MicrRÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE

MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE Dr. TAHAR MOULAY SAIDA

FACULTE : TECHNOLOGIE

DEPARTEMENT : INFORMATIQUE

MÉMOIRE DE MASTER

Option :

Modélisation Informatique des Connaissances et du Raisonnement

---

Le deep learning et la bioinformtique pour analyser la degradation du vaccin covid-19 a arnm

---

Présenté par :

KACIMI Aymen Issam Eddine

MEDJDOUBI Aymen

Encadré par :

Dr. BOUARARA Hadj Ahmed

Année Universitaire 2021-2022

RÉSUMÉS

Abstract — There is increasing interest in mRNA vaccines as potential alternatives to conventional methods of preventing several diseases, including Covid-19. Our goal is to improve the stability of mRNA vaccines. For this we have proposed a system called IA-AVC (artificial intelligence to analyze the covid-19 vaccine) based on the use of recurrent deep learning to extract knowledge from RNA sequences by proposing different configuration of the RNN (recurent neural network) like LSTM and BERT.

For the experiments we have varied different hyperparameters such as the batch size, lerning rate, dropout... .ect. The results obtained were tested on the OpenVaccine dataset offered by Stanford University and validated by evaluation measures such as loss, accuracy, precision recall. In terms of comparison we can say that the configuration based on LSTM with a success rate of 60% can be used to accelerate the search for mRNA vaccines and provide a refrigerator stable vaccine against SARS-CoV-2.

Keywords :bioinformatics, recurrent neural network, bert LSTM, covid vaccine.

Résumé — Les vaccines à ARNm suscitent un intérêt accrue en tant qu'alternatives potentielles aux méthodes conventionnelles de prévention de plusieurs maladies, dont le Covid-19. Notre objectif est L'amélioration de la stabilité des vaccins à ARNm. Pour cela nous avons proposé un système appelé IA-AVC(intelligence artificielle pour analyser le vaccin covid-19) basée sur l'utilisation du deep learning récurent afin d'extraire de la connaissance à partir des séquences d'ARN en proposant différentes configuration du RNN(recurent neural network) comme le LSTM et le BERT.

Pour les expérimentations, nous avons variés différent hyperparamètres comme le batch size, lerning rate, dropout....ect. Les résultats obtenus ont été testé sur le dataset OpenVaccine proposé par l'Université de Stanford et validés par les mesures d'évaluation comme le loss, accuracy, precision recall. En termes de comparaison, nous pouvons dire que la configuration basée sur le LSTM avec un taux de succées de 60% peut étre utiliser pour accélérer la recherche de vaccins à ARNm et fournir un vaccin stable au réfrigérateur contre le SRAS-CoV-2.

Mots clés : Bioinformatics, mRNA vaccine, recurrent neural network, bert LSTM, covid vaccine.

الملخص ---

هناك اهتمام متزايد بلقاحات الرنا المرسال كبدائل محتملة للطرق التقليدية للوقاية من العديد من الأمراض ، بما في ذلك كوفيد ٩١. هدفنا هو تحسين استقرار لقاحات الرنا المرسال. لهذا، اقترحنا نظاماً يسمى ذ ا-ت ل ك (الذكاء الاصطناعي لتحليل لقاح كوفيد ٩١) استناداً إلى استخدام التعلم العميق المتكرر لاستخراج المعرفة من تسلسل الحمض النووي الريبي من خلال اقتراح تكوين مختلف لـ الشبكة العصبية المتكررة مثل ذاكرة طويلة المدى و بيرت.
بالنسبة للتجارب، قمنا بتغيير معلمات تشعبية مختلفة مثل حجم الدفعة، ومعدل التعلم ، والتسرب ... إلخ. تم اختبار النتائج التي تم الحصول عليها على مجموعة بيانات) اللقاح مفتوح(التي(تقدمها جامعة ستانفورد وتم التحقق من صحتها من خلال مقاييس التقييم مثل الفقد والدقة واسترجاع الدقة. من حيث المقارنة، يمكننا القول أنه يمكن استخدام التكوين القائم على ذاكرة طويلة المدى بمعدل نجاح ٠٦٪ لتسريع البحث عن لقاحات الرنا المرسال وتوفير لقاح مستقر للثلاجة ضد فيروس كورونا المتلازمة التنفسية الحادة الوخيمة ٢.
الكلمات المفتاحية:المعلوماتية الحيوية ، لقاح الرنا المرسال ، الشبكة العصبية المتكررة ، بيرت ، الذاكرة طويلة المدى ، لقاح كوفيد

REMERCIEMENT

Tout d'abord, je tiens à remercier Dieu de nous avoir accordé toute la détermination, la volonté et la force pour qu'on puisse réaliser ce modeste travail.

Je voudrais également être reconnaissante infiniment à mon encadreur Dr.BOUARARA HADJAHMED pour ses conseils, sa patience, sa disponibilité et son soutien tout au long de cette période.

Je voudrais exprimer ma reconnaissance envers mes parents Ahmed et Salhi.Z mon frère et mes sœurs, qui m'ont apporté leur support moral et les discussions animées tout au long de ma démarche.

Nous tenons à exprimer notre profonde gratitude et nos sincères remerciements aux membres de jury d'avoir accepté de juger notre travail et de l'avoir enrichi.

Kacimi Aymen I/E

Je suis Redevable à un certain nombre de personnes pour leur soutien pendant que je menais les travaux relatifs à ce travail.
J'adresse tout d'abord ma profonde reconnaissance à mon encadreur , BOUARARA HADJ AHMED, pour son soutien continu sa patience, sa motivation et ses connaissances immenses. Ses conseils éclairés m'ont aidé tout au long de l'élaboration de ce travail et qui ont permis de mieux cerner les difficultés de mes recherches.
Je remercie chaleureusement les deux êtres les plus chers au monde : mes parents qui ont bercé ma vie de tendresse, d'amour et d'encouragements et qui ont œuvré pour ma réussite avec leur indéfectible soutien, leurs précieux conseils,leur constante présence à mes côtés aux sacrifices qu'ils ont consentis pour m'aider à mieux avancer dans la vie.
Mes remerciements vont naturellement à tous les membres du jury qui ont bien voulu prendre le temps de lire ce modeste travail et de l'évaluer.
Merci enfin à mes amis pour leur aide qui m'a permis de réaliser dans de bonnes conditions ce travail avec le concours de leurs conseils et orientations.

Medjdoubi Aymen

Table des sigles et acronymes

IA : *Intelligence Artificielle*
DL : *Deep Learning.*
COVID-19 : *Coronavirus Disease 19.*
SARS-CoV-2 : *severe acute respiratory syndrome coronavirus 2.*
LSTM : *Long Short Term Memory*
ML : *Machine Learning*
RNN : *Recurrent Neural Network..*
CNN : *Convolutional Neural Network.*
ADAM : *Adaptive Moments.*
BERT : *Bidirectional Encoder Representations from Transformers*
mRNA : *Messenger Ribonucleic Acid.*
ADN : *Acide Désoxyribonucléique.*
DNN : *deep neural network*
BBTT : *Backpropagation Through Time.*
SGD : *descente de gradient stochastique.*
DBN : *Deep Belief Networks.*
FNN : *Feed-forward Neural Networks.*
SVM : *Support Vector Machine.*

TABLE DES MATIÈRES

LISTE DES TABLEAUX

TABLE DES FIGURES

LISTE DES FIGURES **Page**

CHAPITRE 1

INTRODUCTION GÉNÉRALE

Au cours des deux dernières décennies, il y a eu un intérêt croissant dans le domaine des technologies basées sur l'ARN dans la création de vaccins prophylactiques. Ils sont largement considérés comme des alternatives plausibles aux approches conventionnelles en raison de leur puissance élevée, de leur capacité de fabrication rapide et à faible coût et de leur administration relativement sûre. Ils font actuellement l'objet de recherches pour de nombreuses maladies, le vaccin Pfizer-BioNTech contre le SRAS-CoV-2 étant le premier à être approuvé pour un usage humain [10].

L'un des principaux obstacles à la création de telles thérapeutiques est la fragilité de la molécule d'ARN ; ils sont susceptibles de se dégrader rapidement en quelques minutes à quelques heures et, en tant que tels, doivent être lyophilisés ou incubés à basse température pour rester stables. Dans la pandémie actuelle, les vaccins sont considérés comme le moyen le plus prometteur de contrôler le nouveau coronavirus, mais avec les limitations actuelles des vaccins à ARNm, une délivrance in vivo efficace de ces molécules d'ARNm semble improbable ; il n'atteindrait probablement qu'une fraction de la population, principalement reléguée aux pays avec un niveau de développement des infrastructures plus élevé.

Par conséquent, la recherche sur la stabilité et la dégradation des molécules d'ARN a suscité un intérêt continu, consistant à ce jour en grande partie sur des approches statistiques traditionnelles et des modèles biophysiques. Cependant, on ne sait toujours pas exactement quelles parties des molécules d'ARN sont plus sujettes à la dégradation spontanée et donc difficiles à prédire avec précision la réactivité et la dégradation de l'ARNm. Par conséquent, l'expérimentation, un

processus incroyablement long, est la méthode par défaut pour déterminer ces valeurs.

1.1 Objectif

Dans ce projet, notre objectif est de présenter trois approches d'apprentissage en profondeur possible à ce problème grâce à l'utilisation de l'ensemble de données Stanford OpenVaccine.Troiss variantes de réseaux neuronaux récurrents (RNN) sont utilisées, les réseaux de mémoire à long terme (LSTM) semple réseau neuronaux récurrents, ainsi Bidirectional Encoder Representations from Transformers (BERT) Ces modèles sont appliqués et comparés pour évaluer si les méthodes d'apprentissage automatique peuvent fournir des résultats utiles pour prédire la réactivité et la dégradation des molécules d'ARNm.

1.2 La description des chapitres

Le reste de ce travail est structuré en 5 chapitres :

— **Chapitre 2** : Explique s'est quoi le coronavirus (COVID-19) , leurs symptômes et Effets, Les différents types de vaccin de cette pandémie, avec l'utilisation d'IA pendant cette pandémie

— **Chapitre 3** : une sera une description de l'apprentissage approfondi, leur origine le réseau de neurones, le fonctionnement de l'apprentissage profond et les différents models de deep learning.

— **Chapitre 4** : Explique notre contribution pour prédite stabilité de vaccin utilisant le réseau de neurones récurrents et comment fonctionner.

— **Chapitre 5** :raite le choix de plateforme pour l'implémentation de notre modèle, les expérimentations, l'évaluation de performance.

— **Chapitre 6** :Discute la conclusion et les futures travaux.

CHAPITRE 2

COVID - 19

2.1 Introdiction

Au cours des deux dernières décennies, le monde a connu un certain nombre de foyers de maladies infectieuses qui ont montré un taux élevé de propagation.
À l'heure actuelle, l'inquiétude grandit face à la propagation continue de le COVID-19 dans certaines régions du monde et le maintien d'un taux réduit de propagation de l'épidémie dans d'autres est un véritable défi. Les gouvernements, les employeurs, les travailleurs et leurs organisations sont confrontés à d'énormes défis lorsqu'ils tentent de lutter contre la pandémie de COVID-19 et de protéger la sécurité et la santé sur les lieux de travail.

2.2 Coronavirus (COVID-19) :

2.2.1 Qu'est-ce qu'un coronavirus (COVID-19) ?

Les coronavirus sont une grande famille de virus qui sont communs aux animaux et peuvent occasionnellement causer des maladies chez les animaux ou les humains. Les gens sont infectés par ces virus qui peuvent ensuite se propager à d'autres personnes : chez l'homme, plusieurs coronavirus sont connus pour provoquer des infections respiratoires allant du rhume aux maladies plus graves telles que le syndrome respiratoire du MoyenOrient (MERS) et le syndrome respiratoire aigu sévère (SARS). Le coronavirus le plus récemment découvert provoque la maladie du coronavirus COVID-19 (OMS, 2020 D)[2].
La maladie du coronavirus (COVID-19) est une maladie infectieuse causée par un coronavirus nouvellement découvert connu sous le nom de coronavirus 2 du syndrome respiratoire aigu sévère (SARS-CoV-2). Les premiers cas humains de COVID-19 ont été identifiés dans la ville de Wuhan,

en Chine, en décembre 2019. (OMS, 2020).

2.2.2 Comment la COVID-19 se propage-t-elle ?

Lorsqu'une personne atteinte de la COVID-19 tousse ou expire, elle libère des gouttelettes de liquide infecté. Si les personnes se tiennent à moins d'un mètre d'une personne atteinte de la COVID-19, elles peuvent l'attraper en respirant des gouttelettes crachées ou expirées par elles. En outre, la plupart de ces gouttelettes tombent sur des surfaces et des objets proches tels que des tables de bureau ou des téléphones. Les gens peuvent attraper la COVID-19 en touchant des surfaces contaminées sur des objets, puis en touchant leurs yeux, leur nez ou leur bouche.

2.2.3 Quels sont les symptômes du COVID-19 ?

Les symptômes les plus courants de la COVID-19 sont la fièvre, la fatigue et la toux sèche. Certains patients peuvent avoir des courbatures et des douleurs, une congestion nasale, un nez qui coule, un mal de gorge ou une diarrhée. Ces symptômes sont généralement légers et commencent progressivement. Certaines personnes sont infectées, mais ne présentent aucun symptôme et ne se sentent pas mal. Cependant, selon les connaissances actuelles, environ 1 cas sur 6 de COVID-19 entraîne une maladie grave et le développement de difficultés respiratoires (OMS 2020 D). Les personnes les plus susceptibles de développer une maladie grave sont les personnes âgées et les personnes ayant des problèmes médicaux sous-jacents.

2.2.4 Effets de la pandémie de COVID-19 dans la vie quotidienne.

COVID-19 a rapidement affecté notre vie quotidienne, nos entreprises, perturbé le commerce et les mouvements mondiaux. L'identification de la maladie à un stade précoce est vitale pour contrôler la propagation du virus, car il se propage très rapidement d'une personne à l'autre. La plupart des pays ont ralenti leur fabrication des produits.[11] Les différentes industries et secteurs sont touchés par la cause de cette maladie ; il s'agit notamment de l'industrie pharmaceutique, du secteur de l'énergie solaire, du tourisme, de l'industrie de l'information et de l'électronique. Ce virus crée des effets d'entraînement importants sur la vie quotidienne des citoyens, ainsi que sur l'économie mondiale.
Actuellement, les impacts de COVID-19 dans la vie quotidienne sont importants et ont des conséquences de grande envergure. Ceux-ci peuvent être divisés en différentes catégories[12] :

- Soins de santé :
  - Défis dans le diagnostic, la quarantaine et le traitement des cas suspects ou confirmés.
  - Fardeau élevé du fonctionnement du système médical existant.
  - Les patients atteints d'autres maladies et problèmes de santé sont négligés.
  - Surcharge des médecins et autres professionnels de la santé, qui courent un risque très élevé.

  - Surcharge des magasins médicaux
  - Exigence de haute protection.
  - Perturbation de la chaîne d'approvisionnement médicale.
- Social :
  - Le secteur des services n'est pas en mesure de fournir son service approprié.
  - Annulation ou report de sports et de tournois à grande échelle.
  - Éviter les déplacements nationaux et internationaux et l'annulation des services.
  - Perturbation de la célébration d'événements culturels, religieux et festifs.
  - Stress excessif parmi la population.
  - Distanciation sociale avec nos pairs et les membres de la famille.
  - Fermeture des hôtels, restaurants et lieux religieux.
  - Fermeture de lieux de divertissement tels que cinémas et théâtres, clubs sportifs, gymnases, piscines, etc.
  - Report des examens.
- Économique :
  - Ralentissement de la fabrication des biens de première nécessité.
  - Perturber la chaîne d'approvisionnement des produits.
  - Pertes dans les affaires nationales et internationales.
  - Faible flux de trésorerie sur le marché.
  - Ralentissement significatif de la croissance du chiffre d'affaires.

Cette COVID-19 a affecté les sources d'approvisionnement et affecte l'économie mondiale. Il existe des restrictions pour voyager d'un pays à un autre. Pendant le voyage, un certain nombre de cas sont identifiés comme positifs lors des tests, en particulier lorsqu'ils effectuent des visites internationales.5 Tous les gouvernements, organisations de santé et autres autorités se concentrent en permanence sur l'identification des cas touchés par la COVID-19. Les professionnels de la santé sont confrontés à de nombreuses difficultés pour maintenir la qualité des soins de santé de nos jours.

2.3 Les différents types de vaccins contre la COVID-19 :

En décembre 2020, plus de 200 vaccins candidats contre la COVID-19 étaient en cours de développement. Sur ceux-ci, au moins 52 vaccins candidats sont au stade des essais sur l'homme. Plusieurs autres vaccins candidats sont actuellement en phase I/II, et entreront dans la phase III au cours des prochains mois[2].

2.3.1 Pourquoi y a-t-il autant de vaccins en développement ?

En général, de nombreux vaccins candidats feront l'objet d'une évaluation avant d'être considérés comme sûrs et efficaces. Par exemple, sur l'ensemble des vaccins qui sont étudiés

chez les animaux de laboratoire et en laboratoire, environ sept sur 100 seront considérés comme suffisamment efficaces pour passer au stade des essais cliniques chez l'homme. Sur les vaccins qui parviennent à l'étape des essais cliniques, seul un sur cinq aboutit à un succès. Le fait d'avoir plusieurs vaccins différents en cours de développement augmente les chances de trouver un ou plusieurs vaccins efficaces et sûrs pour les populations prioritaires visées.

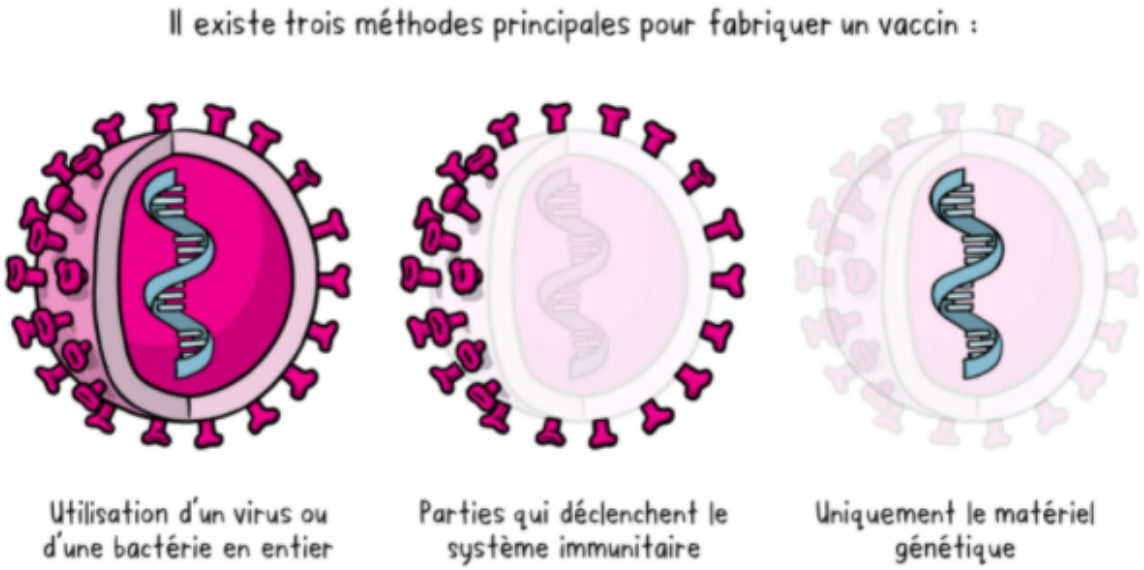


FIGURE 2.1: Les méthodes principales pour fabriquer un vaccin.[2]

2.4 Les différents types de vaccins :

Il existe trois méthodes principales de fabrication d'un vaccin. Leurs différences résident dans la question de savoir s'ils utilisent un virus ou une bactérie en entier ; uniquement les parties du germe qui déclenche le système immunitaire· ; ou uniquement le matériel génétique qui fournit les instructions pour la fabrication de protéines spécifiques et non pas le virus en entier.

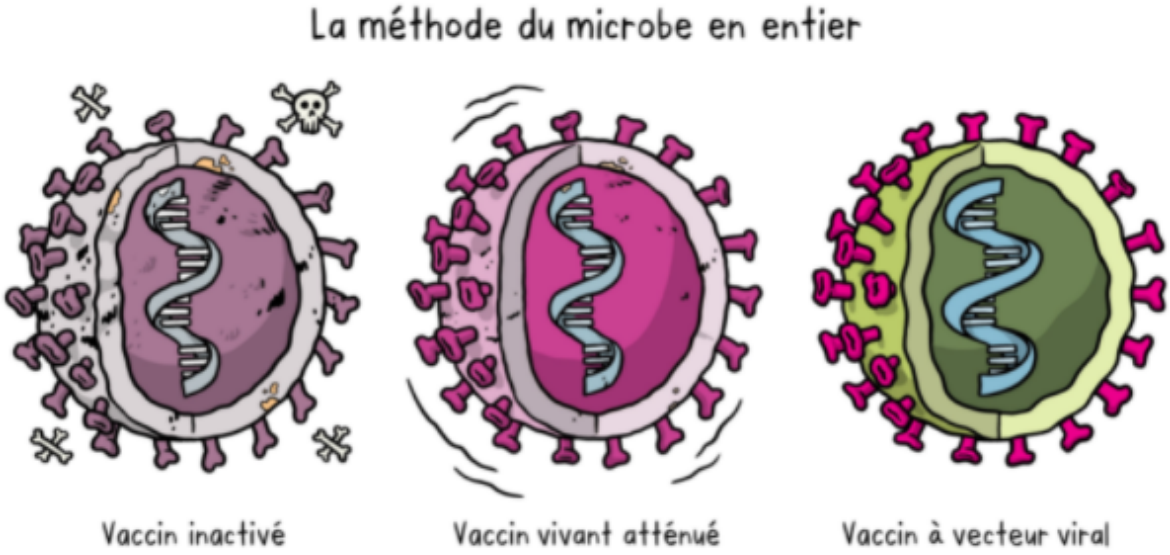


FIGURE 2.2: Les méthodes du microbe en entier.[2]

2.4.1 Vaccin inactivé :

La première façon de fabriquer un vaccin est de prendre le virus ou la bactérie porteuse de la maladie, ou un très semblable à celui-ci, et de l'inactiver ou de le tuer à l'aide de produits chimiques, de chaleur ou de rayonnements. Cette méthode utilise une technologie qui a fait ses preuves chez l'homme – en effet, c'est ainsi que sont fabriqués les vaccins contre la grippe et la poliomyélite – et les vaccins peuvent être fabriqués à une échelle raisonnable.
Toutefois, cette méthode nécessite des installations de laboratoire spéciales pour cultiver le virus ou la bactérie en toute sécurité, elle peut aussi avoir un temps de production relativement long, et le vaccin qui en sera issu sera probablement administré en deux ou trois doses.

2.4.2 Vaccin vivant atténué :

Un vaccin vivant atténué utilise une version vivante, mais affaiblie du virus ou une version très similaire. Le vaccin antirougeoleux-anti-ourlien-antirubéoleux (ROR) et le vaccin contre la varicelle et le zona sont des exemples de ce type de vaccin. Cette méthode utilise une technologie similaire au vaccin inactivé et peut être fabriquée à grande échelle. Cependant, ce type de vaccins ne convient pas aux personnes dont le système immunitaire est affaibli.

2.4.3 Vaccin à vecteur viral :

Ce type de vaccin utilise un virus sûr pour fournir des sous-parties spécifiques – appelées protéines – du germe voulu afin qu'il puisse déclencher une réponse immunitaire sans provoquer de maladie. Pour ce faire, les instructions pour la fabrication de ces fragments particuliers de

l'agent pathogène voulu sont injectées dans un virus sûr. Le virus sûr sert alors de plateforme ou de vecteur pour relâcher la protéine dans l'organisme. La protéine déclenche la réponse immunitaire. Le vaccin contre Ebola est un vaccin à vecteur viral et ce type de vaccin peut être développé rapidement.

FIGURE 2.3: La méthode des sous-unités.[2]

Un vaccin sous-unité n'utilise que les parties très spécifiques (les sous-unités) d'un virus ou d'une bactérie que le système immunitaire doit reconnaître. Il ne contient pas le microbe en entier et n'utilise pas un virus sûr comme vecteur. Les sous-unités peuvent être des protéines ou des sucres. La plupart des vaccins figurant sur le calendrier d'administration pendant l'enfance sont des vaccins sous-unité, protégeant les individus contre des maladies telles que la coqueluche, le tétanos, la diphtérie et la méningite à méningocoque.**La méthode des sous-unités**

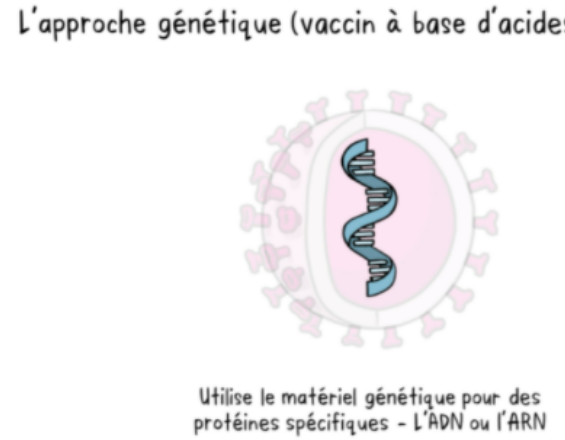


FIGURE 2.4: La méthode des sous-unités.[2]

Contrairement aux méthodes qui utilisent un microbe en entier affaibli ou mort ou des parties d'un microbe, un vaccin à base d'acides nucléiques utilise simplement un fragment de matériel génétique qui fournit les instructions pour des protéines spécifiques, et non pas le microbe en entier. L'ADN et l'ARN sont les instructions que nos cellules utilisent pour fabriquer des protéines. Dans nos cellules, l'ADN est d'abord transformé en ARN messager, qui est ensuite utilisé comme modèle pour fabriquer des protéines spécifiques.**L'approche génétique (vaccin à base d'acides nucléiques)**

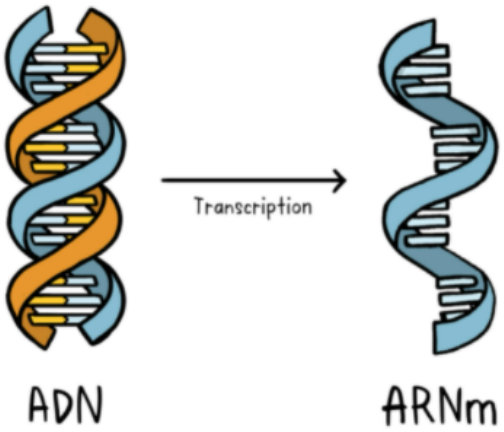


FIGURE 2.5: ADN et ARNm.[2]

Un vaccin à base d'acides nucléiques fournit un ensemble spécifique d'instructions à nos cellules, que ce soit sous forme d'ADN ou d'ARNm, pour que celles-ci fabriquent la protéine spécifique que nous souhaitons que notre système immunitaire puisse reconnaître et combattre. La technique à base d'acides nucléiques est une nouvelle façon de développer des vaccins. Avant la pandémie de COVID-19, aucun vaccin de ce type n'avait encore été soumis au processus complet d'approbation pour une utilisation chez l'homme, bien que certains vaccins à ADN, y compris contre certains cancers, faisaient l'objet d'essais sur l'homme. En raison de la pandémie, la recherche dans ce domaine a progressé très rapidement et certains vaccins à ARNm contre la COVID-19 obtiennent une autorisation d'utilisation d'urgence, ce qui signifie qu'ils peuvent désormais être administrés à des personnes, au-delà d'une utilisation uniquement dans le cadre d'essais cliniques.

2.5 Intelligence artificielle, solutions technologiques contre COVID-19 :

Les algorithmes d'intelligence artificielle (IA) ont montré un grand potentiel pour prédire, diagnostiquer, classer les images et les tendances épidémiologiques des maladies. Ainsi, l'application de l'IA peut être immédiatement appliquée pour lutter contre la COVID-19. Cependant, la lutte contre la COVID-19 dépend de nombreuses variables, notamment un diagnostic rapide, un dépistage, une stratification précise des patients sévères et des traitements appropriés.
La table 2.1 montre la contribution des modèles d'IA pour lutter contre la COVID-19 :

Application	Modèle	Description
Identifier les médicaments disponibles qui pourrait agir sur les protéines virales de SARS-CoV-2 utilisant la molécule Transformateur-Drogue Cible Interaction (MT-DTI)	Transférer l'apprentissage et amarrage moléculaire	les Applications de l'IA pour la réorientation des médicaments COVID-19
Trouver des médicaments approuvés qui peuvent inhiber COVID-19 en utilisant g a cible médicamenteuse basée sur l'apprentissage en profondeur modèle d'interaction appelée Molécule Transformateur-Drogue Cible Interaction (MT-DTI)	Transférer l'apprentissage et amarrage moléculaire	les Applications de l'IA pour la réorientation des médicaments COVID-19
Identifier la thérapeutique antivirale cibles pour la réutilisation de médicaments par en utilisant la tige DeepNEU plateforme cellulaire et validée simulations informatiques d'artificiel cellules pulmonaires.	Machine hybride profonde système d'apprentissage avec éléments entièrement connectés RNN, CM et systèmes évolutifs (GA)	les Applications de l'IA pour la réorientation des médicaments COVID-19
Identifier les médicaments potentiels pour SARS-CoV-2 utilisant la machine algorithmes d'apprentissage	Algorithmes d'apprentissage automatique	les Applications de l'IA pour la réorientation des médicaments COVID-19
Classification de COVID-19 et normal	modèle de détection de la COVID-19	CNN
Classification de COVID-19, pneumonie et sain	modèle de détection de la COVID-19	CNN
Classification de COVID-19, Pneumonie et normal	modèle de détection de la COVID-19	CNN
Gravité de la COVID-19	prédicteurs de la gravité de la maladie	GBM

Gravité de la COVID-19	prédicteurs de la gravité de la maladie	RNN
Gravité de la COVID-19	prédicteurs de la gravité de la maladie	CNN
Risque de mortalité	prédire la mortalité de COVID-19, en utilisant un grand nombre de données cliniques, de laboratoire et d'images.	ANN
Risque de mortalité	prédire la mortalité de COVID-19, en utilisant un grand nombre de données cliniques, de laboratoire et d'images.	SVM
Risque de mortalité	prédire la mortalité de COVID-19, en utilisant un grand nombre de données cliniques, de laboratoire et d'images.	Ensemble modèle
Risque de mortalité	prédire la mortalité de COVID-19, en utilisant un grand nombre de données cliniques, de laboratoire et d'images.	LR

TABLE 2.1: Application de l'IA contre COVID-19

2.6 Conclusion

L'application de l'IA dans la lutte contre la pandémie a montré un grand potentiel de diverses manières, notamment en prédisant la tendance épidémique, en suivant les patients, en stratifiant les patients asymptomatiques et en trouvant des médicaments potentiels réutilisés. Toutes les études présentaient un manque de taille d'échantillon, une validation externe et une évaluation de modèle inappropriée· ; par conséquent, l'utilisation de ces résultats serait une décision optimiste. Les recherches futures avec un échantillon de grande taille et une interprétation appropriée pourraient être évaluées à l'aide de plusieurs ensembles de données avant de les considérer dans le cadre clinique du monde réel. De plus, les candidats médicaments réutilisés peuvent également être évalués par des expériences cliniques. De plus, des études sont nécessaires pour évaluer l'efficacité réelle des modèles d'IA et calculer la rentabilité dans la pratique clinique. Pour avoir le vrai goût de l'IA pour lutter contre COVID-19.

CHAPITRE 3

DEEP LEARNING

3.1 Introdiction

L'IA est basée sur une démarche d'apprentissage afin de reproduire une partie de l'intelligence humaine à travers une application, un système ou un processus. La reconnaissance faciale, la perception visuelle et autre sont des exemples de systèmes d'intelligence artificielle.
La machine Learning (ML) est un sous-domaine de l'IA qui utilise les réseaux neuronaux artificiels (ANN) pour imiter la façon dont les êtres humains prennent des décisions. La machine Learning permet aux ordinateurs de développer des modèles d'apprentissage par eux-mêmes, sans aucune programmation, à partir de gros ensembles de données. La couche immédiatement inférieure est occupée par le Deep Learning (DL), est l'une des nombreuses approches de la machine Learning qui connue un grand succès dans ces cinq dernières années, Pour illustrer la relation entre ces termes, nous pouvons utiliser des cercles concentriques :

— Intelligence artificielle IA (intelligence artificielle) : le cercle plus large est l'idée qui a émergé en premier dans ce domaine.
— Apprentissage automatique (machine Learning) : au milieu, il a prospéré plus tard après l'IA.
— Apprentissage approfondi (Deep Learning) : le plus petit cercle est une expansion de l'IA actuellement.

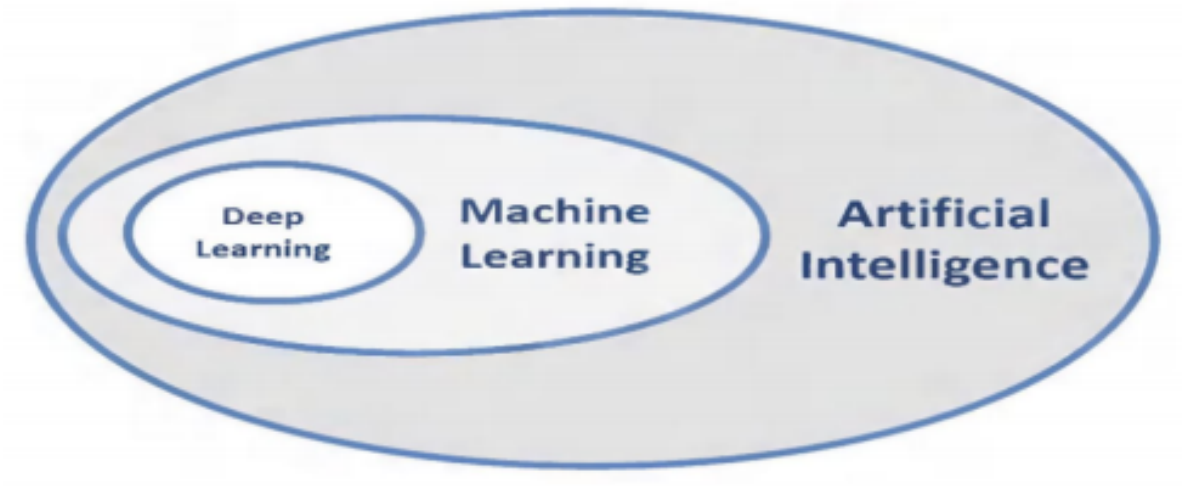


FIGURE 3.1: La relation Entre L'IA et ML et le Deep Learning.

3.1.1 Intelligence artificielle :

L'intelligence artificielle est née dans les années 50, quand une poignée de pionniers du domaine naissant de l'informatique, ont commencé à se demander si les ordinateurs pouvaient être amenés à « penser », une question dont nous explorons encore aujourd'hui les ramifications. Une définition concise du champ serait la suivante : l'effort d'automatiser les tâches intellectuelles normalement effectuées par les humains.
En tant que tel, l'IA est un domaine général qui englobe l'apprentissage automatique et l'apprentissage en profondeur, mais qui comprend également beaucoup plus d'approches qui n'impliquent aucun apprentissage. Les programmes d'échecs initiaux, par exemple, ne concernaient que des règles codées en dur élaborées par des programmeurs et ne se qualifiaient pas comme apprentissage automatique.
Pendant un temps assez long, de nombreux experts ont estimé que l'intelligence artificielle au niveau humain, pouvait être obtenue en faisant en sorte que les programmeurs fabriquent à la main un ensemble suffisamment large de règles explicites pour manipuler les connaissances.
Cette approche est connue sous le nom d'IA symbolique et elle était le paradigme dominant de l'IA des années 50 et à la fin des années 80. Elle a atteint son pic de popularité durant le boom des systèmes experts des années 80.
Bien que l'IA symbolique se soit révélée appropriée pour résoudre des problèmes logiques bien définis, comme jouer aux échecs, il était difficile de trouver des règles explicites pour résoudre des problèmes flous plus complexes, tels que la classification des images, la reconnaissance de la parole et la traduction. Une nouvelle approche est apparue pour prendre la place de l'IA symbolique : l'apprentissage automatique[13].

3.1.2 Machin learning :

L'apprentissage automatique est un domaine de recherche de l'informatique Il s'agit de méthodes d'identification et de mise en œuvre de systèmes et d'algorithmes Les ordinateurs peuvent apprendre, ce domaine est généralement lié à l'intelligence artificielle, plus précisément l'intelligence computationnelle.
L'intelligence informatique est une méthode d'analyse de données qui Créez automatiquement des modèles d'analyse. En d'autres termes, permettez à l'ordinateur Développer des concepts, évaluer, prendre des décisions et planifier des choix futurs[14].
L'ensemble du processus d'apprentissage nécessite un ensemble de données comme suit : Ensemble de données pour l'entrainement :c'est la base de connaissance utilisée pour entrainer, notre l'algorithme d'apprentissage, pendant cette phase, les paramètres du modèle peuvent être réglés (ajustés) en fonction des performances obtenues.
Ensemble de données pour le test : cela est utilisé juste pour évaluer les performances du modèle sur les données non-vues La théorie de l'apprentissage utilise des outils mathématiques dérivés de la théorie des probabilités et de la théorie de l'information. Cela vous permet d'évaluer l'optimalité de certaines méthodes par rapport aux autres.
On peut citer trois types d'algorithme d'apprentissage automatique :

— Apprentissage supervisé.
— Apprentissage non supervisé.
— Apprentissage par renforcement.

3.1.2.1 Apprentissage supervisé :

L'apprentissage supervisé est la tâche d'apprentissage automatique la plus simple et la plus connue. Il est basé sur un certain nombre d'exemples pré classifiés, dans lesquels est connu a priori la catégorie à laquelle appartient chacune des entrées utilisées comme exemples. Dans ce cas, la question cruciale est le problème de généralisation, après l'analyse d'un échantillon d'exemples, le système devrait produire un modèle qui devrait fonctionner pour toutes les entrées possibles.
L'ensemble de données pour l'entrainement, est constitué de données étiquetées, c'est-à-dire d'objets et de leurs classes associées. Cet ensemble d'exemples étiquetés constitue donc l'ensemble d'apprentissage.
Afin de mieux comprendre ce concept, prenons un exemple : un utilisateur reçoit chaque jour un grand nombre d'e-mails, certains sont des e-mails d'entreprises importants et d'autres sont des e-mails indésirables non sollicités ou des spams.
Un algorithme supervisé sera présenté avec un grand nombre d'e-mails qui ont déjà été étiquetés par l'utilisateur comme spam ou non spam. L'algorithme fonctionnera sur toutes les données étiquetées, faire des prédictions sur l'e-mail et voir si c'est un spam ou non.

Cela signifie que l'algorithme examinera chaque exemple et fera une prédiction pour chacun pour savoir si l'e-mail est un spam ou pas. La première fois, l'algorithme fonctionne sur toutes les données non étiquetées, la plupart des e-mails seront mal étiquetés, car il peut fonctionner assez mal au début. Cependant, après chaque exécution, l'algorithme compare sa prédiction au résultat souhaité (l'étiquette). Au fur et à mesure, l'algorithme apprendra à améliorer ses performances et sa précision.
Dans l'exemple que nous avons utilisé, nous avons décrit un processus dans lequel un algorithme apprend à partir de données étiquetées (emails qui ont été catégorisés comme spam ou non-spam). Dans certains cas, le résultat n'est pas nécessairement discret et il se peut que nous n'ayons pas un nombre fini de classes dans lesquelles classer nos données. Par exemple, nous essayons peut-être de prédire l'espérance de vie d'un groupe de personnes en fonction de paramètres de santé préétablis. Dans ce cas, comme le résultat est une fonction continue (nous pouvons spécifier une espérance de vie comme un nombre réel exprimant le nombre d'années que la personne devrait vivre), nous ne parlons pas d'une tâche de classification, mais plutôt d'un problème de régression.
Dans un problème de régression, l'ensemble d'apprentissage est une paire formée par un objet et une valeur numérique associée. Il existe plusieurs algorithmes d'apprentissage supervisé qui ont été développés pour la classification et la régression. Parmi tous, les arbres de décision, les règles de décision, les réseaux de neurones et les réseaux bayé siens. [14] [15][16]

3.1.2.2 Apprentissage non supervisé :

La deuxième classe d'algorithmes d'apprentissage automatique est appelée apprentissage non supervisé, dans ce cas, nous n'étiquetons pas les données au préalable, nous laissons plutôt l'algorithme arriver à sa conclusion.
Ce type d'apprentissage est important, car il est beaucoup plus commun dans le cerveau humain que l'apprentissage supervisé.
Les algorithmes d'apprentissage non supervisé sont particulièrement utilisés pour les problèmes de clustering, où, étant donné un ensemble d'objets, nous voulons être en mesure de comprendre et d'afficher la relation entre eux. La méthode standard consiste à définir une mesure de similarité entre deux objets, puis à trouver tout groupe d'objets plus similaires les uns aux autres que les objets d'autres clusters.
Par exemple, dans le cas précédent de spam/non-spam, l'algorithme peut être en mesure de trouver les points communs de tous les messages de spam (par exemple, il y a des mots mal orthographiés). Bien que cela puisse fournir une meilleure classification que la classification aléatoire, il n'est pas clair si le spam/non-spam peut être facilement distingué.[14] [15][16]

3.1.2.3 Apprentissage par renforcement :

L'apprentissage par renforcement est une méthode d'intelligence artificielle qui met l'accent sur l'apprentissage du système par l'interaction entre le système et l'environnement. Grâce à l'apprentissage par renforcement, le système ajuste ses paramètres en fonction de la rétroaction reçue de l'environnement, puis fournit une rétroaction sur la décision.
Par exemple, un système qui simule un joueur d'échecs qui utilise les résultats des étapes précédentes pour améliorer ses performances est un système par apprentissage par renforcement. La recherche actuelle sur l'apprentissage par renforcement est hautement interdisciplinaire et comprend des chercheurs spécialisés dans les algorithmes génétiques, les réseaux de neurones, la psychologie et la technologie de contrôle. [14] [16]

3.2 Apprentissage profond ou Deep Learning :

3.2.1 Définition :

Le Deep Learning ou apprentissage profond est un sous-domaine de l'apprentissage automatique qui concerne les algorithmes inspirés de la structure et de la fonction du cerveau, appelés réseaux de neurones artificiels (La pratique, de tous les algorithmes de DL sont des réseaux neuronaux [9]).

3.2.2 Histoire de Deep Learning :

Année	Contributeur	Contribution
300 AC	Aristotle	introduction de l'associationnisme, début de l'histoire des humains qui essayent de comprendre le cerveau
1873	Alexander Bain	introduction du Neural Groupings comme les premiers modèles de réseaux de neurones
1943	McCulloch and Pitts	introduction du McCulloch–Pitts (MCP) modèle considérer comme L'ancêtre des réseaux de neurones artificielles
1949	Donald Hebb	considérer comme le père des réseaux de neurones, il introduit la règle d'apprentissage de Hebb qui servira de fondation pour les réseaux de neurones modernes
1958	Frank Rosenblatt	introduction du premier perceptron
1974	Paul Werbos	introduction de la retro propagation
1980	Teuvo Kohonen	introduction des cartes auto organisatrices
1980	Kunihiko Fukushima	introduction du Neocognitron, qui a inspiré les réseaux de neurones convolutifs
1982	John Hopfield	introduction des réseaux de Hopfield
1985	Hilton and Sejnowski	introduction des machines de Boltzmann
1986	Paul Smolensky	introduction d'Harmonium, qui sera connu plus tard comme machines de Boltzmann restreintes
1986	Michael I. Jordan	définition et introduction des réseaux de neurones récurrente
1990	Yann LeCun	introduction de LeNet et montra la capacité des réseaux de neurones profond
1997	Schuster and Paliwal	introduction des réseaux de neurones récurrente bidirectionnelle
1997	Hochreiter and Schmidhuber	introduction de LSTM, qui ont résolu le problème du vanishing gradient dans les réseaux de neurones récurrent
2006	Geoffrey Hinton	introduction des Deep belief Network
2009	Salakhutdinov and Hinton	introduction des Deep Boltzmann Machines
2012	Alex Krizhevsky	introduction de AlexNet qui remporta le challenge ImageNet

TABLE 3.1: Les étapes majeures du Deep Learning [1]

3.3 Réseaux de neurones artificiels :

3.3.1 Définition :

Les réseaux de neurones sont des modèles d'apprentissage automatique capables de représenter une relation entre des données d'un espace X et un espace de sortie Y. C'est un modèle de traitement de l'information qui simule le fonctionnement d'un système nerveux biologique. C'est

similaire à la façon dont le cerveau manipule l'information.

3.3.2 Les neurones :

3.3.2.1 Le neurone biologique :

Il est connu que le cerveau humain est formé de 86 à 100 milliards de neurones (cellule nerveuse). Chaque neurone est connecté en moyenne, à plusieurs milliers d'autres neurones. Ces connexions sont appelées synapses.
Les neurones sont des unités simples de traitement de transmission des signaux chimiques et électriques. Chaque neurone est formé d'un corps appelé soma, un axone et un certain nombre de dendrites.

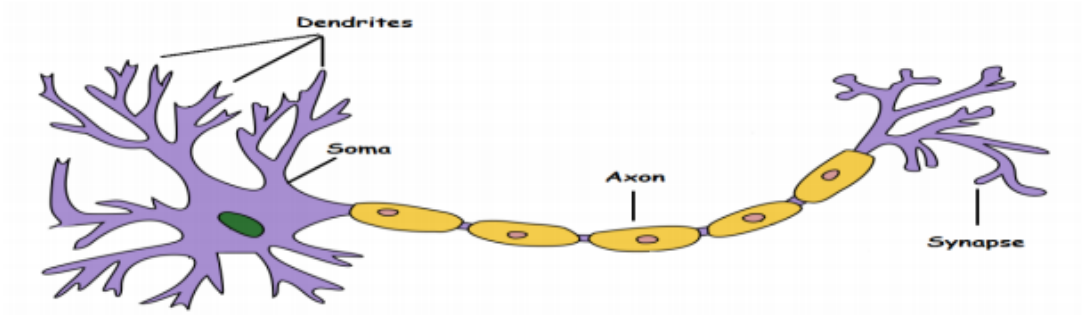


FIGURE 3.2: Présente la forme du neurone biologique.

Le neurone reçoit de l'information (entrées) des autres neurones via ses dendrites, et lorsque ce signal d'entrée dépasse un certain seuil, le neurone "faire ou se déclenche" - en fait, une réaction chimique se produit, ce qui provoque l'envoi d'une impulsion électrique, connue sous le nom de potentiel d'action. En bas de l'axone (la sortie du neurone), vers des synapses qui connectent le neurone aux dendrites d'autres neurones. Donc le réseau de neurones résultat de l'interconnexion de ces neurones simple est extrêmement complexe et puissant.

3.3.2.2 Neurone artificiel :

Est un modèle mathématique du neurone biologique inventé par McCulloch and Pitts (1943) et il est aussi appelé le neurone de McCulloch et Pitts (M-P).
Un neurone artificiel imite un neurone biologique dans certaines fonctions. Ces systèmes neuronaux sont présentés à l'aide des notions mathématiques. Les signaux en entrée sont formulés par les valeurs $x_i, i[1,N], iN$, les dendrites sont les poids W_i, le corps cellulaire, c'est la fonction de combinaison et l'axone, c'est la valeur en sortie. Un neurone artificiel peut être composé

d'un ensemble de nœuds ou de neurones qui sont associés à une valeur x_i et un poidsW_i. Ces valeurs sont combinées par une fonction de combinaison et le résultat est traité par une fonction d'activation f.

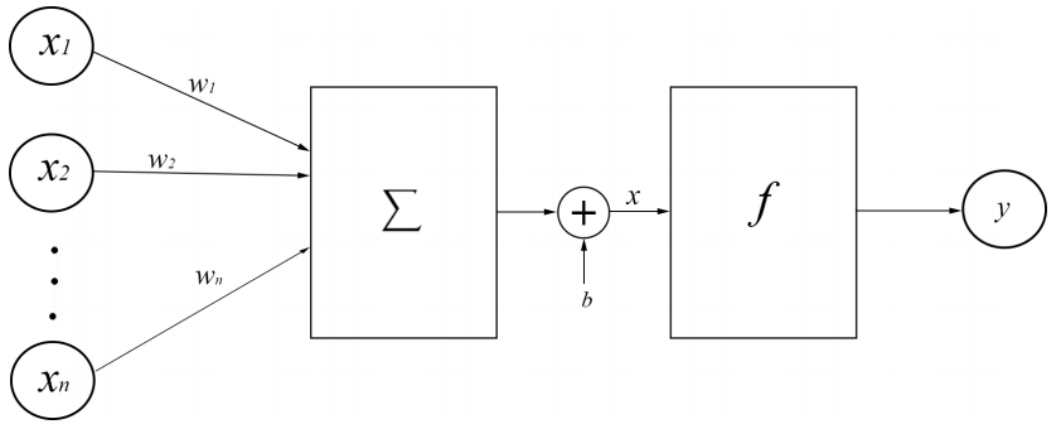


FIGURE 3.3: un neurone artificiel.

La sortie du neurone y, appelée activation de sortie, est calculée selon la formule suivante :

$$y = f(\sum_{i=0}^{n} w_i x_i + b) \tag{3.1}$$

Où f est la fonction d'activation

3.3.3 Les fonctions d'activation :

La fonction d'activation est une caractéristique importante du réseau neuronal. Neurone devrait être activé ou non est décidé par la fonction d'activation. Il calcule la somme pondérée des entrées et ajoute le biais. C'est une transformation non linéaire de la valeur d'entrée. Après la transformation, cette sortie est envoyée à la couche suivante. Sans une fonction d'activation, un réseau de neurones est juste un modèle de régression linéaire.

3.3.3.1 Types de fonction d'activation :

La Fonction sigmoïde :

sont les fonctions les plus utilisées dans la création de réseaux neurones artificiels. Prend une entrée réelle et la réduit entre 0 et 1.

$$f(x) = \frac{1}{1+e^{-x}} \tag{3.2}$$

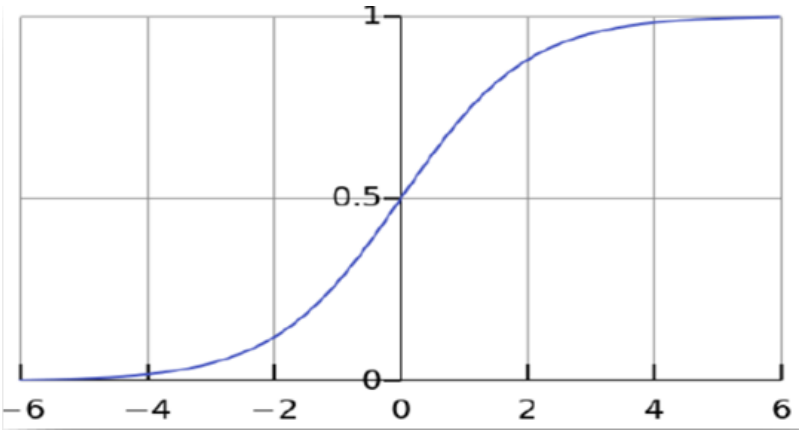


FIGURE 3.4: Représentation du graphe de fonction sigmoïde.

[17]

La Fonction ReLu : C'est une fonction d'activation très simple. Supposons que l'entrée est la valeur X et si X est positif, la sortie sera X sinon 0. La fonction ReLu (unité linéaire rectifiée) est :

(3.3) $$f(x) = max(0, X)$$

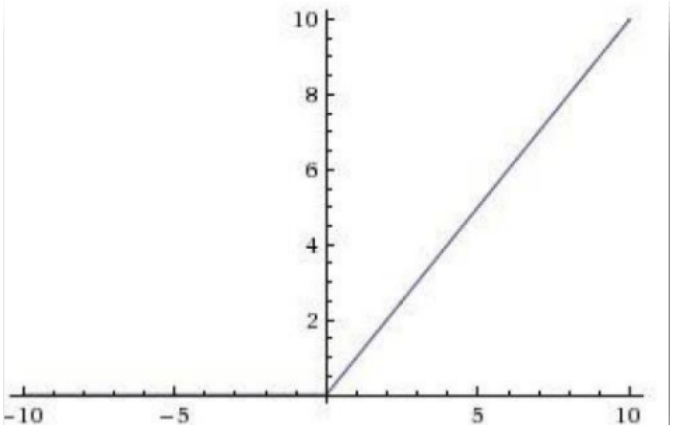


FIGURE 3.5: Représentation graphique de la fonction ReLu.

[17]

La fonction Softmax : Elle réalise une exponentielle normalisée (c'est-à-dire que la somme des sorties totalise 1). En combinaison avec la fonction d'erreur d'entropie croisée, elle permet de modifier les réseaux perceptrons multicouches pour l'estimation des probabilités de classes.

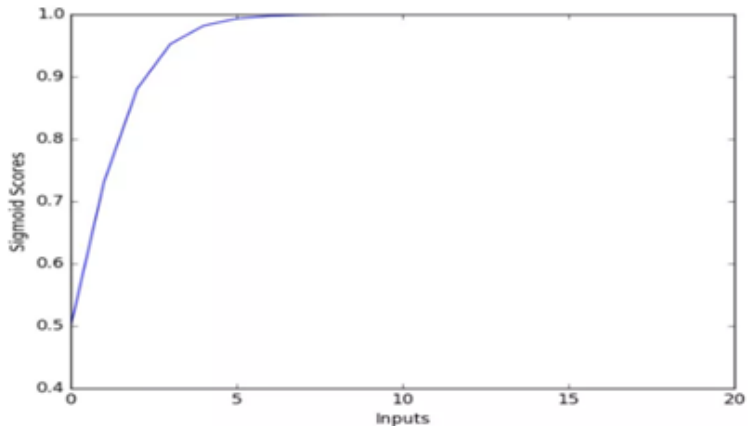


FIGURE 3.6: Représentation du graphe de fonction softmax.

[17]

$$F(x) = \frac{e^{x_i}}{\sum_{i=1}^{k} e^{x_i}} \tag{3.4}$$

La fonction Tangente hyperbolique : prend une entrée de valeur réelle et la réduit à une valeur dans [-1, 1].

$$F(x) = tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}} \tag{3.5}$$

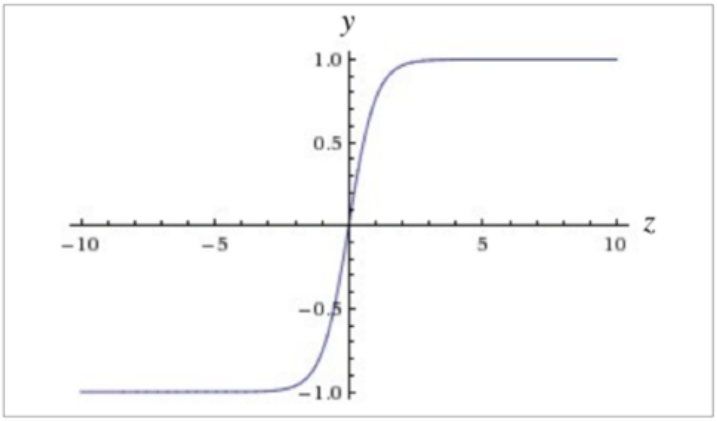


FIGURE 3.7: Représentation du graphe de fonction Tangente hyperbolique.

[17]

3.3.4 Les différents types de modèles

Il existe un grand nombre de variantes d'architectures profondes. La plupart d'entre elles sont dérivées de certaines architectures parentales originales. Il n'est pas toujours possible de

comparer les performances de toutes les architectures, car elles ne sont pas toutes évaluées sur les mêmes ensembles de données. Le Deep Learning est un domaine à croissance rapide, et de nouvelles architectures, variantes ou algorithmes apparaissent toutes les semaines.

3.3.4.1 Les réseaux de neurones convolutionnels :

Les réseaux de neurones convolutifs sont à ce jour les modèles les plus performants pour des images. Désignés par l'acronyme CNN, (CNN pour Convolutional Neural Network), ils comportent deux parties bien distinctes. En entrée, une image est fournie sous la forme d'une matrice de pixels. Elle a deux dimensions pour une image aux niveaux de gris.
La couleur est représentée par une troisième dimension, de profondeur 3 pour représenter les couleurs fondamentales [Rouge, Vert, Bleu]
La première partie d'un CNN est la partie convolutive à proprement parler. Elle fonctionne comme un extracteur de caractéristiques des images. Une image est passée à travers d'une succession de filtres ou noyaux de convolution, créant de nouvelles images appelées cartes de convolutions. Certains filtres intermédiaires réduisent la résolution de l'image par une opération de maximum local. Enfin, les cartes de convolutions sont mises à plat et concaténées en un vecteur de caractéristiques, appelé code CNN.
Ce code CNN en sortie de la partie convolutive est ensuite branché en entrée d'une deuxième partie, constituée de couches entièrement connectées. Le rôle de cette partie est de combiner les caractéristiques du code CNN pour classer l'image.
La sortie est une dernière couche comportant un neurone par catégorie. Les valeurs numériques obtenues sont généralement normalisées entre 0 et 1, de somme 1, pour produire une distribution de probabilité sur les catégories [3].

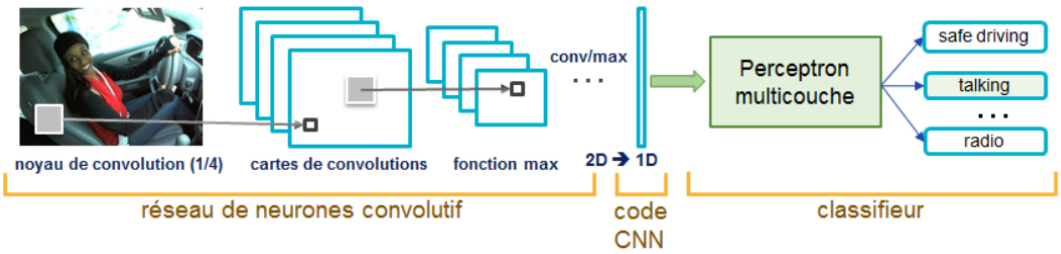


FIGURE 3.8: Architecture standard d'un réseau de neurone convolutionnel [3]

3.3.4.2 Les réseaux neuronaux récurrents (RNN)

Les réseaux neuronaux récurrents (RNNs) sont des réseaux de neurones qui comportent des cycles dans leur graphe de connectivité. Ces cycles permettent au réseau d'entretenir une information en se l'envoyant à lui-même. Cela change la dynamique du réseau de neurones et l'amène à s'autoentretenir. Ces modèles étaient souvent plébiscités notamment pour le traitement

automatique de la parole, et plus généralement de séquences, car leurs caractéristiques leur permettent d'apprendre, de stocker et de prendre en compte l'information contextuelle passée lors de traitement de l'information à l'instant présent [18].
Si la séquence que nous traitons est une phrase de 3 termes par exemple, le réseau va être déroulé en un réseau neuronal à 3 couches, une couche pour chaque mot, la figure suivante représente cette idée :

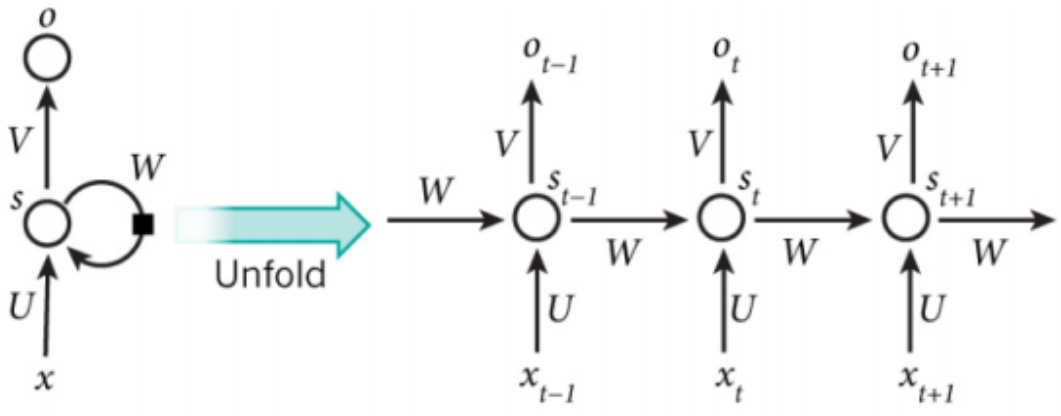


FIGURE 3.9: Un exemple d'un réseau récurrent qui se déroule[4]

3.3.4.3 Feed-forward Neural Networks (FNN)

C'est le réseau hautement interconnecté de neurones artificiels inspiré par le système nerveux humain, travaillant en union pour effectuer une tâche donnée et, à la fin, donne la décision basée sur les poids et les bias pour les données d'entrée complexes.

3.3.4.4 AutoEncoder (AE)

Il s'agit généralement d'un réseau de neurones de type Feed-Forward qui vise à apprendre une représentation compressée et distribuée (encoding) d'un dataset.

3.3.4.5 Deep Belief Networks (DBN)

Il s'agit d'un modèle génératif composé de plusieurs couches de RBM et d'autoencodeurs. Lorsqu'il est apprend sans supervision, il peut apprendre à restreindre ses entrées de manière probabiliste et agit comme des détecteurs de features et avec supervision pour effectuer la classification.
Il existe d'autre réseaux neuronaux, une excellente ressource qui résume peut-être toutes les architectures sont en[5]. La figure suivante montre une représentation de ce travail :

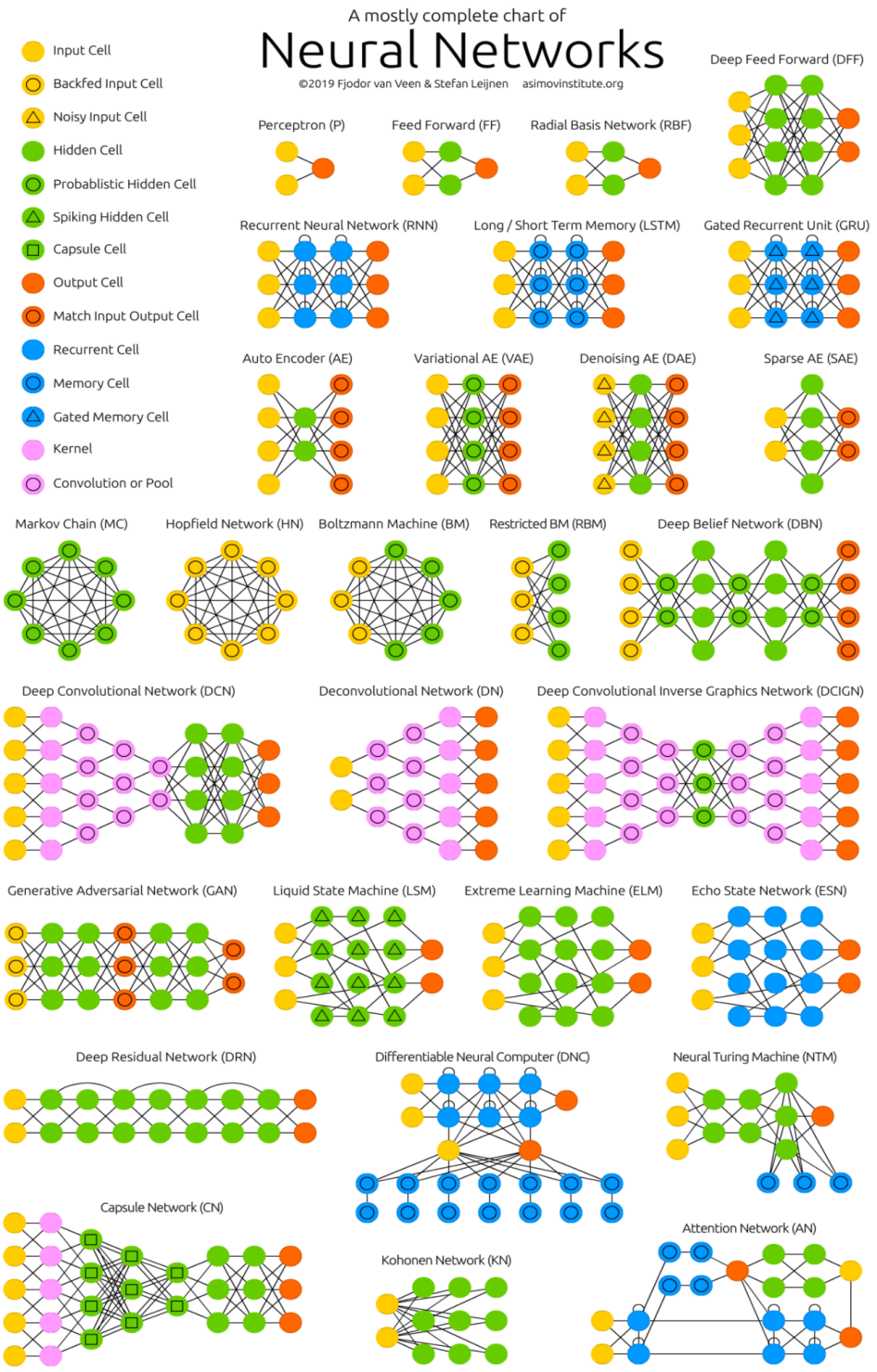


FIGURE 3.10: Un résumé des types d'architectures de réseaux de neurones [5].

3.4 L'apprentissage en Deep Learning .

3.4.1 Gradient Descent :

En même temps, Bernard Widrow et son élève Ted Hoff ont introduit une règle d'apprentissage légèrement modifiée, appelée μ LMS. Au lieu de corriger les erreurs de classification, ils ont proposé d'utiliser l'erreur carrée comme mesure de la qualité [6].

$$l(y, f(x)) = 1/2\|y - f(x)\|_2^2 = 1/2 \sum_{i=1}^{M} (y^i - f^i(x))^2 \tag{3.6}$$

Et minimiser l'erreur carrée moyenne (MSE)

$$L(w) = 1/N \sum_{i=1}^{N} l(y_i, f(x_i)) \tag{3.7}$$

Sur les objets de l'ensemble d'apprentissage. À cet effet, ils ont suggéré d'utiliser une méthode de gradient descent.
Il existe trois principaux types de variantes de l'algorithme de descente de gradient.

3.4.1.1 Batch gradient descent :

C'est la descente de gradient classique, on calcule le gradient de la fonction coût aux paramètres pour tout l'ensemble d'apprentissage· :

$$W = W - \alpha \nabla_W L(W) \tag{3.8}$$

Comme on doit calculer le gradient pour tout l'ensemble de données pour exécuter juste une seule mise à jour, cette méthode peut être très lente et irréalisable si les données ne peuvent pas être stockées en mémoire. Avec cette méthode on est sûr de converger vers l'optimum global si la fonction cout est convexe et vers un optimum local si elle est non convexe.

3.4.1.2 Descente de gradient stochastique :

SGD (la descente de gradient stochastique) met à jour les paramètres pour chaque exemple de l'ensemble de données x^i et label y^i :

$$W = W - \alpha \nabla_W L(x^i, y^i; W) \tag{3.9}$$

Cette méthode est plus rapide, mais les mises à jour des paramètres trop fréquentes causent à la fonction objective des oscillations, ces oscillations d'une part permettent d'atterrir dans des minimums locaux potentiellement meilleurs, mais d'autre part rendent la convergence plus difficile.

3.4.1.3 Mini-batch gradient descent :

Cette méthode prend le meilleur des deux méthodes et met à jour les paramètres pour chaque minigroupes de n exemples :

$$W = W - \alpha \nabla_W L(x^{i;i+n}, y^{i;i+n}; W) \tag{3.10}$$

Cette méthode réduit la variance des mises à jour des paramètres ce qui conduit à une convergence plus stable. Généralement les minigroupes contiennent de 32 à 256 exemples. C'est une méthode de choix pour entraîner un réseau de neurones.

La descente de gradient classique n'offre pas toujours une bonne convergence et pose quelques challenges qui ont besoin d'être résolus :

- Le choix du taux d'apprentissage est difficile, s'il est trop petit cela peut emmener à une convergence trop lente, s'il est trop grand il peut causer des oscillations à la fonction coût voire même ne pas converger du tout.
- Le même taux d'apprentissage est appliqué à tous les paramètres. Si les caractéristiques des observations dont on dispose ont une fréquence différente, on peut vouloir appliquer une plus grande mise à jour aux paramètres pour les caractéristiques qui reviennent plus rarement.
- Un autre challenge de la minimisation des fonctions coût non convexe très courantes dans les réseaux de neurones, c'est éviter d'être piéger dans des optimums locaux.

3.4.2 Algorithmes d'optimisation de la descente de gradient

3.4.2.1 Adagrad

Adagrad [19] est un algorithme d'optimisation basé sur la descente de gradient qui ne fait qu'adapter le taux d'apprentissage aux paramètres, effectuant de plus grandes mises à jour pour les caractéristiques peu fréquentes et de plus petites pour les caractéristiques plus fréquentes. Dean et al.[20] ont trouvé que Adagrad a grandement amélioré la robustesse de SGD et utilisé pour l'apprentissage de grands réseaux de neurones chez Google qui -entre autres- ont appris à reconnaître les chats dans les vidéos Youtube [21]. De plus Pennington et al. [22]ont utilisé Adagrad pour l'apprentissage de GloVe word embeddings, comme les mots les moins fréquents demandent de plus grandes mises à jour que ceux qui sont plus fréquents.
Précédemment, il a été effectué les mises à jour des paramètres θ en utilisant le même taux d'apprentissage η. Adagrad utilise un taux d'apprentissage diffèrent pour chaque paramètre θ i et à chaque étape t.
Soit $g_{t,i}$ le gradient de la fonction objective sachant le paramètre θ i à l'étape t :

$$g_{t,i} = \nabla_\theta j(\theta_i) \tag{3.11}$$

La mise à jour de chaque paramètre θ i à l'étape t devient :

$$\theta_{t+1,i} = \theta_{t,i} - \eta . g_{t,i} \tag{3.12}$$

Dans sa règle de mise à jour, Adagrad modifie le taux d'apprentissage général η à chaque étape t pour chaque paramètre θ i basé sur les gradients passés calculés pour θ i· :

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} . g_{t,i} \tag{3.13}$$

$G_t \in \Re^{d*d}$ Est une matrice diagonale où chaque élément i, i est la somme des carrés de gradient sachant θ i à l'étape t et généralement fixé à $1e8$ est utilisé pour éviter la division par zéro. Fait intéressant, sans l'opération racine carrée, l'algorithme produit des résultats plus mauvais.
Un des avantages de Adagrad c'est qu'il élimine le besoin de modifier manuellement le taux d'apprentissage.
L'inconvénient majeur de Adagrad est l'accumulation des carrés des gradients dans le dénominateur. Comme chaque terme ajouté est positif, l'accumulation des sommes continue d'augmenter au cours de l'apprentissage ce qui va faire baisser le taux d'apprentissage et éventuellement devenir infinitésimalement petit au point que l'algorithme n'est plus capable d'acquérir de nouvelles connaissances.

3.4.2.2 RMSprop

RMSprop est une méthode de taux d'apprentissage adaptative non publiée proposée par Geoff Hinton dans Lecture 6e de sa Coursera Class [23]. Au lieu d'accumuler tous les carrés des gradients précédents, on restreint la fenêtre des gradients accumulés à une taille fixée w.
Au lieu de stocker les w carrés des gradients précédents, on applique une moyenne mobile exponentielle des carrés des gradients précédents. La moyenne courante $E[g^2]_t$ à l'étape t dépend uniquement de la moyenne précédente et de gradient courant.

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} . g_t \tag{3.14}$$

Hinton suggère que γ soit fixé a 0.9, tandis qu'une bonne valeur par défaut pour le taux d'apprentissageη est de 0.001.

3.4.2.3 Adam optimizer

Adaptive Moments (ADAM) optimiser [24] est une autre méthode qui calcule un taux d'apprentissage adaptatif pour chaque paramètre. En plus de stocker une moyenne décroissante exponentielle des précédents carrés des gradients v_t comme RMSprop, ADAM conserve aussi moyenne décroissante exponentielle des précédents gradients m_t.

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$

(3.15) $$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

m_t et v_t sont respectivement des estimations d'ordre 1 (la moyenne) et d'ordre 2(la variance) de gradient. Comme mt et vt sont initialisés comme des vecteurs de 0, les auteurs ont observé qu'ils sont biaisés vers zero surtout durant les premières étapes, et surtout lorsque le coefficient de décroissance est petit ($\beta 1, \beta 2$proche de 1) alors ils ont calculé une correction des biais pour les estimations du premier et du deuxième ordre :

$$m_t = \frac{m_t}{1-\beta_1^t}$$

(3.16) $$v_t = \frac{v_t}{1-\beta_2^t}$$

Puis vient la mise à jour des paramètres θ :

(3.17) $$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{v_t + \epsilon}} m_t$$

Les auteurs proposent des valeurs par défaut de 0.9 pour $\beta 1$, 0.999 pour $\beta 2$ et 10^{-8} pour ϵ

3.4.3 Backpropagation algorithm :

Appliquant les idées de Bryson [25] et Kelley [26] aux perceptrons multicouches, Paul Werbos [27] a proposé une extension de la règle d'apprentissage µLMS, appelée algorithme de backpropagation.
Considérons un perceptron multicouche avec K couches de neurones : y0 = x vecteur d'entrée de taille 1 × N, et les K couches suivantes :

(3.18) $$y_k = \phi_k(y_{k-1}.w_k), \forall k = 1,...,K$$

avec les fonctions d'activation $\phi k(z)$, oùy_K est le vecteur de sortie de la taille 1 × M. Si les fonctions $\phi k(z)$ sont différenciables, nous pouvons calculer les gradients pour la couchey_{k1} en utilisant les gradients pour la couche y_K en utilisant la règle :

(3.19) $$\frac{\partial L}{\partial y_{k-1}^i} = \sum_{j=1}^{M} \frac{\partial L}{\partial y_k^i} . \frac{\partial y_k^j}{\partial y_{k-1}^i} = \sum_{j=1}^{M} \frac{\partial L}{\partial y_k^i} . \left.\frac{\partial \phi_k(z)}{\partial z}\right|_{z=yK-1w_k^j} . w_k^{ij}$$

Ici w_k^j est le vecteur de poids, reliant la couche k−1 au neurone j sur la couche k. Comme nous pouvons le voir, les gradients de la couche précédente $\partial L/\partial y_{k-1}$ dépendent uniquement des

gradients de la couche actuelle $\partial L/\partial y_k$, de la dérivée de la fonction d'activation zk(z) et des poids de couche w_k. Par conséquent, il est possible de calculer de manière itérative les gradients pour toutes les couches de k−1 à 0, en utilisant $\partial J/\partial y_k$ lorde la première itération. Les gradients de poids $\partial L/\partial_{wk}$ peuvent également être calculés à l'aide de la règle :

$$\frac{\partial L}{\partial w_k^{ij}} = \frac{\partial L}{\partial y_k^i} \cdot \frac{\partial y_k^j}{\partial w_k^{ij}} = \frac{\partial L}{\partial y_k^i} \cdot \frac{\partial \phi_k(z)}{\partial z}|_{z=y_{K-1}w_k^j} \cdot y_{k-1}^i \tag{3.20}$$

et peut donc être obtenu de manière itérative pour toutes les couches de K à 1 une fois que les gradients L/∂_{yK} sont disponibles.

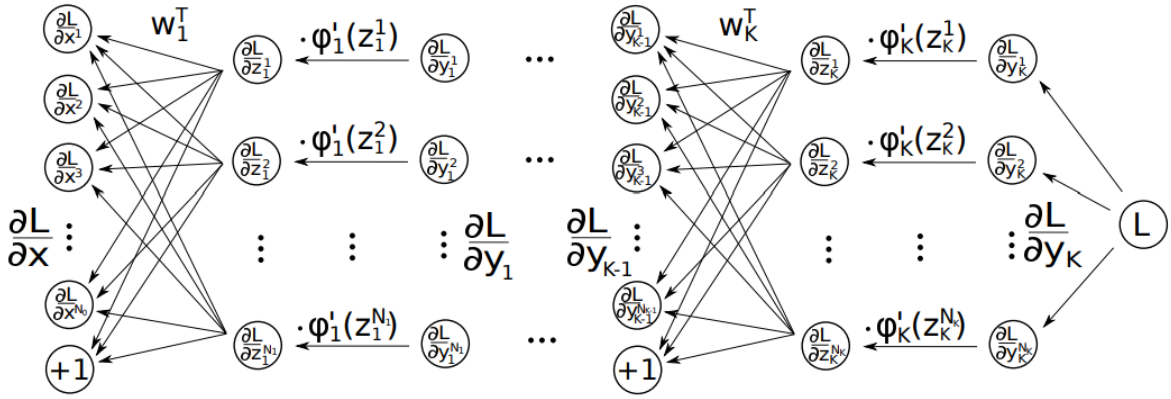


FIGURE 3.11: Etapes de Backpropagation[6]

3.5 Bio-informatique :

3.5.1 Définition :

La bio-informatique est un domaine multidisciplinaire qui utilise des méthodes informatiques, pour résoudre un problème biologique. Pour la plupart des membres de la communauté scientifique, la bio-informatique est l'étude de la façon dont la bio-information est représentée et analysée dans les systèmes biologiques, en particulier les informations obtenues au niveau moléculaire [28].
D'après Claverie et al. [29] « la bio-informatique est la discipline de l'analyse de l'information biologique, en majorité sous la forme de séquences génétiques et de structures de protéines. C'est le décryptage de la bio information (Computational Biology" en anglais). La bio-informatique est donc une branche théorique de la Biologie. Son but, comme tout volet théorique d'une discipline, est d'effectuer la synthèse des données disponibles (à l'aide de modèles et de théories), d'énoncer des hypothèses généralisatrices (ex. : comment les protéines se replient ou comment les espèces évoluent), et de formuler des prédictions (ex. : localiser ou prédire la fonction d'un gène) »

D'après Andrade et Sander[30] « Bioinformatics is a science of recent creation that uses biological data, completed by computational methods, to derive new biological knowledge » Selon les définitions données à la bioinformatique, nous pouvons résumer les activités de cette discipline en trois points principaux [31]· :

— Acquisition et organisation des données biologiques.
— Conception de logiciels pour l'analyse, la comparaison et la modélisation des données.
— Analyse des résultats produits par les logiciels.

3.5.2 Deep learning et bioinformatique :

Grâce aux progrès des technologies à haut débit, un déluge de données biologiques et médicales a été obtenu au cours des dernières décennies, notamment des données liées aux images médicales, aux séquences biologiques et aux structures protéiques. Certaines applications réussies de l'apprentissage en profondeur dans les domaines biomédicaux.
un résumé des applications est présentés dans le tableau suivant :

Sujet	DL architecture	Brève description
Analyse des images médicales	CNN	Segmentation de la tumeur cérébrale
		Segmentation du pancréas en TDM
		Segmentation du cartilage du genou
		Prédire les descriptions sémantiques à partir d'images médicales
		Micro-saignements cérébraux à partir d'images RM
		Détection des exsudats dans les images du fond d'œil
	SAE	Segmentation de l'hippocampe à partir du cerveau des nourrissons
		Caractérisation histologique peau saine et cicatrisation des plaies
		Score du pourcentage de densité mammographique et de la texture mammographique liés au risque de cancer du sein
		Détection du disque optique à partir de la photographie du fond de l'œil
	DBN	Segmentation du ventricule gauche du cœur à partir des données RM
		Discriminer les maladies à base de rétine

	DNN	La segmentation des tumeurs cérébrales en images RM
		Segmentation RM de la prostate
		Segmentation d'instance de glande
		Segmentation sémantique des tissus dans les images CT
		Détection de la mitose dans les images histologiques du cancer du sein
	RNN	Classification des modèles de synchronisation EEG (électroencéphalographie) pour la prédiction des crises
		Détection de déchéance basée sur EEG
		Prédiction des crises d'épilepsie
Séquençage génomique et analyse de l'expression génique	DNN	Inférence d'expression génique
		Identification des régions de régulation cis et des domaines de synchronisation de réplication
		Prédiction de l'amplificateur
		Prédiction des modèles d'épissage dans les tissus individuels et des différences dans les modèles d'épissage entre les tissus

		Annotation de la pathogénicité des variantes génétiques
	DBN	Modélisation des préférences de liaison structurelle et prédiction des sites de liaison des protéines de liaison à l'ARN
		Prédiction de la jonction d'épissage au niveau de l'ADN
		Prédiction des sites de liaison des facteurs de transcription
		Annotation et interprétation du génome non codant
		Prédiction des effets de variante non codants de novo à partir de la séquence
	RNN	Prédiction des cibles précurseurs de miARN et miARN
		Détection des jonctions d'épissure à partir de séquences d'ADN

TABLE 3.2: deep learning et bioinfotmatique

3.6 Conclusion

Dans ce chapitre, nous avons vu qu'est-ce que le Deep Leraning et leur historique, nous avons vu aussi les notions de base concernant les réseaux de neurones artificielles leurs différentes architectures et aussi L'apprentissage en Deep Learning, et finalement nous avons vu bioinformatique et l'application de DL sur ces domaines.

CHAPITRE 4

L'IA IN ARNM-VACCIN COVID

4.1 Introdiction

Dans le chapitre précédent, nous avons vu deep learning et leur fonctionnement avec leurs types et utilisation sur bio-informatique, Les réseaux de neurones (RNN) est l'un de ces types qui a démontré une excellente réussite dans les problèmes de séquence. Pour cela, on a choisi d'utiliser Les réseaux de neurones pour prédiction de la stabilité/réactivité de la dégradation de la séquence ARN.
Premièrement, on va expliquer l'architecture de réseaux RNN, puis on va parler Les réseaux Long Short-Term Memory (LSTM), ce dernier virsion améliorer réseaux de neurones récurrents. Et aussi on va parler sur BERT et finalement on va faire une description de nos modèles et ses architectures.

4.2 Architecture globale :

Description Architecture de notre système global : Quand un malade fait un test de diagnostic pour la maladie corona et que le résultat du test est positif, un échantillon du sang du patient est prélevé pour l'envoyer au laboratoire afin de l'étudier et de connaître le type de cette souche et comment se déroule le processus de développement génétique de ce virus.
Dans la troisième étape, une étude scientifique moderne est menée sur le virus et comment il se développe et comment il se développe à partir de lui-même à l'intérieur du corps. Voici le rôle du scientifique dans l'enregistrement et le suivi de ce qui se passe à l'intérieur des cellules, comment il s'active et comment il est inhibé. Plus tard, les caractéristiques de ce virus sont connues et comment il se défend. Ici, le rôle des scientifiques est de libérer des antiviraux à ce virus pour le combattre en réalisant séquence mRNA.

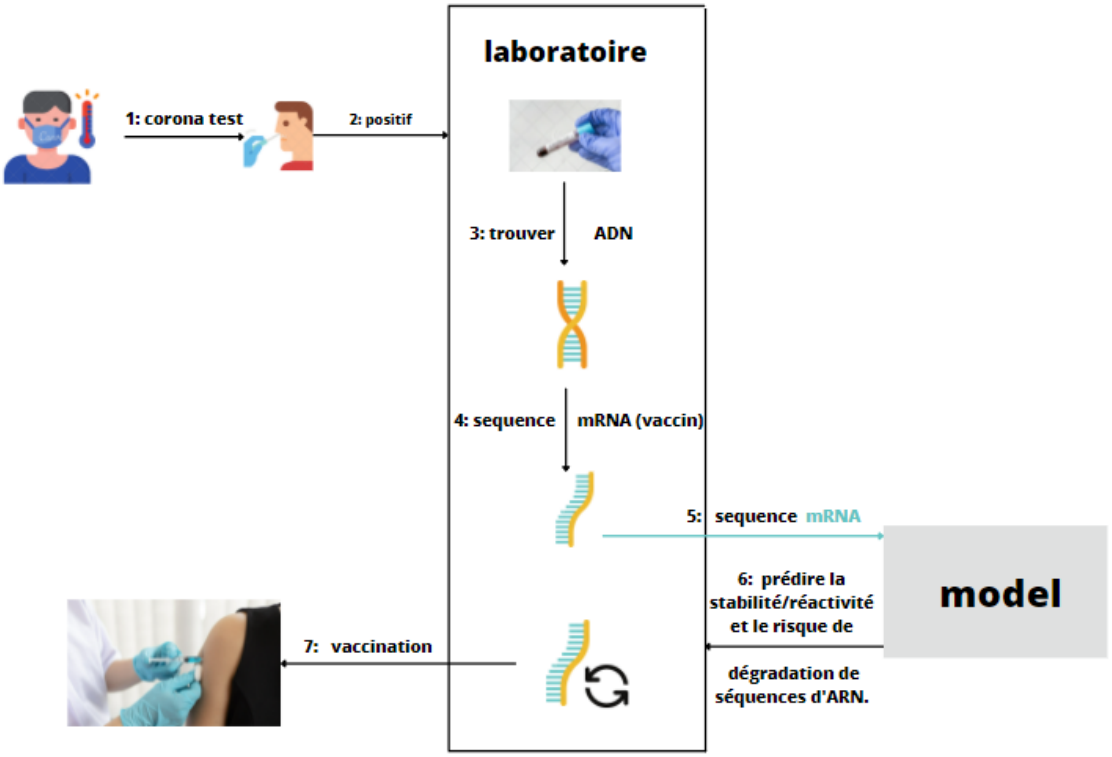


FIGURE 4.1: Architecture globale.

Après trouver la séquence mRNA, notre modèle prend ces structures pour fait une prédiction de la stabilité/réactivité de la dégradation de la séquence ARN. Ces prédictions peuvent être très utiles dans le développement de vaccins à ARNm, car elles peuvent réduire le nombre de séquences synthétisées et testées en aidant à identifier les candidats les plus prometteurs.
Enfin le laboratoire trouver la résulta de notre modèle pour faites un mis ajour ou bien un développement de la séquence mRNA avant vaccination.

4.3 Réseaux de neurones récurrents :

Les humains ne pensent pas à partir de zéro à chaque seconde. Lorsque vous lisez un livre, vous comprendrez chaque mot en fonction de votre compréhension des mots précédents. Nous n'oublierons pas tout et recommencerons à réfléchir Nos pensées ont une persistance.
Les réseaux de neurones traditionnels ne peuvent pas le faire, ce qui est un inconvénient majeur. Par exemple, supposons que vous souhaitiez classer les événements qui se produisent à chaque étape d'un film. On ne sait pas comment les réseaux de neurones traditionnels utilisent leur raisonnement sur les événements précédents du film pour notifier les événements ultérieurs. RNN (les réseaux de neurones récurrents) traitent ce problème. Ce sont des réseaux avec des

boucles, permettant aux informations de persister.
L'idée derrière RNN est d'utiliser des informations séquentielles. Dans les réseaux de neurones traditionnels, nous supposons que toutes les entrées (et sorties) sont indépendantes les unes des autres. Mais pour de nombreuses tâches, c'est une très mauvaise idée. Si vous voulez prédire le mot suivant dans une phrase, vous devez connaître le mot qui est apparu avant.
Les RNN sont appelés boucles, car ils effectuent la même tâche sur chaque élément de la séquence et la sortie dépend des calculs précédents. Une autre façon de penser aux RNN est qu'ils ont une "mémoire" qui peut capturer des informations sur ce qui a été calculé jusqu'à présent. En théorie, les RNN peuvent utiliser des informations dans n'importe quelle séquence longue, mais en pratique, ils se limitent à passer en revue quelques étapes. Voici à quoi ressemble un RNN typique :

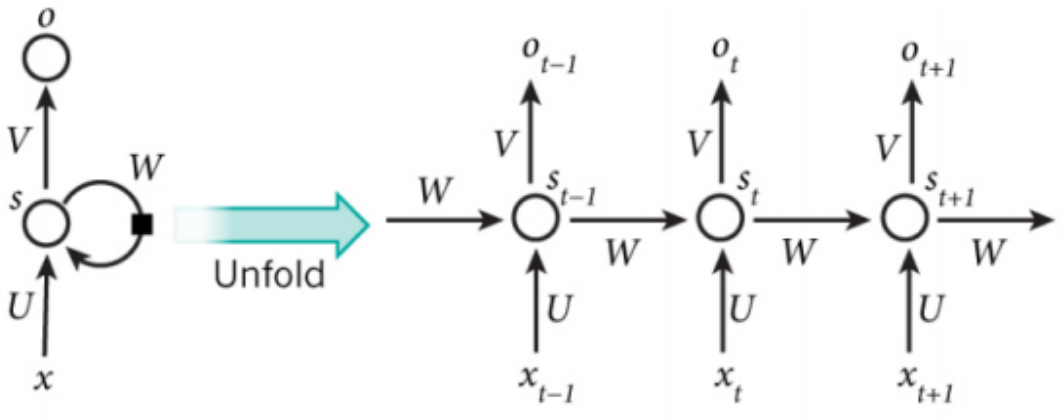


FIGURE 4.2: Un exemple d'un réseau récurrent qui se déroule. [4]

Les formules qui régissent les calculs dans un RNN sont les suivantes :
• x_t est l'entrée au moment t.
• U, V, W sont les paramètres que le réseau va apprendre des données de l'apprentissage.
• s_t est l'état caché au moment t. C'est la « mémoire » du réseau. st est calculé en fonction de l'état caché précédent et de l'entrée à l'étape actuelle :

$$s_t = f(Ux_t + Ws_{t-1}) \tag{4.1}$$

Où f est une fonction non linéaire.
• ot est la sortie au moment t. Par exemple, si on veut prédire le prochain mot dans une phrase, ce serait un vecteur de probabilités dans un vocabulaire.

$$o_t = softmax(Vs_t) \tag{4.2}$$

4.3.1 Apprentissage

Pour entraîner le RNN, une version légèrement modifiée de la rétro propagation est utilisée, appelée Backpropagation Through Time (BBTT) [32] [33][. Les paramètres étant Toujours partagés dans le réseau, le gradient de chaque sortie dépend non seulement du calcul à l'instant présent, mais aussi des étapes précédentes. Ensuite, nous appliquons la règle de la chaîne.
On considère la prédiction du réseau à l'instant t. Nous traitons la séquence entière (comme une phrase complète) comme un seul exemple d'apprentissage, donc l'erreur totale est la somme des erreurs à chaque instant (chaque mot).
Le but est de calculer le gradient de l'erreur pour les paramètres U, V, W et d'apprendre de bon paramètre en utilisant Stochastic Gradient Descent (SGD). Comme on a sommé les erreurs, on somme également les gradients à chaque étape pour un seul exemple d'apprentissage :

$$\frac{\delta j}{\delta W} = \sum_{t} \frac{\delta j_t}{\delta W} \tag{4.3}$$

Avec· : J la fonction de coût représentant l'erreur entre la vérité terrain g(x) (liée à la donnée d'entrainement x) et la valeur prédite y.
Pour calculer ces gradients, nous utilisons la règle de la chaîne. Par exemple l'erreur au moment t = 3

$$\frac{\delta j3}{\delta v} = \frac{\delta j3}{\delta o3}\frac{\delta o3}{\delta v} \tag{4.4}$$

$$\frac{\delta j3}{\delta o3}\frac{\delta o3}{\delta z3}\frac{\delta z3}{\delta v} \tag{4.5}$$

Avec z3 = V s3. La remarque importante ici, c'est que dépend uniquement des valeurs du moment actuel o3, s3. Mais c'est différent pour $\frac{\delta j3}{\delta w}$ (et pour U). Pour comprendre pourquoi, on écrit la règle en chaîne suivante :

$$\frac{\delta j3}{\delta w} = \frac{\delta j3}{\delta o3}\frac{\delta o3}{\delta s3}\frac{\delta s3}{\delta w} \tag{4.6}$$

On note que $s3 = f(u_x 3 + w_s 3)$ dépend de s3 qui dépend de W et de s1 et ainsi de suite. Donc si on prend la dérivée par rapport à W on ne peut pas traiter s2 comme une constante, on doit appliquer à nouveau la règle de la chaîne et ce qu'on obtient, c'est ceci :

$$\frac{\delta j2}{\delta w} = \sum_{k=0}^{2} \frac{\delta j2}{\delta o2}\frac{\delta o2}{\delta s2}\frac{\delta s2}{\delta sk}\frac{\delta sk}{\delta w} \tag{4.7}$$

On somme la contribution au gradient de chaque moment. En d'autres termes, puisque W est utilisé à chaque moment jusqu'à la sortie, on doit rétro propager le gradient de t = 3 vers t = 0 à travers tout le réseau· :

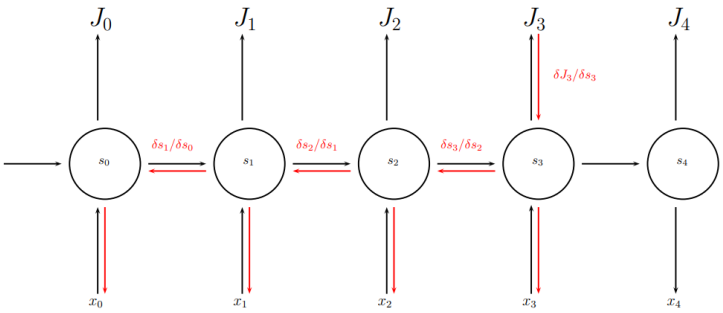


FIGURE 4.3: Backpropagation Through Time.

Les RNN ont des difficultés à apprendre des dépendances sur le long terme et les interactions entre des mots qui sont éloignés les uns des autres [2]. C'est problématique parce que la signification d'une phrase est souvent déterminée par des mots qui ne sont pas très proches. Dans ce cas précis si la fonction d'activation est une tanh ou bien une sigmoïde alors nous aurons le problème du vanishing gradient

$$\frac{\delta E}{\delta w} = \frac{\delta E}{\delta y_t} * \frac{\delta y_t}{\delta h_t} * \frac{\delta h_t}{\delta y_{t-1}} * \frac{\delta y_{t-1}}{\delta h_{t-2}} ... \tag{4.8}$$

C'est pour cette raison qu'on se tourne vers la fonction d'activation ReLu qui elle ne souffre pas de ce problème, mais il existe une solution encore plus utilisée, c'est l'utilisation des architectures Long Short Term Memory (LSTM) [34].

4.3.2 Application :

Les RNN ont connu un grand succès dans de nombreuses tâches de traitement du langage naturel. Les plus grands exploits des RNN ont été accomplie par l'architecture LSTM car ils sont bien meilleurs pour capturer des dépendances à long terme.

* ***Modélisation du langage et génération de texte***[35–37] : Compte tenu d'une séquence de mots, nous voulons prédire la probabilité de chaque mot sachant les mots précédents.

* ***Traduction automatique***[38–40] : La traduction automatique est semblable à la modélisation du langage en ce sens que l'entrée est une séquence de mots dans une langue source (par exemple, l'arabe). Nous voulons générer une séquence de mots dans une langue cible (par exemple, l'anglais). Une différence majeure est que notre sortie ne commence que lorsque nous avons vu l'entrée complète, car le premier mot de les phrases traduites nécessite des informations capturées à partir de la séquence d'entrée complète.

* ***La reconnaissance vocale*** [41] :La reconnaissance vocale est une technique informatique qui permet d'analyser la voix humaine pour la transcrire sous la forme d'un texte exploitable par une machine.

* ***Description des images***[42–44] : Ensemble avec les réseaux de neurones convolutifs, les RNN ont été utilisés pour générer des descriptions pour des images non étiquetées. Il est tout à fait incroyable de voir à quel point cela semble fonctionner. Le modèle combiné aligner sur des caractéristiques trouvées dans les images.

4.4 Les réseaux Long Short-Term Memory (LSTM) :

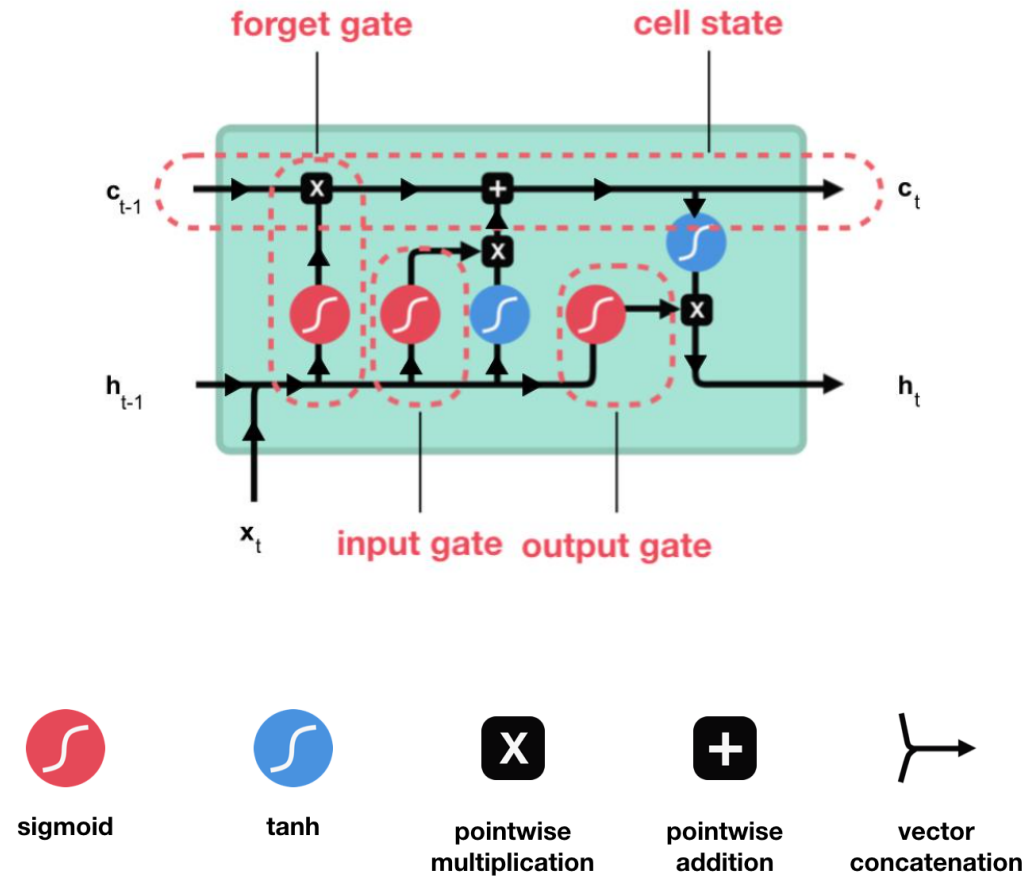


FIGURE 4.4: Cellule LSTM [7].

Les réseaux de mémoire à long terme à court terme généralement appelés simplement (LSTM : Long Short Term Memory) sont un type spécial de RNN. Ils ont été introduits par Hochreiter Schmidhuber (1997). Les Réseaux neuronaux récurrents présentés dans la section précédente sont capables d'apprendre des règles de mise à jour de séquence arbitraire en théorie. Dans la

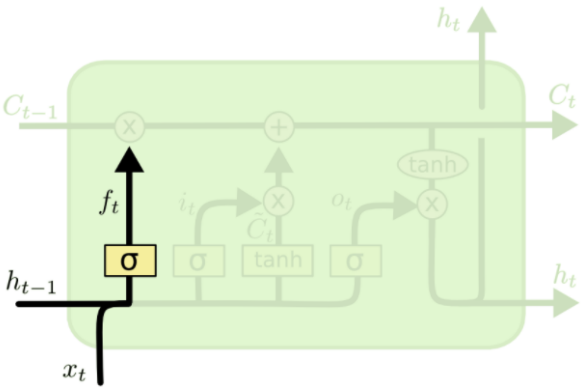
pratique, cependant, ces modèles oublient généralement rapidement le passé [45]. C'est ce qu'on appelle le problème de la disparition de gradient [46]. Et c'est pourquoi ils ont inventé le LSTM. La cellule LSTM est une adaptation de la couche récurrente qui permet aux signaux plus anciens des couches profondes de se déplacer vers la cellule du présent, La figure suivante représente une chaîne cellules LSTM :

* Forget gate (porte d'oubli)
* Input gate (porte d'entrée)
* Output gate (porte de sortie)
* Cell state (état de la cellule)

Les calculs se déroule comme suit [7, 47] :

4.4.1 Porte d'oubli (forget gate) :

Cette porte décide de quelle information doit être conservée ou jetée.



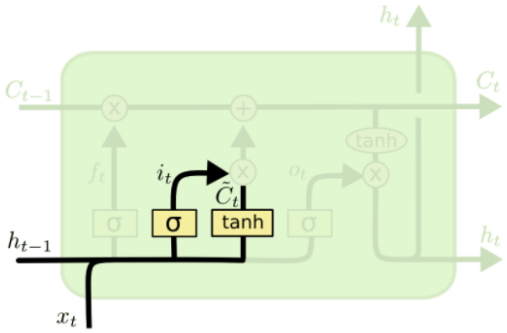
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

FIGURE 4.5: Porte d'oubli[7]

Avec :
h_t : La sortie a l'instant $(t-1)$.
x_t : L'entrée courant a l'instant t.
b_f : C'est le bais.
W_f : C'est le poids .
σ : C'est la fonction sigmoïde.

4.4.2 Porte d'entrée (input gate) :

La porte d'entrée a pour rôle d'extraire l'information de la donnée courante· :

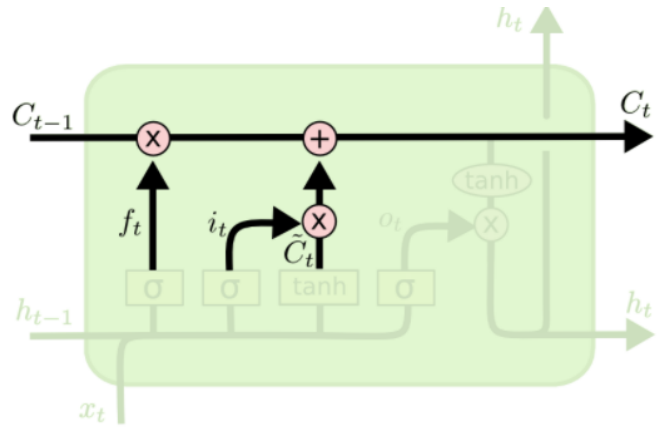


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

FIGURE 4.6: Porte d'entrée (input gate) [7].

— Sigmoïde va renvoyer un vecteur pour lequel une coordonnée proche de 0 signifie que la coordonnée en position équivalente dans le vecteur concaténé n'est pas importante. À l'inverse, une coordonnée proche de 1 sera jugée "importante"
— Tanh va simplement normaliser les valeurs (les écraser) entre -1 et 1 pour éviter les problèmes de surcharge de l'ordinateur en calculs.
— Le produit des deux permettra donc de ne garder que les informations importantes, les autres étant quasiment remplacées par 0

Avec :
tanh : C'est la fonction d'activation tangente hyperbolique.
c_t : Une valeur candidate



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

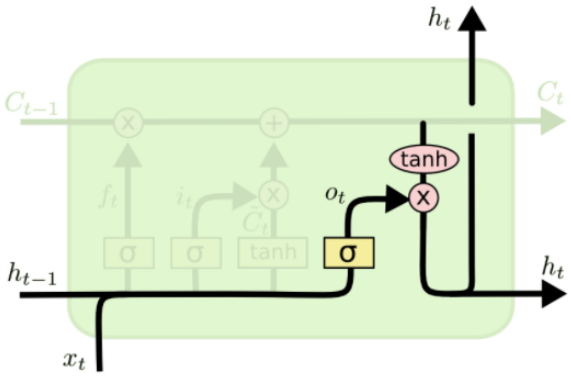
FIGURE 4.7: Porte d'entrée[7].

L'état de la cellule se calcule assez simplement à partir de la porte d'oubli et de la porte d'entrée : d'abord on multiplie coordonnée à coordonnée la sortie de l'oubli avec l'ancien état de la cellule. Cela permet d'oublier certaines informations de l'état précédent qui ne servent pas pour la nouvelle prédiction à faire. Ensuite, on additionne le tout (coordonnée à coordonnée) avec la

sortie de la porte d'entrée, ce qui permet d'enregistrer dans l'état de la cellule ce que le LSTM (parmi les entrées et l'état caché précédent) a jugé pertinent.
Avec :
c_t : État interne.

4.4.3 Porte de sortie (output gate) :

La porte de sortie doit décider de quel sera le prochain état caché, qui contient des informations sur les entrées précédentes du réseau et sert aux prédictions.



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

FIGURE 4.8: porte de sortie [7].

Pour ce faire, le nouvel état de la cellule calculé juste avant est normalisé entre -1 et 1 grâce à tanh. Le vecteur concaténé de l'entrée courante avec l'état caché précédent passe, pour sa part, dans une fonction sigmoïde dont le but est de décider des informations à conserver (proche de 0 signifie que l'on oublie et proche de 1 que l'on va conserver cette coordonnée de l'état de la cellule).
Avec :
h_t : La sortie.

4.4.4 Bidirectional Long Short-Term Memory :

L'idée des BDLSTM vient du RNN bidirectionnel [48], qui traite les données de séquence dans les deux sens vers l'avant et vers l'arrière avec deux couches cachées distinctes. Les BDLSTM connectent les deux couches cachées à la même couche de sortie. Il a été prouvé que les réseaux bidirectionnels sont nettement meilleurs que les réseaux unidirectionnels dans de nombreux domaines, comme la classification des phonèmes [49] et la reconnaissance vocale [50].
La structure d'une couche BDLSTM dépliée, contenant une couche LSTM avant et une couche

LSTM arrière, La séquence de sortie de la couche avant, est calculé de manière itérative en utilisant des entrées dans une séquence positive de temps T- n à T – 1, tandis que la séquence de sortie de couche arrière, est calculé en utilisant les entrées inversées du temps T- 1 à T – n. Les sorties de couche avant et arrière sont calculées en utilisant les équations de mise à jour LSTM standard.

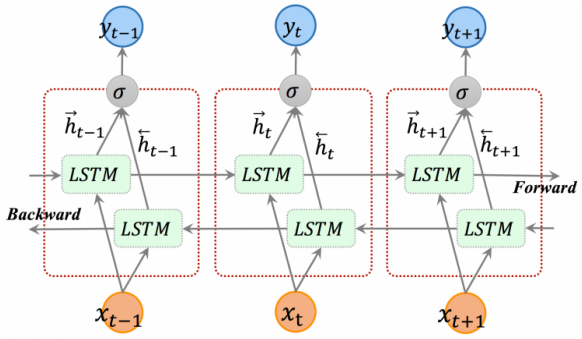


FIGURE 4.9: Architecture dépliée de LSTM bidirectionnel avec trois étapes consécutives[8]

4.5 Bert :

BERT (Devlin et al., 2018)[51] est un modèle neuronal de plongements lexicaux utilisant une succession de couches Transformer 3 (Vaswani et al., 2017) afin de produire des représentations contextualisées.
Ce modèle est entraîné sur deux tâches : une tâche de Modélisation du Langage Masquée (Masked Language Modelling - MLM) et une tâche de Prévision de la Phrase Suivante (Next Sentence Prediction - NSP). Cette section décrit l'architecture de BERT ainsi que la procédure employée pour l'entraîner. Dans tout ce qui suit, nous ferons la distinction entre deux phases :

* la phase de pré-entraînement, qui permet au modèle d'apprendre à produire des plongements contextualisés via les deux tâches MLM et NSP

* la phase d'adaptation à une tâche, où BERT est utilisé comme générateur de plongements au sein d'un modèle plus large qui est intégralement entraîné sur une tâche cible.

Encodeur :
Bert est composé d'un empilement de N = 6 couches identiques. Chaque couche a deux sous-couches. Le premier est un mécanisme d'auto-attention à plusieurs têtes et le second est un simple réseau d'anticipation entièrement connecté en termes de position. Nous utilisons une connexion résiduelle [52] autour de chacune des deux sous-couches, suivie d'une normalisation de couche [53]. C'est-à-dire que la sortie de chaque sous-couche est LayerNorm(x + Sublayer(x)),

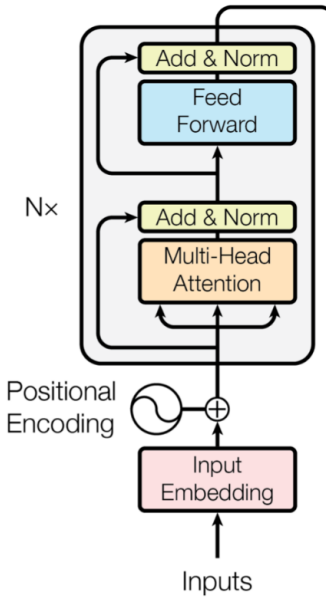


FIGURE 4.10: Modele BERT .

où Sublayer(x) est la fonction implémentée par la sous-couche elle-même. Pour faciliter ces connexions résiduelles, toutes les sous-couches du modèle, ainsi que les couches d'insertion, produisent des sorties de dimension d_{model} = 512.

Explication :

1. Les entrées sont des données d'entrée après remplissage.
2. Initialisez la matrice d'intégration, mappez les entrées sur l'intégration de jetons via la recherche d'intégration.
3. Créez un codage positionnel via les fonctions sin et cos, indiquant les informations de position absolue d'un jeton et ajoutez-le à l'incorporation du jeton, puis abandonnez.

$$PE_{(}pos,2i) = \sin pos(10000^{2i/d_{m}odel}) \tag{4.9}$$

$$PE_{(}pos,2i+1) = \cos pos(10000^{2i/d_{m}odel}) \tag{4.10}$$

4. Attention à plusieurs têtes.
5. Ajoutez les résultats de l'intégration de jetons et de l'attention multitêtes et effectuez une normalisation de couche (les deux dimensions sont les mêmes).
6. Passez le résultat de <5> à travers 2 couches de Dense (couches entièrement connectées), où activation de la couche 1 = relu et activation de la couche 2 = Aucune.

7. La même fonction que <5>.

4.6 Notre modele :

Nous allons proposer et évalue trois deep learning model pour résoudre cette problématique (Long Short-Term Memory (LSTM), réseaux de neurones récurrents (RNN), Bidirectional Encoder Representations from Transformers (BERT)) L'architecture de notre model et composé de 3 couches BDLSTM (model 1) et SimpleRNN (model2) et une couche bert(model3) :

- **Embedding layer** : ce que nous avons ici sont des caractères décrivant la séquence d'ARN (A, G, U et C), la structure ((, .,)), et le bpRNA (S, M, I, B, H, E, X). Alors ce n'est pas possible d'utiliser word2vec ou GloVE. Donc nous utilisent petite Embedding de dimension 200 pour chacun de ces caractères.
- **TFOpLambda layer** : pour concaténer les trois séquences en une seule séquence. Il en résulte donc un tenseur de forme (bs, seq len, 3*100) [bs :batch size , seq len : séquence taille].
- **SpatialDropout1D layer** : Cette version remplit la même fonction que Dropout.
  - **Le Dropout pour La régularisation :** La régularisation est toute modification que nous apportons à un algorithme d'apprentissage dans le but de réduire son erreur de généralisation, mais pas son erreur d'apprentissage. Il s'agit des techniques pour éviter le overfitting, l'une de ses techniques est le Dropout.

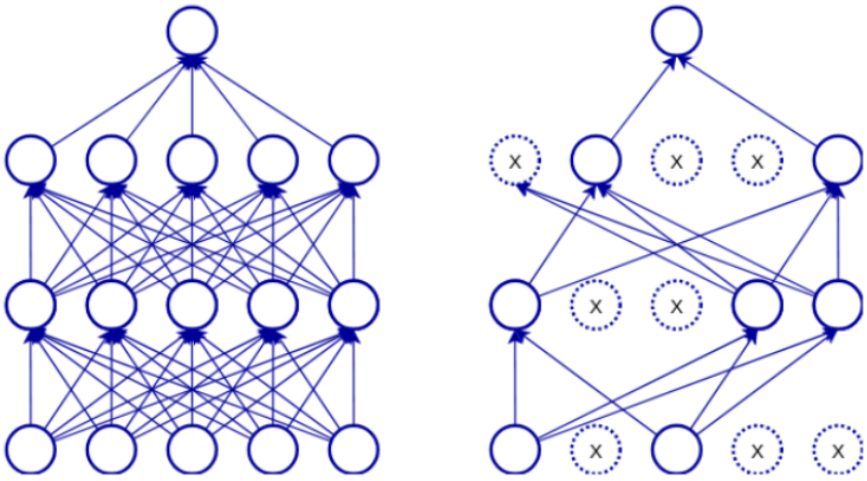


FIGURE 4.11: l'application du dropout après une couche complètement connectée [9].

Dropout est une technique, qui peut être appliquée à la sortie de certaines des couches du réseau. Il enlèvera aléatoirement et périodiquement certains des neurones (ainsi que leurs connexions d'entrée et de sortie) du réseau [9].

La figure suivante illustre l'application du dropout après une couche complètement connectée :

Lorsque les neurones sont supprimés au hasard du réseau au cours de l'apprentissage, les autres neurones sont obligés de répondre et de contrôler la représentation requise pour faire des prédictions pour les neurones manquants. Cette méthode améliore la généralisation, car elle oblige les couches à apprendre le même "concept" avec des neurones différents.

— **SlicingOplambda layer· :** pour tronquer la séquence, car la prédiction se fait juste pour les premiers 68 séquence.
— **Dense layer :** qui est une couche de neurones classique, complètement connectée avec la couche précédente et la couche suivante.
— **flatten layer. :**Une opération d'aplatissement pour redéfinit la forme de l'input.

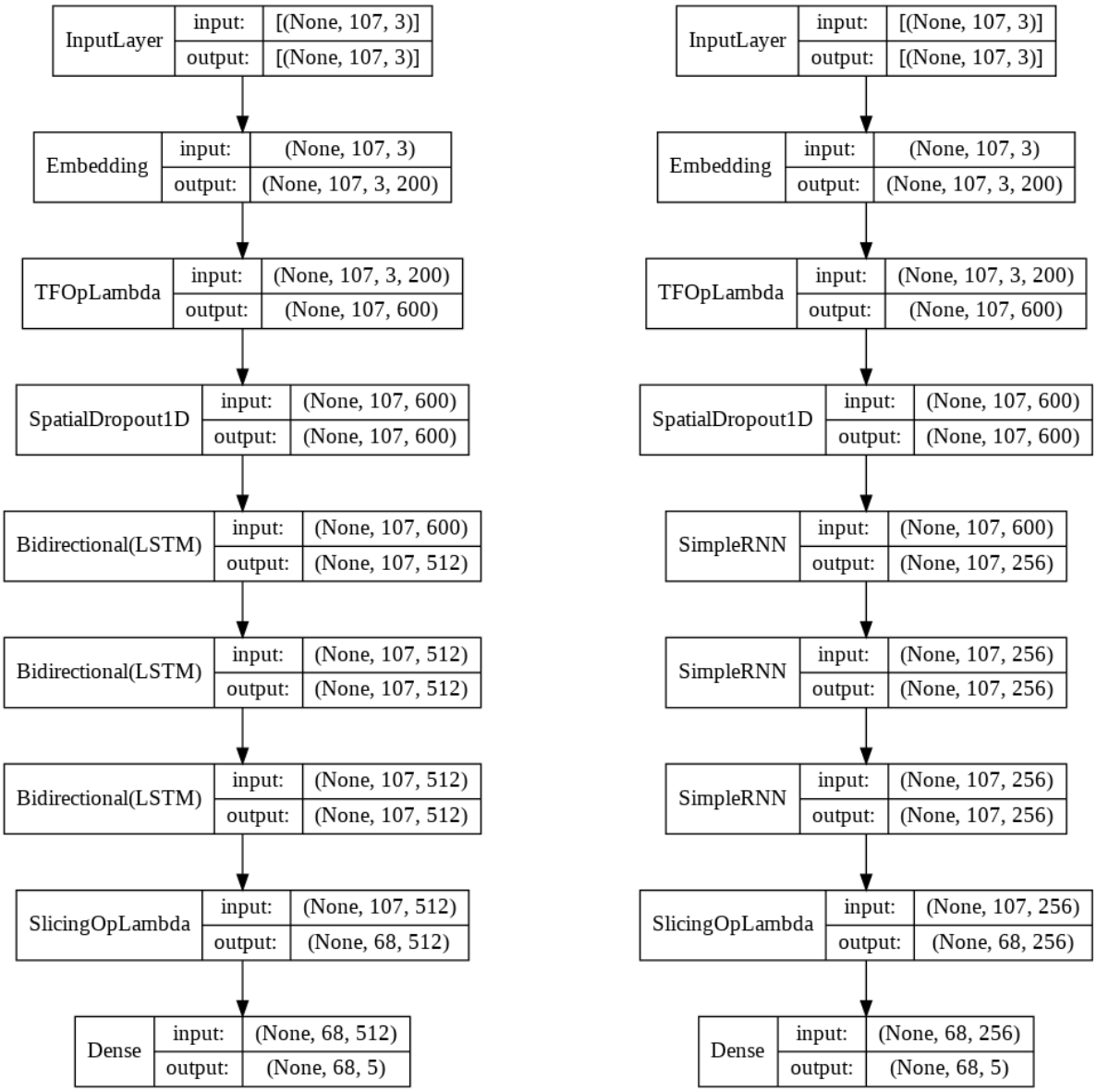


FIGURE 4.12: L'architecture proposée de modèle 1 et modèle 2.

Comme nous pouvons le voir dans la figure 4.12, nous avant fais le mème traitement pour les deux modèles RNN et LSTM.
Au début input layer ensuit une coche Embedding layer ensuit un lambda layer pour changer forme, un spatialdropout pour améliore accuracy (trois couches BDLSTM pour modèle 1 et trois couches simpleRNN pour modele 2). Une couche Lambda pour changer la taille de la séquence et Dense layer pour sortie.

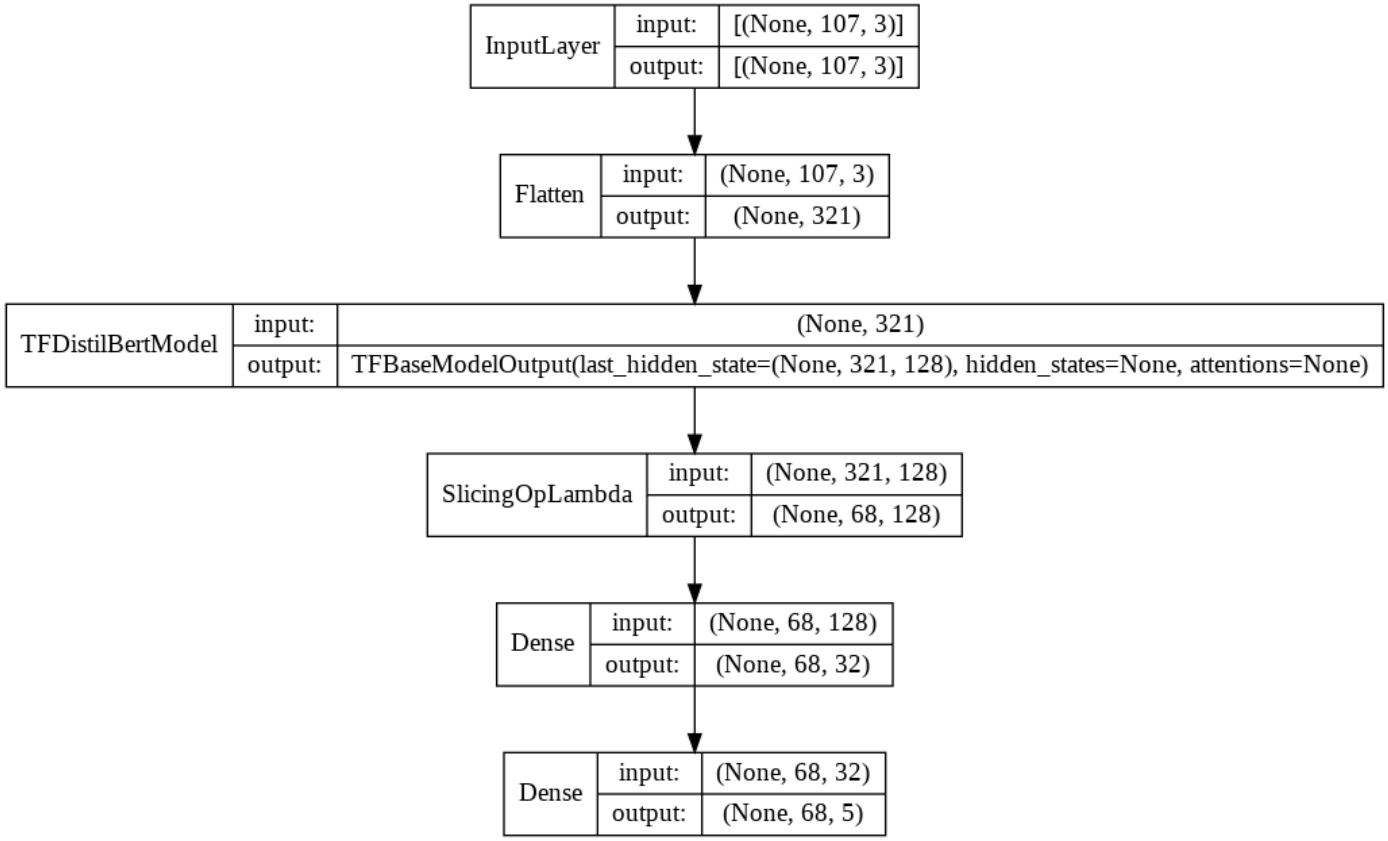


FIGURE 4.13: L'architecture proposée de modèle 3.

Comme nous pouvons le voir dans la figure 4.13, qui représente notre architecture proposer pour le modèle Bret.
Au début une couche input ensuit une couche flatten qui change la forme de (107,3) à un Seul vecteur de longueur (321), une couche Bert ensuit une couche lambda pour changer la séquence de longueur 107 à 68 en suit Dense layer pour output.

4.7 comparaison en terme de nombre de paramètre

les figures 4.15, 4.14 et 4.12 représentent résumé avec chaque couche et leurre nombre de paramètres.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_5 (InputLayer)         [(None, 107, 3)]          0
_________________________________________________________________
embedding_4 (Embedding)      (None, 107, 3, 100)       1400
_________________________________________________________________
tf.reshape_4 (TFOpLambda)    (None, 107, 300)          0
_________________________________________________________________
spatial_dropout1d_4 (Spatial (None, 107, 300)          0
_________________________________________________________________
simple_rnn_3 (SimpleRNN)     (None, 107, 256)          142592
_________________________________________________________________
simple_rnn_4 (SimpleRNN)     (None, 107, 256)          131328
_________________________________________________________________
simple_rnn_5 (SimpleRNN)     (None, 107, 256)          131328
_________________________________________________________________
tf.__operators__.getitem_3 ( (None, 68, 256)           0
_________________________________________________________________
dense_3 (Dense)              (None, 68, 5)             1285
=================================================================
Total params: 407,933
Trainable params: 407,933
Non-trainable params: 0
_________________________________________________________________
```

FIGURE 4.14: résumé de model RNN.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_4 (InputLayer)         [(None, 107, 3)]          0
_________________________________________________________________
embedding_3 (Embedding)      (None, 107, 3, 200)       2800
_________________________________________________________________
tf.reshape_3 (TFOpLambda)    (None, 107, 600)          0
_________________________________________________________________
spatial_dropout1d_3 (Spatial (None, 107, 600)          0
_________________________________________________________________
bidirectional_3 (Bidirection (None, 107, 512)          1755136
_________________________________________________________________
bidirectional_4 (Bidirection (None, 107, 512)          1574912
_________________________________________________________________
bidirectional_5 (Bidirection (None, 107, 512)          1574912
_________________________________________________________________
tf.__operators__.getitem_2 ( (None, 68, 512)           0
_________________________________________________________________
dense_2 (Dense)              (None, 68, 5)             2565
=================================================================
Total params: 4,910,325
Trainable params: 4,910,325
Non-trainable params: 0
_________________________________________________________________
```

FIGURE 4.15: résumé de model LSTM.

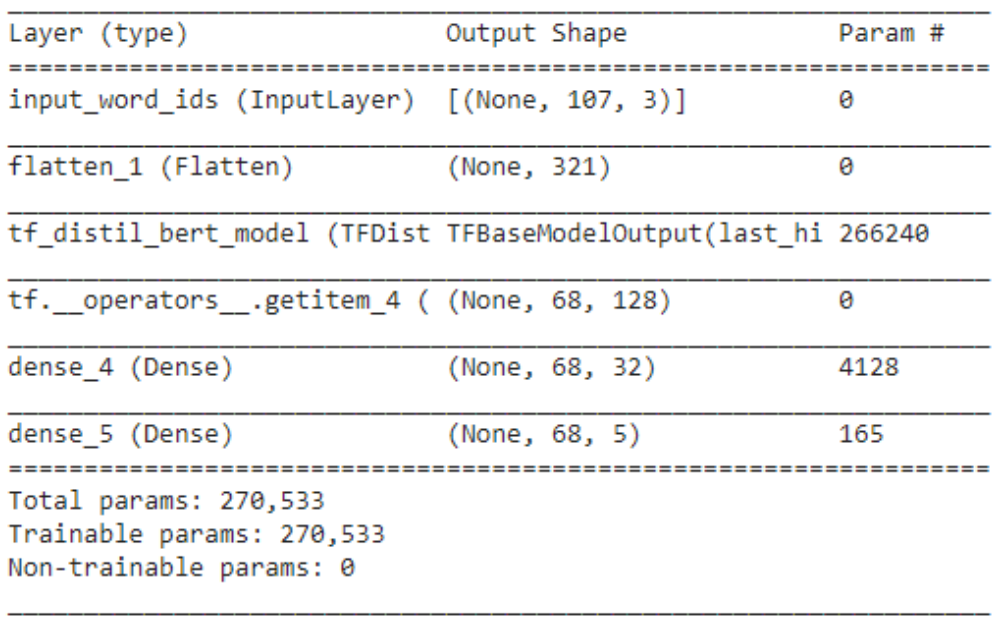


FIGURE 4.16: résumé de modèle Bert.

la table 4.7 afficher les nombre de Paramètres de chaque model avec le temps exécution :

* **note :** les model sant excuter sur google colab (lstm prend 3h sur I5 6éme génération)

model	Total Paramètre	Paramètres entraînables	Paramètres non entraînables	temps exécution
LSTM	4,910,325	4,910,325	0	210s
RNN	407 ,933	407 ,933	0	840s
Bert	270,533	270,533	0	73s

Comme nous pouvons le voir dans le tableau, le modèle bert a le plus petit nombre de paramètres, cela signifie que le temps d'exécution est petit et cela nous aide à modifier la configuration.
Le modèle rnn nous donne un nombre raisonnable de paramètres, mais le temps d'exécution est trop élevé, ce qui n'est pas ce que nous attendons.
le modèle LSTM donne un grand nombre de paramètres qui aident la précision à augmenter, et le temps d'exécution est raisonnable et qui nous aide à changer la configuration du modèle

4.8 Conclusion

La conception de l'architecture du modèle est importante pour améliorer les résultats, mais d'autres facteurs tels que l'ensemble de données et les paramètres d'apprentissage jouent également un rôle très important. La sélection de l'ensemble de données et l'expérience d'optimisation du modèle sont introduites dans chapitre suivent.

CHAPITRE 5

RESULTATS EXPERIMENTATIONS ET COMPARAISONS

5.1 Introduction

Dans le chapitre précédent, nous avons vu l'architecture proposée, mais avant de vérifier notre modèle, nous avons fait une série d'expériences pour sélectionner les meilleurs hyperparamètres pour optimiser les performances du modèle. Ensuite, nous commencerons par les descriptions des implémentations utiles, la sélection des ensembles de données et des paramètres.

5.2 Outils d'implémentation

5.2.1 Les softwares :

Plusieurs frameworks open sources sont disponibles dans la littérature, la grande majorité supporte le langage Python. Voici une liste non exhaustive de quelques frameworks.

5.2.1.1 Python

FIGURE 5.1: Python logo.

Python est un langage de programmation interprété, de haut niveau et à usage général. Créée par Guido van Rossum et publiée pour la première fois en 1991, la philosophie de conception de Python met l'accent sur la lisibilité du code avec son utilisation notable d'espaces blancs importants. Ses constructions de langage et son approche orienté objet visent à aider les programmeurs à écrire un code clair et logique pour des projets à petite et à grande échelle.
Python est typé dynamiquement et ramassé. Il prend en charge plusieurs paradigmes de programmation, y compris la programmation structurée (en particulier procédurale), orienté objet et fonctionnelle. Python est souvent décrit comme un langage « ·piles incluses· » en raison de sa bibliothèque standard complète. Nous utilisons Python 3.6 pour notre projet.

5.2.1.2 TensorFlow :

FIGURE 5.2: TensorFlow logo.

TensorFlow est une bibliothèque logicielle gratuite et open source pour le flux de données et la programmation différentiable pour une gamme de tâches. Il s'agit d'une bibliothèque mathématique symbolique, également utilisée pour les applications d'apprentissage automatique telles que les réseaux de neurones. Il est utilisé à la fois pour la recherche et la production chez Google. TensorFlow a été développé par l'équipe Google Brain pour une utilisation interne à Google. Il a été publié sous la licence Apache 2.0 le 9 novembre 2015.

```
import import tensorflow as tf
```

5.2.1.3 Keras :

FIGURE 5.3: Keras logo.

Keras est une bibliothèque de réseau de neurones open source écrite en Python. Il est capable de s'exécuter sur TensorFlow, Microsoft Cognitive Toolkit, R, Theano ou PlaidML. Conçu pour permettre une expérimentation rapide avec les réseaux de neurones profonds, il se concentre sur la convivialité, la modularité et l'extensibilité. Il a été développé dans le cadre de l'effort de recherche du projet ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System), et son principal auteur et mainteneur est François Chollet, un ingénieur de Google. Chollet est également l'auteur du modèle de réseau de neurones profonds XCeption.
En 2017, l'équipe TensorFlow de Google a décidé de prendre en charge Keras dans la bibliothèque principale de TensorFlow. Chollet a expliqué que Keras a été conçu pour être une interface plutôt qu'un cadre d'apprentissage automatique autonome. Il offre un ensemble d'abstractions de niveau supérieur et plus intuitif qui facilite le développement de modèles d'apprentissage en profondeur, quel que soit le backend de calcul utilisé. Microsoft a également ajouté un backend CNTK à Keras, disponible à partir de CNTK v2.0.

```
import tensorflow.keras.layers as L
```

5.2.1.4 Numpy

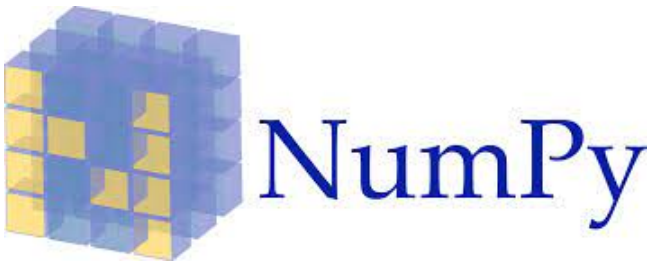


FIGURE 5.4: Numpy logo.

NumPy est une bibliothèque pour le langage de programmation Python, ajoutant la prise en charge de grands tableaux et matrices multidimensionnels, ainsi qu'une grande collection de fonctions mathématiques de haut niveau pour opérer sur ces tableaux.

```
import numpy as np
```

5.2.1.5 Scikit-learn

FIGURE 5.5: Scikit-learn.

Scikit-learn (anciennement scikits.learn et également connu sous le nom de sklearn) est une bibliothèque logicielle gratuite d'apprentissage automatique pour le langage de programmation Python. Il comporte divers algorithmes de classification, de régression et de clustering, notamment les machines vectorielles de support, les forêts aléatoires, l'amplification de gradient, k-means et DBSCAN, et est conçu pour interagir avec les bibliothèques numériques et scientifique Python NumPy et SciPy.

```
from sklearn.model_selection import train_test_split
```

5.2.1.6 Google Colab

FIGURE 5.6: Google Colab.

Colaboratory est un projet de recherche Google créé pour aider à diffuser l'enseignement et la recherche en apprentissage automatique. Il s'agit d'un environnement de notebook Jupyter qui ne nécessite aucune configuration pour être utilisé et qui s'exécute entièrement dans le cloud.

5.2.2 Le hardware :

Nous avons implémenté notre modèle et faire toutes les expérimentations en Google cloud

5.2.2.1 Hardware spécification

CPU Model Name	Intel(R) Xeon(R)
RAM	12GB
Disk Space	25GB

TABLE 5.1: Hardware spécification

5.3 Openvaccin Stanford Dataset :

L'ensemble de données utilisé pour évaluer le modèle est appelé ensemble de données «Stanford OpenVaccine» [54]. Il se compose de 6034 séquences d'ARN. L'ensemble d'apprentissage se compose de 2400 de ces séquences, qui ont une longueur de 107 bases nucléotidiques. L'ensemble de données de test se compose de 3634 séquences d'une longueur de 130 bases nucléotidiques. En raison des limites de l'expérience (selon la collecte de données des chercheurs de Stanford [54]), il est impossible de mesurer la base finale de la séquence. Par conséquent, il n'y en a que 68 (pour une séquence de longueur 107) et 91 (pour une séquence de longueur 130). Séquence) La première base de chaque séquence a des données expérimentales qui lui sont liées.
Chaque séquence est associée à trois prédicteurs : la séquence elle-même (décrite dans les bases A, G, C et U), la structure attendue de la molécule et le type de boucle prédite (dérivée de la simulation de la structure secondaire de l'ARN molécule). Une matrice de probabilité de paires de bases est également fournie pour chaque séquence individuelle, indiquant la probabilité de certaines interactions de paires de bases. Il fournit également cinq valeurs déterminées expérimentalement pour les 68 premières paires de bases de la séquence (ci-après dénommées « target »)· : valeur de réactivité, valeur de dégradation à pH 10, valeur de dégradation après ajout de magnésium. pH 10, valeur de dégradation à 50 ° avec ajout de magnésium et valeur de dégradation à 50 ° avec ajout de magnésium. Pour plus d'informations, voir le tableau.

	Class	Description	SYMBOLE
Sequence	Input	Une séquence de 107 lettres correspondant aux quatre bases de la séquence.	A, G, U, U, C.
Structure	Input	Structure attendue de la molécule (longueur = 107). '(' Et ')' font référence à une interaction de paires de bases. Tous les '.' Au milieu sont associés à aucune interaction BP.	(..()...)()(... ...
Predicted loop type	Input	Structure secondaire prédite de la molécule d'ARN en différents points. "S" fait référence à une structure de tige, boucle multiple "M", boucle interne "I", renflement "B", boucle en épingle à cheveux "H", extrémité pendante "E", boucle externe "X".	E, S, M, I, H, ...
Reactivity	Target	Valeurs de réactivité à chaque point individuel de la séquence	0.1633, 0.1314, ...
Deg pH 10	Target	Valeurs de dégradation à pH 10	2.983, 0.2526, ...
Deg pH 10 Mg	Target	Valeurs de dégradation à pH 10 avec Mg.	3.506, 0.2635,...
Deg 50∘C	Target	Valeurs de dégradation à 50∘C	0.6382, 1.3228 ,....
Deg 50∘C Mg	Target	Valeurs de dégradation à 50∘C avec Mg.	0.3581, 1.4552 ,....

TABLE 5.2: Informations de dataset

5.4 Configuration :

La configuration suivante a été considéré dans toutes les étapes de l'apprentissage de notre modèle :

Prétraitement des données de dataset :ce que nous avons ici sont des caractères décrivant la séquence d'ARN (A, G, U et C), la structure ((, .,)), et le bpRNA (S, M, I, B, H, E, X). alor Alors ce n'est pas possible d'utiliser word2vec ou GloVE. Nous avant utilise fonction pour identifie chaque caractère « token2int »

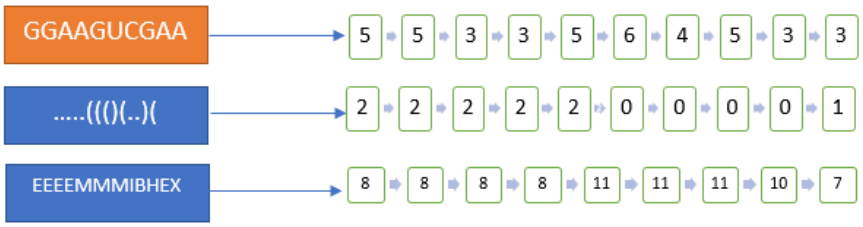


FIGURE 5.7: Prétraitement des données de dataset.

Listing 5.1: preprocess inputs

```
#identifier chaque caractre avec un nombre
token2int = {x:i for i, x in enumerate('().ACGUBEHIMSX')}
#fonction qui change chaque caractre dans le dataset
def preprocess_inputs(df, token2int, cols=['sequence', 'structure',
    'predicted_loop_type']):
    return pandas_list_to_array(
        df[cols].applymap(lambda seq: [token2int[x] for x in seq])
    )
#appel de la fonction
train_inputs = preprocess_inputs(train, token2int)
```

L'apprentissage de modèle : Les modèles seront évalués en fonction de l'erreur produite dans la prédiction des valeurs cibles. Nous avons choisi la fonction mean column-wise root mean squared error (MCRMSE) :

$$MCRMSE = \frac{1}{N_t}\sum_{j-1}^{N_1}\sqrt{\frac{1}{n}\sum_{i-1}^{n}(y_{ij}-\hat{y}_{ij})^2} \tag{5.1}$$

Listing 5.2: MCRMSE fonction

```
def MCRMSE(y_true, y_pred):
    colwise_mse = tf.reduce_mean(tf.square(y_true - y_pred), axis=1)
    return tf.reduce_mean(tf.sqrt(colwise_mse), axis=1)
```

Car Les modèles du concours Kaggle sont notés en fonction de MCRMSE. et pour la précision

$$accuracy = \frac{TP+TN}{TP+TN+FP+FN} \tag{5.2}$$

5.5 Expérimentations

Nous avons fait plusieurs expérimentations avec des diffèrents batch size et dropout, optimazers, La précision 5.2 (accuracy) et le taux d'erreur (Loss) 5.1 d'apprentissage et de validation

sont utilisés pour évaluer la performance du modèle.

Epoch s'écoule lorsqu'un ensemble de données (dataset) entier est transmis à travers le réseau de neurones et les poids sont mis à jour exactement une fois. Si l'ensemble de données ne peut pas être transmis à l'algorithme en une seule fois, il doit être divisé en mini-batchs.

Batch size est le nombre total d'échantillons d'apprentissage présents dans un seul mini-batch.

Iteration indique le nombre de mises à jour des paramètres de réseau.

code source en python pour changement de la configuration

code python pour changement d l'algorithm

```
tout=[]
for mod in [1,2,3,4,5,6,7,8]:
    if mod==1:
        opt=tf.keras.optimizers.RMSprop()
    elif mod==2:
        opt=tf.keras.optimizers.Adam()
    elif mod==3:
        opt=tf.keras.optimizers.Adadelta()
    elif mod==4:
        opt=tf.keras.optimizers.Adagrad()
    elif mod==5:
        opt=tf.keras.optimizers.Adamax()
    elif mod==6:
        opt=tf.keras.optimizers.SGD()
    elif mod==7:
        opt=tf.keras.optimizers.Nadam()
    elif mod==8:
        opt=tf.keras.optimizers.Ftrl()
    dr=0
    print('model'+str(m)+'.h5')
    model_lstm = model_Rnn(embed_size=len(token2int),dropout=dr,opt=opt)
    history = model.fit(
```

code python pour changement de learning rate

```
for i in [0.0001,0.0002,0.0003,0.0004,0.0005,0.0006,
          0.0007,0.0008,0.0009,0.001]:
  print('model'+str(i)+'.h5')
  model = model(embed_size=len(token2int),
  dropout=0,opt=opt=tf.keras.optimizers.Adam(learning_rate=i))
  history = model.fit(
```

code python pour changement de batch size

```
for i in [0,32,64,128,256]:
  print('model'+str(i)+'.h5')
  model = model(embed_size=len(token2int),
  dropout=0,opt=opt=tf.keras.optimizers.Adam(learning_rate=0.0008))
  history = model.fit(
      x_train, y_train,
      validation_data=(x_val, y_val),
      batch_size=i,
```

code python pour changement de Dropout

```
for i in [0,0.2,0.5]:
   print(  model  +str(i)+  . h5 )
   model = model(embed_size=len(token2int),
   dropout=i,opt=opt=tf.keras.optimizers.Adam(learning_rate=0.0008))history =
       model.fit(
```

5.5.1 Model RNN

5.5.1.1 Algorithm d'optimisation :

la figure 5.8 présente la précision et le taux d'erreur d'apprentissage obtenu avec les défirent algorithme d'optimisation

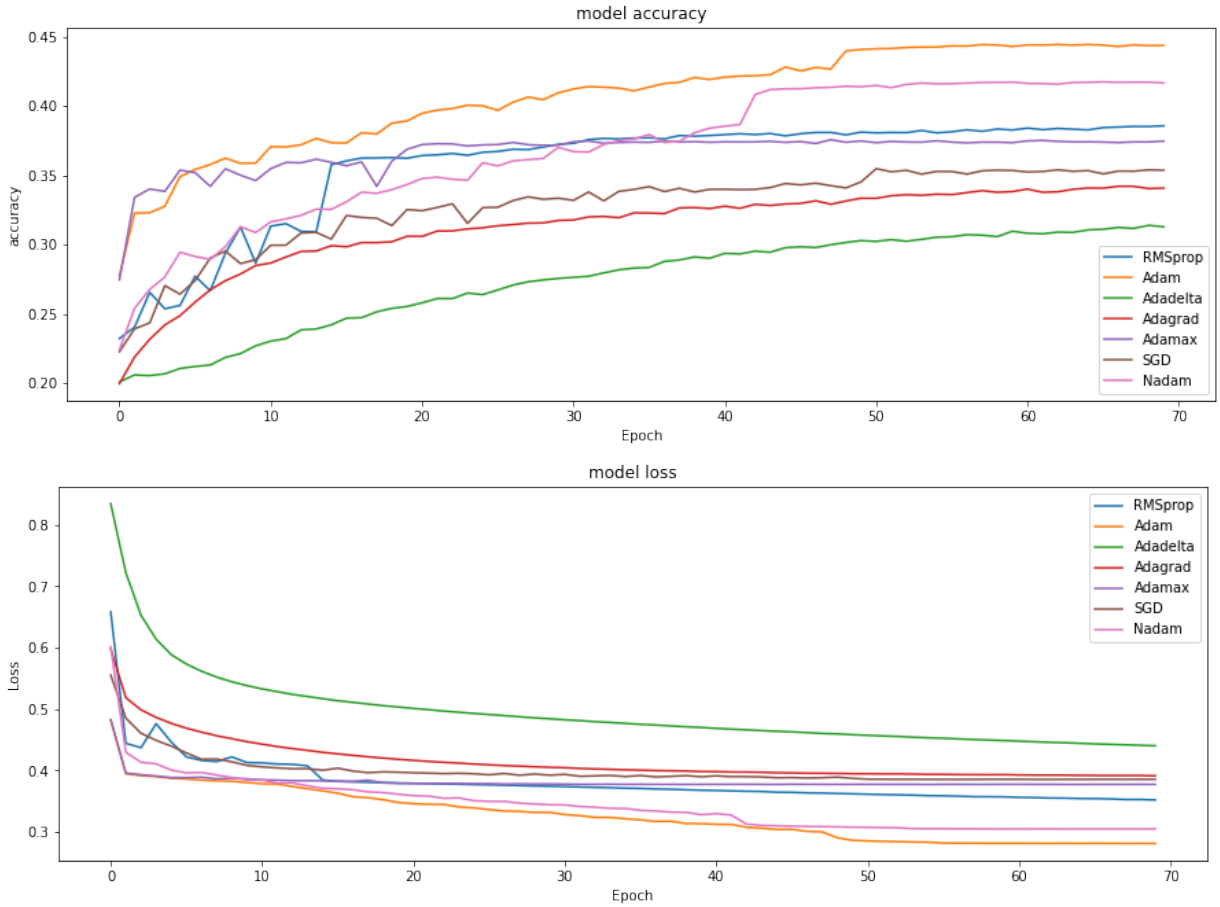


FIGURE 5.8: La précision et le taux d'erreur d'apprentissage avec diffèrents Algorithmes (model RNN)

Pour la validation, les résultats dans la figure 5.9 sont obtenues.

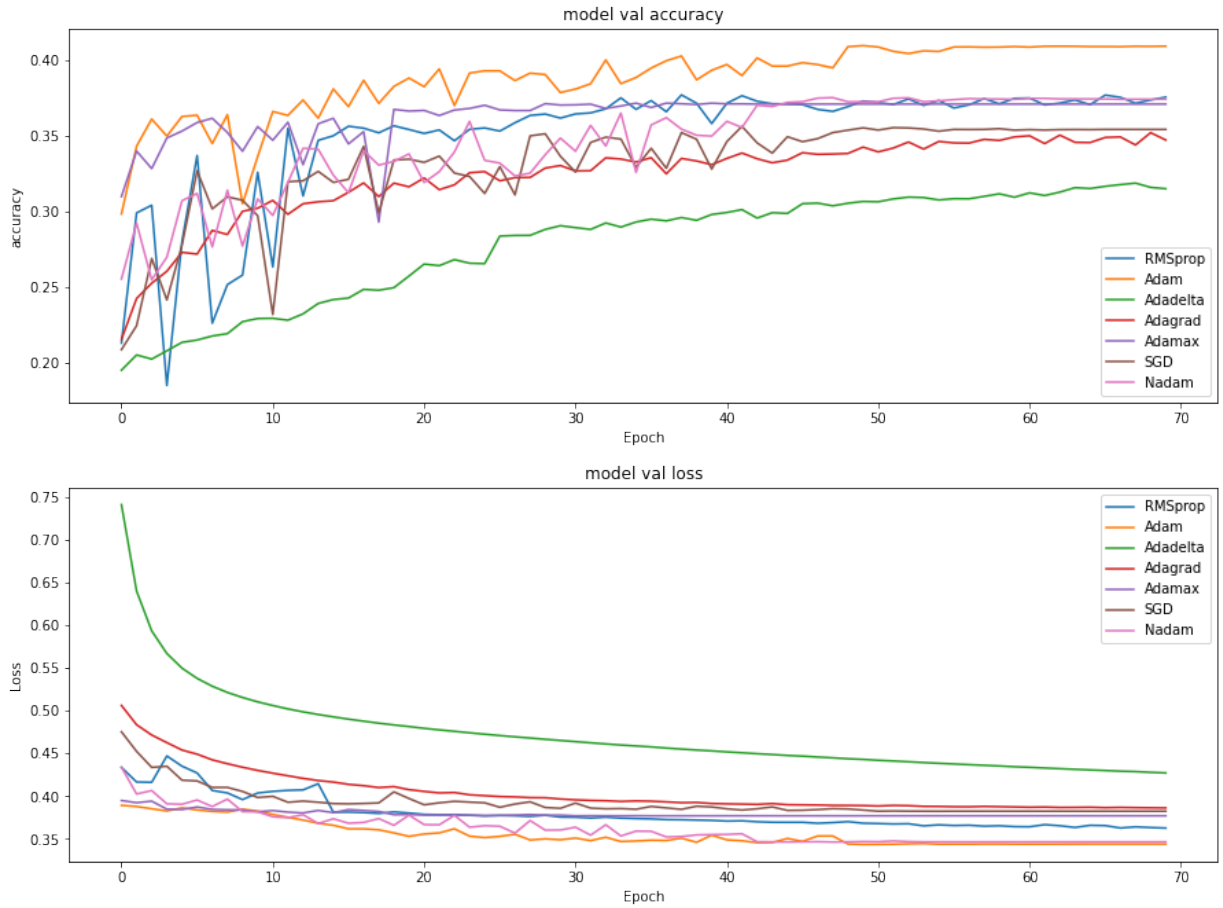


FIGURE 5.9: La précision et le taux d'erreur de validation avec diffèrents Algorithme (model RNN)

Après la comparaison, l'algorithme **adam** du réseau donne le meilleur résultat.

5.5.1.2 learning rate

La figure 5.10 présente la précision et le taux d'erreur d'apprentissage obtenu avec learning rate 0.0001-0.001

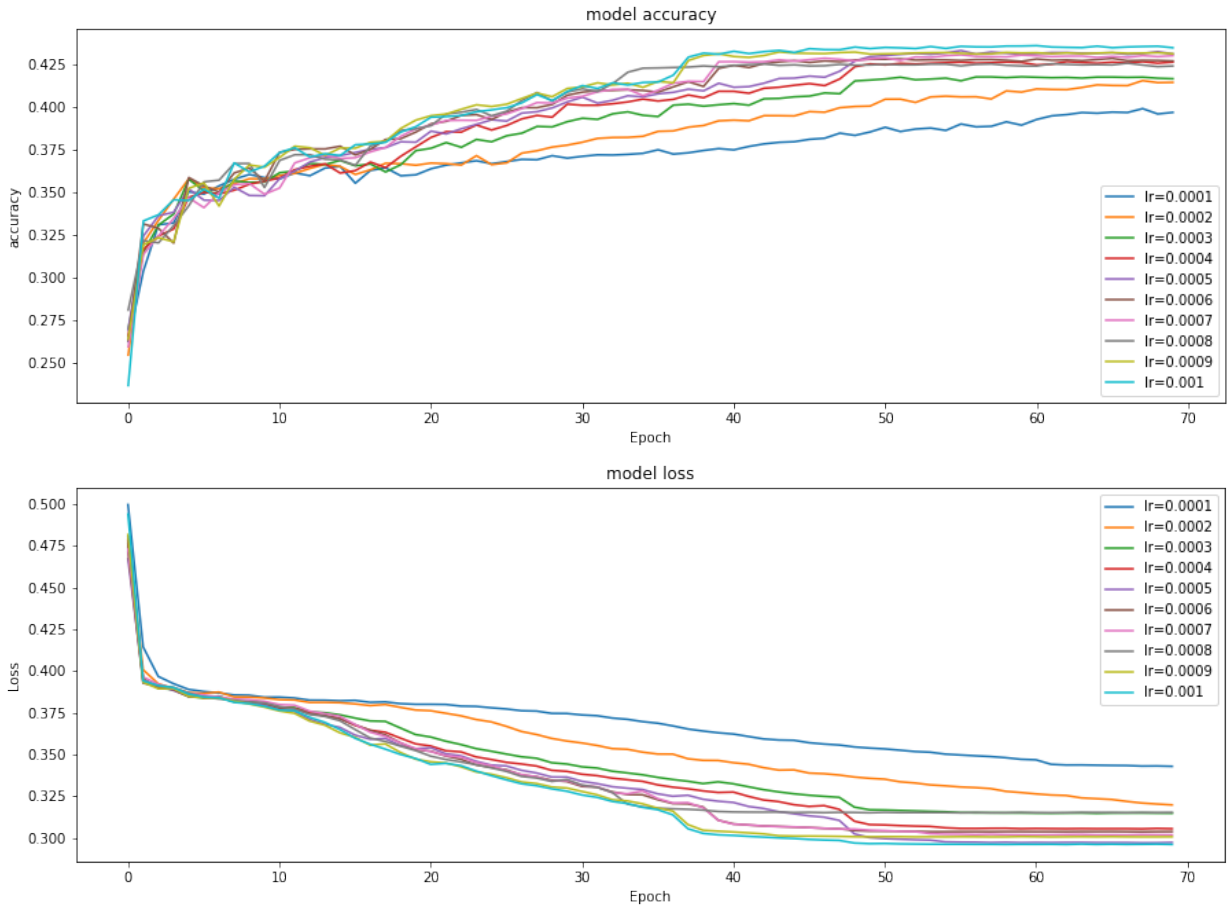


FIGURE 5.10: La précision et le taux d'erreur d'apprentissage avec diffèrents learning rate (model RNN)

Pour la validation, les résultats dans la figure 5.11 sont obtenues.

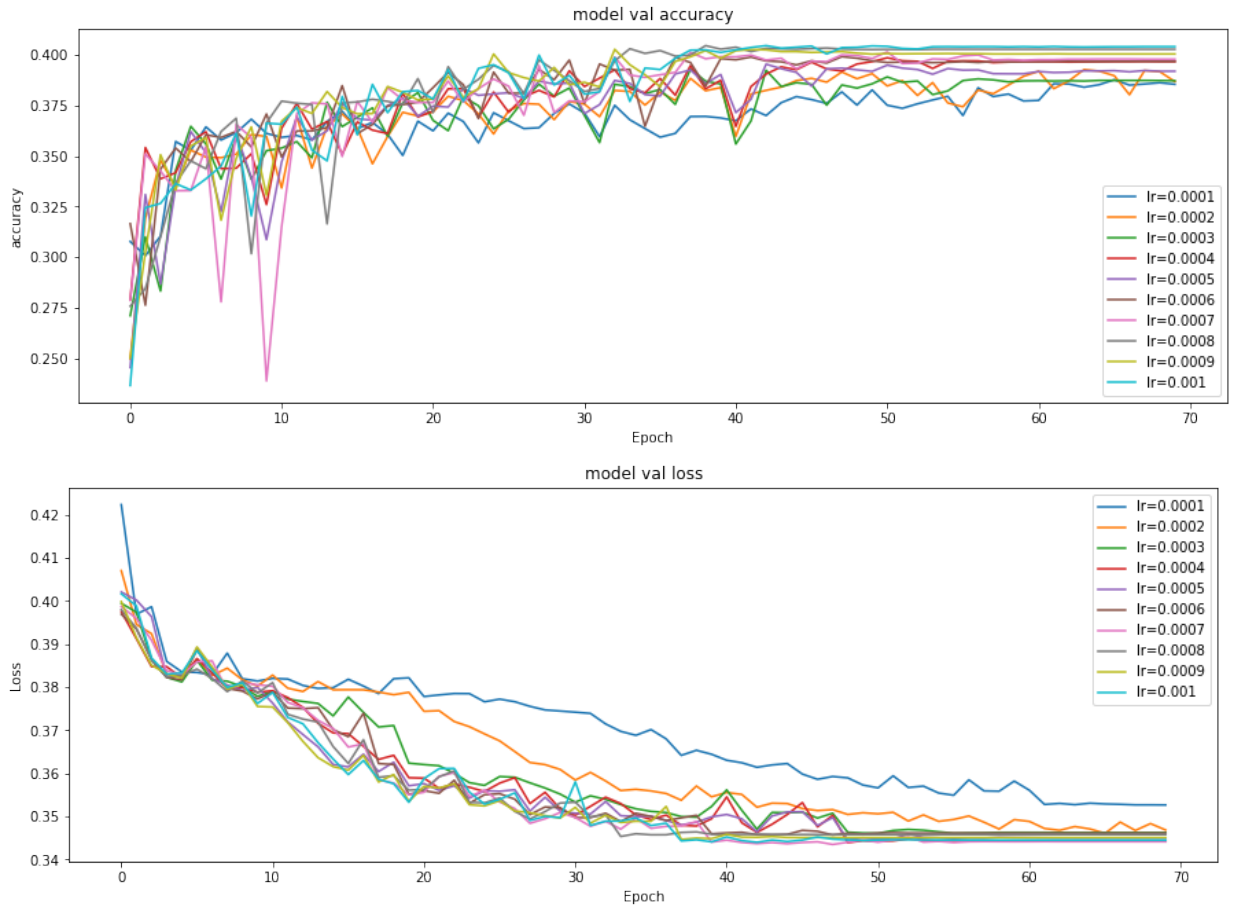


FIGURE 5.11: La précision et le taux d'erreur de validation avec différents learning rate (model RNN)

Après la comparaison, learning rate avec la valeur **0.0009** donne le meilleur résultat.

5.5.1.3 Batch size :

La figure 5.12 présente la précision et le taux d'erreur d'apprentissage optenues avec batch-size 0, 32, 64 128, 256.

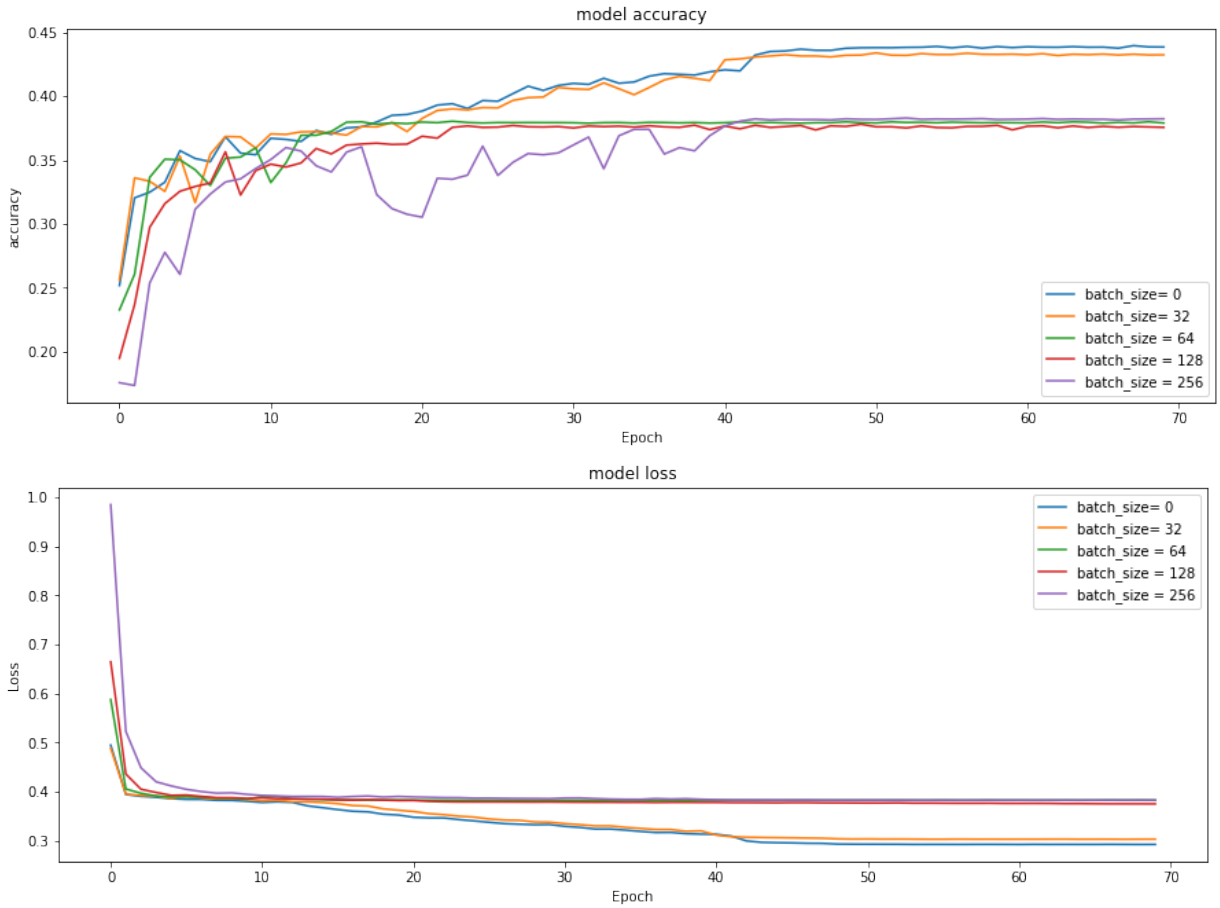
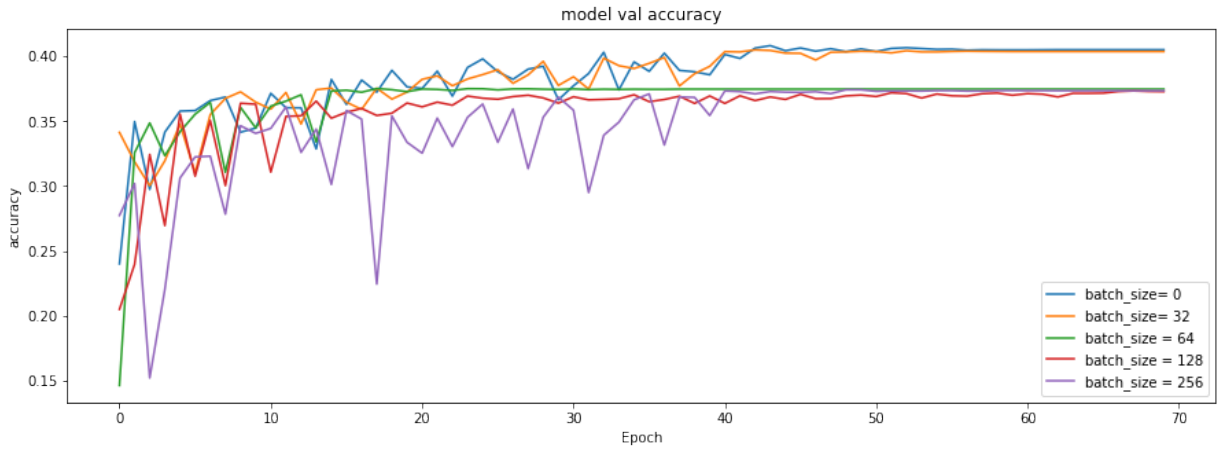


FIGURE 5.12: La précision et le taux d'erreur d'apprentissage avec diffèrents batch-size (model RNN)

Pour la validation, les résultats dans la figure 5.13 sont obtenues.



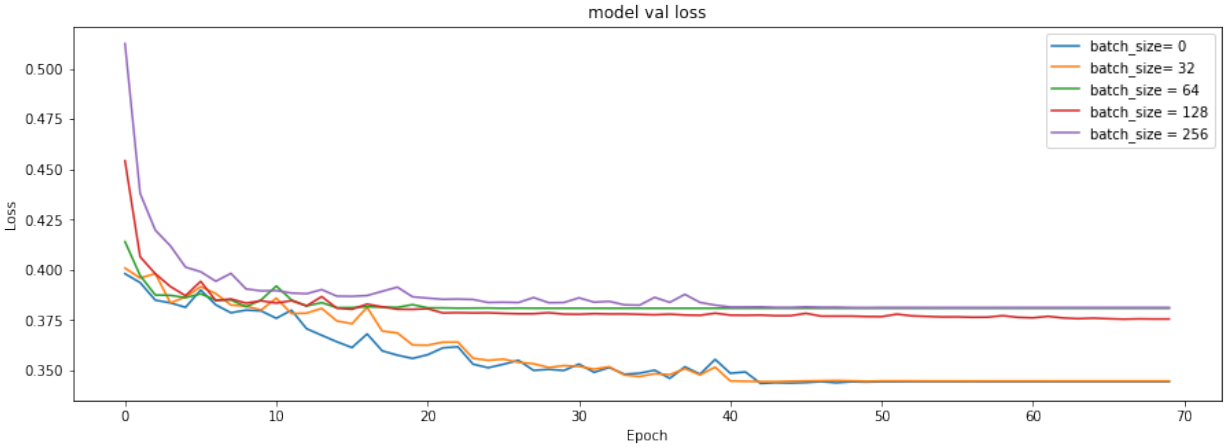
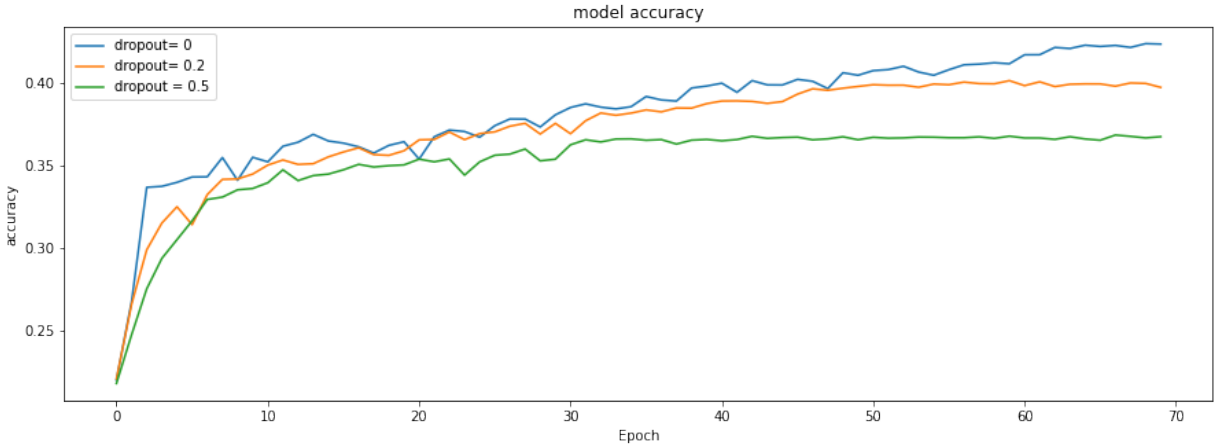


FIGURE 5.13: La précision et le taux d'erreur de validation avec différents batch-sizes

Après la comparaison, le batch-sise avec la valeur **0** donne le meilleur résultat.

5.5.1.4 Dropout

Les résultats dans la figure 5.14 représentent la précision et le taux d'erreur durant la phase de validation avec les taux de dropout 0, 0.2, 0.5.



Pour la validation, les résultats dans la figure 5.15 sont obtenues.

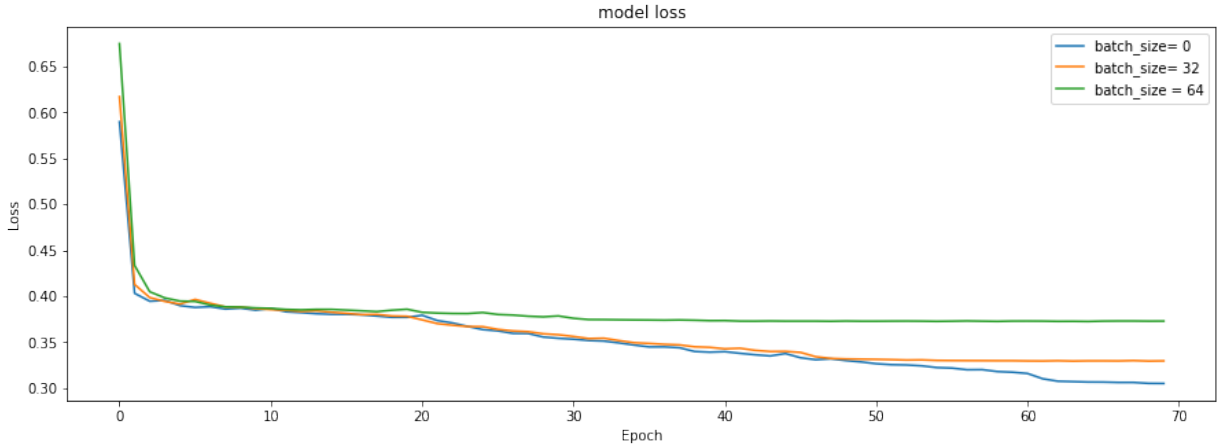


FIGURE 5.14: La précision et le taux d'erreur d'apprentissage avec diffèrents dropout (model RNN)

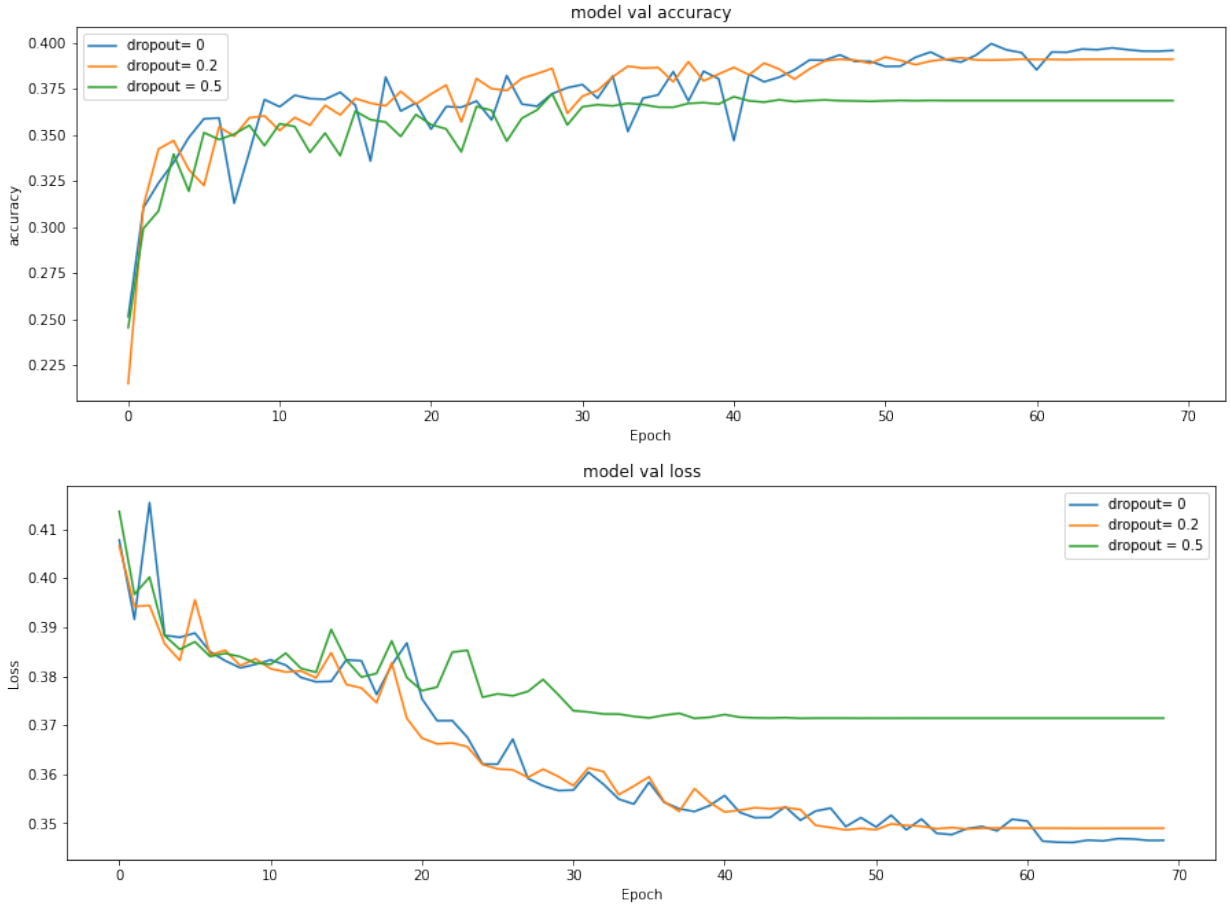


FIGURE 5.15: La précision et le taux d'erreur de validation avec différents dropout(model RNN)

Après la comparaison, le dropout avec la valeur 0 donne le meilleur résultat.

5.5.2 Model LSTM

5.5.2.1 Algorithm d'optimisation :

présente la précision et le taux d'erreur d'apprentissage optenues avec les défirent algorithme d'optimisation

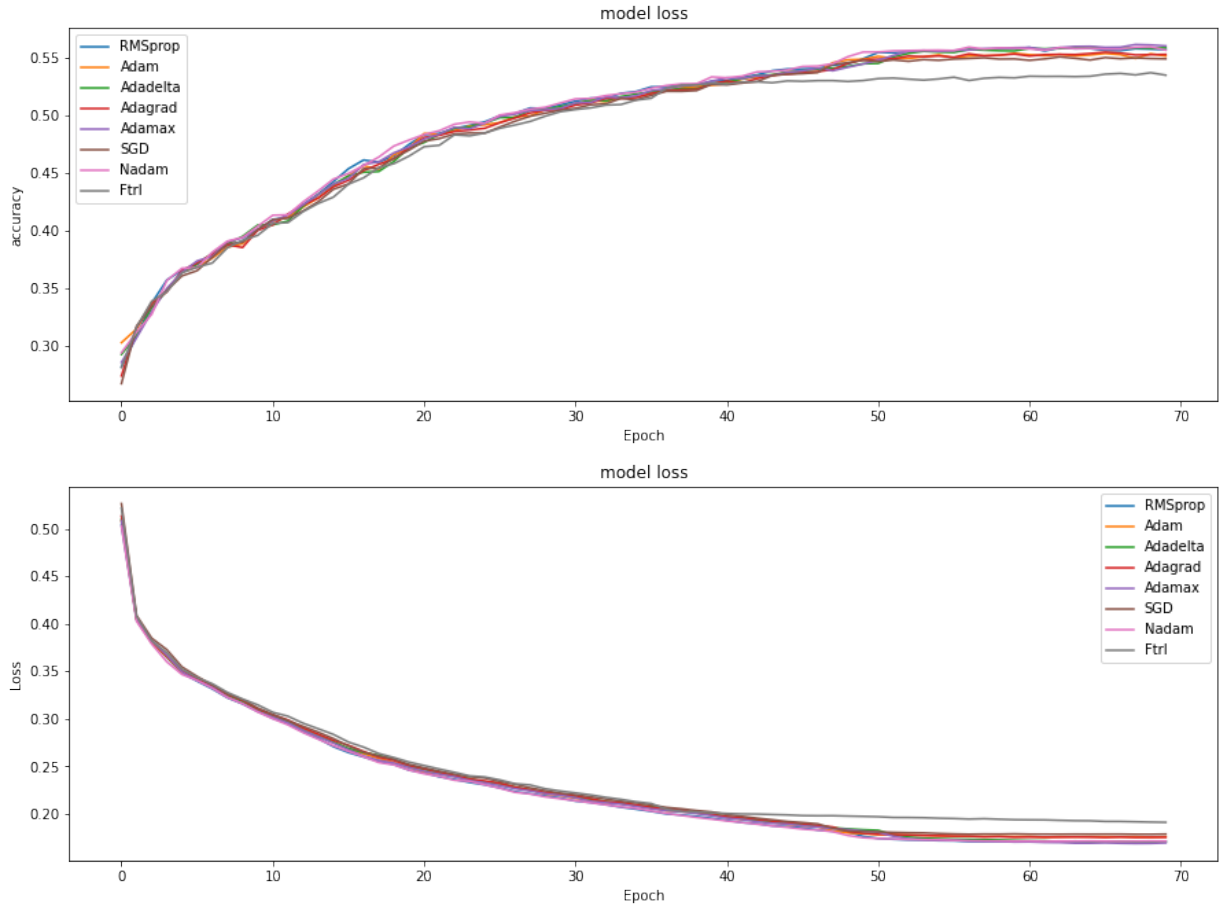


FIGURE 5.16: La précision et le taux d'erreur d'apprentissage avec diffèrents Algorithms (model LSTM)

Pour la validation, les résultats dans la figure 5.17 sont obtenues.

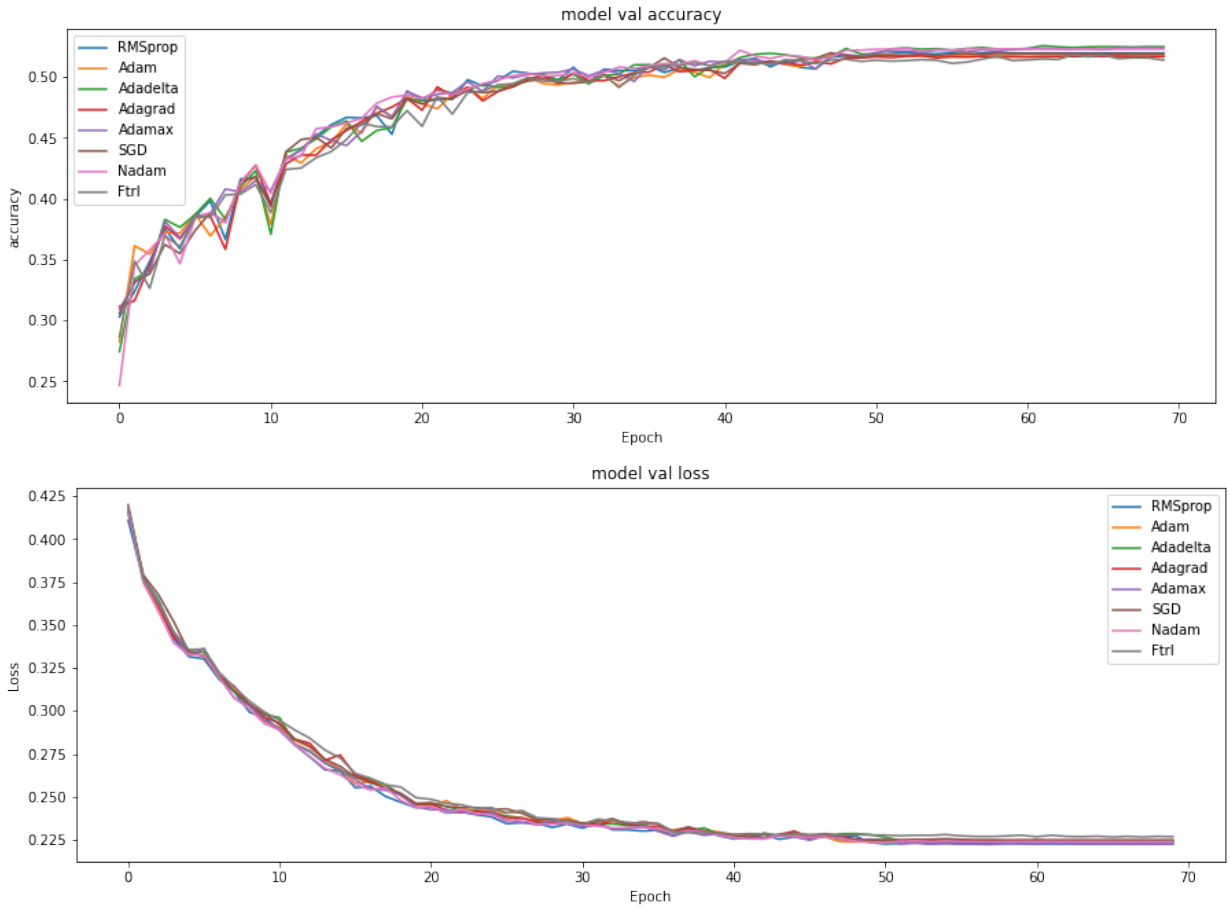


FIGURE 5.17: La précision et le taux d'erreur de validation avec diffèrents Algorithms(model LSTM)

Après la comparaison, l'algorithme **Adamax** pour mettre à jour les poids du réseau donne le meilleur résultat.

5.5.2.2 learning rate

La figure 5.18 présente la précision et le taux d'erreur d'apprentissage optenues avec learning rate 0.0001-0.001

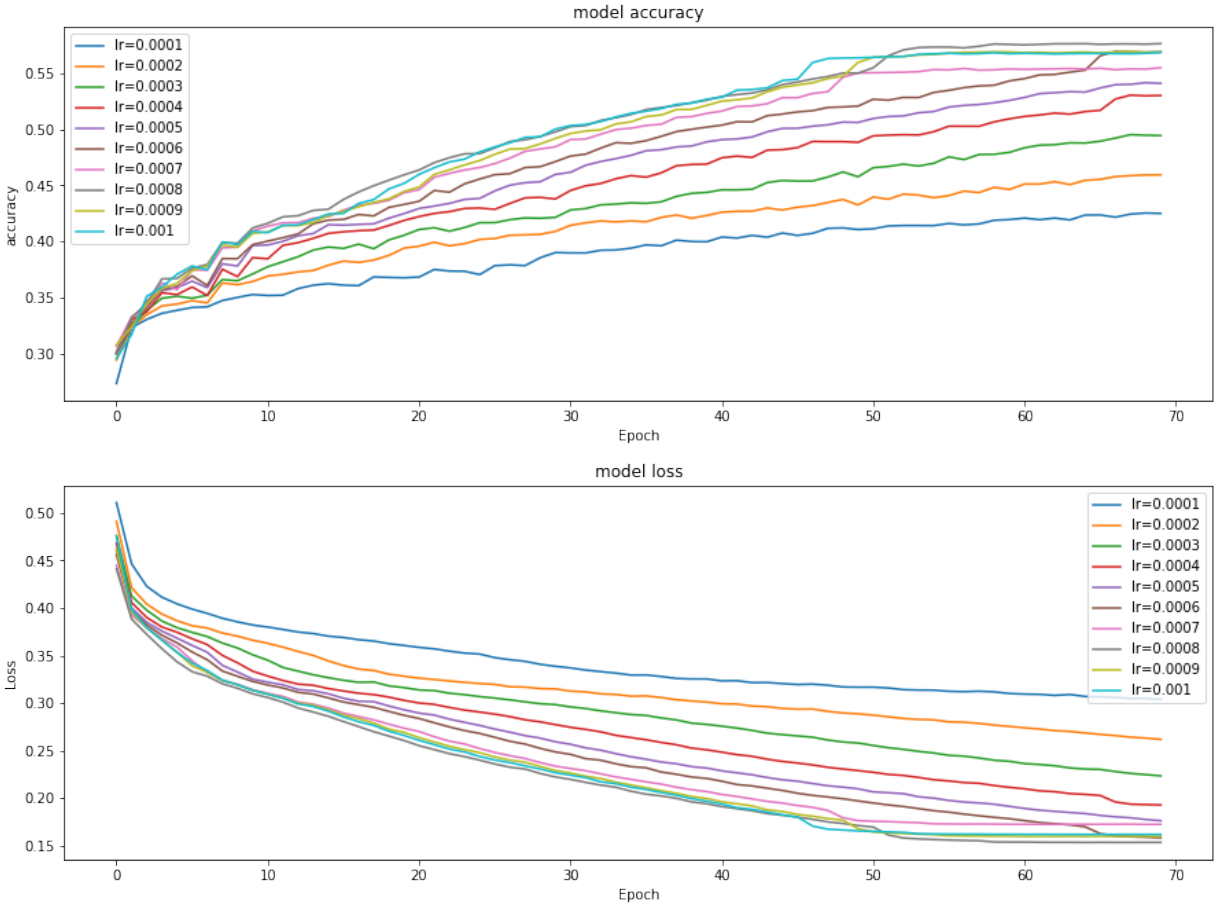
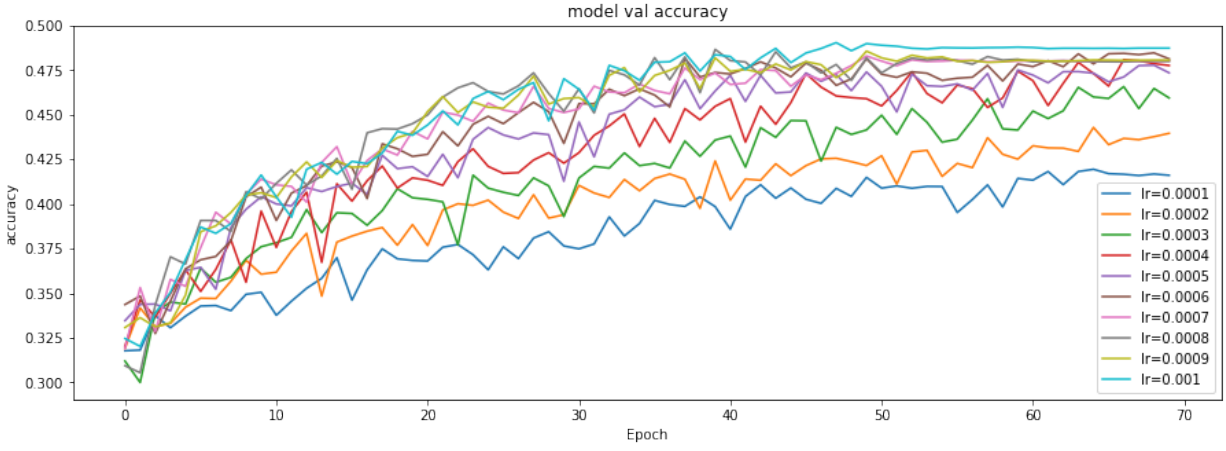


FIGURE 5.18: La précision et le taux d'erreur d'apprentissage avec diffèrents learning rate (model LSTM)

Pour la validation, les résultats dans la figure 5.19 sont obtenues.



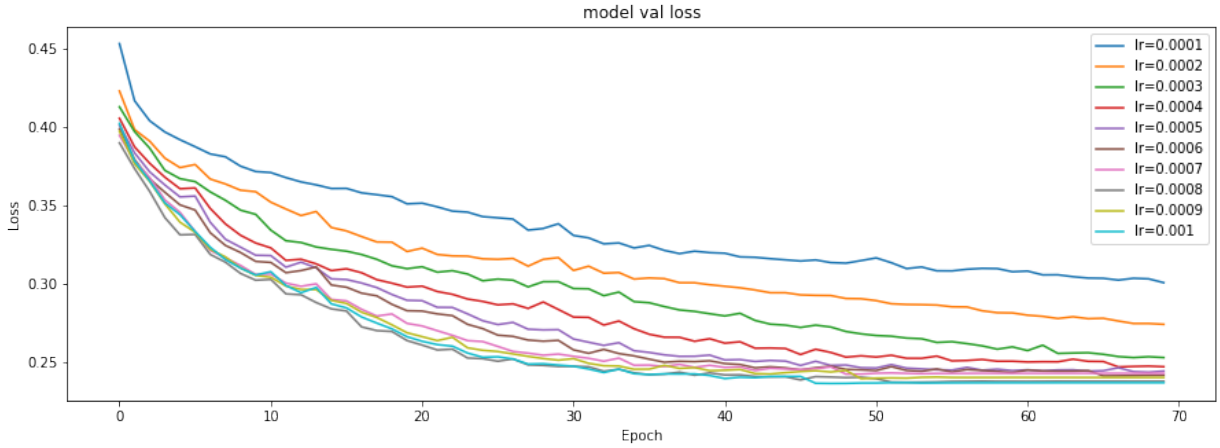
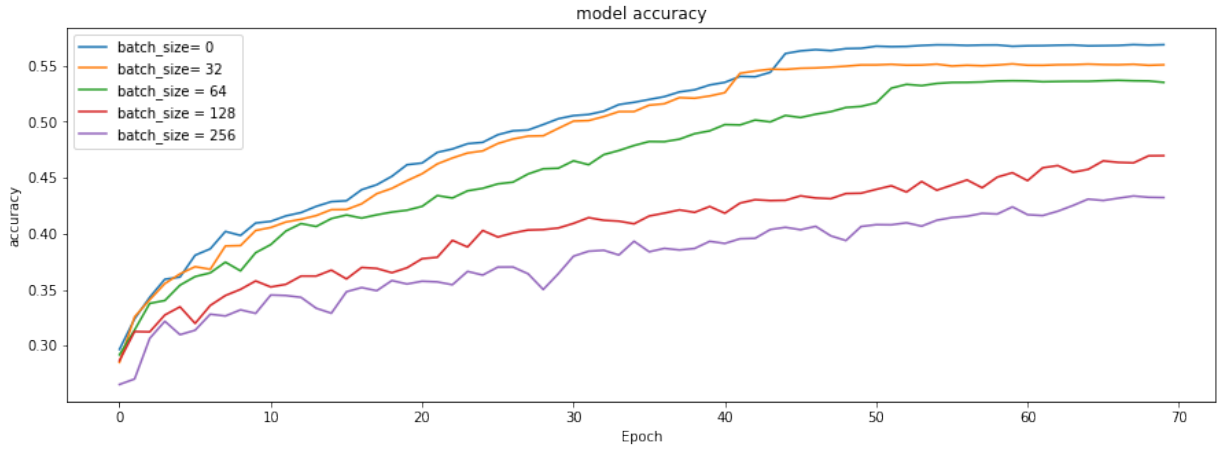


FIGURE 5.19: La précision et le taux d'erreur de validation avec différents learning rate (model LSTM)

Après la comparaison, learning rate avec la valeur **0.0008** donne le meilleur résultat.

5.5.2.3 Batch size :

La figure 5.20 présente la précision et le taux d'erreur d'apprentissage optenues avec batch-size 0, 32, 64 128, 256.



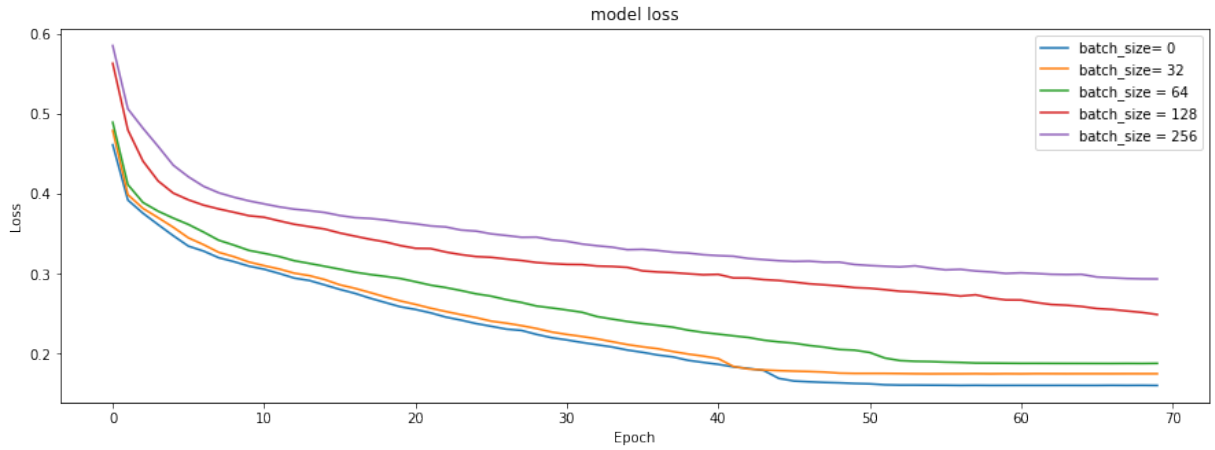
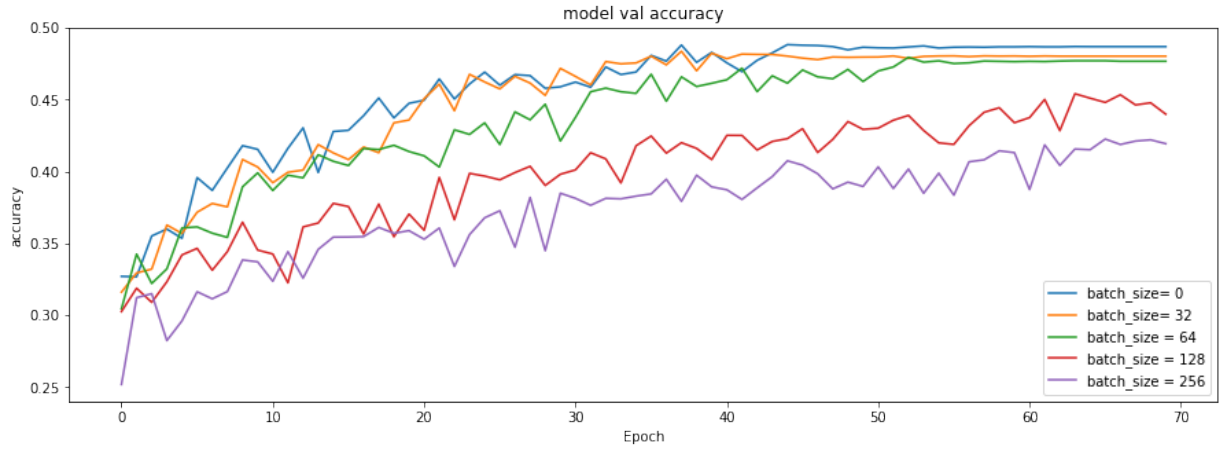


FIGURE 5.20: La précision et le taux d'erreur d'apprentissage avec diffèrents batch-size (model LSTM)

Pour la validation, les résultats dans la figure 5.21 sont obtenues.



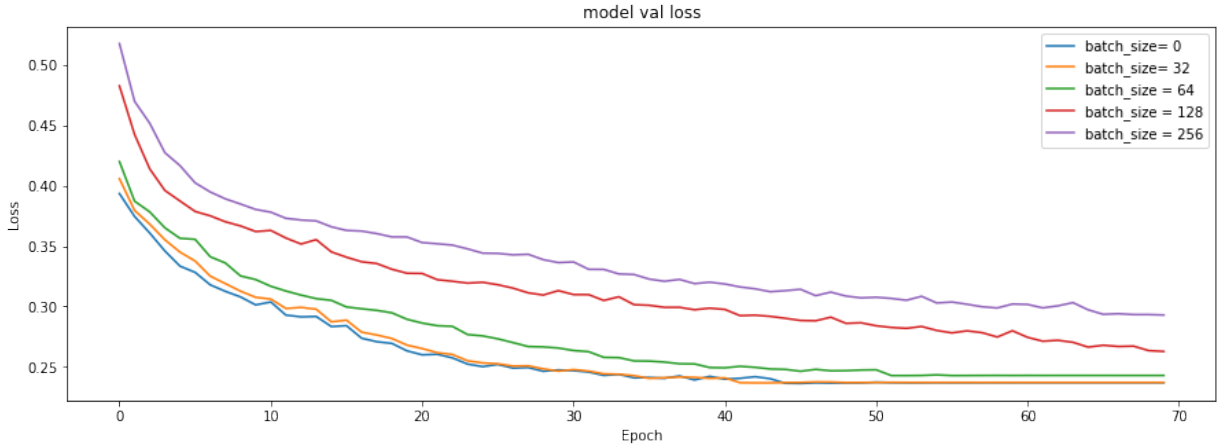
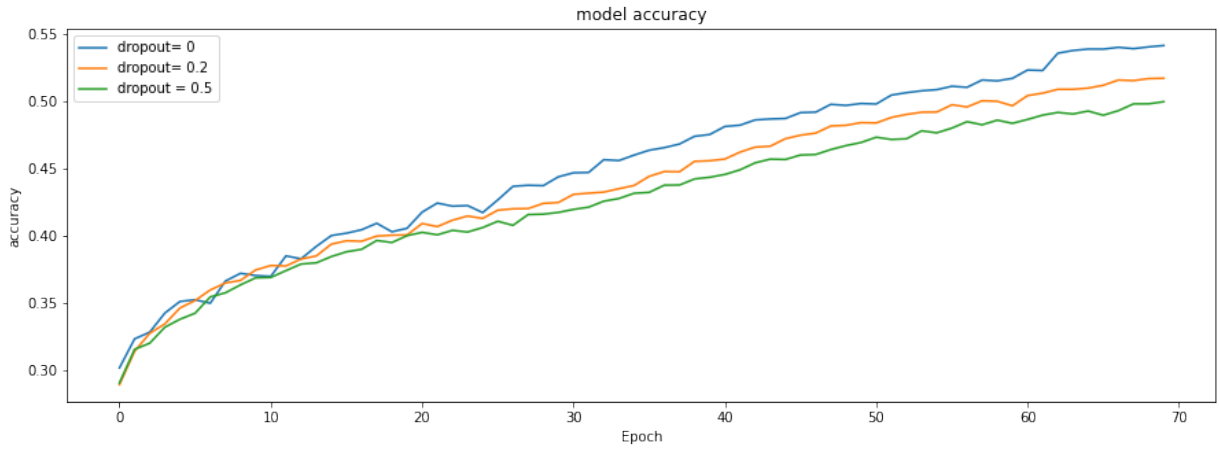


FIGURE 5.21: La précision et le taux d'erreur de validation avec différents batch-sizes (model LSTM)

Après la comparaison, le batch-sise avec la valeur **0** donne le meilleur résultat.

5.5.2.4 Dropout

code python pour changement de Dropout Les résultats dans la figure 5.22 représentent la précision et le taux d'erreur durant la phase de validation avec les taux de dropout 0, 0.2,0.5.



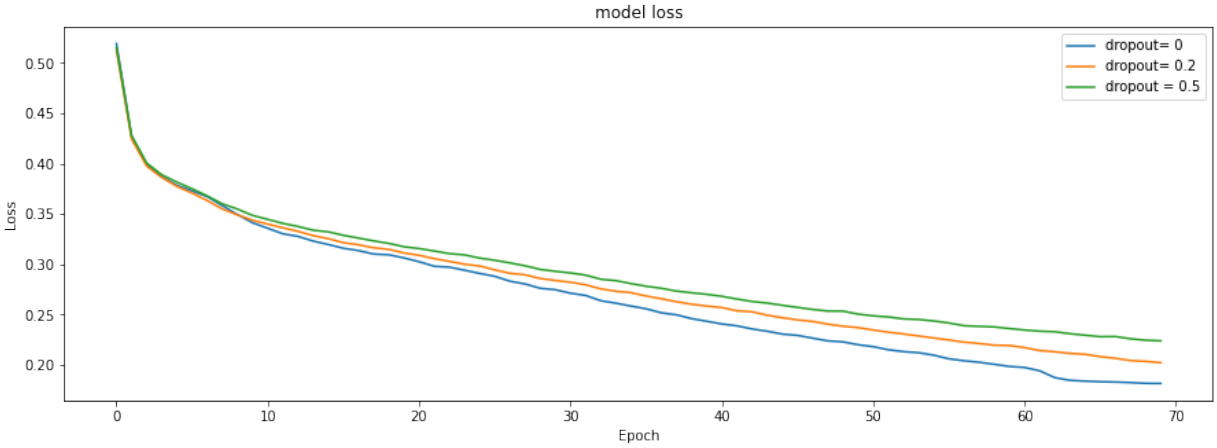
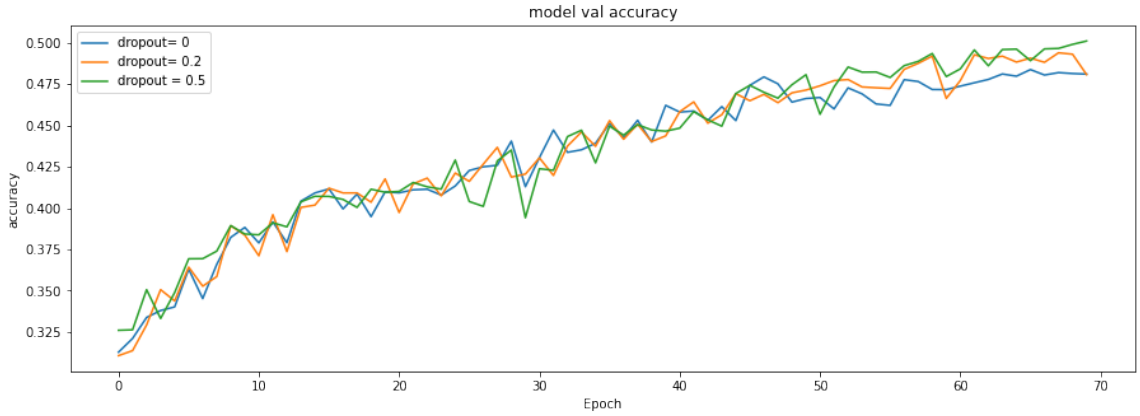


FIGURE 5.22: La précision et le taux d'erreur d'apprentissage avec diffèrents dropout(model LSTM)

Pour la validation, les résultats dans la figure 5.23 sont obtenues.



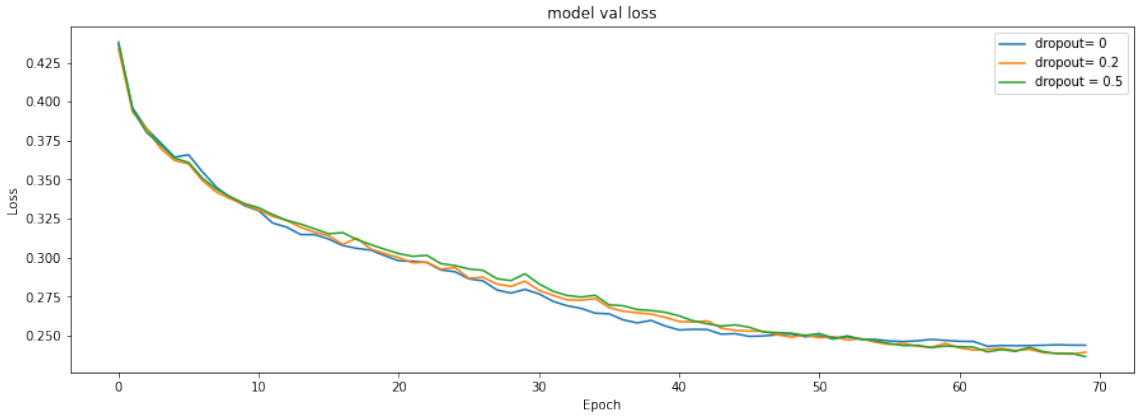
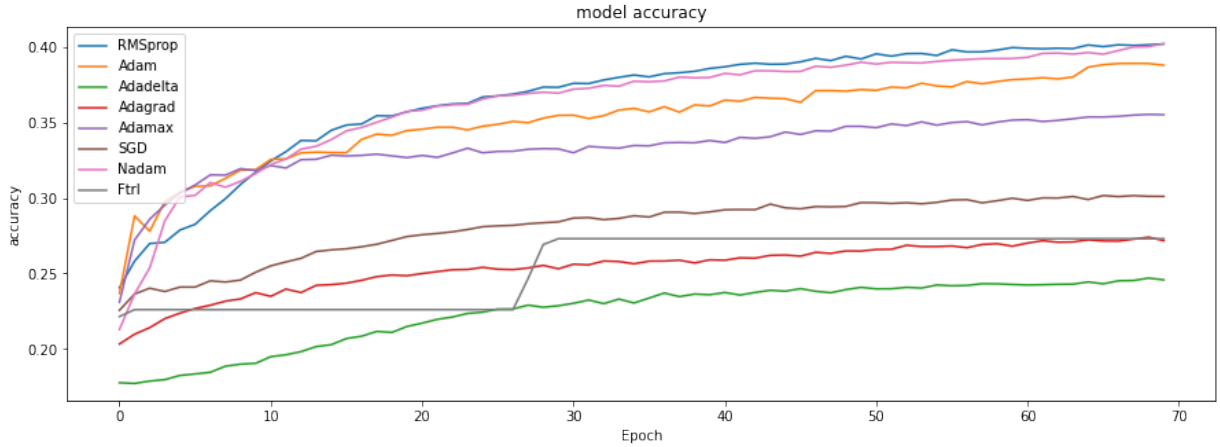


FIGURE 5.23: La précision et le taux d'erreur de validation avec différents dropout (model LSTM)

5.5.3 Model Bert

5.5.3.1 Algorithm d'optimisation :

présente la précision et le taux d'erreur d'apprentissage obtenues avec les défirent algorithme d'optimisation



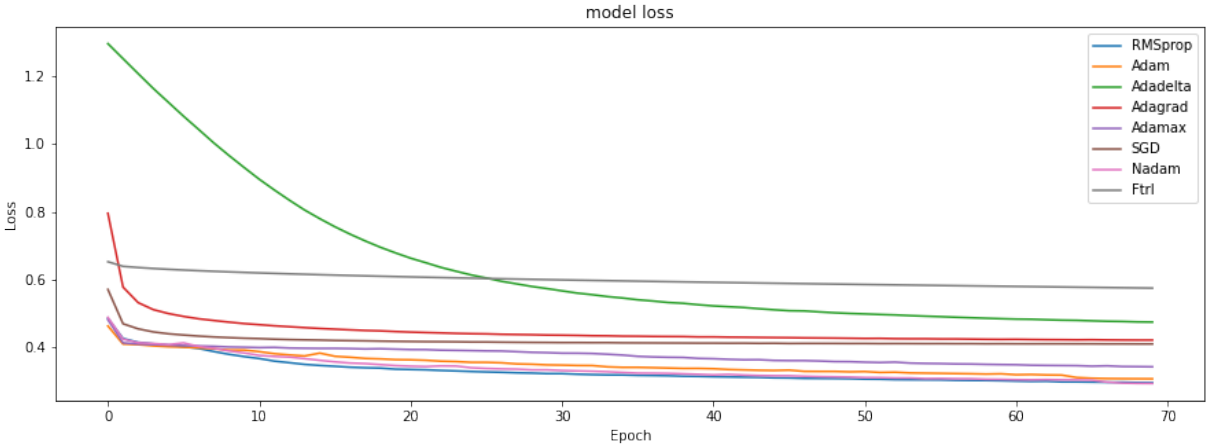
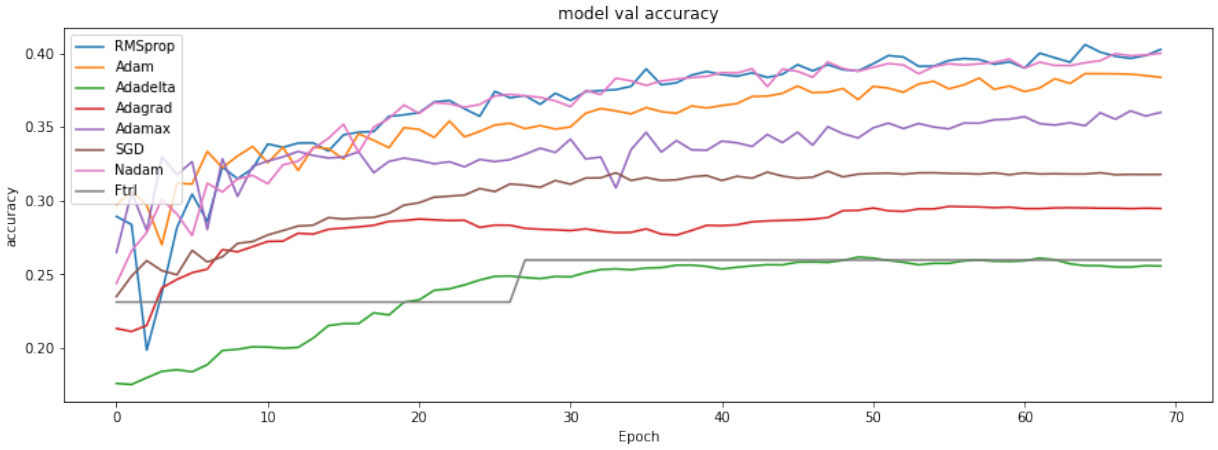


FIGURE 5.24: La précision et le taux d'erreur d'apprentissage avec différents Algorithms (model Bert)

Pour la validation, les résultats dans la figure 5.25 sont obtenues.



Après la comparaison, l'algorithme **RMSprop** pour mettre à jour les poids du réseau donne le meilleur résultat.

5.5.3.2 learning rate

La figure 5.26 présente la précision et le taux d'erreur d'apprentissage obtenues avec learning rate 0.0001-0.001

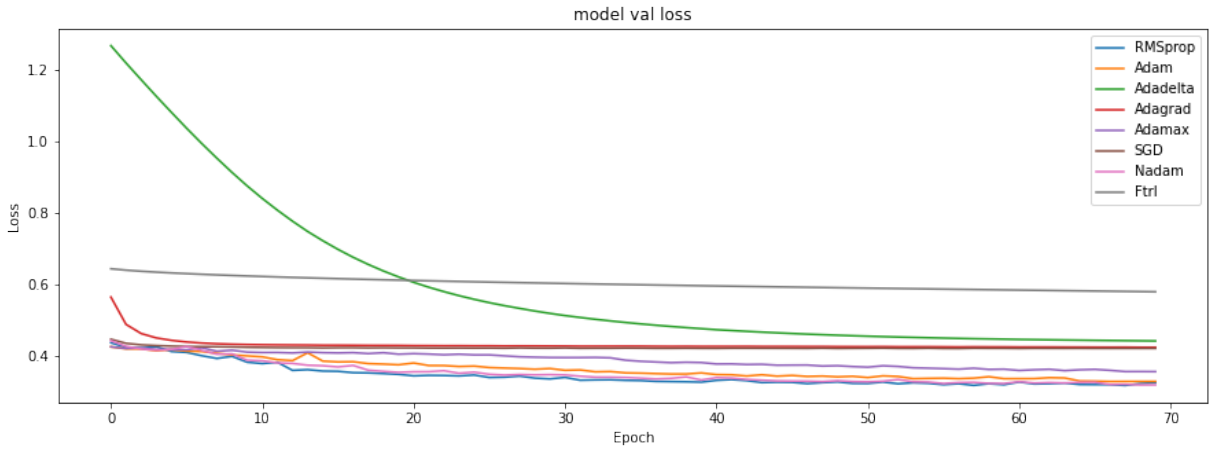


FIGURE 5.25: La précision et le taux d'erreur de validation avec diffèrents Algorithms (model Bert)

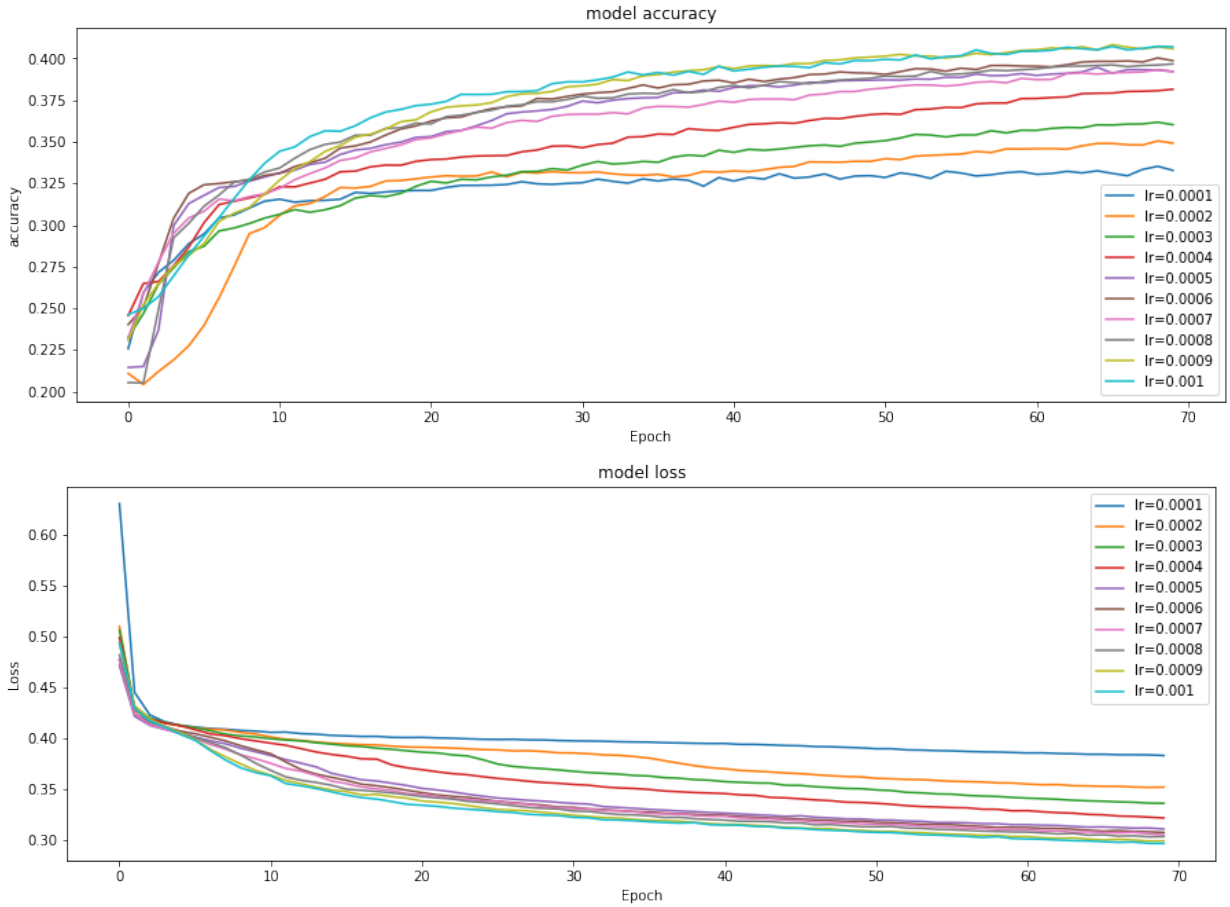


FIGURE 5.26: La précision et le taux d'erreur d'apprentissage avec diffèrents learning rate (model Bert)

Pour la validation, les résultats dans la figure 5.27 sont obtenues.

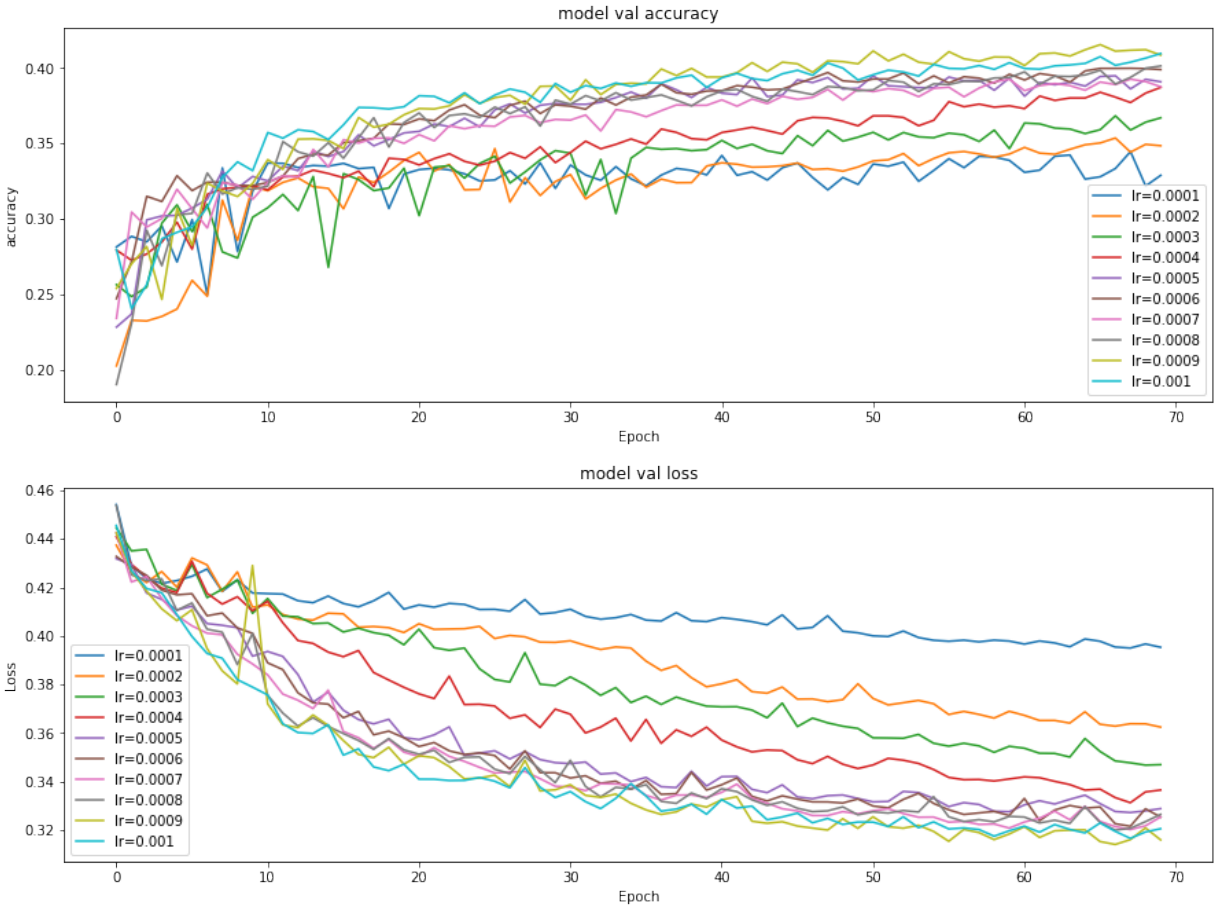


FIGURE 5.27: La précision et le taux d'erreur de validation avec différents learning rate (model Bert)

Après la comparaison, learning rate avec la valeur **0.0009** donne le meilleur résultat.

5.5.3.3 Batch size :

La figure 5.28 présente la précision et le taux d'erreur d'apprentissage obtenu avec batch-size 0, 32, 64 128, 256

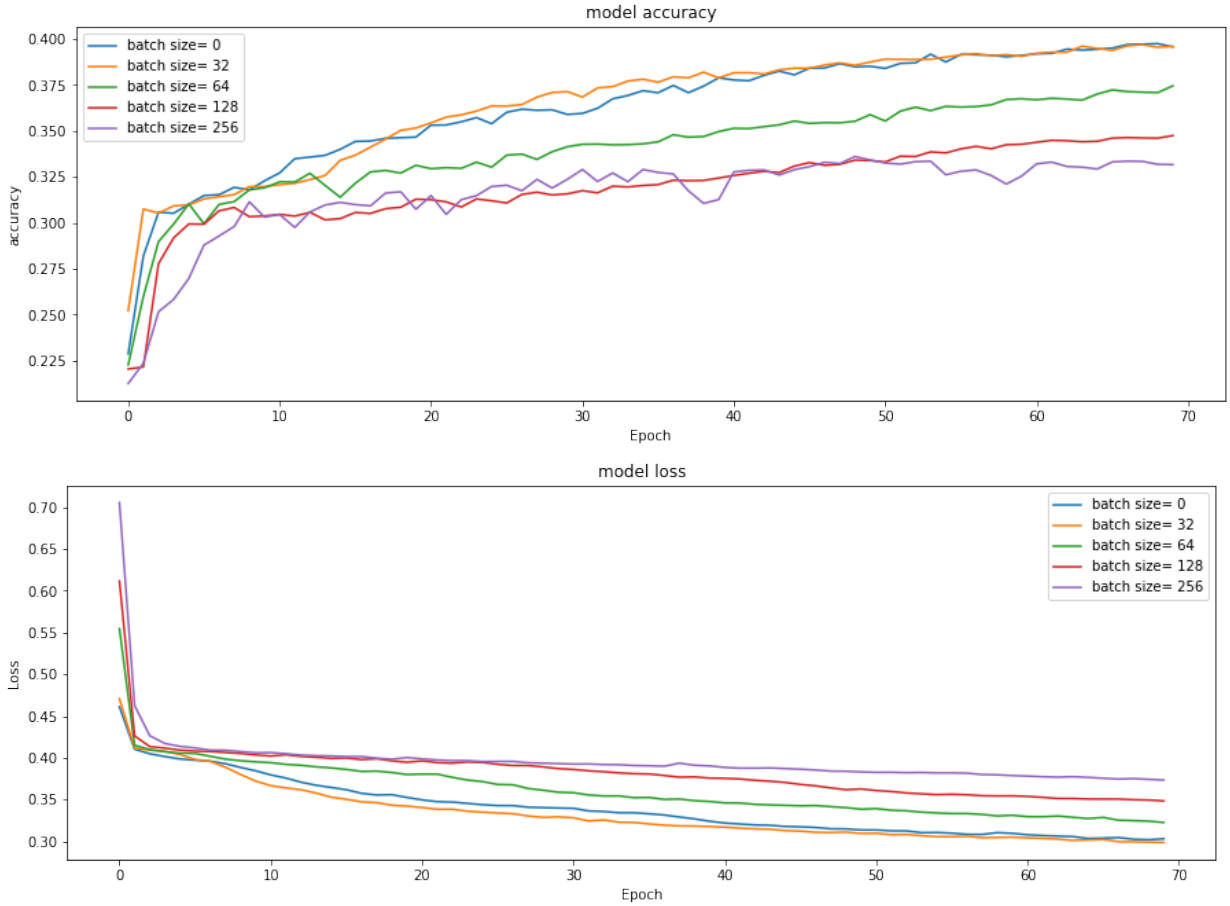


FIGURE 5.28: La précision et le taux d'erreur d'apprentissage avec diffèrents batch-size (model Bert)

Pour la validation, les résultats dans la figure 5.29 sont obtenues.

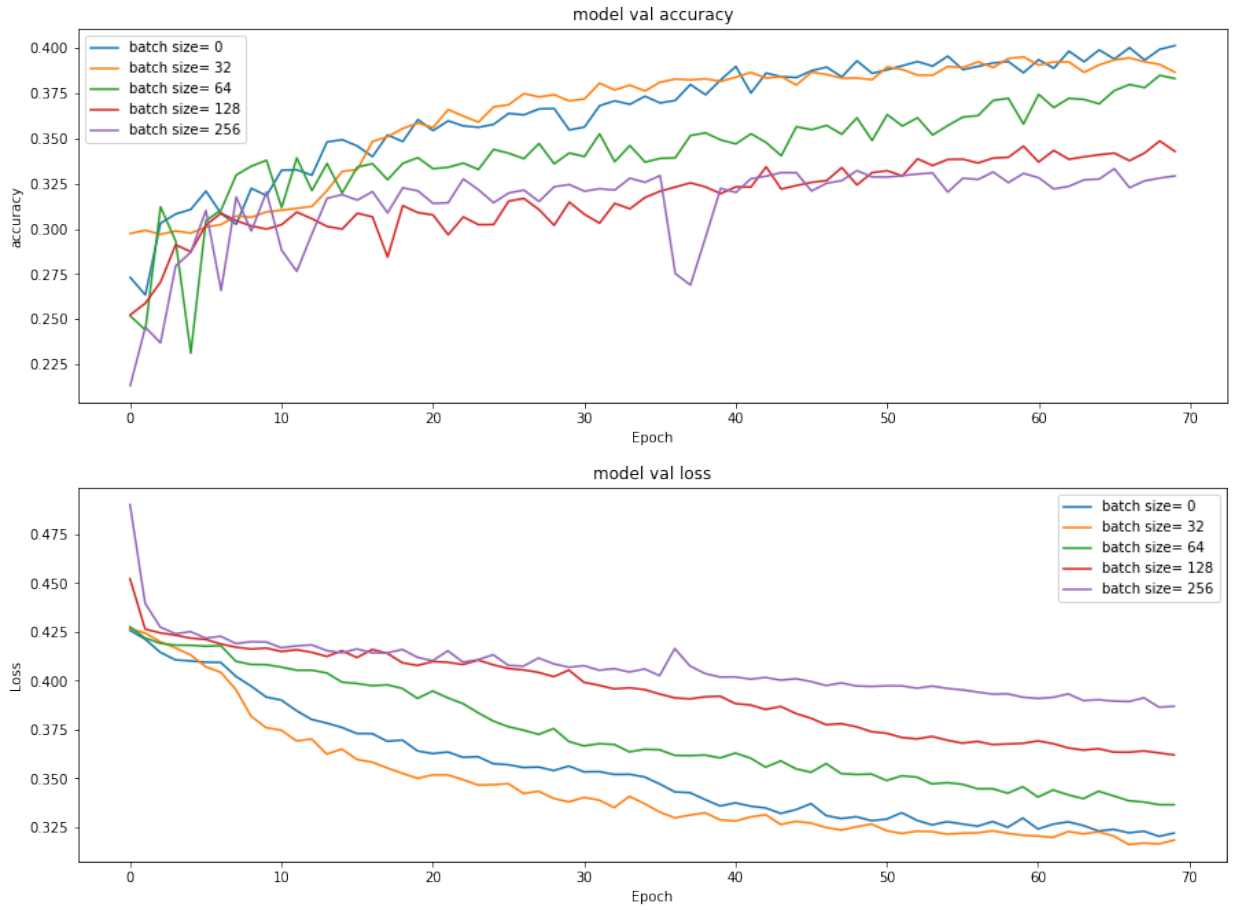


FIGURE 5.29: La précision et le taux d'erreur de validation avec différents batch-sizes (model Bert)

Après la comparaison, le batch-sise avec la valeur **32** donne le meilleur résultat.

5.5.3.4 Dropout

Les résultats dans la figure 5.30 représentent la précision et le taux d'erreur durant la phase de validation avec les taux de dropout 0, 0.2,0.5.

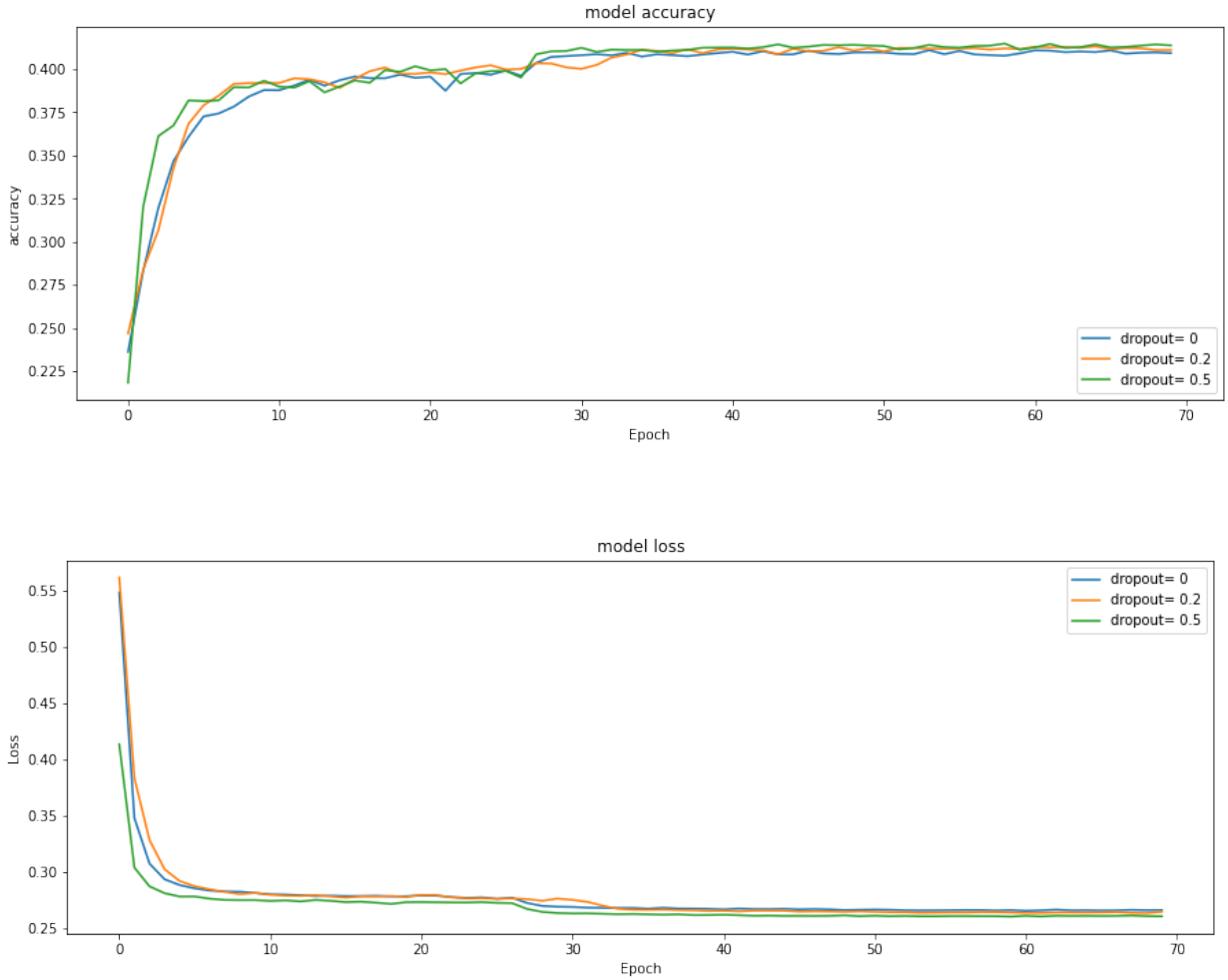
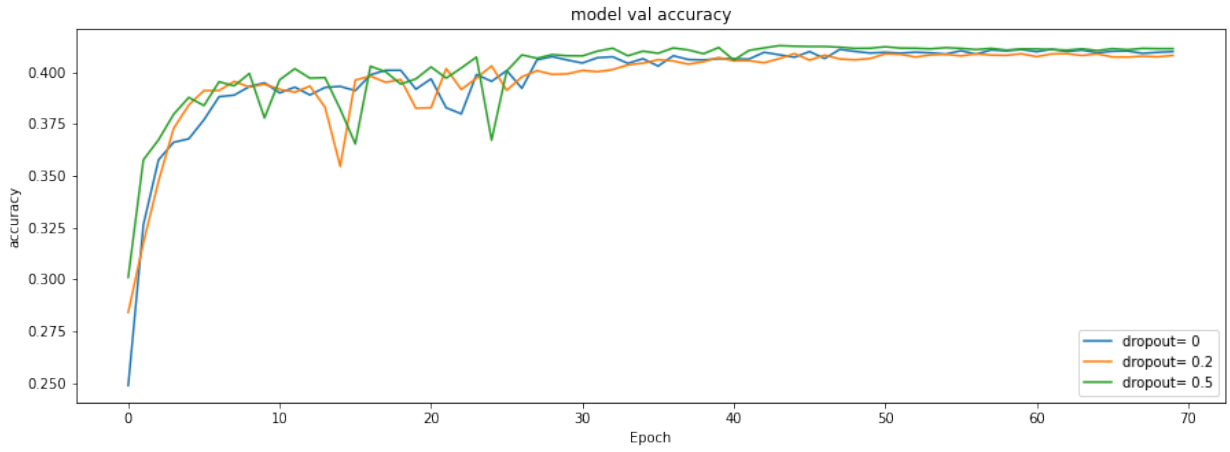


FIGURE 5.30: La précision et le taux d'erreur d'apprentissage avec diffèrents dropout (model Bert)

Pour la validation, les résultats dans la figure 5.31 sont obtenues.



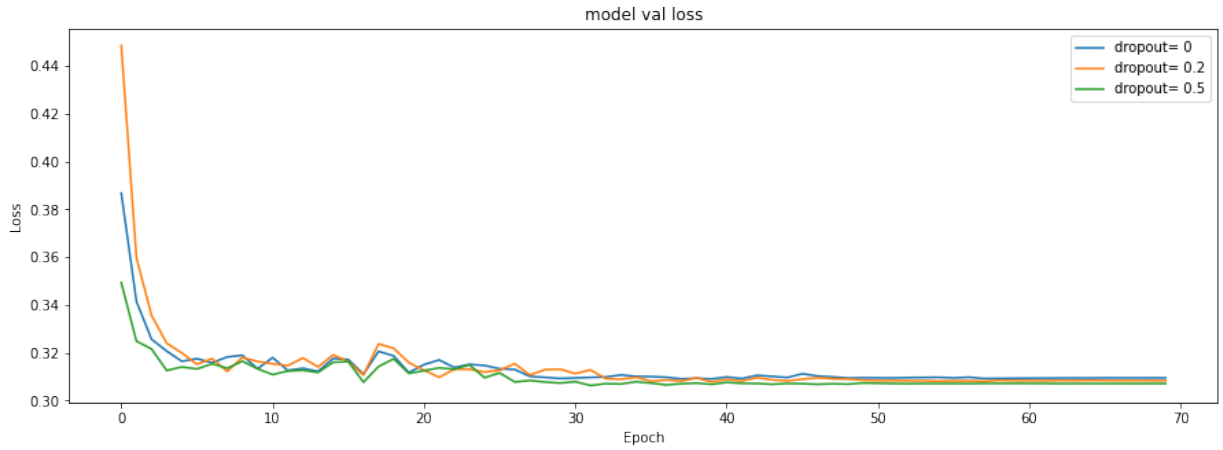


FIGURE 5.31: La précision et le taux d'erreur de validation avec différents dropout (model Bert)

Après la comparaison, dropout avec la valeur **0.5** donne le meilleur résultat.

5.6 Configuration finale

Après les expérimentations, nous avons atteidre au configurations des paramètres dans le tableau 5.3

Modèle	Algorithm d'optimisation	learning rate	Batch size	Dropout
RNN	Adam	0.001	0	0
LSTM	Adamax	0.0008	0	0
Bert	RMSprop	0.0009	32	0.5

TABLE 5.3: configurations du modèles

5.7 Résultats finaux

Les figures 5.33,5.33 et 5.32 représentent loss et accuracy, Précision, Rappelle et f1 score de chaque modèle avec utilisation de la method validation croisée k-fold, k=4 pour amilioré les modèles.

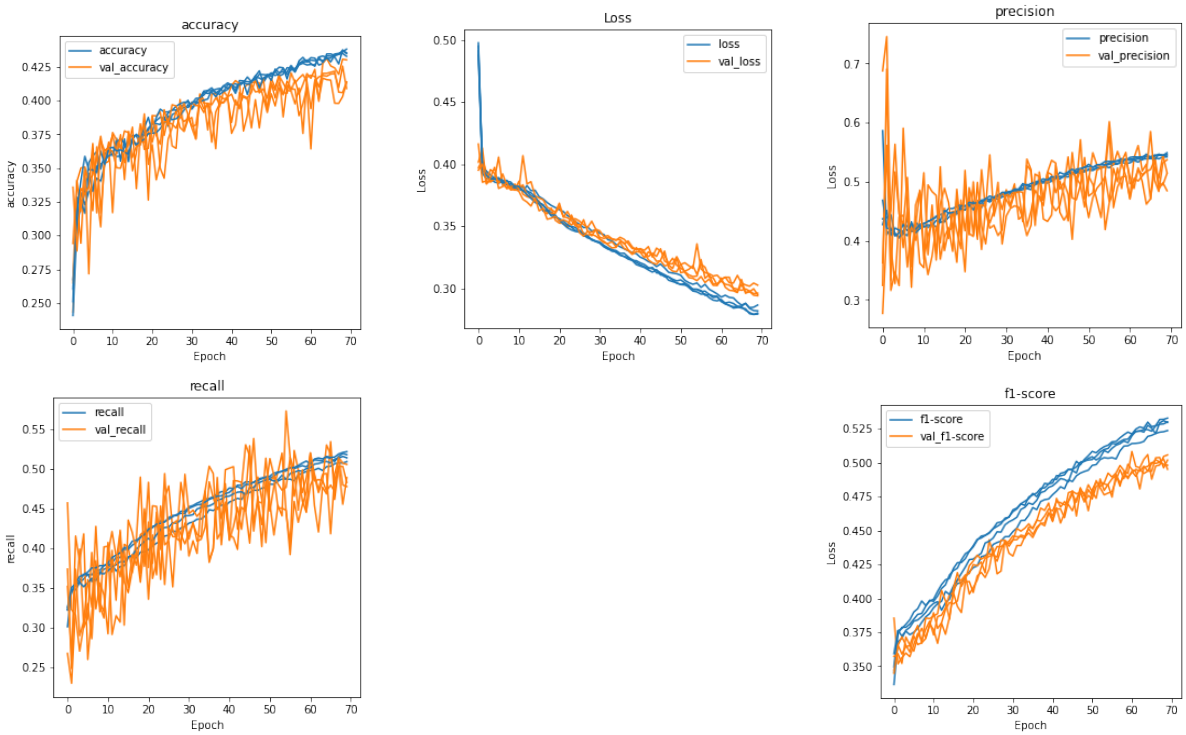


FIGURE 5.32: l'erreur et accuracy, Précision, Rappelle et f1 score du model RNN

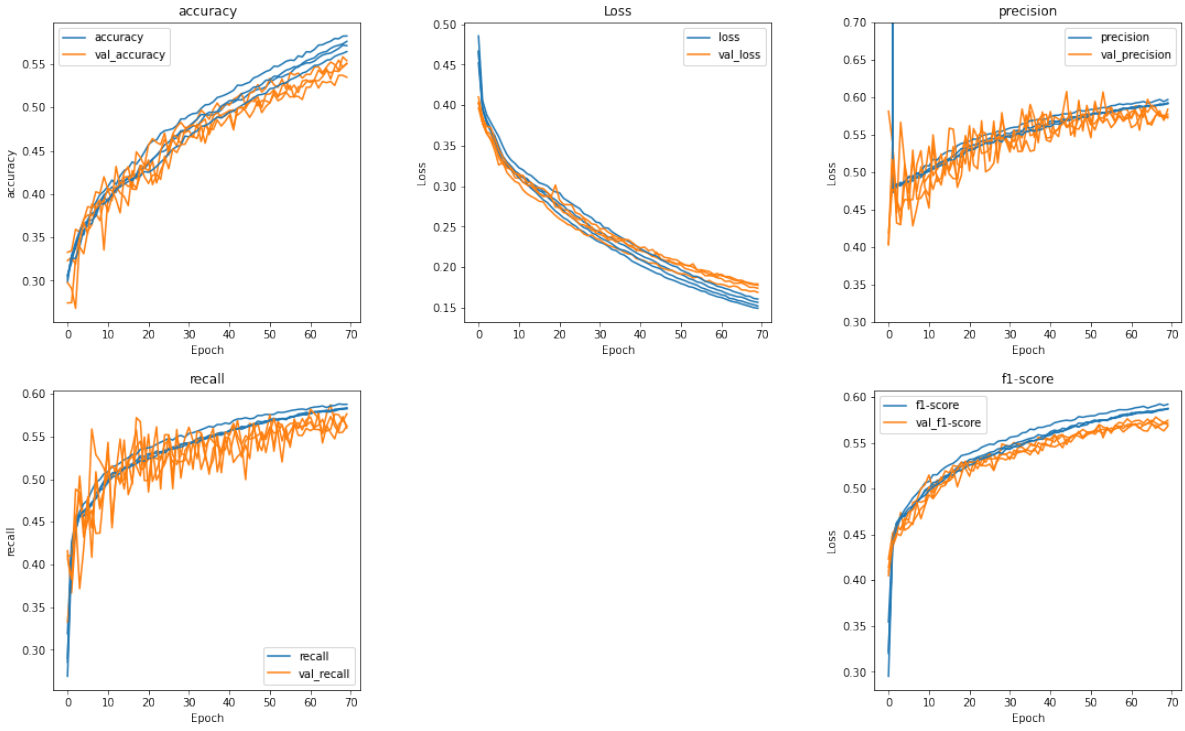


FIGURE 5.33: l'erreur et accuracy, Précision, Rappelle et f1 score du model LSTM

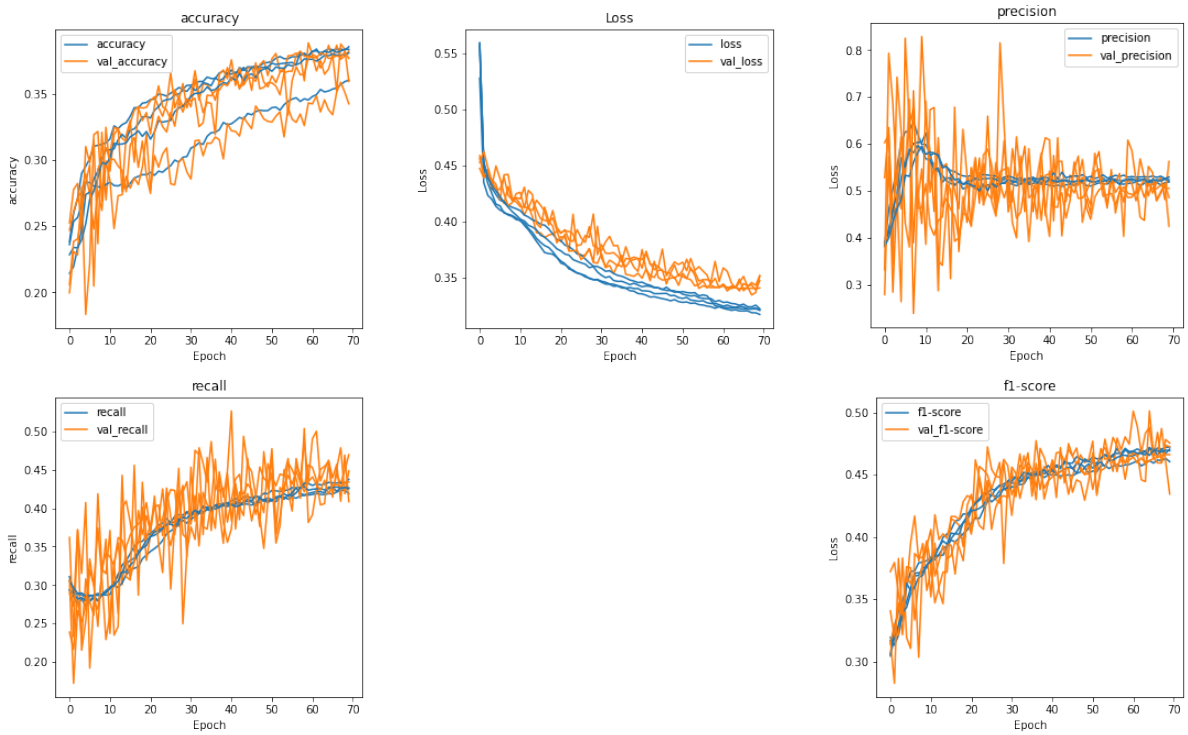


FIGURE 5.34: l'erreur et accuracy, Précision, Rappelle et f1 score du model Bert

Dans le tableau suivant, nous résumons le résultat obtenu pour les trois modèle proposé :

	Training			validation		
	LSTM	RNN	BERT	LSTM	RNN	BERT
Loss	0.1492	0.2792	0.3173	0.1690	0.2939	0.3515
accuracy	0.5826	0.4352	0.3810	0.5541	0.4134	0.3595
Précision	0.5968	0.5453	0.5293	0.5725	0.5371	0.4244
Rappelle	0.5877	0.5221	0.4254	0.5766	0.4780	0.4484
F1 score	0.5921	0.5329	0.4704	0.5745	0.5058	0.4344

Les trois figure Les figures 5.33,5.33 et 5.32 représentent les résulta finaux de la phase d'apprentissage La première remarque que nous allons vus, c'est instabilité de la phase validation Cela peut se produire, car l'ensemble de données de validation contient trop peu d'exemples par rapport à l'ensemble de données d'apprentissage.
De plus, comme on peut le voir dans le tableau 5.7, les performances du modèle de base LSTM sont les meilleures. Le taux d'erreur (1,3 à 1,6) et la précision (1,4 à 0,2) sont respectivement meilleurs que RNN et Bert. La précision proposée par le trois modèles est controversé avec Résultats de convergence (0,05 à 0,06), mais le modèle de Bert donne des résultats de f1 score (0,16 à 0,09) et un rappel de (0,12 à 0,06) que LSTM et RNN respectivement.

5.8 Comparaisons enter les trios modéle

les figures 5.35 5.36, 5.37 5.38 représentant la précision et l'erreur de chaque modèle.

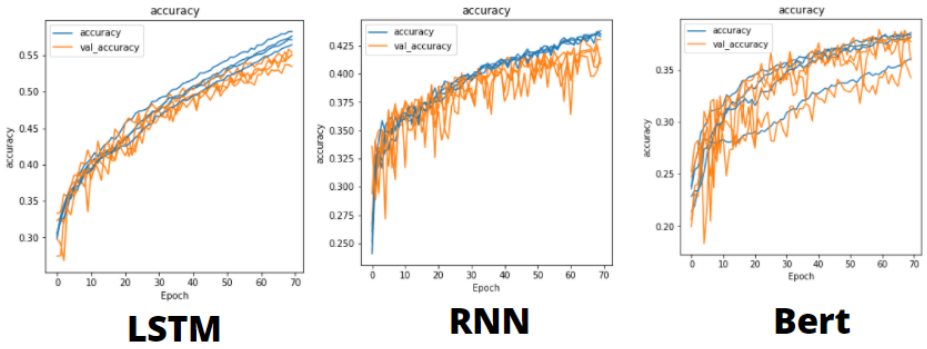


FIGURE 5.35: La précision du modèle LSTM, RNN, BERT respectivement

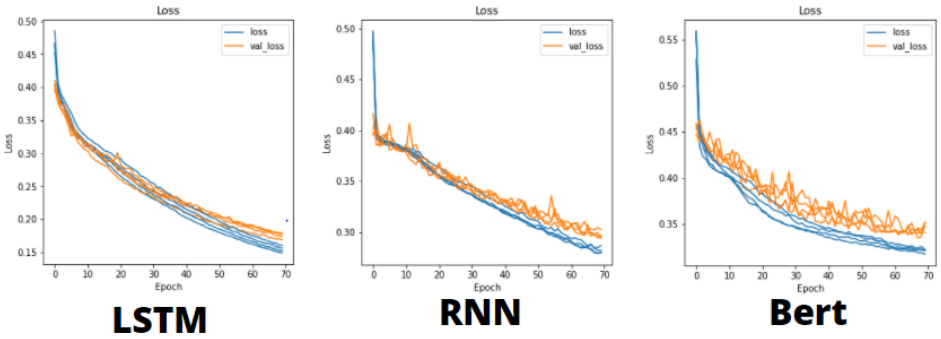


FIGURE 5.36: l'erreur du modèle LSTM, RNN, BERT respectivement

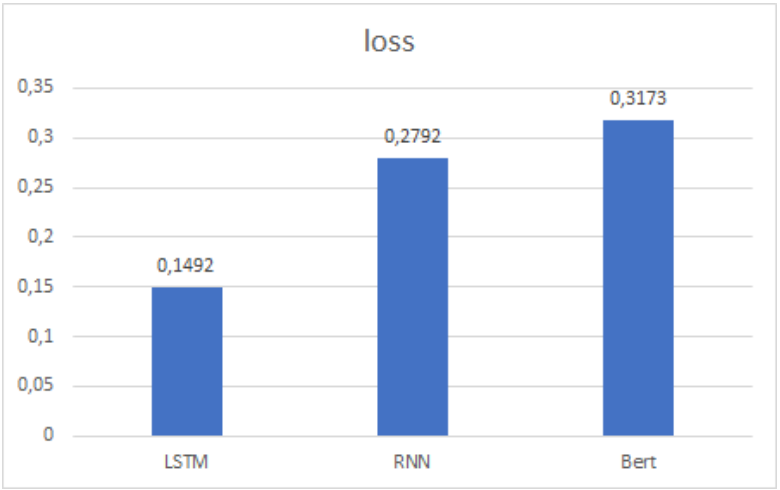


FIGURE 5.37: Comparaison de la précision entre les modèles proposés

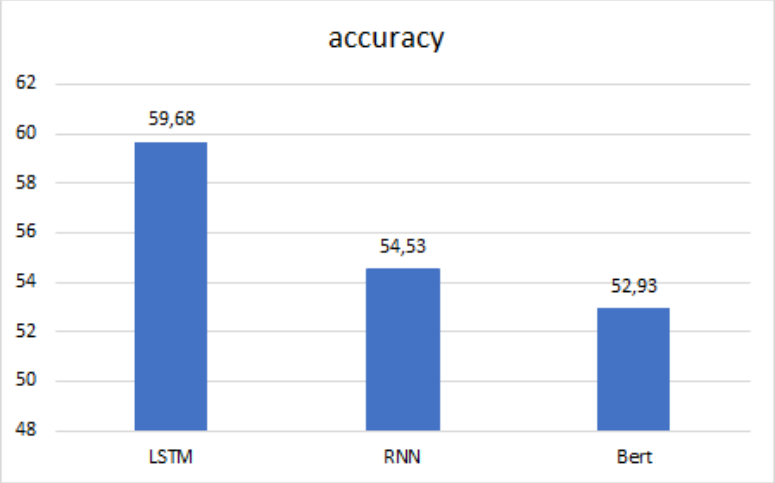


FIGURE 5.38: Comparaison de l'erreur entre les modèles proposés

Comme on peut le constater après la phase initiale d'apprentissage et de validation, le LSTM conserve les meilleures performances avec loss (0,1492) et précision (0,5826) puis RNN et Bert avec loss (0,2792, 0,3173) et précision (0,4352, 0,3810). Mais on ne peut pas comparer les model juste sur accuracy et loss.
L'un des avantages de l'utilisation d'un ensemble de données d'un concours Kaggle est que nous avons pu soumettre des soumissions tardives au concours et voir à quel point nos modèles se comparent aux soumissions les plus performantes du concours. Les modèles du concours Kaggle sont notés en fonction de l'erreur (MCRMSE) de leurs prédictions.

De plus, seuls les éléments suivants : réactivité, deg_Mg_pH10 et deg_Mg_50C valeurs prédites sont réellement notés pour les classements de la compétition.
La table 5.8 afficher les résulta de nos modèles :

modèl	Public score	Private score
Lstm	0.30054	0.40900
RNN	0.41405	0.49526
Bert	0.36695	0.49751

Comme nous pouvons le voir ici, le modèle de base LSTM a donné les meilleurs résultats parmi les trois modèles que nous avons proposé. Le meilleur score public était de 0.30054 et le meilleur score privé était de 0.40900.
malgré le model bert donne les mauvais résultat sur la phase de trainig mais sur la phase de test, il donne un bon résulta que le modèle rnn sur public était et un score proche sur privé était.

5.9 Conclusion

Dans ce chapitre, nous avons vu notre part de contribution au problème de mRNA stabilité, et les expérimentations que nous avons faites avec les différents paramètres nous permettent d'améliorer la performance de nos modèles en matière de précision et taux d'erreur.

CHAPITRE 6

CONCLUSION GÉNÉRALE

6.1 Conclusion

Bien que la création de molécules d'ARNm stables reste difficile, les ensembles de données de séquences et les informations correspondantes deviennent de plus en plus populaires et largement disponibles. Grâce à l'utilisation d'architectures d'apprentissage en profondeur, des prédictions raisonnables des caractéristiques structurelles peuvent être obtenues, comme le démontre ce travail. Cette technologie a le potentiel d'augmenter la vitesse et l'efficacité de la découverte de vaccins à ARNm et a d'autres implications dans d'autres domaines de recherche connexes.

Cependant, les méthodes décrites dans ce travail ne sont pas sans limitations. Il convient de noter que même si un résultat raisonnablement précis est produit, l'erreur ne peut être ignorée compte tenu de l'échelle de la valeur prédite (1-4) ; en fait, cela équivaut 30% sur mean squared error. Par conséquent, il peut prédire de manière incorrecte la stabilité de molécules importantes qui peuvent être négligées pendant la phase de découverte. Malgré ces limites, les résultats de ce travail montrent que cet algorithme de prédiction est réalisable et a le potentiel de gagner du temps dans le processus de recherche, ce qui est une denrée particulièrement précieuse lors des épidémies. À long terme, ces technologies peuvent également aider à mieux comprendre les raisons de la stabilité de certaines molécules d'ARN et contribuer au développement de technologies associées. On espère que ce travail aidera également d'autres scientifiques des données à créer de meilleurs modèles prédictifs pour ce domaine.

6.2 Future works :

— Appliquer notre travail sur des données de l'Algérie

— Assurer un cryptage des ADN et ARN afin de renforcer la vie privée des personnes vacciner
— Création de virus pour avoir un futur et un pouvoir dans la guerre biologique
— Analysée la composition du virus grâce à l'intelligence artificielle pour prédire les prochains version du virus

BIBLIOGRAPHIE

[1] Haohan Wang and Bhiksha Raj.
On the origin of deep learning.
arXiv preprint arXiv :1702.07800, 2017.

[2] organisation mondiale de la santé.
`https://www.who.int/fr/home`.

[3] Mohammed Zakaria Mokri.
Classification des images avec les réseaux de neurones convolutionnels.
PhD thesis, 09-01-2018, 2017.

[4] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton.
Deep learning.
nature, 521(7553) :436–444, 2015.

[5] `http://www.asimovinstitute.org/neural-network-zoo/18Mai2019`.

[6] Sergey Demyanov.
Regularization methods for neural networks and related models.
PhD thesis, 2015.

[7] `https://penseeartificielle.fr/comprendre-lstm-gru-fonctionnement-schema/#Le_probleme_des_reseaux_de_neurones_recurrents_RNN`.

[8] Zhiyong Cui, Ruimin Ke, Ziyuan Pu, and Yinhai Wang.
Deep bidirectional and unidirectional lstm recurrent neural network for network-wide traffic speed prediction.
arXiv preprint arXiv :1801.02143, 2018.

[9] Ivan Vasilev, Daniel Slater, Gianmario Spacagna, Peter Roelants, and Valentino Zocca.
Python Deep Learning : Exploring deep learning techniques and neural network architectures with PyTorch, Keras, and TensorFlow.
Packt Publishing Ltd, 2019.

[10] Fernando P Polack, Stephen J Thomas, Nicholas Kitchin, Judith Absalon, Alejandra Gurtman, Stephen Lockhart, John L Perez, Gonzalo Pérez Marc, Edson D Moreira, Cristiano Zerbini, et al.
Safety and efficacy of the bnt162b2 mrna covid-19 vaccine.
New England Journal of Medicine, 383(27) :2603–2615, 2020.

[11] YH Jin, L Cai, ZS Cheng, H Cheng, T Deng, YP Fan, C Fang, D Huang, LQ Huang, Q Huang, et al.
Management ftzhowunc, research team e-bmcocie, promotive association for m, health c. a rapid advice guideline for the diagnosis and treatment of 2019 novel coronavirus (2019-ncov) infected pneumonia (standard version).
Military Medical Research, 7, 2020.

[12] Abid Haleem, Mohd Javaid, and Raju Vaishya.
Effects of covid 19 pandemic in daily life.
Current medicine research and practice, 2020.

[13] Francois Chollet et al.
Deep learning with Python, volume 361.
Manning New York, 2018.

[14] Giancarlo Zaccone, Md Rezaul Karim, and Ahmed Menshawy.
Deep learning with TensorFlow.
Packt Publishing Ltd, 2017.

[15] Ala Eddine DJOKHRAB.
Planification et Optimisation de Trajectoire d'un Robot Manipulateur à 6 DDL par des Techniques Neuro-Floues.
PhD thesis, Université Mohamed Khider-Biskra, 2015.

[16] Marc Parizeau.
Réseaux de neurones.
GIF-21140 et GIF-64326, 124, 2004.

[17] Nikhil Buduma and Nicholas Locascio.
Fundamentals of deep learning : Designing next-generation machine intelligence algorithms.
" O'Reilly Media, Inc.", 2017.

[18] Dan Claudiu Ciresan, Ueli Meier, Jonathan Masci, Luca Maria Gambardella, and Jürgen Schmidhuber.
Flexible, high performance convolutional neural networks for image classification.
In *Twenty-second international joint conference on artificial intelligence*, 2011.

[19] John Duchi, Elad Hazan, and Yoram Singer.
Adaptive subgradient methods for online learning and stochastic optimization.
Journal of machine learning research, 12(7), 2011.

[20] Jeffrey Dean, Greg S Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V Le, Mark Z Mao, Marc'Aurelio Ranzato, Andrew Senior, Paul Tucker, et al.
Large scale distributed deep networks.
2012.

[21] Liat Clark.
Google's artificial brain learns to find cat videos.
Wired UK, www. wired. com, 2012.

[22] Jeffrey Pennington, Richard Socher, and Christopher D Manning.
Glove : Global vectors for word representation.
In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[23] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky.
Neural networks for machine learning lecture 6a overview of mini-batch gradient descent.
Cited on, 14(8), 2012.

[24] Diederik P Kingma and Jimmy Ba.
Adam : A method for stochastic optimization.
arXiv preprint arXiv :1412.6980, 2014.

[25] Henry J Kelley.
Gradient theory of optimal flight paths.
Ars Journal, 30(10) :947–954, 1960.

[26] Arthur E Bryson.
A gradient method for optimizing multi-stage allocation processes.
In *Proc. Harvard Univ. Symposium on digital computers and their applications*, volume 72, page 22, 1961.

[27] Paul Werbos.
Beyond regression :" new tools for prediction and analysis in the behavioral sciences.
Ph. D. dissertation, Harvard University, 1974.

[28] Edward H Shortliffe, Edward H Shortliffe, James J Cimino, and James J Cimino.
Biomedical informatics : computer applications in health care and biomedicine.
Springer, 2014.

[29] Jean-Michel Claverie, Stéphane Audic, and Chantal Abergel.
La bioinformatique : une discipline stratégique pour l'analyse et la valorisation des génomes,.
Disponible à l'adresse, 236 :4, 2000.

[30] Miguel A Andrade and Chris Sander.
Bioinformatics : from genome data to biological knowledge.
Current Opinion in Biotechnology, 8(6) :675–683, 1997.

[31] David W Mount and David W Mount.
Bioinformatics : sequence and genome analysis, volume 1.
Cold spring harbor laboratory press Cold Spring Harbor, NY, 2001.

[32] Michael C Mozer.
A focused backpropagation algorithm for temporal.
Backpropagation : Theory, architectures, and applications, 137, 1995.

[33] Paul J Werbos.
Generalization of backpropagation with application to a recurrent gas market model.
Neural networks, 1(4) :339–356, 1988.

[34] Sepp Hochreiter and Jürgen Schmidhuber.
Long short-term memory.
Neural computation, 9(8) :1735–1780, 1997.

[35] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur.
Recurrent neural network based language model.
In *Eleventh annual conference of the international speech communication association*, 2010.

[36] Tomáš Mikolov, Stefan Kombrink, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur.
Extensions of recurrent neural network language model.
In *2011 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 5528–5531. IEEE, 2011.

[37] Ilya Sutskever, James Martens, and Geoffrey E Hinton.
Generating text with recurrent neural networks.
In *ICML*, 2011.

[38] Shujie Liu, Nan Yang, Mu Li, and Ming Zhou.
A recursive recurrent neural network for statistical machine translation.
2014.

[39] Ilya Sutskever, Oriol Vinyals, and Quoc V Le.
Sequence to sequence learning with neural networks.
arXiv preprint arXiv :1409.3215, 2014.

[40] Michael Auli, Michel Galley, Chris Quirk, and Geoffrey Zweig.
Joint language and translation modeling with recurrent neural networks.
2013.

[41] Alex Graves and Navdeep Jaitly.
Towards end-to-end speech recognition with recurrent neural networks.
In *International conference on machine learning*, pages 1764–1772. PMLR, 2014.

[42] Andrej Karpathy and Li Fei-Fei.
Deep visual-semantic alignments for generating image descriptions.
In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3128–3137, 2015.

[43] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan.
Show and tell : A neural image caption generator.
In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3156–3164, 2015.

[44] Jeffrey Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell.
Long-term recurrent convolutional networks for visual recognition and description.
In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2625–2634, 2015.

[45] Bharath Ramsundar and Reza Bosagh Zadeh.
TensorFlow for deep learning : from linear regression to reinforcement learning.
" O'Reilly Media, Inc.", 2018.

[46] Sepp Hochreiter.
The vanishing gradient problem during learning recurrent neural nets and problem solutions.
International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, 6(02) :107–116, 1998.

[47] `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`.

[48] Mike Schuster and Kuldip K Paliwal.
Bidirectional recurrent neural networks.
IEEE transactions on Signal Processing, 45(11) :2673–2681, 1997.

[49] Alex Graves and Jürgen Schmidhuber.
Framewise phoneme classification with bidirectional lstm and other neural network architectures.

Neural networks, 18(5-6) :602–610, 2005.

[50] Alex Graves, Navdeep Jaitly, and Abdel-rahman Mohamed.
Hybrid speech recognition with deep bidirectional lstm.
In *2013 IEEE workshop on automatic speech recognition and understanding*, pages 273–278. IEEE, 2013.

[51] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova.
Bert : Pre-training of deep bidirectional transformers for language understanding.
arXiv preprint arXiv :1810.04805, 2018.

[52] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun.
Deep residual learning for image recognition.
In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[53] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton.
Layer normalization.
arXiv preprint arXiv :1607.06450, 2016.

[54] `https://www.kaggle.com/c/stanford-covid-vaccine/overview`.