# Apprentissage statistique et équations différentielles stochastiques

Nathan Doumèche

Mémoire sous la direction de Gérard Biau

15 juin 2022


## Table des matières

# 1 Introduction

## 1.1 Motivations

Les méthodes d'**intelligence artificielle** (IA) regroupent l'ensemble des techniques s'inspirant de l'intelligence humaine pour effectuer des tâches. Le domaine de l'intelligence artificielle en tant qu'objet de recherche autonome naît dans les années 1950 avec notamment les Conférences de Dartmouth. Néanmoins, l'échec à obtenir des résultats substantiels donna lieu dans les années 1980 à un désintérêt progressif prenant le nom d'*AI winter*. Il faudra alors attendre les années 2010 et l'amélioration de la puissance de calcul des ordinateurs pour que l'IA redevienne un domaine actif avec en 2012 le gain de la compétition de reconnaissance d'images *ImageNet* par un algortihme d'IA et en 2016 la performance d'AlphaGo contre le champion du monde de Go. Les algorithmes d'IA ont aujourd'hui prouvé leur efficacité pour la réalisation de nombreuses tâches : reconnaissance de parole, classification d'images, prévision de séries temporelles, modélisation physique, génération de signaux... En conséquence, ils défient les performances humaines dans divers cas pratiques : diagnostic précoce de cancers, voiture autonome, prévisions boursières, traduction automatique... Cela étant, le fonctionnement précis de ces algorithmes est aujourd'hui encore mal connu. Mieux modéliser ces algorithmes permettrait premièrement d'améliorer leurs **performances** et deuxièmement de leur accorder plus de **confiance** dans les décisions qu'ils prennent.

## 1.2 Le cadre de l'apprentissage supervisé

Parmi les techniques d'intelligence artificielle, nous nous bornerons ici au cadre de l'**apprentissage supervisé**. C'est une méthode de création d'algorithmes destinés à apprendre une certaine tâche à partir de données d'entraînement sur lesquelles le résultat de la tâche est connu [Mohri et al., 2018].

Considérons un exemple simple. Supposons que l'on veuille définir un algorithme qui apprenne la relation entre la taille $X$ et le poids $Y$ d'une personne. La relation physique, que l'on veut faire apprendre à l'algorithme, est $Y = X^3$. Dans la pratique, cette relation n'est pas parfaitement suivie et on dispose de $n$ données indépendantes $\mathcal{D}_n = (X_i, Y_i)_{1 \leq i \leq n}$ vérifiant la relation $Y_i = f(X_i) + \epsilon_i$ avec $(\epsilon_i)_{1 \leq i \leq n}$ une famille i.i.d. de bruits, et $f(x) = x^3$. On appelle $X$ les **features** et $Y$ les **labels**. $f$ est la fonction que l'on cherche à faire apprendre à l'algorithme supervisé.

Pour cela, on définit premièrement une **architecture** pour les fonctions que l'on va considérer comme des bons candidats. Ici, on considère par exemple un ensemble de fonctions indexées par $\lambda \in \mathbb{R} : g(\lambda)(x) = x^\lambda$. Le problème devient alors paramétrique et il suffit de déterminer le paramètre $\lambda^*$ optimal. L'algorithme d'apprentissage supervisé devra donc apprendre $\lambda^* = 3$, car dans ce cas $g(\lambda^*) = f$.

Pour que l'algorithme puisse trouver le $\lambda$ optimal, nous définissons une **fonction de perte** $\mathcal{L}$ qui sert à mesurer l'écart de performances entre les fonctions candidates $g(\lambda)$ et la fonction réelle que l'on cherche à apprendre $f$. Par exemple, considérons $\mathcal{L}(u) = u^2$. Le paradigme fondamental de l'apprentissage supervisé est le suivant : la fonction $g(\lambda^*)$ qui minimise **l'erreur empirique** $\mathcal{L}_n(\lambda) = \sum_{i=1}^n \mathcal{L}\left(y_i,\ g(\lambda)(x_i)\right)$ est un bon candidat pour approximer $f$. On cherche donc à calculer $\lambda^* = \underset{\lambda}{\operatorname{argmin}}\ \mathcal{L}_n$ avec l'idée que **minimiser l'erreur empirique** fournit un bon estimateur du minimiseur du risque global. La théorie de Vapnik-Chervonenkis permet de démontrer cette heuristique dans certains cas [Vershynin, 2020].

Pour minimiser l'erreur empirique, la plupart des algorithmes utilise une **descente de gradient**. En partant d'une valeur arbitraire $\lambda = \lambda_0$, ils calculent le gradient de l'erreur empirique $\mathcal{L}_n$ en le paramètre $\lambda$. Puis, ils suivent le gradient pour faire baisser l'erreur empirique : $\lambda \mapsto \lambda - \nabla_\lambda \mathcal{L}_n$. Les seuls points fixes de cette méthode sont les minima locaux de l'erreur empirique. Dans la pratique, des techniques plus sophistiquées de descente de gradient sont utilisées, notamment la **descente de gradient stochastique**. Elle consiste en la procédure $\lambda \mapsto \lambda - \gamma \nabla_\lambda \mathcal{L}_E$ où $E$ est un sous-ensemble de $\{1, \ldots, n\}$ tiré alétoirement, $\mathcal{L}_E = \sum_{i \in E} \mathcal{L}\left(y_i,\ g(\lambda)(x_i)\right)$ et $\gamma$ est le **learning rate**. Le learning rate est un exemple d'**hyperparamètre**, c'est-à-dire de paramètre qui ne varie pas pendant la descente de gradient et qu'il faut donc régler auparavant.

Un problème immédiat intervenant dans le choix d'architecture est celui du **surapprentissage**

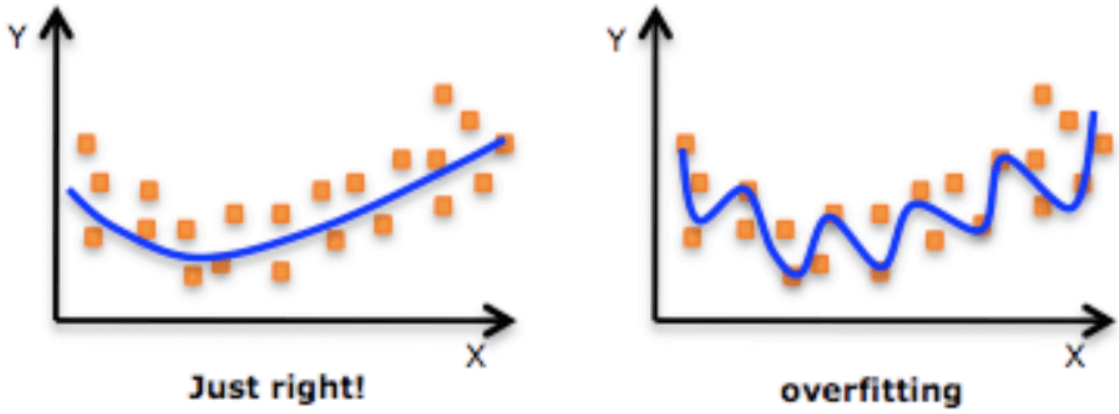


FIGURE 1 – Surapprentissage

(overfitting en anglais) : l'algorithme ne doit pas apprendre le bruit. En effet, si l'architecture considérée contient trop de fonctions, il devient possible à l'algorithme d'obtenir une erreur nulle $\mathcal{L}_n(\lambda^*) = 0$. Cela étant, cela ne correspond pas à une réalité physique et lorsqu'on donnera à l'algorithme de nouvelles données, $X_{new}$, il y a de fortes chances pour que la prédiction $Y_{new}$ qu'il réalise soit de mauvaise qualité.

## 1.3 Aspect stochastique des réseaux de neurones

De nombreuses architectures populaires d'algorithmes d'apprentissage supervisé se regroupent sous la notion de réseau de neurones.

**Définition 1.1** (Réseau de neurones)**.** *Un réseau de neurones est une architecture s'inspirant de l'organisation du cerveau humain en entités individuelles appelées neurones qui communiquent entre elles par des connexions [Shalev-Shwartz and Ben-David, 2014].*

**Exemple 1.2** (MLP)**.** *Le réseau de neurones le plus simple est le* ***multilayer perceptron****. Supposons que l'on veuille créer un algorithme capable de lire des chiffres à partir des images des chiffres. En entrée (input), l'algorithme prend donc l'ensemble des pixels de l'image du chiffre et la sortie (output) correspond à la probabilité que l'image corresponde au chiffre* 0*. La fonction globale correspondant à l'algorithme est la composition d'une succession de fonctions élémentaires paramétriques représentées par des flèches sur la Figure 2. Les résultats intermédiaires représentés par des cercles sont appelés les neurones. Chaque alignement de neurones est appelé* ***couche*** *(layer). Un choix habituel de fonction élémentaire paramétrique est la composition d'une fonction non-linéaire* $s$ *appelée* ***fonction d'activation*** *et d'une opération affine. Chaque flèche* $i$ *correspond alors à la fonction* $g_i(x) = A_i x + b_i$ *et l'arrivée sur un neurone caché correspond à l'application d'une fonction d'activation* $s$*. La fonction relu :* $x \mapsto x\mathbf{1}_{x\geq 0}$ *est un choix courrant de fonction d'activation.*

**Remarque 1.3** (Propriétés des réseaux de neurones)**.** *À ce jour, les propriétés des réseaux de neurones sont encore peu connues, d'autant plus que de nouveaux réseaux de neurones sont régulièrement découverts. Aussi la plupart des critères du cadre de l'apprentissage supervisé sont des domaines de recherche actif : choix d'une architecture pour un certain type de tâche, vitesse de convergence en fonction du nombre de données, réglage des hyperparamètres, robustesse de l'output au bruit dans les données, effet de la descente de gradient sur l'algorithme... Nous verrons comment la modélisation des réseaux de neurones comme des équations différentielles stochastiques permet de répondre partiellement à ces problèmes.*

**Exemple 1.4** (Drop-out)**.** *Pour éviter le surapprentissage, une des méthodes est de s'assurer de la diffusion de la décision dans tout le réseau de neurones. Par exemple, il s'agit d'éviter que le résultat de l'algorithme ne dépende exclusivement d'un neurone. Pour cela, une technique appelée* ***drop-out*** *consiste à mettre aléatoirement certains des paramètres à* 0 *pendant la descente de gradient.*

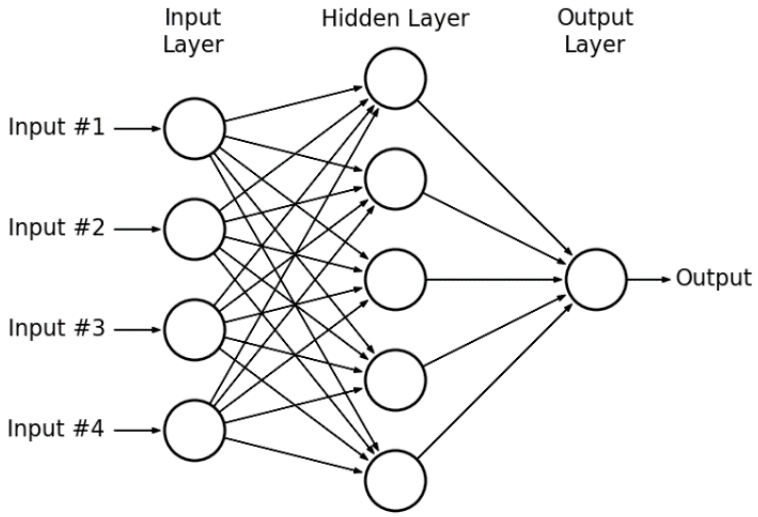


FIGURE 2 – Multilayer perceptron

**Remarque 1.5.** *De nombreuses sources d'aléas motive une modélisation stochastique du fonctionnement des réseaux de neurones :*

- *les données et les labels $(X, Y)$ suivent des lois de probabilités,*
- *l'initialisation des paramètres $\lambda_0$ est souvent stochastique,*
- *certaines méthodes de régularisation comme le drop-out sont par essence stochastiques,*
- *la descente de gradient stochastique.*

## 1.4 Équations différentielles stochastiques

Nous ferons plus tard le lien entre réseaux de neurones et équations différentielles. De surcroît, leur caractère stochastique amène à considérer des équations différentielles qui sont elles-mêmes des processus stochastiques : les équations différentielles stochastiques.

Tout comme l'IA, le calcul stochastique est une branche très récente de l'histoire des mathématiques. Bien que la notion de mouvement brownien émerge au XIX$^{ième}$ siècle pour décrire le mouvement désordonné de grains de pollen sur un liquide, il faudra attendre la construction probabiliste de Kolmogorov du début du XX$^{ième}$ siècle pour que le calcul stochastique prenne naissance, avec notamment la construction de l'intégrale d'Itô en 1951.

La notion d'équation différentielle stochastique (**EDS**) permet de généraliser la notion d'équation différentielle. Elle permet de considérer par exemple des équations différentielles qui dépendent de paramètres aléatoires. Par exemple, il est possible de considérer l'EDS $dY_t = XY_t dt$ où $X \sim \mathcal{N}(\mu, \sigma^2)$ qui a évidemment pour solution le proessus $Y_t = Y_0 \exp(Xt)$.

Il devient également possible de considérer des EDS avec des dérives stochastiques. Par exemple, si $X$ est un processus à temps continu à trajectoires à variation finie (par exemple dérivables), il est également possible de de considérer l'équation différentielle $dY_t = Y_t dX_t$. Sa solution est alors $Y_t = Y_0 \exp(X_t - X_0)$.

La construction de l'intégrale stochastique et des équations différentielle stochastique est technique mais peut être consultée dans Rogers and Williams [1994]. L'**équation différentielle stochastique** $dY_t = f(Y_t)dX_t$, où $X$ est un processus, correspond en réalité à l'équation $Y_t - Y_0 = \int_0^t f(Y_u)dX_u$. Elle ne dépend donc que de la définition de l'intégrale contre le processus $X$. Or, il existe plusieurs manières d'intégrer contre des processus stochastiques. Nous allons en présenter deux : les intégrales d'Itô et de Stratonovich. Elle permettent toutes deux d'intégrer contre une classe de processus appelés **semi-martingales continues** et qui regroupe les sommes de processus dont les trajectoires sont à variations finies et des martingales locales qui sont les limites de martingales

à temps continu (comme le mouvement brownien). L'**intégrale d'Itô** du processus $Y$ contre le processus $X$, notée $\int_0^t Y_u dX_u$ est une généralisation de l'intégrale de Riemann. En effet, elle résulte de la limite de la méthode des rectangles. C'est donc un objet-limite naturel à considérer.

**Proposition 1.6** (Méthode des rectangles)**.** *Si $X$ est une semimartingale continue et si $H$ est à trajectoires continues, alors pour toute suite de subdivisions emboîtées $0 = t_0^n < t_1^n < \cdots < t_{p_n}^n = t$ de pas tendant vers 0,*

$$\sum_{i=0}^{p_n-1} H_{t_i} \left(X_{t_{i+1}} - X_{t_i}\right) \xrightarrow{\mathbb{P}} \int_0^t H_s dX_s.$$

L'**intégrale de Stratonovich** est une autre intégrale stochastique qui vérifie le théorème fondamental de l'analyse. L'intégrale de Stratonovich de $Y$ contre $X$ se note $\int_0^t Y_t \circ dX_t$. Si $f$ est une fonction $C^3$, elle vérifie donc $f(X_t) = f(X_0) + \int_0^t f'(X_u) \circ dX_u$.

**Proposition 1.7** (Méthode des trapèzes)**.** *Si $X$ est une semimartingale continue et si $H$ est à trajectoires continues, alors pour toute suite de subdivisions emboîtées $0 = t_0^n < t_1^n < \cdots < t_{p_n}^n = t$ de pas tendant vers 0,*

$$\sum_{i=0}^{p_n-1} \frac{H_{t_{i+1}} + H_{t_i}}{2} \left(X_{t_{i+1}} - X_{t_i}\right) \xrightarrow{\mathbb{P}} \int_0^t H_s \circ dX_s.$$

# 2 Dynamique des réseaux de neurones

## 2.1 Les réseaux de neurones

**Définition 2.1** (Réseau de neurones profond dense)**.** *On appelle réseau de neurones profond dense à $l-1$ couches cachées de tailles $(n_1, \ldots, n_{l-1})$ l'architecture :*

$$g(W_1, a_1, \ldots, W_l, a_l)(x) = f_l \circ \ldots \circ f_1(x),$$

*avec $\forall i < l$, $f_i(u) = s(W_i u + a_i)$, $f_l(u) = W_l u + a_l$, $W_i \in \mathcal{M}_{n_{i-1}, n_i}(\mathbb{R})$, $a_i \in \mathbb{R}^{n_i}$, $n_0$ est la dimension de l'entrée, $n_l$ est la dimension de la sortie, et $s$ une fonction d'activation.*

**Définition 2.2** (Sigmoïde)**.** *sigmoid : $x \mapsto \frac{1}{1+\exp(-x)}$ est une fonction d'activation usuelle.*

Un certain nombre de résultats théoriques permettent de rendre compte du fonctionnement des réseaux de neurones. Comme pour toute architecture, il faut d'abord s'assurer de leur **expressivité**, c'est-à-dire de leur capacité à reconstruire des fonctions complexes. Se pose ensuite les questions d'**implémentation pratique** des réseaux de neurones : à quelle valeur initialiser les poids, comment éviter le surapprentissage et comment choisir les hyperparamètres ? À un niveau plus avancé, la question de l'**explicabilité** consiste à réellement décrire le fonctionnement d'un réseau de neurones. Cela consiste notamment à montrer l'intérêt de certaines structures pour effectuer certaines tâches.

La démonstration de l'expressivité des réseaux de neurones est très ancienne et est l'objet du Théorème 2.3. On dit alors que les réseaux de neurones sont des **approximateurs universels**. Cependant, ce résultat était établit pour des réseaux à une couche cachée dont la taille pouvait être arbitraire (**wide neural networks**), alors que la pratique montrait que des réseaux de neurones avec des couches de tailles bornées mais en plus grand nombre (**narrow neural networks**) parvenaient à de meilleurs résultats. Delalleau and Bengio [2011] exhibe une famille de fonctions qui se représente avec un nombre exponentiel de neurones pour un wide neural network mais avec un nombre linéaire de neurones pour un narrow neural network. Lu et al. [2017] montre quant à lui que les narrow neural networks sont des approximateurs universels et qu'ils peuvent représenter tout wide neural network à $n$ neurones, avec un nombre de neurones polynomiale en $n$. Il y a donc un équilibre à trouver entre le nombre de couches et le nombre de neurones par couche.

**Théorème 2.3** (Théorème d'approximation universelle, Cybenkot [1989])**.** *Pour tout $p \geq 0$, les réseaux de neurones avec une sortie de dimension $p$, à une couche cachée et avec pour fonction d'activation la sigmoïde sont denses dans l'ensemble des fonctions continues sur $[0,1]^p$ pour la norme $||.||_\infty$.*

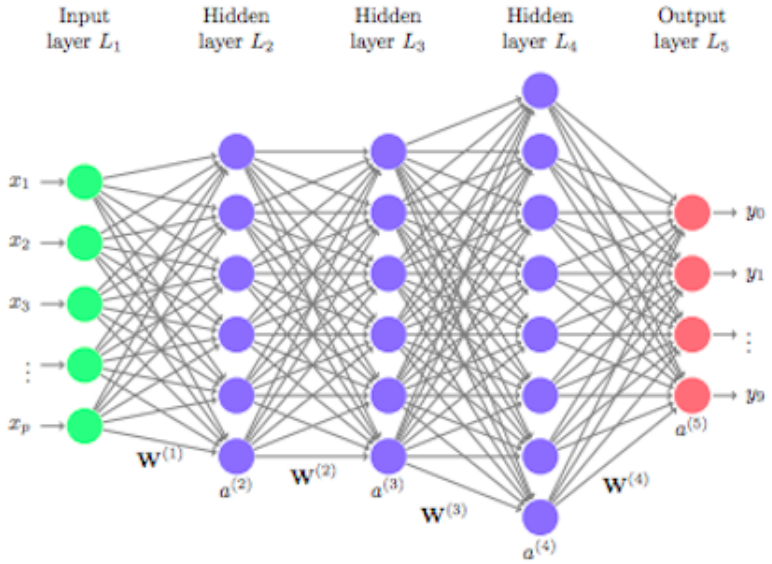


FIGURE 3 – Réseau de neurones à 3 couches cachées de tailles $(6, 6, 8)$

## 2.2 Initialisation des réseaux de neurones

### Processus et noyaux gaussiens

**Définition 2.4** (Processus). *Soit $(\Omega, \mathcal{F}, \mathbb{P})$ un espace probabilisé, un processus $X$ sur l'ensemble $T$ est une fonction* $\begin{array}{l} \Omega \xrightarrow{X} \mathbb{R}^T \\ \omega \mapsto (X_i(\omega))_{i \in T} \end{array}$ *telle que $\forall t \in T, X_t$ est une variable aléatoire.*

**Remarque 2.5** (Loi d'un processus à trajectoires continus). *L'intérêt de la notion de processus à temps continu réside dans la notion de trajectoire. Soit $\omega \in \Omega$, la trajectoire issue de $\omega$ est la fonction $t \mapsto X_t(\omega)$. Aussi, il est naturel de se demander si le processus $X$ peut être vu comme une variable aléatoire directement sur l'espace des trajectoires, ici l'espace des fonctions continues $C^0(\mathbb{R})$. Par définition, un processus est mesurable pour la tribu cylindrique sur $C^0(\mathbb{R})$. Cela étant, il est intéressant de pouvoir borner les trajectoires du processus, et donc de considérer l'évènement $(\forall t \in [0,1], |X_t| \leq 1)$ ou autrement dit la topologie de la convergence uniforme sur les compacts.*

**Théorème 2.6.** *La tribu cylindrique sur $C^0(\mathbb{R})$ coïncide avec la tribu issue de la topologie de la convergence uniforme sur les compacts.*

*Démonstration.* Le lecteur peut se référer à Billingsley [1999]. □

**Définition 2.7** (Lois marginales). *Les lois marginales d'un processus sur $T$ sont l'ensemble des lois des processus sur $\tau$ pour $\tau$ un sous-ensemble fini de $T$.*

**Corollaire 2.8.** *Deux processus à trajectoires continues qui ont les mêmes lois marginales ont la même loi.*

**Définition 2.9** (Vecteur Gaussien). *Un vecteur de variables aléatoires $(X_1, \ldots, X_n)$ est dit gaussien si toute combinaison linéaire des $(X_i)_{1 \leq i \leq n}$ suit une loi gaussienne.*

**Proposition 2.10** (Densité). *Soit $X = (X_1, \ldots, X_n)$ un vecteur gaussien. Soit $m = \mathbb{E}(X)$ et $Q = Cov(X)$. Si $detQ \neq 0$, alors $X$ admet pour densité*

$$\mathcal{N}(m, Q)(dx) = \frac{1}{\sqrt{(2\pi)^n detQ}} \exp\left(\frac{-1}{2}\langle Q^{-1}(x-m),\ x-m\rangle\right) dx.$$

**Définition 2.11** (Processus Gaussien). *Un processus $X$ sur $\mathbb{R}$ est dit gaussien si, pour tout $n \in \mathbb{N}^*$, et pour tout $(t_1, \dots, t_n) \in \mathbb{R}^n$, $(X_{t_1}, \dots, X_{t_n})$ est un vecteur gaussien. Il est dit centré si pour tout $t \in \mathbb{R}$, $\mathbb{E}(X_t) = 0$.*

**Corollaire 2.12** (Noyau gaussien). *La loi d'un processus gaussien centré à trajectoires continues $X$ est uniquement déterminé par son noyau $Q : (t_1, t_2) \mapsto Cov(X_{t_1}, X_{t_2})$.*

**Initialisation des réseaux de neurones**

Le théorème suivant montre comment normaliser un réseau de neurones profond avec un nombre fini de couches mais un grand nombre de neurones pour qu'il converge en tant que processus.

**Théorème 2.13** (Initialisation des réseaux infinis). *Soit $l \in \mathbb{N}$. Considérons une suite de réseaux de neurones profonds denses à $l$ couches de taille $(n_1, \dots, n_l) \in \mathbb{N}^l$. L'initialisation indépendante des coefficients des matrices $W_i$ par des lois $\mathcal{N}\left(0, \frac{1}{n_i}\right)$ et les coefficients des $a_i$ par des lois $\mathcal{N}(0,1)$ fait que la suite de réseaux de neurones converge en loi quand $\min(n_1, \dots, n_l) \to \infty$. La limite de cette suite est un processus continu gaussien centré de noyau $Q = \Sigma^{(l+1)}$ défini par récurrence par*

$$\begin{cases} \Sigma^{(1)}(x,y) = \frac{1}{n_0}\langle x, y\rangle + 1 \\ \Sigma^{(k+1)}(x,y) = \underset{f\sim\mathcal{N}(0,\Sigma^{(k)})}{\mathbb{E}}\left[s(f(x)) \times s(f(y))\right] + 1 \end{cases}.$$

*Démonstration.* Nous adaptons la preuve de Jacot et al. [2018] et raisonnons par récurrence. Pour $l = 0$, il n'y a pas de couche cachée. Le réseau de neurones agit donc comme la fonction

$$g(x) = W^{(0)}x + a^{(0)}.$$

Soit $n_0$ la dimension de l'entrée et $n_{l+1}$ la dimension de la sortie et $g = (g_1, \dots, g_p)$, on a donc bien que $g_i(x) = a_i^{(0)} + \sum_{j=1}^{n_0} W_{j,i}^{(0)} x_i \sim \mathcal{N}\left(0, 1 + \frac{1}{n_0}||x||_2^2\right)$ par indépendance des $a$ et des $W$. On vérifie pareillement que pour tout $n \in \mathbb{N}^*$, pout tout $(x_1, \dots, x_n) \in (\mathbb{R}^{n_0})^n$, pour tout $(i_1, \dots, i_n) \in \{1, \dots, l\}^n$, et pout tout $(\lambda_1, \dots, \lambda_n) \in \mathbb{R}^n$, $\lambda_1 g_{i_1}(x_1) + \dots + \lambda_n g_{i_n}(x_n)$ est une variable gaussienne. $g$ est donc bien un processus gaussien. Son noyau se calcule pareillement : $\Sigma^{(1)}(x,y) = \frac{1}{n_0}\langle x, y\rangle + 1$.

Supposons que la proposition soit vraie pour $l$ et montrons la pour $l+1$ et appelons $z^{(k)}$ le contenu de la $k^{ième}$ couche cachée avant l'application de la fonction d'activation. Autrement dit, si $k = 1$, $z^{(1)}(x) = W^{(0)}(x) + a^{(0)}$ et sinon $z_k(x) = W^{(k-1)} f_{k-1} \circ \dots \circ f_1(x) + a^{(k-1)}$. Nous avons montré que $z^{(l+1)}$ est un processus gaussien de noyau $\Sigma^{l+1}$ et nous voulons montrer que $z^{(l+2)}$ est également un processus gaussien de noyau $\Sigma^{l+2}$. Or,

$$z^{(l+2)} = W^{(l)} s\left(z^{(l+1)}\right) + a^{(l)}.$$

Conditionnellement à $z^{(l+1)}$, $z^{(l+2)}$ est une gaussienne de noyau

$$K^{(l+1)}\left(z^{(l+1)}(\cdot),\ z^{(l+1)}(\cdot)\right)$$

avec

$$K^{(l+1)}(x,y) = \left(\frac{1}{n_l}\langle s(x), s(y)\rangle + 1\right) Id_{n_l}.$$

On note $\phi(m, Q)(\lambda) = \exp(i\langle\lambda, m\rangle - \frac{1}{2}\lambda^T Q \lambda)$ la fonction caractéristique d'un vecteur gaussien. La fonction caractéristique de $z^{(l+2)}(x)$ est alors

$$\begin{aligned} \psi(l+1)(\lambda) &= \mathbb{E}\left[\exp\left(i\lambda\left(W^{(l)} s\left(z^{(l+1)}(x)\right) + a^{(l)}\right)\right)\right] \\ &= \mathbb{E}\left[\mathbb{E}\left(\exp\left(i\lambda\left(W^{(l)} s\left(z^{(l+1)}(x) + a^{(l)}\right)\right)\right) | z^{(l+1)}\right)\right] \\ &= \mathbb{E}\left[\phi\left(0, K^{(l+1)}\left(z^{(l+1)}(x),\ z^{(l+1)}(x)\right)\right)(\lambda)\right]. \end{aligned}$$

Comme $\min(n_1, \ldots, n_l) \to \infty$, $z^{(l+1)}$ converge en loi vers un processus gaussien de noyau $\Sigma^{(l+1)}$ et, à $n_{l+1}$ fixé, comme $x \mapsto \mathbb{E}\left[\phi\left(0, K^{(l+1)}(x,x)\right)(\lambda)\right]$ est une fonction continue bornée,

$$\mathbb{E}\left[\phi\left(0, K^{(l+1)}\left(z^{(l+1)}(x), z^{(l+1)}(x)\right)\right)(\lambda)\right] \to \underset{Z_{n_{l+1}} \sim \mathcal{N}(0, \Sigma^{(l+1)}(x,x))}{\mathbb{E}} \left[\phi\left(0, K^{(l+1)}\left(Z_{n_{l+1}}, Z_{n_{l+1}}\right)\right)(\lambda)\right].$$

De plus, quand $n_{l+1} \to \infty$,

$$K^{(l+1)}\left(Z_{n_{l+1}}, Z_{n_{l+1}}\right) \xrightarrow{p.s} \underset{Z \sim \mathcal{N}(0, \Sigma^{(l+1)}(x,x))}{\mathbb{E}} \left[\left(\langle s(Z), s(Z)\rangle + 1\right) Id_{n_l}\right].$$

La convergence presque sûre est à comprendre sur l'espace probabilisé limite $\Omega$ portant la suite de variables $(Z_{n_{l+1}})_{n_{l+1} \in \mathbb{N}}$ muni de la tribu cylindrique. (Les $Z_{n_{l+1}}$ ne sont pas des variables aléatoires qui ont une réalité physique, au sens où elles ne représentent pas une donnée initiale comme les poids du réseau par exemple, mais bien une limite en loi des $z^{(l+1)}$. La construction de $\Omega$ est donc purement formelle et résulte du théorème de Daniell-Kolmogorov.) Par convergence dominée sur $\Omega$, comme $\phi$ est bornée par 1, on en déduit que

$$\psi(l+1)(\lambda) \xrightarrow{\min(n_1, \ldots, n_{l+1}) \to \infty} \phi\left(0, \underset{Z \sim \mathcal{N}(0, \Sigma^{(l+1)}(x,x))}{\mathbb{E}} \left[\left(\langle s(Z), s(Z)\rangle + 1\right) Id_{n_l}\right]\right)(\lambda).$$

Le théorème de Paul-Lévy affirme donc que $z^{(l+2)}(x)$ converge en loi vers une gaussienne centrée de noyau $\Sigma^{(l+2)}(x,x)$. Exactement le même schéma de preuve avec des expressions plus longues permette de montrer que $z^{(l+2)}$ est un processus gaussien de noyau $\Sigma^{(l+2)}$. □

**Remarque 2.14.** *Ce noyau est appelé **NNGP** (Neural Network Gaussian Process).*

## 2.3 Descente de gradient stochastique

La recherche d'un minimiseur du risque empirique est un problème NP difficile comme le montre le Théorème 20.7 de Shalev-Shwartz and Ben-David [2014]. C'est pourquoi en pratique, les paramètres du réseaux de neurones sont estimés par une descente de gradient. La méthode de la **backpropagation** permet en effet de calculer le gradient du risque empirique en fonction des paramètres du réseau de neurones avec une complexité polynomiale. Cela étant, les algorithmes de descente de gradient ne convergent que vers des minima locaux du risque empirique. Certains travaux montrent cependant que ces minima locaux sont proches du minimum global [Choromanska et al., 2014]. Nous présentons ici un objet mathématique, le NTK, qui permet de modéliser la descente de gradient stochastique et de donner des éléments pour calculer un hyperparamètre important : le learning rate.

**Définition 2.15** (NTK)**.** *Considérons un réseau de neurones profond représenté par la fonction $g : \mathbb{R}^{n_0} \to \mathbb{R}^{n_l}$. On regroupe ses $N$ paramètres dans $\theta = (W^{(0)}, a^{(0)}, \ldots, W^{(l)}, a^{(l)})$. On lui associe alors un noyau, le Neural Tangent Kernel (NTK) :*

$$\forall (x,y) \in \mathbb{R}^{2 \times n_0}, \quad \Theta(x,y) = \sum_{i=1}^{N} \partial_{\theta_i} g(x) \otimes \partial_{\theta_i} g(y)$$

*où $\otimes$ désigne le produit tensoriel canonique.*

**Proposition 2.16** (Convergence du NTK)**.** *Sous les mêmes conditions que le Théorème 2.13, le NTK converge en probabilité vers un noyau défini récursivement par*

$$\begin{cases} \Theta_\infty^{(1)} = \Sigma^{(1)} \\ \Theta_\infty^{(k+1)}(x,y) = \Theta_\infty^{(k)}(x,y) \underset{f \sim \mathcal{N}(0, \Sigma^{(k)})}{\mathbb{E}} \left[s'(f(x)) \times s'(f(y))\right] + \Sigma^{(k+1)}(x,y) \end{cases}.$$

$$\Theta \xrightarrow{\mathbb{P}} \Theta_\infty^{(l)} \otimes Id_{n_l}.$$

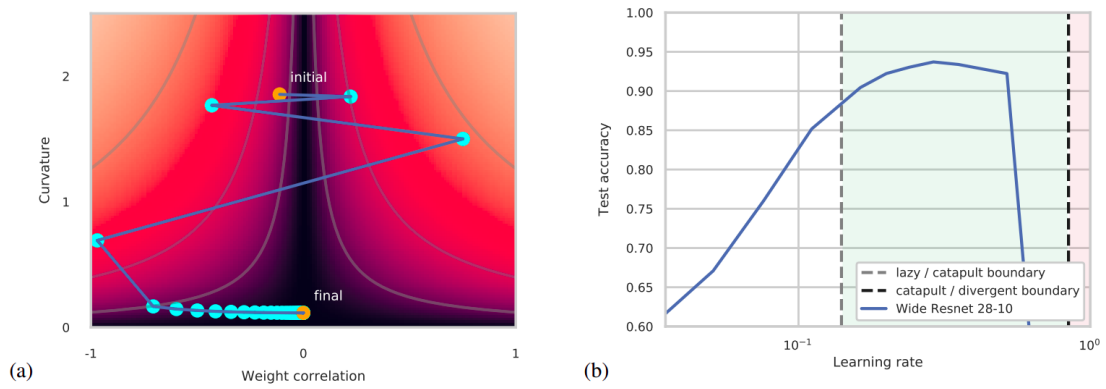


FIGURE 4 – Trois phases d'apprentissage

*Démonstration.* Il s'agit du Théorème 1 de Jacot et al. [2018]. La preuve reprend essentiellement celle développée ici pour le Théorème 2.13. □

**Théorème 2.17** (Dynamique des réseaux infinis)**.** *Sous les mêmes conditions que le Théorème 2.13, lors de la descente de gradient stochastique, les réseaux de neurones profonds convergent vers la solution d'une équation différentielle stochastique dont les paramètres dépendent du NTK.*

*Démonstration.* Il s'agit du Théorème 2 de Jacot et al. [2018]. □

**Théorème 2.18** (Learning rate et valeurs propres du NTK)**.** *Soit $\lambda$ la plus grande valeurs propres des noyaux évaluées sur l'ensemble des données. Trois phases existent durant l'entraînement qui dépendent du learning rate $\gamma$ :*

- *si $\gamma < 2/\lambda$, on parle de **lazy phase**. La convergence de la descente de gradient vers un minimum local est lente,*
- *si $2/\lambda < \gamma < 4/\lambda$, on parle de **catapult phase**. La descente de gradient est optimale.*
- *si $4/\lambda < \gamma$, on parle de **divergent phase**. La descente de gradient ne converge pas.*

*La Figure 4 illustre ce phénomène.*

*Démonstration.* Voir Lewkowycz et al. [2020]. □

**Remarque 2.19** (Enjeux de recherche)**.** *Le calcul des NNGP et des NTK des différentes architectures neuronales est un objet de recherche en soi. Par exemple, ce calcul a été mené récemment pour les algorithmes utilisant la structure de l'attention [Hron et al., 2020]. De plus, certaines études s'intéressent aux conséquences qu'imposent la convergence du NTK sur l'architecture. Hanin and Nica [2019] montre que si $\sum_{i=1}^{l} \frac{1}{n_i} \to 0$, alors la suite de réseaux de neurones converge encore en loi. En outre, comme le NTK décrit la dynamique stochastique des réseaux de neurones, il est naturel qu'il donne des critères sur les hyperparamètres liés à la descente de gradient. Mieux comprendre ces liens, peut-être jusqu'à expliquer l'étrange phénomène de double descent, est un enjeu important.*

# 3 Equations différentielles neuronales

## 3.1 Réseaux de neurones et équations différentielles

La modélisation du fonctionnement intrinsèque des réseaux de neurones comme des équations différentielles est un paradigme récent mais très actif. L'idée fut introduite en 2018 par Chen et al. [2018] lors d'un papier qui fut jugé comme l'un des meilleurs de la conférence NeurIPS, une des plus prestigieuses dans le domaine du Machine Learning.

**Définition 3.1** (Réseau de neurones résiduel)**.** *Considérons en entrée une série temporelle* $(x_t)_{t\in[0,1]}$*. On appelle réseau de neurones résiduel à* $l$ *couches l'architecture :*

$$g(l)(x) = f_l \circ \ldots \circ f_1(x),$$

*avec* $f_i(u) = u + \frac{1}{l}h(u, x_{\frac{i}{l}}, \frac{i}{l})$ *et* $h$ *une fonction continue.*

**Exemple 3.2.** *Le ResNet, introduit par He et al. [2015], est un réseau de neurones résiduel extrêmement performant pour la classification d'images. Ce fut le premier réseau de neurones résiduel inventé.*

**Définition 3.3** (NODE)**.** *L'approche de Chen et al. [2018] consiste à remarquer que les réseaux de neurones résiduels sont des schémas d'Euler associés à des équations différentielles appelées* ***équations différentielles neuronales*** *(NODE)*

$$dY_t = h(Y_t, x_t, t)dt.$$

*La solution de l'équation différentielle pour* $Y_1$ *correspond alors à la sortie du réseau de neurones résiduel. Il suffit alors de remplacer le réseau de neurone par un solveur d'équations différentielle.*

**Exemple 3.4** (Nouveaux algorithmes résiduels)**.** *Depuis la publication du papier de Chen et al. [2018], de nombreux réseaux de neurones performants ont vu émerger leur équivalent résiduel. Les Recurrent Neural Networks (RNNs), qui permettent de traiter des données temporelles pour de la prévision par exemple, ont un équivalent continu introduit par Rubanova et al. [2019]. Il en va de même des Long Short-Term Memories (LSTMs) avec Fermanian et al. [2021], des Gated Recurrent Networks (GRUs) avec Brouwer et al. [2019]. Les VAEs, qui permettent de procéder à de l'augmentation de données, ont équivalent récursif introduit par [Fabius and van Amersfoort, 2015].*

## 3.2 L'exemple du RNN résiduel

L'intérêt de considérer les modèles résiduels provient du fait que leur proximité avec les équations différentielles rend la démonstration de certains critères plus facile. Nous développons dans ce paragraphe un cas particulier : une majoration du risque global pour les RNN résiduel.

**Définition 3.5** (RNN)**.** *Les réseaux de neurones récurrents sont des réseaux de neurones prenant en entrée des données temporelles* $(x_t)_{t\in[0,1]} \in (\mathbb{R}^p)^{[0,1]}$*. Ils sont construits à partir d'une architecture vérifiant*

$$g(A, B, c)(x) = \psi(u_l),$$

*avec* $u_0 = 0$*,* $\forall 0 \leq k \leq l$*,* $u_{k+1} = s(Au_k + Bx_{k/l} + c)$*,* $A \in \mathbb{R}$*,* $B \in \mathcal{M}_{p,1}(\mathbb{R})$*,* $c \in \mathbb{R}$*,* $s$ *une fonction d'activation et* $\psi$ *une fonction continue.*

**Définition 3.6** (Classification et prévision)**.** *Un RNN peut être employé pour la classification et pour la prévision de séries temporelles. La* ***classification*** *consiste en le regroupement des données selon des catégories. Par exemple, imaginons que l'on veuille classer les battements cardiaques en trois catégories : sain, tachycardie et bradycardie [Singh et al., 2018]. Le RNN prendrait en entrée un enregistrement des battements cardiaques sur une fenêtre de temps limité* $(x_t)_{t\in[0,T]}$ *et associerait à* $x$ *la probabilité qu'il s'agisse d'un battement sain* $p_s$*, ou de tachycardie* $p_t$*, ou de bradycardie* $p_b$*. Le RNN est alors une fonction de* $\mathbb{R}^{[0,T]}$ *dans* $\mathbb{R}^3$*. La* ***prévision*** *consiste en la reconstruction d'une série temporelle* $(y_t)_{t\in[0,T]}$ *à partir d'une autre série temporelle* $(x_t)_{t\in[0,T]}$*. Par exemple, il est possible de prédire la consommation électrique française* $y$ *à partir des données de température* $x$ *[Yahya et al., 2018]. Le RNN est alors une fonction de* $\mathbb{R}^{[0,T]}$ *dans* $\mathbb{R}^{[0,T]}$*.*

**Définition 3.7** (RNN résiduel)**.** *Les réseaux de neurones récurrents résiduel sont des réseaux de neurones prenant en entrée des données temporelles* $(x_t)_{t\in[0,1]} \in (\mathbb{R}^p)^{[0,1]}$*. Ils sont construits à partir d'une architecture vérifiant*

$$g(A, B, c)(x) = \psi(u_l),$$

*avec* $u_0 = 0$*,* $\forall 0 \leq k \leq l$*,* $u_{k+1} = u_k + \frac{1}{l}s(Au_k + Bx_{k/l} + c)$*,* $A \in \mathbb{R}$*,* $B \in \mathcal{M}_{p,1}(\mathbb{R})$*,* $c \in \mathbb{R}$*,* $s$ *une fonction d'activation et* $\psi$ *une fonction continue.*

**Proposition 3.8** (Du RNN résiduel à la NODE). *Supposons que l'entrée $x$ soit à variation bornée et que $s$ soit lipschitzienne. L'équation différentielle neuronale*

$$\begin{cases} h_0 = 0 \\ dh_t = s(Ah_t + Bx_t + c)dt \end{cases}$$

*admet une unique solution $h$. De plus, il existe une constante $C$ telle que*

$$\forall 1 \leq j \leq k,\ ||u_k - h_{k/l}||_2 \leq \frac{C}{l}.$$

*Démonstration.* Il s'agit de la Proposition 1 de Fermanian et al. [2021]. □

**Définition 3.9** (Erreur empirique et risque). *On rappelle qu'étant donnée une fonction de perte $\mathcal{L}$, l'erreur empirique pour le paramétrage $\lambda$ est $\mathcal{L}_n(\lambda) = \sum_{i=1}^n \mathcal{L}\left(y_i,\ g(\lambda)(x_i)\right)$. Le* ***risque*** *associé au paramétrage est $R(\lambda) = \mathbb{E}\left(\mathcal{L}\left(y_i,\ g(\lambda)(x_i)\right)\right)$.*

**Théorème 3.10** (Ecart erreur empirique et risque). *Considérons un RNN résiduel utilisé en classification ou en prédiction. On suppose que $h$ est lipschitzienne et que la fonction d'activation est soit sigmoid et soit tanh. Soit $\lambda_n^*$ le paramètre qui minimise l'erreur empirique $\mathcal{L}_n$. Fermanian et al. [2021] calcule alors des constantes $c_1$, $c_2$ et $c_3$ telles que, avec une probabilité d'au moins $1-\delta$,*

$$R(\lambda_n^*) \leq \mathcal{L}_n(\lambda_n^*) + \frac{c_1}{l} + \frac{c_2}{\sqrt{n}} + c_3\sqrt{\frac{\ln\left(\frac{1}{\delta}\right)}{n}}.$$

## 3.3 Signature et apprentissage

Nous donnons ici les arguments utilisés par Fermanian et al. [2021] pour établir l'inégalité du Théorème 3.10 car ils établissent un lien très intéressant entre algorithmes résiduels et un objet mathématique appelé la signature. De plus, à plus haut niveau, cela établit un lien entre les algorithmes de Machine Learning et un type d'équations différentielles plus générales encore que les EDS : les équations différentielles rugueuses.

**Définition 3.11** (Signature). *La signature est un objet algébrique qui est au fondement de la théorie des chemins rugueux developpée dans les années 1990s par Terry Lyons. Etant donnés une façon d'intégrer, notée $\int$, et un chemin intégrable $(x_t)_{t\in[0,T]}$, on associe au chemin $x$ la suite d'intégrales itérées*

$$\begin{aligned} S_{[0,T]}(x) &= \left(1, S^1_{[0,T]}(x), ..., S^n_{[0,T]}(x), ...\right) \\ &= \left(1, \int_0^T dx_{t_1}, ..., \int_0^T \int_0^{t_1} ... \int_0^{t_{n-1}} dx_{t_n} \otimes ... \otimes dx_{t_1}, ...\right). \end{aligned}$$

*$S_{[0,T]}(x)$ s'appelle la* ***signature*** *de $x$. Si $x$ est un chemin dans $\mathbb{R}^d$ et soit $(e_1, ..., e_d)$ la base canonique de $\mathbb{R}^d$, le produit tensoriel est à comprendre de la façon suivante :*

$$S^n_{[0,T]}(x) = \sum_{1\leq i_1,...,i_n\leq d} \left(\int_0^T \int_0^{t_1} ... \int_0^{t_{n-1}} dx^{i_n}_{t_n} ... dx^{i_1}_{t_1}\right) e_{i_n} \otimes ... \otimes e_{i_1} \in \left(\mathbb{R}^d\right)^{\otimes n}.$$

Fermanian et al. [2021] généralisent un résultat connu (cas où $F$ est linéaire, voir Lyons et al. [2004]) et montrent que la solution d'une équation différentielle contre un chemin $x$ peut s'écrire comme une opération linéaire sur la signature.

**Théorème 3.12** (Développement en signature). *Soit $F \in C^\infty\left(\mathbb{R}^d\right)$ et $x$ un processus à variation finie, la solution de l'équation différentielle*

$$dY_t = F(Y_t)d\boldsymbol{x}_t$$

*peut s'écrire sous la forme*

$$Y_t = Y_0 + \sum_{k=1}^{\infty} \mathcal{D}^k(F, Y_0) \left( S^k_{[0,t]}(\boldsymbol{x}) \right),$$

*où les opérateurs $\mathcal{D}^k(F, Y_0)$ sont linéaires.*

**Remarque 3.13.** *Ils montrent alors que la solution d'une NODE est une opération linéaire sur la signature. La signature est donc une transformation qui simplifie l'expression du réseau de neurones, on parle de* ***noyau****. Le cadre des méthodes à noyau est un cadre bien connu des statistiques, à partir duquel ils prouvent le Théorème 3.10.*

**Remarque 3.14** (Mémoire de M2)**.** *La généralisation de ce type de résultat à d'autres types d'algorithmes que le RNN résiduel et au cas où $x$ n'est pas à variation finie mais suit une équation différentielle stochastique était l'enjeu de mon mémoire de M2.*

# 4 Conclusion et ouverture

L'objectif de cette Introduction au Domaine de Recherche était de présenter les applications du calcul stochastique, et plus généralement de la modélisation probabiliste, à l'apprentissage statistique. La plupart des difficultés techniques (construction des intégrales stochastiques, convergence de la descente de gradient vers une EDS, convergence des algorithmes résiduels vers des EDS, diversité dans les architectures neuronales...) ont été évitées à dessein pour mettre en lumière un certain nombre de gains concrets qu'apporte la modélisation stochastique des réseaux de neurones. Les sujets traités ici sont loin de regrouper l'ensemble des applications des équations différentielles stochastiques au Machine Learning. Cependant, les Théorèmes 2.13, 2.18 et 3.10 visent à illustrer le potentiel de cette approche en apportant des résultats à différents endroits du cadre de l'apprentissage supervisé présenté dans le paragraphe 1.2, à savoir la conformité de la méthode de minimisation du risque empirique et le réglage des hyperparamètres d'initialisation du réseau et de pas d'apprentissage.

D'autres résultats théoriques intéressants n'ont pas pu être présentés ici. Par exemple, l'ajout d'un bruit brownien dans les NODEs aboutit à des EDS appelées les NSDEs qui sont plus stables que les NODEs [Liu et al., 2019]. De plus, montrer la convergence des algorithmes vers des équations différentielles stochastiques est un sujet de recherche actuel [Peluchetti and Favaro, 2019]. Enfin, l'étude des liens entre apprentissage et équations différentielles rugueuses est un sujet extrêmement récent et prometteur [Morrill et al., 2020, Ni et al., 2020] et est un des sujets de la thèse que je poursuivrai avec le Professeur Gérard Biau.

# Références


Patrick Billingsley. Convergence of probability measures, second edition, 1999.

Edward De Brouwer, Jaak Simm, Adam Arany, and Yves Moreau. Gru-ode-bayes : Continuous modeling of sporadically-observed time series, 2019.

Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. *CoRR*, abs/1806.07366, 2018. URL `http://arxiv.org/abs/1806.07366`.

Anna Choromanska, Mikael Henaff, Michael Mathieu, Gérard Ben Arous, and Yann LeCun. The loss surfaces of multilayer networks, 2014. URL `https://arxiv.org/abs/1412.0233`.

G. Cybenkot. Approximation by superpositions of a sigmoidal function, 1989.

Olivier Delalleau and Yoshua Bengio. Shallow vs. deep sum-product networks. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc., 2011. URL `https://proceedings.neurips.cc/paper/2011/file/8e6b42f1644ecb1327dc03ab345e618b-Paper.pdf`.

Otto Fabius and Joost R. van Amersfoort. Variational recurrent auto-encoders, 2015.

Adeline Fermanian, Pierre Marion, Jean-Philippe Vert, and Gérard Biau. Framing rnn as a kernel method : A neural ode approach, 2021.

Boris Hanin and Mihai Nica. Finite depth and width corrections to the neural tangent kernel, 2019. URL https://arxiv.org/abs/1909.05989.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

Jiri Hron, Yasaman Bahri, Jascha Sohl-Dickstein, and Roman Novak. Infinite attention : Nngp and ntk for deep attention networks, 2020. URL https://arxiv.org/abs/2006.10540.

Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel : Convergence and generalization in neural networks. *CoRR*, abs/1806.07572, 2018. URL http://arxiv.org/abs/1806.07572.

Aitor Lewkowycz, Yasaman Bahri, Ethan Dyer, Jascha Sohl-Dickstein, and Guy Gur-Ari. The large learning rate phase of deep learning : the catapult mechanism, 2020.

Xuanqing Liu, Tesi Xiao, Si Si, Qin Cao, Sanjiv Kumar, and Cho-Jui Hsieh. Neural sde : Stabilizing neural ode networks with stochastic noise, 2019. URL https://arxiv.org/abs/1906.02355.

Zhou Lu, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang. The expressive power of neural networks : A view from the width, 2017. URL https://arxiv.org/abs/1709.02540.

Terry J. Lyons, Michael Caruana, and Thierry Levy. Differential equations driven by rough paths, 2004.

Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. Foundations of machine learning, second edition, 2018.

James Morrill, Cristopher Salvi, Patrick Kidger, James Foster, and Terry Lyons. Neural rough differential equations for long time series, 2020. URL https://arxiv.org/abs/2009.08295.

Hao Ni, Lukasz Szpruch, Magnus Wiese, Shujian Liao, and Baoren Xiao. Conditional sig-wasserstein gans for time series generation, 2020. URL https://arxiv.org/abs/2006.05421.

Stefano Peluchetti and Stefano Favaro. Infinitely deep neural networks as diffusion processes, 2019. URL https://arxiv.org/abs/1905.11065.

L.C.G. Rogers and David Williams. Diffusions, markov processes, and martingales- volume 2 : Ito calculus, 2nd edition, 1994.

Yulia Rubanova, Ricky T. Q. Chen, and David Duvenaud. Latent odes for irregularly-sampled time series. *CoRR*, abs/1907.03907, 2019. URL http://arxiv.org/abs/1907.03907.

Shai Shalev-Shwartz and Shai Ben-David. Understanding machine learning : From theory to algorithms, 2014.

Shraddha Singh, Saroj Kumar Pandey, Urja Pawar, and Rekh Ram Janghel. Classification of ecg arrhythmia using recurrent neural networks. *Procedia Computer Science*, 132 :1290–1297, 2018. ISSN 1877-0509. doi : https ://doi.org/10.1016/j.procs.2018.05.045. URL https://www.sciencedirect.com/science/article/pii/S1877050918307774. International Conference on Computational Intelligence and Data Science.

Roman Vershynin. High-dimensional probability, an introduction with applications in data science, 2020.

Muhammad Amri Yahya, Sasongko Pramono Hadi, and Lesnanto Multa Putranto. Short-term electric load forecasting using recurrent neural network (study case of load forecasting in central java and special region of yogyakarta). In *2018 4th International Conference on Science and Technology (ICST)*, pages 1–6, 2018. doi : 10.1109/ICSTC.2018.8528651.