil du temps selon des règles déterminées [4, 3]. L'état représente l'ensemble des variables nécessaires pour caractériser complètement le système à un instant donné.

3.1.1. Systèmes à temps continu vs discret

Les systèmes dynamiques se divisent en deux catégories principales selon la nature du temps [4, 25] :

1. **Systèmes à temps continu** : L'évolution est décrite par des équations différentielles ordinaires (EDO). Par exemple, pour un système d'état $x(t)$ (Cas général) :

$$\frac{dx}{dt} = f(x, t)$$

où f est une fonction définissant la dynamique du système [3, 4].

2. **Systèmes à temps discret** : L'évolution se produit à des instants discrets, décrite par des équations aux différences. Par exemple :

$$x_{n+1} = F(x_n)$$

où F est une fonction de transition [25].

3.1.2. Exemples classiques

Plusieurs systèmes dynamiques classiques illustrent des comportements fondamentaux [1, 3] :

— **Pendule amorti** : Un pendule soumis à la gravité et au frottement, dont le mouvement décroît progressivement vers l'équilibre.
— **Modèle de Hodgkin-Huxley** : Introduit en 1952 pour décrire les mécanismes d'excitation des neurones, ce modèle comporte quatre variables d'état, des non-linéarités fortes et des oscillations auto-entretenues [1]. Ce modèle a contribué à l'attribution du prix Nobel de physiologie ou médecine à Alan Hodgkin et Andrew Huxley en 1963.

— **Système de Lorenz-96** : Un modèle simplifié de processus atmosphériques utilisé pour étudier la prédictibilité météorologique et le chaos [1].

3.2. Points fixes, attracteurs et bassins d'attraction

3.2.1. Points fixes et stabilité

Un point fixe (ou point d'équilibre) x^* d'un système dynamique est un état qui ne change pas dans le temps [3, 26] :

$$f(x^*) = 0 \quad \text{(pour les systèmes continus)}$$
$$F(x^*) = x^* \quad \text{(pour les systèmes discrets)}$$

Les points fixes peuvent être stables ou instables. Un point fixe stable attire les trajectoires voisines : si le système commence près de ce point, il y converge avec le temps. Un point fixe instable repousse les trajectoires voisines [3, 4].

3.2.2. Attracteurs

Un attracteur est un ensemble vers lequel le système tend à évoluer, indépendamment (dans certaines limites) de ses conditions initiales [27**?**]. Les attracteurs peuvent prendre différentes formes :

— **Point fixe** : Le système converge vers un état stationnaire unique.
— **Cycle limite** : Le système oscille périodiquement le long d'une trajectoire fermée.
— **Attracteur quasi-périodique** : Oscillations avec plusieurs fréquences incommensurables.
— **Attracteur chaotique** : Comportement apparemment aléatoire mais déterministe, extrêmement sensible aux conditions initiales.

Le modèle de Hodgkin-Huxley mentionné précédemment possède un attracteur de type cycle limite, représentant les oscillations neuronales [1].

3.2.3. Bassins d'attraction

Le bassin d'attraction d'un attracteur est l'ensemble de toutes les conditions initiales qui conduisent le système vers cet attracteur à long terme [27]. Dans l'espace des phases (l'espace de tous les états possibles), chaque attracteur possède son propre bassin.

Pour les systèmes avec plusieurs attracteurs, les bassins peuvent avoir des structures géométriques très complexes [28]. Les frontières entre bassins peuvent être fractales, ce qui signifie que deux conditions initiales arbitrairement proches peuvent conduire à des attracteurs différents—une propriété ayant des implications profondes pour la prédictibilité du système.

3.3. Valeurs propres et fonctions propres

3.3.1. Concepts mathématiques

Les valeurs propres et vecteurs propres sont des concepts fondamentaux de l'algèbre linéaire. Pour une matrice A, un vecteur ϕ est un vecteur propre avec valeur propre λ si [29] :

$$A\phi = \lambda\phi$$

Cela signifie que l'application de A à ϕ se réduit simplement à une mise à l'échelle par le facteur λ, sans changement de direction [29].

Pour les opérateurs linéaires agissant sur des espaces de fonctions (de dimension potentiellement infinie), on parle de fonctions propres et valeurs propres [30, 31]. Une fonction $\phi(x)$ est une fonction propre d'un opérateur D avec valeur propre λ si :

$$D\phi = \lambda\phi$$

3.3.2. Importance pour les systèmes dynamiques

Les valeurs propres et fonctions propres jouent un rôle crucial dans l'analyse des systèmes dynamiques [29]. Elles permettent de :

- Décomposer des comportements complexes en modes plus simples
- Déterminer la stabilité d'équilibres et d'orbites périodiques
- Identifier les échelles de temps caractéristiques du système
- Comprendre comment les perturbations se propagent et s'amplifient ou décroissent

Dans le contexte de l'article de Moore et al. [1], les fonctions propres de l'opérateur de Koopman fournissent des coordonnées dans lesquelles les dynamiques non linéaires deviennent linéaires, facilitant considérablement l'analyse.

4. L'opérateur de Koopman : linéariser le non-linéaire

4.1. Motivation et principe fondamental

4.1.1. Le défi des systèmes non linéaires

La plupart des systèmes dynamiques réels sont non linéaires, ce qui signifie que le principe de superposition ne s'applique pas : la somme de deux solutions n'est généralement pas elle-même une solution [6]. Cette non-linéarité engendre des phénomènes complexes comme les décalages de fréquence, la génération d'harmoniques, le chaos, et les bifurcations, rendant l'analyse et le contrôle très difficiles.

Les systèmes linéaires, en revanche, sont beaucoup plus faciles à analyser. Ils admettent des solutions sous forme de superpositions de modes propres, peuvent être décomposés spectralement, et disposent d'une vaste théorie mathématique pour leur analyse et leur contrôle [6, 32].

4.1.2. L'idée révolutionnaire de Koopman

En 1931, Bernard Koopman a proposé une perspective révolutionnaire : au lieu d'étudier directement l'évolution non linéaire de l'état $x(t)$ d'un système, on peut étudier l'évolution de fonctions (observables) de cet état [6, 33]. Cette approche transforme le problème non linéaire en dimension finie en un problème linéaire en dimension infinie. Bien que cette dimension infinie puisse sembler problématique, la linéarité ainsi obtenue permet d'utiliser des outils mathématiques puissants et de trouver des approximations de dimension finie très efficaces.

Formellement, pour un système dynamique $x_{k+1} = F(x_k)$ (temps discret) ou $\frac{dx}{dt} = f(x)$ (temps continu), l'opérateur de Koopman $\mathcal{K}$ agit sur des fonctions (observables) $g(x)$ de l'état plutôt que sur l'état lui-même [6, 32] :

$$\mathcal{K}g(x) = g(F(x))$$

pour les systèmes discrets, ou pour la version continue :

$$\frac{d}{dt}g = \mathcal{K}g$$

où $\mathcal{K}$ est le générateur infinitésimal [32].

L'aspect remarquable est que **l'opérateur de Koopman est toujours linéaire**, même lorsque la dynamique sous-jacente F est fortement non linéaire [6, 33]. Cette linéarité découle directement de la linéarité de l'opérateur de composition.

4.2. Fonctions propres de Koopman et coordonnées intrinsèques

4.2.1. Décomposition spectrale

Si l'on peut trouver les fonctions propres $\phi_j(x)$ et valeurs propres λ_j de l'opérateur de Koopman [6, 32] :

$$\mathcal{K}\phi_j = \lambda_j \phi_j$$

alors toute observable $g(x)$ peut être décomposée en termes de ces fonctions propres :

$$g(x) = \sum_{j=1}^{\infty} b_j \phi_j(x)$$

L'évolution temporelle de g devient alors simplement [6] :

$$g(x(t)) = \sum_j b_j \cdot \lambda_j^t \cdot \phi_j(x(0))$$

pour les systèmes discrets, ou $g(x(t)) = \sum_j b_j e^{\lambda_j t} \phi_j(x(0))$ pour les systèmes continus.

4.2.2. Linéarisation dans les coordonnées de Koopman

Les fonctions propres de Koopman définissent un système de coordonnées naturel $\phi(x) = [\phi_1(x), \phi_2(x), \ldots]$ dans lequel les dynamiques non linéaires originales deviennent complètement linéaires [6, 32] :

$$\phi(x(t+1)) = \Lambda\phi(x(t))$$

où Λ est une matrice diagonale contenant les valeurs propres. C'est cette propriété que Moore et al. exploitent pour simplifier radicalement l'analyse de systèmes complexes [1].

Par exemple, toute quantité conservée d'un système dynamique (comme l'énergie dans un système conservatif) est une fonction propre de Koopman correspondant à la valeur propre $\lambda = 1$ (ou $\lambda = 0$ pour le générateur continu) [6, 32].

4.3. Approximations de dimension finie : DMD et extensions

4.3.1. Le défi pratique

En théorie, l'opérateur de Koopman agit sur un espace de dimension infinie de fonctions. En pratique, il faut trouver des approximations de dimension finie exploitables numériquement pour effectuer des calculs concrets [6, 34].

4.3.2. Dynamic Mode Decomposition (DMD)

La décomposition en modes dynamiques (DMD) est devenue l'algorithme numérique le plus populaire pour approximer l'opérateur de Koopman [35, 6]. Développée par Schmid et Sesterhenn en 2008, la DMD analyse des séquences temporelles de données pour extraire des modes cohérents, chacun associé à une fréquence d'oscillation et un taux de croissance/décroissance fixes [35, 36].

Pour une séquence de snapshots (instantanés) de l'état du système $\{x_1, x_2, \ldots, x_N\}$, la DMD cherche une matrice A telle que [35] :

$$x_{k+1} \approx Ax_k$$

Les valeurs propres et vecteurs propres de A fournissent une approximation des valeurs propres et fonctions propres de Koopman dans la base des snapshots.

4.3.3. Extended DMD et apprentissage profond

Une limitation de la DMD standard est qu'elle utilise directement les variables d'état comme observables. L'Extended DMD (EDMD) permet d'utiliser des dictionnaires de fonctions non linéaires des variables d'état, améliorant considérablement la capacité à capturer des dynamiques complexes [35, 34].

L'approche de Moore et al. [1] va plus loin en utilisant des réseaux de neurones profonds pour apprendre automatiquement les meilleures fonctions observables (fonctions propres de Koopman) directement à partir des données, sans avoir à spécifier manuellement un dictionnaire de fonctions.

5. L'approche de Moore, Mann et Chen : IA rencontre Koopman

5.1. Architecture globale du système

5.1.1. Réseau encodeur-décodeur avec structure Koopman

Le cadre développé par Moore et al. [1] combine élégamment les autoencodeurs neuronaux avec la théorie de l'opérateur de Koopman. L'architecture comprend trois composantes principales :

1. **Encodeur neuronal** : Un réseau de neurones profond qui mappe l'état de haute dimension $x \in \mathbb{R}^n$ vers un espace latent de dimension réduite $z \in \mathbb{R}^d$ (où $d \ll n$). Cruciallement, cet encodeur apprend à approximer les fonctions propres de l'opérateur de Koopman.
2. **Dynamique linéaire latente** : Dans l'espace latent, la dynamique est contrainte à être linéaire et peut être représentée par une simple matrice $\mathcal{G}$ telle que $z_{k+1} = \mathcal{G} z_k$. Cette contrainte de linéarité reflète directement la propriété fondamentale de l'opérateur de Koopman. Cette matrice $\mathcal{G}$ constitue une approximation de dimension finie de l'opérateur de Koopman $\mathcal{K}$ introduit précédemment.
3. **Décodeur neuronal** : Un réseau de neurones qui reconstruit l'état original x à partir de la représentation latente z.

Cette architecture force le réseau à apprendre une représentation où les dynamiques complexes et non linéaires dans l'espace d'état original deviennent de simples évolutions linéaires dans l'espace latent.

5.1.2. Fonction de perte multi-objectifs

L'entraînement du réseau utilise une fonction de perte combinant plusieurs termes [1] :

— **Perte de reconstruction** : Mesure l'écart entre l'entrée originale et sa reconstruction, assurant que l'encodeur-décodeur préserve l'information essentielle.

— **Perte de prédiction** : Compare les prédictions du modèle linéaire latent avec les trajectoires réelles observées, garantissant que la dynamique latente capture fidèlement l'évolution temporelle.
— **Régularisation spectrale** : Pénalise le spectre (valeurs propres) de la matrice $\mathcal{G}$ pour encourager l'apprentissage de dynamiques stables ou neutralement stables.
— **Perte d'auxiliaire** : Encourage l'encodeur à approximer des fonctions propres véritables de l'opérateur de Koopman en imposant des contraintes supplémentaires sur leur comportement.

5.2. Réduction dimensionnelle adaptative

5.2.1. Détermination automatique de la dimension

Un défi majeur dans la modélisation de systèmes dynamiques est de déterminer le nombre minimal de dimensions nécessaires pour capturer fidèlement le comportement du système. Moore et al. [1] abordent ce problème en analysant le spectre (valeurs propres) de la matrice de dynamique latente $\mathcal{G}$.

Les valeurs propres révèlent les échelles de temps caractéristiques du système. Les modes associés à des valeurs propres proches de zéro (pour le générateur continu) ou de un (pour les systèmes discrets) correspondent à des processus lents, tandis que ceux avec des valeurs propres plus éloignées correspondent à des processus rapides qui décroissent ou divergent rapidement [1, 6].

5.2.2. Exemples de réduction

L'article démontre des réductions spectaculaires pour plusieurs systèmes classiques [1] :

— **Système de Lorenz-96** : Réduit de 40 dimensions originales à seulement 14 dimensions latentes, tout en maintenant des prédictions précises sur de longs horizons temporels.
— **Modèle de Hodgkin-Huxley** : Les quatre variables d'état du modèle neuronal sont effectivement compressées en un nombre réduit de coordonnées de Koopman capturant les oscillations auto-entretenues.

Cette capacité de réduction dimensionnelle est rendue possible par le fait que, bien que le système ait de nombreuses variables, celles-ci partagent souvent de l'information et évoluent de manière corrélée. Les fonctions propres de Koopman identifient les véritables degrés de liberté indépendants [1].

5.3. Découverte de structures invariantes

5.3.1. Identification des attracteurs

Le cadre ne se contente pas de prédire l'évolution future—il identifie automatiquement les structures géométriques fondamentales du système dyna-

mique, notamment les attracteurs [1, 2].

En analysant les fonctions propres apprises et leurs valeurs propres associées, le système peut localiser :

— Les points fixes (équilibres)
— Les cycles limites (orbites périodiques attractives)
— Les attracteurs chaotiques
— Les bassins d'attraction de chacun de ces attracteurs

Pour les praticiens, identifier ces structures est essentiel car elles représentent les « repères » d'un nouveau paysage dynamique [2]. Une fois qu'on connaît les points stables, le reste du système commence à avoir du sens.

5.3.2. Exemple : pendule magnétique

L'article étudie un pendule magnétique expérimental, un système présentant plusieurs attracteurs correspondant aux différentes positions d'équilibre vers lesquelles le pendule peut converger selon ses conditions initiales [1]. Le réseau neuronal a appris à :

1. Identifier automatiquement ces multiples attracteurs
2. Cartographier leurs bassins d'attraction respectifs
3. Prédire correctement vers quel attracteur le système convergera pour une condition initiale donnée

Remarquablement, le modèle a même correctement inféré que deux conditions initiales très proches dans l'espace d'état pouvaient conduire à des attracteurs opposés, un comportement révélant la structure fractale de la frontière entre bassins [1].

6. Fonctions de Lyapunov neuronales : analyse de stabilité automatisée

6.1. Théorie classique de Lyapunov

6.1.1. Qu'est-ce qu'une fonction de Lyapunov ?

Dans l'analyse des systèmes dynamiques, déterminer la stabilité d'un point d'équilibre est fondamental. La méthode directe de Lyapunov, développée par le mathématicien russe Aleksandr Lyapunov à la fin du 19ème siècle, fournit un outil puissant pour cette analyse [26, 37].

Une fonction de Lyapunov $V(x)$ pour un système dynamique avec un point d'équilibre à l'origine est une fonction scalaire satisfaisant [26, 37] :

1. **Définie positive** : $V(x) > 0$ pour tout $x \neq 0$, et $V(0) = 0$
2. **Dérivée définie négative** : $\dot{V}(x)\frac{dV}{dt} < 0$ pour tout $x \neq 0$

Intuitivement, une fonction de Lyapunov joue le rôle d'une « fonction d'énergie » qui décroît strictement le long des trajectoires du système. Si une telle fonction existe, alors le point d'équilibre est asymptotiquement stable : toute trajectoire commençant suffisamment près de l'équilibre convergera vers celui-ci [26].

6.1.2. Le défi de trouver des fonctions de Lyapunov

Un problème majeur est qu'il n'existe pas de méthode générale analytique pour trouver des fonctions de Lyapunov pour des systèmes non linéaires arbitraires [1, 37]. Les chercheurs doivent souvent deviner des formes de fonctions candidates basées sur l'intuition physique et la structure mathématique du problème, un processus largement artisanal.

Pour les systèmes linéaires, des méthodes systématiques existent. Si le système est $\dot{x} = Ax$, on peut chercher une fonction de Lyapunov quadratique $V(x) = x^T Px$ en résolvant l'équation matricielle de Lyapunov [26] :

$$PA + A^T P = -Q \tag{1}$$

pour une matrice définie positive P, avec Q définie positive arbitraire. Mais pour les systèmes non linéaires, aucune recette similaire n'est disponible.

6.2. Construction automatique via fonctions propres

6.2.1. L'innovation de Moore et al.

L'approche de Moore et al. [1] fournit automatiquement des fonctions de Lyapunov neurales comme sous-produit de leur cadre d'apprentissage. L'idée clé est que les fonctions propres de Koopman associées à des valeurs propres à partie réelle négative décroissent naturellement le long des trajectoires du système.

En prenant la magnitude (norme) de telles fonctions propres, on obtient automatiquement une fonction candidate de Lyapunov [1] :

$$V(x) = |\phi_j(x)|$$

où ϕ_j est une fonction propre avec valeur propre λ_j satisfaisant $\mathrm{Re}(\lambda_j) < 0$ où ϕ_j est une fonction propre avec valeur propre λ_j satisfaisant $\mathrm{Re}(\lambda_j) < 0$ (partie réelle négative, indiquant une décroissance temporelle).

6.2.2. Vérification empirique

La condition de définie positive est satisfaite en prenant la magnitude. Pour vérifier la condition de dérivée négative, Moore et al. évaluent empiriquement la fonction candidate sur des trajectoires réelles du système [1].

Par exemple, pour le modèle de Hodgkin-Huxley avec son attracteur de cycle limite, les chercheurs ont :

1. Construit une fonction de Lyapunov neuronale à partir des fonctions propres apprises
2. Superposé des trajectoires réelles sur cette fonction
3. Vérifié que la fonction décroît effectivement le long des trajectoires, confirmant son statut de fonction de Lyapunov valide

6.3. Applications à l'analyse de stabilité

6.3.1. Caractérisation des bassins d'attraction

Les ensembles de niveau (level sets) d'une fonction de Lyapunov fournissent des informations cruciales sur la structure du bassin d'attraction [1, 37]. Chaque ensemble de niveau $\{x : V(x) = c\}$ pour une constante c est *forward-invariant* (invariant vers l'avant) : une trajectoire commençant sur cet ensemble ne peut jamais atteindre un niveau supérieur.

Cela permet de construire des estimations garanties du bassin d'attraction : tout point à l'intérieur d'un certain ensemble de niveau est garanti de converger vers l'attracteur [37, 26].

6.3.2. Exemple du pendule magnétique

Pour le système expérimental du pendule magnétique, Moore et al. ont utilisé leurs fonctions de Lyapunov neurales pour [1] :

— Identifier les multiples bassins d'attraction correspondant aux différents attracteurs
— Visualiser les frontières complexes entre ces bassins
— Prédire la stabilité et la robustesse de chaque attracteur face aux perturbations

Cette capacité d'analyse de stabilité automatisée est particulièrement précieuse pour des systèmes expérimentaux où les équations sous-jacentes sont inconnues ou très complexes [1, 2].

7. Réseaux de neurones récurrents et extension temporelle

7.1. Architecture des RNN et LSTM

Bien que l'approche principale de Moore et al. repose sur des autoencodeurs feedforward (à propagation avant), il est instructif de comprendre comment les réseaux de neurones récurrents (RNN) [1] et leurs variantes peuvent aussi traiter des données séquentielles temporelles [38, 39].

1. **RNN** (Recurrent Neural Network, réseau de neurones récurrent) : réseau de neurones avec boucles de rétroaction permettant de traiter des séquences temporelles en conservant une mémoire des entrées passées.

7.1.1. Principe des RNN

Un réseau de neurones récurrent diffère des réseaux feedforward par la présence de connexions en boucle, permettant à l'information de persister dans le temps [40, 41]. À chaque pas de temps t, un RNN reçoit l'entrée actuelle x_t et l'état caché précédent h_{t-1}, puis calcule :

$$h_t = f(W_x x_t + W_h h_{t-1} + b)$$

où W_x et W_h sont des matrices de poids, b est un biais, et f est une fonction d'activation.

Cette architecture permet au réseau de maintenir une « mémoire » des entrées passées, crucial pour traiter des séquences temporelles [40].

7.1.2. Le problème du gradient évanescent

Un défi majeur des RNN standard est le problème du gradient évanescent : lors de la rétropropagation à travers de nombreux pas de temps, les gradients deviennent exponentiellement petits, rendant difficile l'apprentissage de dépendances à long terme [39, 42].

7.1.3. LSTM : Long Short-Term Memory

Les réseaux LSTM (*Long Short-Term Memory*, soit « mémoire à court et long terme »), introduits par Hochreiter et Schmidhuber en 1997, résolvent ce problème en introduisant une architecture de cellule spécialisée avec plusieurs portes (gates) [39, 42] :

- **Porte d'oubli** : Décide quelles informations de l'état de cellule précédent doivent être oubliées
- **Porte d'entrée** : Détermine quelles nouvelles informations doivent être stockées
- **Porte de sortie** : Contrôle quelles informations de l'état de cellule doivent être utilisées pour la sortie

Ces portes permettent aux LSTM de maintenir des dépendances sur des milliers de pas de temps, les rendant idéaux pour des tâches comme la traduction automatique, la reconnaissance vocale et la prévision de séries temporelles [42, 40].

7.2. Comparaison avec l'approche Koopman

7.2.1. Philosophies complémentaires

Les RNN/LSTM et l'approche Koopman représentent deux philosophies différentes pour modéliser des dynamiques temporelles :

- **RNN/LSTM** : Apprennent des représentations non linéaires récursives de l'historique temporel, capturant des dépendances complexes de manière implicite dans les poids du réseau.

— **Approche Koopman** : Cherchent explicitement à linéariser les dynamiques en trouvant les bonnes coordonnées (fonctions propres), fournissant une structure mathématique interprétable.

7.2.2. Avantages de l'approche Koopman

L'approche de Moore et al. offre plusieurs avantages distinctifs [1, 2] :

1. **Interprétabilité** : Les fonctions propres et valeurs propres ont une signification mathématique claire, révélant les modes oscillatoires, les échelles de temps, et les structures invariantes.
2. **Efficacité de prédiction** : Une fois l'espace latent linéaire appris, les prédictions à long terme sont simples et efficaces en temps de calcul, contrairement aux RNN qui doivent itérer séquentiellement.
3. **Analyse de stabilité** : La structure linéaire facilite l'analyse spectrale et la construction de fonctions de Lyapunov.
4. **Généralisation** : La contrainte de linéarité dans l'espace latent peut améliorer la généralisation en évitant le sur-apprentissage de détails arbitraires.

8. Résultats et implications

8.1. Validation expérimentale

8.1.1. Systèmes étudiés

Moore et al. ont validé leur approche sur un éventail diversifié de systèmes dynamiques [1], incluant :

— **Pendule amorti** : Système mécanique simple avec dissipation d'énergie
— **Oscillateur de Duffing** : Système non linéaire exhibant des bifurcations et du chaos
— **Modèle de Hodgkin-Huxley** : Dynamique neuronale avec oscillations auto-entretenues
— **Système de Lorenz-96** : Modèle atmosphérique haute-dimension
— **Pendule magnétique expérimental** : Données réelles d'un système physique avec bassins d'attraction complexes
— **Système masse-ressort-amortisseur magnétique** : Autre système expérimental avec non-linéarités

Cette diversité démontre la généralité et la robustesse de l'approche à travers différents domaines scientifiques.

8.1.2. Qualité des prédictions

Les résultats montrent que les modèles appris peuvent prédire avec précision l'évolution future sur de longs horizons temporels, capturant fidèlement [1] :

- Les trajectoires dans l'espace d'état original
- Les oscillations intra-puits et inter-puits pour les systèmes multi-stables
- Les bonnes convergences vers les attracteurs corrects
- Les structures fractales des bassins d'attraction

De plus, les prédictions dans l'espace latent sont très peu coûteuses en calcul, car elles ne requièrent que des multiplications matricielles simples plutôt que l'intégration numérique coûteuse d'équations différentielles non linéaires.

8.2. Découverte scientifique automatisée

8.2.1. Vers des « machines scientifiques»

Ce travail s'inscrit dans un objectif à long terme du General Robotics Lab de Boyuan Chen : développer des « machine scientists » (machines scientifiques) qui assistent à la découverte scientifique automatisée [2]. En reliant l'IA moderne au langage mathématique des systèmes dynamiques, cette recherche pointe vers un futur où l'IA fait plus que reconnaître des patterns—elle pourrait aider à découvrir les règles fondamentales façonnant à la fois le monde physique et les systèmes vivants.

8.2.2. Design expérimental actif

L'équipe explore comment le cadre pourrait guider le design expérimental en sélectionnant activement quelles données collecter pour révéler plus efficacement la structure d'un système [2]. Plutôt que de collecter passivement des données, un système d'IA pourrait proposer des expériences ciblées maximisant l'information gagnée, accélérant considérablement le processus de découverte.

8.2.3. Extension à des modalités de données riches

Des travaux futurs visent à appliquer la méthode à des formes plus riches de données, incluant la vidéo, l'audio et les signaux de systèmes biologiques complexes [2]. Par exemple, analyser directement des enregistrements vidéo de phénomènes physiques pour en extraire automatiquement les lois dynamiques sous-jacentes représenterait une avancée majeure.

8.3. Applications potentielles

8.3.1. Ingénierie et contrôle

La capacité de réduire des systèmes complexes à des modèles linéaires de dimension réduite facilite grandement leur contrôle. Les techniques de

contrôle linéaire, beaucoup mieux développées que leurs équivalents non linéaires, deviennent applicables une fois qu'on dispose d'une représentation linéaire dans les coordonnées de Koopman [6, 2].

8.3.2. Prévision et surveillance

Pour des systèmes comme les circuits électriques, les processus industriels, ou les signaux physiologiques, identifier les signes précurseurs d'instabilité est crucial. Les fonctions de Lyapunov neurales et l'analyse spectrale peuvent fournir des indicateurs d'alerte précoce lorsqu'un système s'approche d'un point de basculement (tipping point) [2, 43].

8.3.3. Science du climat et écologie

Les systèmes climatiques et écologiques sont notoirement complexes, avec de nombreuses variables interdépendantes et des comportements non linéaires. Appliquer ces techniques pourrait aider à identifier les modes dominants de variabilité, améliorer les prévisions, et comprendre les mécanismes de transitions abruptes [1].

9. Limitations et perspectives

9.1. Défis actuels

9.1.1. Exigences en données

Bien que l'approche soit data-driven, elle nécessite des quantités substantielles de données de haute qualité couvrant l'espace d'état du système. Pour des systèmes expérimentaux complexes, obtenir de telles données peut être coûteux ou difficile [1].

9.1.2. Interprétabilité des fonctions propres

Tandis que les valeurs propres ont une interprétation claire (échelles de temps), les fonctions propres apprises par les réseaux de neurones peuvent être difficiles à interpréter physiquement. Relier ces fonctions à des quantités physiques significatives reste un défi [1].

9.1.3. Systèmes hautement chaotiques

Pour des systèmes exhibant un chaos fort avec des exposants de Lyapunov très positifs, les prédictions à long terme demeurent fondamentalement limitées par la sensibilité exponentielle aux conditions initiales, quelle que soit la qualité du modèle [4].

9.2. Directions futures

9.2.1. Intégration de connaissances physiques

Combiner l'apprentissage data-driven avec des contraintes physiques connues (conservation de l'énergie, symétries, etc.) pourrait améliorer la performance et la généralisabilité tout en réduisant les besoins en données [34].

9.2.2. Systèmes stochastiques

Étendre le cadre pour gérer des dynamiques intrinsèquement stochastiques plutôt que purement déterministes élargirait considérablement le champ d'application, notamment pour les systèmes biologiques et financiers [37].

9.2.3. Apprentissage continuel

Développer des systèmes capables de mettre à jour continuellement leurs modèles à mesure que de nouvelles données arrivent, s'adaptant à des changements de régime ou de paramètres, serait précieux pour des applications en temps réel [2].

10. Conclusion

10.1. Synthèse des contributions

Cet article a présenté une synthèse accessible d'une avancée majeure à l'intersection de l'intelligence artificielle et de la théorie des systèmes dynamiques. Nous avons progressivement introduit les concepts fondamentaux de l'apprentissage automatique—réseaux de neurones, rétropropagation, autoencodeurs—avant d'explorer leur application à la découverte automatisée de lois régissant des systèmes dynamiques complexes.

L'approche de Moore, Mann et Chen [1] combine élégamment la puissance des réseaux de neurones profonds avec la rigueur mathématique de la théorie de l'opérateur de Koopman. En apprenant automatiquement des fonctions propres qui linéarisent des dynamiques non linéaires, leur système peut :

- Réduire drastiquement la dimensionnalité de systèmes complexes,
- Identifier automatiquement attracteurs et bassins d'attraction,
- Fournir des fonctions de Lyapunov neurales pour l'analyse de stabilité,
- Générer des prédictions précises et computationnellement efficaces

10.2. Impact scientifique et philosophique

Cette recherche illustre une tendance plus large dans la science contemporaine : le passage d'approches purement déductives (partir de principes fondamentaux) vers des approches hybrides combinant déduction et induction data-driven. L'IA ne remplace pas la compréhension théorique, mais

l'augmente, permettant aux scientifiques de s'attaquer à des systèmes auparavant trop complexes pour une analyse traditionnelle.

Le fait que ces découvertes soient exprimées dans le langage mathématique standard de la théorie des systèmes dynamiques—valeurs propres, fonctions propres, fonctions de Lyapunov—est crucial. L'IA ne produit pas simplement des boîtes noires prédictives, mais des structures mathématiques interprétables que les scientifiques peuvent analyser, valider et utiliser pour développer une intuition plus profonde [2].

10.3. Vision future

Alors que nous progressons dans le 21ème siècle, l'intersection de l'intelligence artificielle et des sciences fondamentales promet de transformer notre capacité à comprendre et contrôler des systèmes complexes. Des prévisions climatiques aux thérapies personnalisées, de la conception de matériaux à la robotique adaptative, les outils développés dans ce domaine auront des répercussions vastes et profondes.

La vision de « scientifiques machines » collaborant avec des chercheurs humains, chacun apportant ses forces complémentaires—créativité et intuition humaines, rigueur computationnelle et capacité de traitement de données massives de l'IA—pourrait accélérer considérablement le rythme de la découverte scientifique [2].

Comme l'a montré le travail de Moore et al., nous sommes à l'aube d'une nouvelle ère où les mystères les plus profonds de la nature pourraient être dévoilés non pas malgré leur complexité, mais à travers elle, grâce aux outils puissants de l'intelligence artificielle moderne combinés à la sagesse intemporelle des mathématiques.

Remerciements

Cette synthèse s'appuie sur l'article « Automated global analysis of experimental dynamics through low-dimensional linear embeddings » de Samuel A. Moore, Brian P. Mann et Boyuan Chen, publié dans *npj Complexity* en décembre 2025, ainsi que sur de nombreuses ressources pédagogiques sur l'apprentissage automatique et les systèmes dynamiques.

Glossaire

Apprentissage automatique (*Machine Learning*) : Branche de l'intelligence artificielle permettant aux algorithmes d'apprendre à partir de données sans être explicitement programmés pour chaque tâche spécifique. Les trois paradigmes principaux sont l'apprentissage supervisé, non supervisé et par renforcement.

Apprentissage profond (*Deep Learning*) : Sous-domaine de l'apprentissage automatique utilisant des réseaux de neurones comportant plusieurs couches cachées. La profondeur du réseau permet d'apprendre des représentations hiérarchiques de plus en plus abstraites des données.

Attracteur (*Attractor*) : Ensemble vers lequel un système dynamique tend à évoluer, indépendamment (dans certaines limites) de ses conditions initiales. Les attracteurs peuvent être des points fixes, des cycles limites, des tores quasi-périodiques ou des attracteurs chaotiques.

Autoencodeur (*Autoencoder*) : Architecture de réseau de neurones entraînée de manière non supervisée pour reconstruire ses propres entrées. En forçant l'information à passer par un goulot d'étranglement de dimension réduite, l'autoencodeur apprend les caractéristiques les plus saillantes des données.

Bassin d'attraction (*Basin of Attraction*) : Ensemble de toutes les conditions initiales qui conduisent un système dynamique vers un attracteur donné à long terme. Les frontières entre bassins peuvent avoir des structures géométriques complexes, voire fractales.

Chaos (*Chaos*) : Comportement d'un système dynamique déterministe caractérisé par une sensibilité extrême aux conditions initiales. Deux états initiaux arbitrairement proches peuvent diverger exponentiellement au cours du temps, rendant les prédictions à long terme pratiquement impossibles malgré le caractère déterministe des équations.

Cycle limite (*Limit Cycle*) : Trajectoire fermée isolée dans l'espace des phases vers laquelle les trajectoires voisines convergent (cycle limite stable) ou divergent (cycle limite instable). Les cycles limites représentent des oscillations auto-entretenues, comme les battements cardiaques ou les oscillations neuronales du modèle de Hodgkin-Huxley.

Descente de gradient (*Gradient Descent*) : Algorithme d'optimisation itératif qui ajuste les paramètres d'un modèle dans la direction opposée au gradient de la fonction de perte. Cette méthode permet de minimiser progressivement l'erreur du modèle lors de l'entraînement.

DMD – Décomposition en modes dynamiques (*Dynamic Mode Decomposition*) : Algorithme numérique permettant d'extraire des modes cohérents à partir de séquences temporelles de données, chacun associé à une fréquence d'oscillation et un taux de croissance ou décroissance. La DMD fournit une approximation de dimension finie de l'opérateur de Koopman.

Encodeur / Décodeur (*Encoder / Decoder*) : Composantes d'un autoencodeur. L'**encodeur** compresse les données d'entrée de haute dimension vers une représentation latente de dimension réduite. Le **décodeur** effectue l'opération inverse, reconstruisant les données originales à partir de cette représentation compressée.

Espace des phases (*Phase Space, State Space*) : Espace mathématique abstrait dont chaque point représente un état possible du système dynamique. Les coordonnées de cet espace correspondent aux variables d'état du système. L'évolution temporelle se traduit par une trajectoire dans cet espace.

Espace latent (*Latent Space*) : Représentation compressée de dimension réduite apprise par un autoencodeur, également appelée goulot d'étranglement (*bottleneck*). Cet espace capture les caractéristiques essentielles des données en éliminant le bruit et les redondances.

Fonction d'activation (*Activation Function*) : Fonction non linéaire appliquée à la sortie d'un neurone artificiel. Elle introduit la non-linéarité nécessaire pour que le réseau puisse apprendre des relations complexes. Les fonctions courantes incluent la sigmoïde, la tangente hyperbolique (tanh) et ReLU (*Rectified Linear Unit*).

Fonction de Lyapunov (*Lyapunov Function*) : Fonction scalaire permettant de prouver la stabilité d'un système dynamique. Si une telle fonction existe, définie positive et à dérivée temporelle négative, le point d'équilibre est asymptotiquement stable. Elle joue intuitivement le rôle d'une fonction d'énergie décroissante.

Fonction de perte (*Loss Function, Cost Function*) : Fonction mathématique quantifiant l'écart entre les prédictions d'un modèle et les valeurs réelles attendues. L'objectif de l'entraînement est de minimiser cette fonction. L'erreur quadratique moyenne (MSE) est un exemple courant pour les problèmes de régression.

Fonction propre (*Eigenfunction*) : Fonction qui, sous l'action d'un opérateur linéaire, est simplement multipliée par un scalaire appelé valeur propre. Dans le contexte de l'opérateur de Koopman, les fonctions propres définissent des coordonnées naturelles où les dynamiques non linéaires deviennent linéaires.

LSTM (*Long Short-Term Memory*) : Architecture de réseau de neurones récurrent conçue pour capturer les dépendances à long terme dans les séquences temporelles. Les cellules LSTM utilisent des mécanismes de portes (oubli, entrée, sortie) pour contrôler le flux d'information, résolvant ainsi le problème du gradient évanescent des RNN classiques.

Observable (*Observable*) : Dans le contexte de la théorie de Koopman, fonction scalaire $g(x)$ de l'état x d'un système dynamique. Plutôt que d'étudier directement l'évolution de l'état, l'approche de Koopman étudie l'évolution des observables, transformant ainsi un problème non linéaire en un problème linéaire.

Opérateur de Koopman (*Koopman Operator*) : Opérateur linéaire de dimension infinie décrivant l'évolution temporelle des observables d'un système dynamique. Bien que le système sous-jacent soit non linéaire, l'opérateur de Koopman est toujours linéaire, ce qui permet d'appliquer des outils d'analyse spectrale puissants.

Point fixe (*Fixed Point, Equilibrium*) : État d'un système dynamique qui reste inchangé au cours du temps. Un point fixe peut être stable (les trajectoires voisines y convergent) ou instable (les trajectoires voisines s'en éloignent). L'analyse de stabilité des points fixes est fondamentale en théorie des systèmes dynamiques.

Rétropropagation (*Backpropagation*) : Algorithme calculant efficacement les gradients nécessaires à l'entraînement des réseaux de neurones multicouches. Basée sur la règle de dérivation en chaîne, la rétropropagation propage l'erreur de la sortie vers l'entrée, permettant d'ajuster chaque poids selon sa contribution à l'erreur totale.

Taux d'apprentissage (*Learning Rate*) : Hyperparamètre α contrôlant l'amplitude des ajustements des poids lors de la descente de gradient. Un taux trop élevé peut empêcher la convergence ; un taux trop faible ralentit l'apprentissage. Le choix de ce paramètre est crucial pour l'efficacité de l'entraînement.

Valeur propre (*Eigenvalue*) : Scalaire λ associé à une fonction propre (ou vecteur propre) caractérisant le facteur de mise à l'échelle sous l'action d'un opérateur linéaire. Dans les systèmes dynamiques, les valeurs propres révèlent les échelles de temps caractéristiques et déterminent la stabilité du système.

Références

[1] S. A. Moore, B. P. Mann, B. Chen, Automated global analysis of experimental dynamics through low-dimensional linear embeddings, npj Complexity 2 (1), publication en ligne : 17 décembre 2025 (2025). doi:10.1038/s44260-025-00062-y. (Cité pages 1, 2, 4, 6, 7, 8, 9, 10, 12, 13, 14, 15, 16, 17, 19, 20, 21 et 22.)

[2] Duke University, This ai finds simple rules where humans see only chaos, ScienceDaily (12 2025).
URL https://www.sciencedaily.com/releases/2025/12/251221091237.htm (Cité pages 1, 2, 15, 17, 19, 20, 21, 22 et 23.)

[3] F. Dablander, A gentle introduction to dynamical systems theory, Blog personnel (12 2020).
URL https://fabiandablander.com/r/Dynamical-Systems.html (Cité pages 2, 8 et 9.)

[4] Wikipedia, Dynamical systems theory, Wikipedia (2004).
URL https://en.wikipedia.org/wiki/Dynamical_systems_theory (Cité pages 2, 8, 9 et 21.)

[5] S. Harley, A. Zinin, Ai learns to build simple equations for complex systems, Phys.org (12 2025).
URL https://phys.org/news/2025-12-ai-simple-equations-complex.html (Cité page 2.)

[6] S. L. Brunton, M. Budišić, E. Kaiser, J. N. Kutz, Modern koopman theory for dynamical systems, SIAM Review (2022).
URL https://www.lri.fr/~gcharpia/deeppractice/2022/chap_5_biblio/Koopman/Modern_koopman_theory.pdf (Cité pages 2, 3, 10, 11, 12, 14 et 21.)

[7] IBM, Supervised vs unsupervised learning: What's the difference?, IBM Think Topics (2024).
URL https://www.ibm.com/think/topics/supervised-vs-unsupervised-learning (Cité pages 3 et 4.)

[8] Google Developers, What is machine learning?, Google Machine Learning (2025).
URL https://developers.google.com/machine-learning/intro-to-ml/what-is-ml (Cité pages 3 et 4.)

[9] GeeksforGeeks, Supervised and unsupervised learning, GeeksforGeeks (9 2017).
URL https://www.geeksforgeeks.org/machine-learning/supervised-unsupervised-learning/ (Cité pages 3 et 4.)

[10] Pecan AI, Three types of machine learning you should know, Pecan AI Blog (2025).
URL https://www.pecan.ai/blog/3-types-of-machine-learning/ (Cité pages 3 et 4.)

[11] Google Developers, Neural networks: Training using backpropagation, Google Machine Learning Crash Course (2025).
URL https://developers.google.com/machine-learning/crash-course/neural-networks/backpropagation (Cité pages 4, 5 et 6.)

[12] GeeksforGeeks, Backpropagation in neural network, GeeksforGeeks (3 2024).
URL https://www.geeksforgeeks.org/machine-learning/backpropagation-in-neural-network/ (Cité pages 4, 5 et 6.)

[13] Built In, How does backpropagation in a neural network work?, Built In (2024).
URL https://builtin.com/machine-learning/backpropagation-neural-network (Cité pages 5 et 6.)

[14] Wikipedia, Autoencoder, Wikipedia (2006).
URL https://en.wikipedia.org/wiki/Autoencoder (Cité pages 5, 6 et 7.)

[15] DataCamp, Loss functions in machine learning explained, DataCamp Tutorial (12 2024).
URL https://www.datacamp.com/tutorial/loss-function-in-machine-learning (Cité page 5.)

[16] CodeSignal, Mean squared error loss, CodeSignal Learn (2024).
URL https://codesignal.com/learn/courses/training-neural-networks-the-backpropagation-algorithm-1/lessons/mean-squared-error-loss (Cité page 5.)

[17] Built In, Mean squared error (mse) vs. mean squared logarithmic error (msle), Built In (7 2025).
URL https://builtin.com/data-science/msle-vs-mse (Cité page 5.)

[18] Wikipedia, Mean squared error, Wikipedia (2003).
URL https://en.wikipedia.org/wiki/Mean_squared_error (Cité page 5.)

[19] M. Mazur, A step by step backpropagation example, Blog personnel (3 2015).
URL https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/ (Cité page 6.)

[20] Wikipedia, Backpropagation, Wikipedia, l'encyclopédie libre (2005).
URL https://en.wikipedia.org/wiki/Backpropagation (Cité page 6.)

[21] V7 Labs, Autoencoders in deep learning: Tutorial & use cases [2024], V7 Labs Blog (12 2025).
URL https://www.v7labs.com/blog/autoencoders-guide (Cité pages 6 et 7.)

[22] IBM, What is an autoencoder?, IBM Think Topics (2023).
URL https://www.ibm.com/think/topics/autoencoder (Cité page 6.)

[23] DataCamp, Introduction to autoencoders: From the basics, DataCamp Tutorial (12 2023).
URL https://www.datacamp.com/tutorial/introduction-to-autoencoders (Cité pages 6 et 7.)

[24] GeeksforGeeks, Autoencoders in machine learning, GeeksforGeeks (2019).
URL https://www.geeksforgeeks.org/machine-learning/auto-encoders/ (Cité page 7.)

[25] M. Brin, G. Stuck, Introduction to Dynamical Systems, Cambridge University Press, 2002. (Cité page 8.)

[26] MIT, Ch. 9 - lyapunov analysis, Underactuated Robotics (2023).
URL http://underactuated.mit.edu/lyapunov.html (Cité pages 9, 15, 16 et 17.)

[27] Wikipedia, Attractor, Wikipedia (2003).
URL https://en.wikipedia.org/wiki/Attractor (Cité page 9.)

[28] arXiv, The basins zoo (4 2025). arXiv:2504.01580. (Cité page 9.)

[29] Wikipedia, Eigenvalues and eigenvectors, Wikipedia (2005).
URL https://en.wikipedia.org/wiki/Eigenvalues_and_eigenvectors (Cité page 10.)

[30] Wikipedia, Eigenfunction, Wikipedia (2003).
URL https://en.wikipedia.org/wiki/Eigenfunction (Cité page 10.)

[31] Lamar University, Differential equations - eigenvalues and eigenfunctions, Paul's Online Math Notes (2022).
URL https://tutorial.math.lamar.edu/classes/de/bvpevals.aspx (Cité page 10.)

[32] UK Fluids Network, Notes on koopman operator theory, UK Fluids Network.
URL https://fluids.ac.uk/files/meetings/KoopmanNotes.1575558616.pdf (Cité pages 11 et 12.)

[33] S. L. Brunton, M. Budišić, E. Kaiser, J. N. Kutz, Modern koopman theory for dynamical systems (2021). arXiv:2102.12086. (Cité page 11.)

[34] M. J. Colbrook, The multiverse of dynamic mode decomposition algorithms, DAMTP, University of Cambridge.
URL http://www.damtp.cam.ac.uk/user/mjc249/pdfs/DMD_multiverse_colbrook.pdf (Cité pages 12, 13 et 22.)

[35] Wikipedia, Dynamic mode decomposition, Wikipedia (2008).
URL https://en.wikipedia.org/wiki/Dynamic_mode_decomposition (Cité pages 12 et 13.)

[36] S. L. Brunton, et al., Dynamic mode decomposition and its variants, Annual Reviews (1 2022).
URL https://www.annualreviews.org/content/journals/10.1146/annurev-fluid-030121-015835 (Cité page 12.)

[37] Emergent Mind, Lyapunov-based stability analysis, Emergent Mind Topics (10 2025).
URL https://www.emergentmind.com/topics/lyapunov-based-stability-analysis (Cité pages 15, 16, 17 et 22.)

[38] C. Olah, Understanding lstm networks, Blog personnel (8 2015).
URL https://colah.github.io/posts/2015-08-Understanding-LSTMs/ (Cité page 17.)

[39] Wikipedia, Long short-term memory, Wikipedia (2007).
URL https://en.wikipedia.org/wiki/Long_short-term_memory
(Cité pages 17 et 18.)

[40] Pathmind, A beginner's guide to lstms and recurrent neural networks, Pathmind Wiki (2015).
URL http://wiki.pathmind.com/lstm (Cité page 18.)

[41] IBM, What is a recurrent neural network (rnn)?, IBM Think Topics (2021).
URL https://www.ibm.com/think/topics/recurrent-neural-networks (Cité page 18.)

[42] GeeksforGeeks, What is lstm - long short term memory?, GeeksforGeeks (2019).
URL https://www.geeksforgeeks.org/deep-learning/deep-learning-introduction-to-long-short-term-memory/ (Cité page 18.)

[43] JuliaDynamics, Attractor basins, tipping points, DynamicalSystems.jl Documentation (2021).
URL https://juliadynamics.github.io/DynamicalSystems.jl/previews/PR156/chaos/basins/ (Cité page 21.)