

ECOLE POLYTECHNIQUE DE L'UNIVERSITÉ FRANÇOIS RABELAIS DE TOURS

Département Informatique

64 avenue Jean Portalis

37200 Tours, France

Tél. +33 (0)2 47 36 14 14

polytech.univ-tours.fr

Projet Recherche & Développement

2017-2018

Mémorisation Intelligente: optimiser par apprentissage automatique

**POLYTECH[®]**
TOURS

Tuteurs académiques

Lei SHANG

Vincent T'KINDT

Maxime MARTINEAU

Étudiant

Zhicong LIU (DI5)



Liste des intervenants

Nom	Email	Qualité
Zhicong LIU	zhicong.liu@etu.univ-tours.fr	Étudiant DI5
Lei SHANG	lei.shang@univ-tours.fr	Tuteur académique, Département Informatique
Vincent T'KINDT	vincent.tkindt@univ-tours.fr	Tuteur académique, Département Informatique
Maxime MARTINEAU	maxime.martineau@univ-tours.fr	Tuteur académique, Département Informatique



Avertissement

Ce document a été rédigé par Zhicong LIU susnommé l'auteur.

L'Ecole Polytechnique de l'Université François Rabelais de Tours est représentée par Lei SHANG, Vincent T'KINDT et Maxime MARTINEAU susnommés les tuteurs académiques.

Par l'utilisation de ce modèle de document, l'ensemble des intervenants du projet acceptent les conditions définies ci-après.

L'auteur reconnaît assumer l'entière responsabilité du contenu du document ainsi que toutes suites judiciaires qui pourraient en découler du fait du non respect des lois ou des droits d'auteur.

L'auteur atteste que les propos du document sont sincères et assument l'entière responsabilité de la véracité des propos.

L'auteur atteste ne pas s'approprier le travail d'autrui et que le document ne contient aucun plagiat.

L'auteur atteste que le document ne contient aucun propos diffamatoire ou condamnable devant la loi.

L'auteur reconnaît qu'il ne peut diffuser ce document en partie ou en intégralité sous quelque forme que ce soit sans l'accord préalable des tuteurs académiques et de l'entreprise.

L'auteur autorise l'école polytechnique de l'université François Rabelais de Tours à diffuser tout ou partie de ce document, sous quelque forme que ce soit, y compris après transformation en citant la source. Cette diffusion devra se faire gracieusement et être accompagnée du présent avertissement.



Pour citer ce document

Zhicong LIU, *Mémorisation Intelligente:: optimiser par apprentissage automatique*, Projet Recherche & Développement, Ecole Polytechnique de l'Université François Rabelais de Tours, Tours, France, 2017-2018.

```
@mastersthesis{
  author={LIU, Zhicong},
  title={Mémorisation Intelligente:: optimiser par apprentissage automatique},
  type={Projet Recherche \& Développement},
  school={Ecole Polytechnique de l'Université François Rabelais de Tours},
  address={Tours, France},
  year={2017-2018}
}
```

Table des matières

Liste des intervenants	a
Avertissement	b
Pour citer ce document	c
Table des matières	i
Table des figures	v
Liste des tableaux	vii
1 Introduction	1
1 Les acteurs du projet	1
2 Contexte de la réalisation.....	2
2.1 Contexte	2
2.1.1 <i>Branch&Bound</i>	3
2.1.2 Mémorisation.....	3
2.1.3 Stratégies de nettoyage	4
2.2 Description du problème.....	4
2.3 Existant	5
2.4 Objectif et besoins.....	5
3 Hypothèses	5
4 Bases méthodologiques	5
2 Description générale	6
1 Environnement du projet	6
2 Caractéristiques des utilisateurs	6

2.1	MOA	6
2.2	MOE	7
3	Fonctionnalités du système	7
3	Etat de l'art	8
1	Problématiques : existant	8
1.1	<i>BB2001</i>	8
1.2	Mémorisation	9
1.3	Stratégies de nettoyage	9
2	Méthodes de Machine Learning	9
2.1	Classification naïve bayésienne	9
2.2	Méthode des k plus proches voisins (<i>KNN</i>)	10
2.3	Machine à vecteurs de support (<i>SVM</i>)	11
2.4	Arbre de décision	12
2.5	Forêt d'arbres décisionnels (<i>RF</i>)	12
2.6	Perceptron	13
2.7	Rétro-propagation du gradient (<i>Back Propagation</i>)	14
3	Classification de séquences	14
4	Réseau de neurones récurrents	14
5	<i>Long Short-Term Memory</i>	15
4	Analyse et conception	16
1	Analyse du problème	16
2	Analyse de données	16
3	Analyse d'entrée/sortie	16
4	Analyse de méthode	18
5	Conception	19
5	Mise en œuvre	21
1	Outils et bibliothèques utilisées	21
2	Implémentation	22
2.1	Composition du projet	22
2.2	Diagramme de classe	22
2.3	Choix techniques	23
2.3.1	Paramètres	23
2.3.2	Définition d'entrées	23
2.3.3	Définition d'étiquettes (<i>labels</i>)	23
2.3.4	Définition de sorties	24
2.3.5	Méthode pour construire le réseau	24
2.3.6	Définition de <i>loss</i>	24

2.3.7	Définition de précision	25
2.3.8	Définition d'optimiseur	25
2.4	Analyse.....	25
2.4.1	Équilibrage de données	25
2.4.2	Mélange de données	25
2.4.3	Réorganisation de données.....	26
2.4.4	Ajustement de paramètres	26
3	Évaluation de performance	26
3.1	Stratégie d'évaluation	26
3.2	Analyse des résultats	26
6	Bilan et conclusion	37
1	Fait	37
2	Perspective	37
3	Bilan sur la qualité	38
	Annexes	39
A	Spécifications fonctionnelles	40
B	Spécifications non fonctionnelles	41
1	Contraintes de développement	41
2	Contraintes de fonctionnement.....	41
C	Cahier de développeur	42
1	Installation et configuration.....	42
2	Lancement du projet	43
D	Plan de développement et organisation	45
1	Tâche 1	45
2	Tâche 2	45
3	Tâche 3	45
4	Tâche 4.....	45
5	Tâche 5	46
6	Diagramme de Gantt	46
E	Dossiers de tests	48
1	Tests unitaires	48
1.1	<i>Test runner</i>	48
1.2	<i>Test case</i>	48
2	Tests fonctionnels	48

2.1	Test d'entraînement	49
2.2	Test de graphe de <i>Tensorflow</i>	50
Webographie		53
Bibliographie		54

Table des figures

1 Introduction

1	Logo de l'École Polytechnique de l'Université de Tours.....	1
2	Logo de l'équipe de recherche ROOT ERL-CNRS 6305	2
3	Logo du Laboratoire d'Informatique de Tours	2
4	Exemple de SMTT	3
5	Exemple d'arbre d'un ensemble de solutions	3
6	Une partie d'arbre d'une instance de 100 tâches.....	4
7	Diagramme des cas d'utilisation	5

2 Description générale

1	Logo du Python	6
2	Logo du Pycharm.....	6

3 Etat de l'art

1	Conception de SVM.....	11
2	Fonction noyau : mappage	12
3	Modèle de perceptron	13
4	Schéma d'un réseau de neurones récurrents	14
5	Schéma du LSTM.....	15
6	Schéma d'une cellule du LSTM.....	15

4 Analyse et conception

1	Données d'une instance.....	17
2	Données de solutions.....	18

3	Tableau de synthèse	19
5 Mise en œuvre		
1	Logo du Tensorflow	21
2	Logo du Numpy	21
3	Composition du projet	22
4	Diagramme de classes	22
5	Données d'entrée	24
6	Pas de mélange et équilibrage de données.....	28
7	<i>Learning rate</i> = 10^{-7}	29
8	<i>Learning rate</i> = 10^{-6}	30
9	<i>Learning rate</i> = 10^{-5}	31
10	<i>Learning rate</i> = 10^{-4}	32
11	<i>Batch size</i> =128.....	33
12	<i>Batch size</i> =256.....	34
13	<i>Batch size</i> =512.....	35
14	Le meilleur modèle rencontré	36
C Cahier de développeur		
1	Installation de tensorflow	42
2	Lancer Git	43
3	Obtenir code de Github.....	43
4	Lancer Tensorboard	43
5	Variables avec Tensorboard	44
6	Exemple de résultat	44
7	Fichier d'évènement	44
8	Fichiers de variables	44
D Plan de développement et organisation		
1	Diagramme de Gantt	47
E Dossiers de tests		
1	Tableau de tests unitaires	49
2	Tableau de tests unitaires	49
3	Résultat de tests unitaires	49
4	Affiche du résultat.....	50
5	Affiche du résultat.....	51
6	Graphe de flux de données.....	51
7	Tableau de tendance du loss.....	52
8	Tableau de tendance de précision d'entraînement	52



Liste des tableaux

5 Mise en œuvre

1	Tableau de synthèse	26
2	Tableau de stratégie d'évaluation.....	27
3	Tableau de résultat de modèle 1.....	27
4	Tableau de résultat de modèle 1.....	27
5	Tableau de résultat de modèle 3.....	27
6	Tableau de résultat de modèle 4.....	28
7	Tableau de résultat de modèle 5.....	28
8	Tableau de résultat de modèle 5.....	28

1

Introduction

Le Projet de Recherche & Développement (PR&D), réalisé en 5ème année, s'inscrit dans la formation dispensée à l'École Polytechnique de l'Université de Tours et constitue une réelle expérience en termes de conduite de projet. Le PR&D se déroule du 21 septembre 2017 au 29 mars 2018 à raison de 2 jours par semaine à temps plein. Il donne lieu à la rédaction d'un rapport avec le cahier de spécification et de deux présentations du travail effectué lors de deux soutenances.

Ce PR&D est orienté vers une étude théorique et aussi un développement algorithmique d'une méthode de résolution appelée mémorisation intelligente où la MOA était représentée par M. Lei SHANG, M. Vincent T'KINDT et M. Maxime MARTINEAU, membres du Laboratoire d'Informatique de Tours. Le projet s'inscrit dans un cadre théorique qui fait l'objet d'une étude de *Machine Learning* en amont, pour aboutir à la solution du problème.

Ce document est donc le rapport du projet "Mémorisation Intelligente". Il définit les besoins, l'environnement du projet et les objectifs à réaliser. Ce rapport présente aussi l'analyse et la conception du problème, l'état de l'art, différentes tâches à effectuer et le planning prévisionnel.

1 Les acteurs du projet

– la maîtrise d'œuvre (MOE) : Zhicong LIU, étudiante en 5ième année de l'Ecole Polytechnique de l'Université de Tours au sein du département Informatique, dans le cadre du PR&D ainsi que ses encadrants Lei SHANG, Vincent T'KINDT et Maxime MARTINEAU de l'équipe de recherche ROOT ERL-CNRS 6305 du Laboratoire Informatique de Tours. Le logo est comme [Figure 1](#), [Figure 2](#) et [Figure 3](#)



Figure 1 – Logo de l'École Polytechnique de l'Université de Tours

Les recherches menées au sein de cette équipe portent sur l'étude et la résolution des problèmes d'ordonnancement et de planification, que ce soit dans le cadre de problématiques "industrielles" ou de problèmes théoriques. Il s'agit donc de problèmes d'optimisation, bien souvent



Figure 2 – Logo de l'équipe de recherche ROOT ERL-CNRS 6305



Figure 3 – Logo du Laboratoire d'Informatique de Tours

combinatoires, pour lesquels des techniques de Recherche Opérationnelle et d'Aide à la Décision sont mises en œuvre. (li.univ-tours.fr)

– la maîtrise d'ouvrage (MOA) : Laboratoire d'Informatique de Tours, M. Lei SHANG, M. Vincent T'KINDT et M. Maxime MARTINEAU.

Ce document a été rédigé par Zhicong LIU et sera validé par les différents acteurs du projet.

2 Contexte de la réalisation

2.1 Contexte

Un problème d'ordonnancement classique consiste à organiser un nombre de tâches (*jobs*), compte tenu de contraintes temporelles et de ressources dans le but d'optimiser une ou plusieurs fonctions objectives.

Exemple du problème d'ordonnancement : n tâches à ordonnancer sur 3 machines, le critère à minimiser est le maximum des dates de fin d'exécution des tâches : $3 \parallel C_{max}$

Nous nous intéressons dans ce projet à une classe particulière de problème d'ordonnancement classique. Ce problème est appelé dans la littérature, *Single Machine Total Tardiness Problem (SMTT)*. Il s'avère être *NP-Difficile*. Dans sa formulation, une seule machine est considérée, tel que cette machine possède un ensemble de tâches et une fonction objective, chaque tâche a une durée de traitement (*processing time*) et une date due (*due date*). Le but est d'organiser les tâches en une séquence pour minimiser le retard total $T(N, S) = \sum_{j=1}^n \max \left\{ \sum_{i=1}^j p_{a_i} - d_{a_j}, 0 \right\}$. On note ce problème par $1 \parallel \sum T_j$ tel que :

- $N = \{1, 2, \dots, n\}$: un ensemble de tâches
- p_j : *processing time* du tâche j
- d_j : *due date* du tâche j
- $S = \{a_1, \dots, a_n\}$: une séquence de tâches

Par exemple dans la Figure 4 : S'il y a 3 tâches 1, 2, 3 sur une seule machine, les dates dues sont respectivement d_1, d_2, d_3 . Pour la tâche 1, le retard est 1. Pour la tâche 2, le retard est 0. Pour la tâche 3, le retard est 1. Donc la somme des retards est 2.

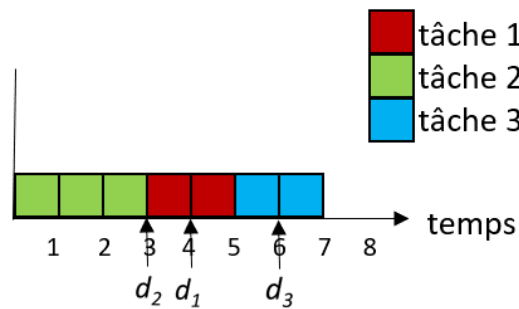


Figure 4 – Exemple de SMTT

2.1.1.1 Branch&Bound

La méthode de l'état de l'art est un algorithme de *Branch&Bound* qui permet de résoudre des problèmes avec jusqu'à 500 jobs.

« L'idée de *Branch&Bound* consiste à diviser le problème en un certain nombre de sous-problèmes qui ont chacun leur ensemble de solutions réalisables. Ainsi, en résolvant tous les sous-problèmes et en prenant la meilleure solution trouvée, on est assuré d'avoir résolu le problème initial. » [WWW3]

Les ensembles de solutions (et leurs sous-problèmes associés) ainsi construits ont une hiérarchie naturelle en arbre, par exemple dans la Figure 5, souvent appelée arbre de recherche. Cet arbre représente une instance. Chaque nœud dans l'arbre représente une solution d'un sous-problème. Après, on évalue un nœud de l'arbre de recherche pour déterminer l'optimum de l'ensemble des solutions réalisables associé au nœud en question. La méthode la plus générale consiste à déterminer un minimum du coût des solutions. Si on arrive à trouver un minimum qui est supérieur au coût de la meilleure solution trouvée jusqu'à présent, on a alors l'assurance que le sous-ensemble ne contient pas l'optimum et donc il peut être coupé. Ainsi, l'efficacité peut être améliorée.

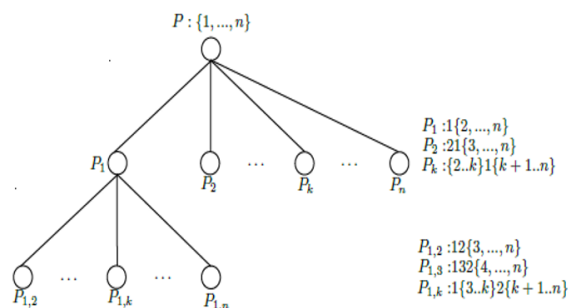


Figure 5 – Exemple d'arbre d'un ensemble de solutions

2.1.1.2 Mémorisation

BB2001 [3] est un algorithme *Branch&Bound* qui utilise un catalyseur : Mémorisation. L'idée est de mémoriser les solutions des sous-problèmes résolus dans la mémoire afin que lorsque ce

sous-problème réapparaît, sa solution soit récupérée directement à partir de la mémoire au lieu de la résoudre plusieurs fois.

Cependant, plus de mémoire est nécessaire qu'avant et la mémoire est très vite saturée car l'arbre (représente des ensembles de sous-problèmes) est très grand. Selon le résultat d'une expérimentation **Figure 6**, seulement une petite partie des solutions mémorisées est utile pour résoudre d'autres sous-problèmes. Donc, ce n'est pas nécessaire de mémoriser si beaucoup de solutions.

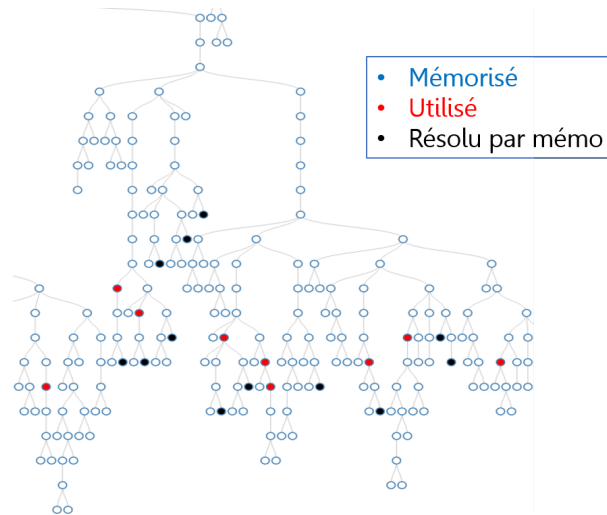


Figure 6 – Une partie d'arbre d'une instance de 100 tâches

Cette partie sera plus détaillée dans la partie d'état de l'art.

2.1.3 Stratégies de nettoyage

Pour résoudre le problème de la mémoire, trois stratégies de nettoyage de mémoire sont testées pour éliminer les solutions inutiles lorsque la mémoire est saturée :

- ✓ FIFO
- ✓ BEFO
- ✓ LUFO

Selon le résultat d'une expérimentation, *LUFO* est meilleure que les autres.

Cette partie sera plus détaillée dans la partie d'état de l'art.

2.2 Description du problème

Comme informé dans le contexte, la stratégie *LUFO* a une bonne performance. Mais elle a des inconvénients.

Par exemple, si une solution est juste stockée dans la mémoire, alors elle n'est pas encore utilisée. Avec la stratégie *LUFO*, elle va être supprimée. Mais en réalité, elle est très utile pour résoudre les sous-problèmes identiques.

Donc, le problème est de déterminer si une solution est utile pour résoudre des autres sous-problèmes en utilisant les méthodes de *Machine Learning*. Si la réponse est oui, on la stocke dans la mémoire, sinon on ne la stocke pas.

Ce problème est indépendant des problèmes de *SMTT* et il peut être considéré comme un problème de classification de séquences.

2.3 Existant

C'est un tout nouveau projet. Donc aucun existant n'est présent. Il faut commencer par zéro.

2.4 Objectif et besoins

L'objectif de ce PR&D est de construire un logiciel pour prévoir si une solution d'un sous-problème sera utilisée.

Les tâches principales consistent de trois parties :

- ✓ Étudier les méthodes de *Machine Learning* pour enfin en choisir une qui est adaptée à ce problème
- ✓ Construire un modèle d'apprentissage
- ✓ Entraîner le modèle en utilisant les données existantes
- ✓ Tester le modèle pour évaluer la performance

Avec ce logiciel, un utilisateur peut saisir les données initiales, traiter des données (équilibrer les données, récupérer les données utiles, mélanger les données) et obtenir le résultat de classification.

La **Figure 7** ci-dessous est le diagramme des cas d'utilisation.

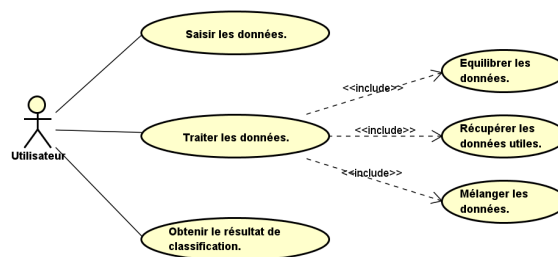


Figure 7 – Diagramme des cas d'utilisation

3 Hypothèses

On suppose que la structure combinatoire des solutions utilisées par la mémorisation peut être apprise.

4 Bases méthodologiques

Afin de conduire et gérer ce projet, l'outil *GanttProject* sera utilisé pour réaliser la planification des tâches et le suivi de celle-ci.

Le projet sera suivi grâce à *Trello* qui me permet de gérer l'avancement du projet (Documentation, Rendez-vous...).

J'ai utilisé *Github* pour la gestion de version du code.

Afin d'avoir un feedback régulier et de pouvoir fournir des livrables fréquemment, je vais mettre en place une soumise hebdomadaire comme compte-rendu.

Durant ce projet, l'ensemble de la documentation est réalisé sous Latex.

2

Description générale

1 Environnement du projet

Le projet est réalisé en langage *Python* sous *Windows*.

- Version : *Python 3.6.2* **Figure 1**.



Figure 1 – Logo du Python

- IDE : *JetBrains Pycharm* **Figure 2**



Figure 2 – Logo du Pycharm

2 Caractéristiques des utilisateurs

Au cours de ce projet, deux types d'utilisateurs sont présents, la MOA et la MOE.

2.1 MOA

La MOA (Lei SHANG) sera utilisateur qui va utiliser l'outil. Celle-ci possède de bonnes connaissances en informatique et en ordonnancement.

2.2 MOE

À la suite de la réalisation, la MOE(Zhicong LIU) ayant des connaissances en *Machine Learning* restera nécessaire. Ceci afin de déployer la solution. En effet, la MOE sera pour rôle de analyser, construire, entraîner et tester le modèle dont on a besoin.

3 Fonctionnalités du système

La fonctionnalité du système est de prévoir si une solution est utile pour déterminer si elle doit être mémorisée.

L'entrée est une séquence qui contient les données organisées d'une solution.

La sortie est un résultat de classification. Il y a deux possibilités, la solution est utile et la solution n'est pas utile.

3

Etat de l'art

Dans cette partie, j'ai d'abord étudié l'existant pour en savoir plus sur le problème. Ensuite j'ai fait une investigation des différentes méthodes de Machine Learning et de classification de séquence qui sont décrits plus en détail ci-dessous. Le but de ces investigations est de bien comprendre le problème de mémorisation intelligente et d'avoir une idée réalisable afin de répondre aux objectifs de ce PR&D à la fin de la partie de recherche.

1 Problématiques : existant

J'ai lit un papier « *Exact Solution of the Single Machine Total Tardiness Problem : the Power of Memorization* » [3] pour étudier l'existant. Cette thèse introduit la définition de *Single Machine Total Tardiness Problem (SMTT)*, la méthode existante pour résoudre ce problème et quelques expérimentations qui produisent ce projet. *Single Machine Total Tardiness Problem (SMTT)* s'avère être NP-Difficile. Dans ce problème, une seule machine est considérée, tel que cette machine possède un ensemble de tâches et une fonction objective, chaque tâche a une certaine durée d'exécution et une date d'échéance. Le but est d'organiser les tâches en une séquence pour minimiser le retard total $T(N, S) = \sum_{j=1}^n \max \left\{ \sum_{i=1}^j p_{a_i} - d_{a_j}, 0 \right\}$. On note ces problèmes par 1 || $\sum T_j$ tel que :

- $N = \{1, 2, \dots, n\}$ un ensemble de tâches
- p_j durée d'exécution du tâche j
- d_j date d'échéance du tâche j
- $S = \{a_1, \dots, a_n\}$ une séquence

1.1 BB2001

BB2001[3] est un algorithme *Branch&Bound* qui utilise un catalyseur : Mémorisation. Les procédures sont :

- ✓ Rechercher la solution dans la mémoire, retourner la solution si on la trouve, sinon exécuter l'étape 2.

- ✓ Diviser le problème et résoudre chaque sous-problème récursivement. Si on ne peut pas le diviser, exécuter l'étape 3.
(Etape 2 est appelée Split)
- ✓ Utiliser Property 2 (une règle d'élimination)[3] pour obtenir la liste des positions possibles pour la tâche la plus longue et la tâche de la plus petite échéance.
- ✓ Combiner Decomposition 1[1] et Decomposition 2 [4] pour faire brancher la tâche la plus longue et la tâche de la plus petite *due date* à chaque position donnée par l'étape 3. Pour chaque cas de ramification, résoudre les sous-problèmes de façon récursive, puis finalement stocker la meilleure solution parmi tous les cas de ramification dans la mémoire.

1.2 Mémorisation

Dans une expérimentation, deux méthodes sont utilisées pour résoudre le problème *SMTT* d'instance avec 700 tâches : *BB2001* et *NoSplit* [3]. *BB2001* est une méthode qui utilise l'algorithme *Branch&Bound* et *NoSplit* est une méthode qui supprime l'étape *Split* dans *BB2001*. Le résultat indique que *NoSplit* est plus rapide que *BB2001*. Quand l'étape *Split* est supprimée, les nœuds sont résolus avec plus de temps lors de leur première occurrence, mais ils ne sont jamais résolus deux fois grâce à la mémorisation. C'est le puissance de la mémorisation.

Cependant, plus de mémoire est nécessaire qu'avant et c'est pourquoi *NoSplit* a rencontré le problème de mémoire sur les instances avec 800 tâches. La mémoire est très vite saturée car l'arbre (représente des ensembles de solutions) est très grand. Selon le résultat d'une autre expérimentation seule une petite partie des solutions mémorisées est utile pour résoudre des autre sous-problèmes. Donc, ce n'est pas nécessaire de mémoriser toutes les solutions rencontrées.

1.3 Stratégies de nettoyage

Trois stratégies de nettoyage de mémoire sont testées pour éliminer les solutions inutiles lorsque la mémoire est saturée.

- ✓ *FIFO* : premier entré, premier sorti.
- ✓ *BEFO* : la solution avec le plus haut niveau dans l'arbre, premier sorti.
- ✓ *LUFO* : le moins utilisé, premier sorti

Selon le résultat d'une expérimentation, *LUFO* est meilleure que les autres.

2 Méthodes de Machine Learning

Machine Learning est un champ d'étude de l'intelligence artificielle, concerne la conception, l'analyse, le développement et l'implémentation de méthodes permettant à une machine d'évoluer par un processus systématique, et ainsi de remplir des tâches difficiles ou problématiques par des moyens algorithmiques plus classiques.

J'ai lit un livre de référence pour apprendre les méthodes de Machine Learning [2].

Quelques méthodes sont listées ci-dessous.

2.1 Classification naïve bayésienne

Pour l'article à classer, l'idée est de calculer la probabilité d'occurrence de chaque catégorie sous la condition d'occurrence de cet article. Qui est le maximum, et cet article appartient à quelle catégorie.

Principe :

- ✓ Supposons qu'il existe un échantillon $x = (a_1, a_2, a_3, \dots, a_n)$ à classer ($a_1, a_2, a_3, \dots, a_n$ sont les caractéristiques indépendantes).
- ✓ Supposons qu'il existe des catégories de classification $Y = \{y_1, y_2, y_3, y_4 \dots y_N\}$
- ✓ Alors $\max(P(y_1|x), P(y_2|x), P(y_3|x) \dots P(y_N|x))$ est la catégorie de classification finale
- ✓ $P(y_k|x) = P(x|y_k) * P(y_k) / P(x)$
- ✓ x est le même pour chaque catégorie, donc il faut savoir $\max P(x|y_k) * P(y_k)$
- ✓ $P(x|y_k) * P(y_k) = P(y_k) * \prod_i (P(a_i|y_k))$
- ✓ $P(a_i|y_k)$ et $P(y_k)$ sont tous disponibles à partir de l'échantillon d'entraînement
 $P(a_i|y_k)$ représente la probabilité d'occurrence d'une caractéristique dans une catégorie
 $P(y_k)$ représente la probabilité d'occurrence d'une catégorie dans toutes les catégories

Processus :

- ✓ Phase de préparation
Déterminer les attributs de caractéristiques et diviser chaque attribut de manière appropriée, puis classer manuellement certains des articles à classer pour former des échantillons d'apprentissage.
- ✓ Phase de d'entraînement
Calculer la probabilité d'occurrence de chaque catégorie dans l'échantillon d'entraînement et la probabilité conditionnelle de chaque attribut pour chaque catégorie.
- ✓ Phase d'application
Utiliser le classificateur pour la classification, l'entrée est le classificateur et l'échantillon à classer, la sortie est la catégorie de classification de l'échantillon

Avantages :

- ✓ Très bonne performance pour les données à petite échelle, adaptées aux tâches de multi-classification, adaptées à l'entraînement incrémentale.

Inconvénients :

- ✓ Sensible à l'expression des données d'entrée (discrètes, continues, extrêmement petites/-grandes). Si les caractéristiques sont dépendantes, il y a des problèmes

2.2 Méthode des k plus proches voisins (KNN)

En donnant un ensemble de données d'entraînement et une nouvelle instance, on cherche les k instances d'entraînement les plus proches de cette nouvelle instance dans l'ensemble de données d'entraînement, puis on compte le nombre de chaque catégorie à laquelle appartient les k instances. La catégorie avec le plus grand nombre est la catégorie de cette nouvelle instance.

Avantages

- ✓ Pensée simple, théorie mature, peut être utilisés pour faire la classification aussi la régression
- ✓ Peut être utilisé pour la classification non linéaire
- ✓ La complexité du temps d'entraînement est $O(n)$
- ✓ Haute précision, pas d'hypothèses sur les données, pas sensible aux valeurs aberrantes

Inconvénients

- ✓ Une grande quantité de calculs
- ✓ Déséquilibre de l'échantillon (c.-à-d. Le nombre d'échantillons dans certaines catégories est fort et le nombre d'autres échantillons est faible)
- ✓ Nécessite beaucoup de mémoire

2.3 Machine à vecteurs de support (SVM)

L'idée de SVM est de résoudre l'hyperplan qui peut correctement classer les échantillons d'entraînement et maximiser leur espacement géométrique.

Principe : [Figure 1](#) [2]

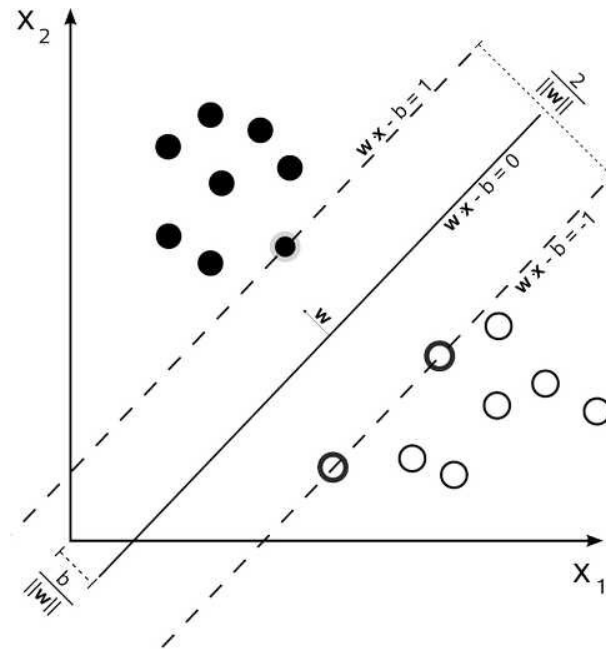


Figure 1 – Conception de SVM

- ✓ Supposons que l'on va diviser les cercles pleins et les cercles creux en deux catégories
- ✓ Il existe plusieurs lignes (hyperplan) pour accomplir cette tâche
- ✓ En SVM, nous cherchons une ligne de démarcation optimale qui rend la marge à chaque côté la plus grande
- ✓ Dans ce cas, les points en gras sont appelés vecteurs de support

Supposons que les échantillons d'entraînement sont linéairement séparables, c'est-à-dire qu'il y a un hyperplan qui peut diviser correctement les échantillons. Cependant, dans les tâches pratiques, peut-être qu'il n'y a pas d'hyperplan qui peut diviser correctement les échantillons. A ce stade, on introduit la fonction noyau pour mapper les échantillons de l'espace d'origine vers un espace de caractéristiques avec une plus élevée dimension comme [Figure 2](#) [2]. Dans le nouvel espace de caractéristiques, les échantillons peuvent être linéairement séparables.

Cependant, dans les tâches pratiques, peut-être qu'il n'y a pas d'hyperplan qui peut diviser correctement les échantillons. Donc, la fonction du noyau est introduite pour mapper les échantillons de l'espace d'origine vers un espace de caractéristiques avec une plus élevée dimension. Dans le nouvel espace de caractéristiques, les échantillons peuvent être linéairement séparables.

Avantages

- ✓ Utiliser la fonction noyau pour mapper l'espace de grande dimension
- ✓ Utiliser la fonction noyau pour résoudre la classification non linéaire
- ✓ L'idée de la classification est très simple, c'est-à-dire maximiser la distance entre l'échantillon et la zone de décision
- ✓ La classification a un bon résultat

Inconvénients

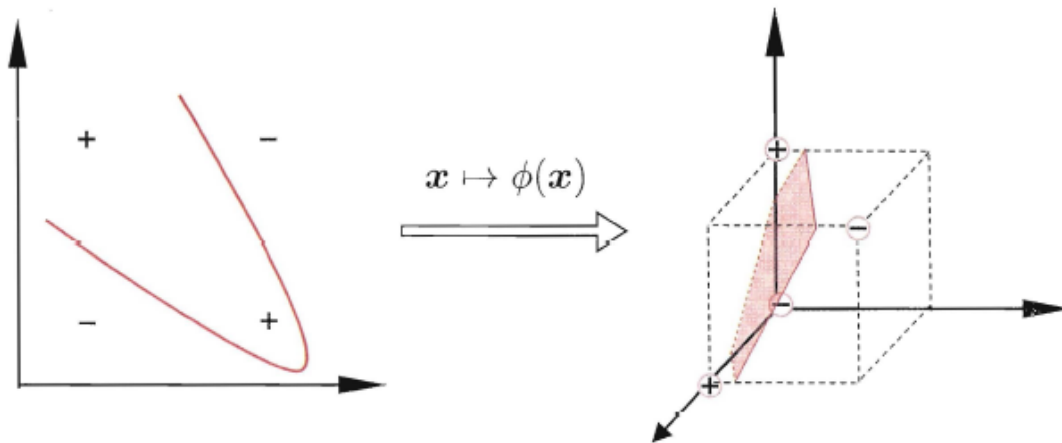


Figure 2 – Fonction noyau : mappage

- ✓ Il est difficile d'entraîner un gros volume de données
- ✓ Ne peut pas directement prendre en charge la multi-classification, mais on peut utiliser des méthodes indirectes pour la faire

2.4 Arbre de décision

Arbre de classification :

- ✓ D'abord, calculer les coefficients de *Gini* des caractéristiques actuelles
- ✓ Sélectionner la caractéristique avec le plus petit coefficient de *Gini* comme caractéristique de division
- ✓ Trouver la catégorie de classification avec le plus petit coefficient de *Gini* comme point de division optimale
- ✓ Diviser l'échantillon actuel en deux catégories, l'un : la catégorie de caractéristique est égale au point de division optimale, l'autre n'est pas égal
- ✓ Faire récursivement la division ci-dessus jusqu'à toutes les feuilles sont vers la même catégorie ou le nombre de feuilles inférieur à un certain seuil

La sortie est la catégorie la plus à laquelle les nœuds feuille appartiennent.

Arbre de régression :

- ✓ Sortir la moyenne de chaque valeur des échantillons dans les nœuds feuille.

Avantages

- ✓ Le calcul est simple, peut être interprété, plus approprié pour traiter l'échantillon qui manque d'attribut, traiter des caractéristiques non pertinentes

Inconvénients

- ✓ La classification par un seul arbre de décision est faible et il est difficile à traiter des variables de valeur continue
- ✓ Il est facile à sur-ajuster

2.5 Forêt d'arbres décisionnels (RF)

La forêt d'arbres décisionnels est composée de nombreux arbres de décision, et il n'y a pas de relation entre eux. Quand on fait la prédiction, il faut déterminer chaque arbre de décision, et enfin utiliser des idées de *Bagging* pour obtenir les résultats (c'est-à-dire, l'idée de voter)

Processus d'apprentissage :

- ✓ Il y a maintenant N échantillons d'entraînement, chaque échantillon a M caractéristiques, il faut construire K arbres
- ✓ Retirer N échantillons avec remplacement à partir de N échantillons d'entraînement comme des ensembles d'entraînement (les échantillons n'ont pas été pris comme échantillons de prédiction, et évaluer leurs erreurs)
- ✓ Retirer m caractéristiques à partir des M caractéristiques ($m \ll M$)
- ✓ Utiliser les échantillons pour créer des arbres de décision dont chaque nœud ne peut pas être divisé ou tous les échantillons pointent vers la même catégorie
- ✓ Répéter K fois étape 2 pour construire la forêt

Processus de prévision :

- ✓ Entrer les échantillons de prédiction dans K arbres pour prévoir séparément
- ✓ S'il s'agit d'un problème de classification, utiliser directement la méthode de vote pour choisir la catégorie de fréquence la plus élevée
- ✓ S'il s'agit d'un problème de régression, utiliser la valeur moyenne de la classification comme résultat

2.6 Perceptron

« Le perceptron peut être vu comme le type de réseau de neurones le plus simple. C'est un classifieur linéaire. Dans sa version simplifiée, le perceptron est mono-couche et n'a qu'une seule sortie à laquelle toutes les entrées sont connectées et les entrées et la sortie sont booléennes. Plus généralement, les entrées peuvent être des nombres réels. » [WWW1] Le modèle de perceptron est comme **Figure 3**

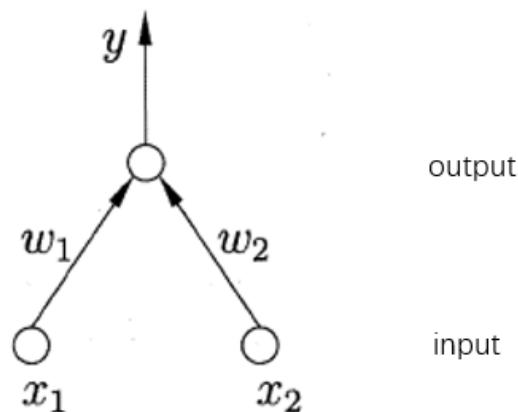


Figure 3 – Modèle de perceptron

Étant donné un ensemble de données d'entraînement, les poids et le seuil peuvent être obtenus par l'apprentissage.

Les règles d'apprentissage sont très simples, pour un échantillon (x, y) , si la sortie de la perceptron est \hat{y} , alors on ajuste les poids :

$$w_i \leftarrow w_i + \Delta w_i$$

$$\Delta w_i = \eta (y - \hat{y})$$

Si la prédiction est correcte, soit $y = \hat{y}$, alors le perceptron ne change pas, sinon les poids seront ajustés.

2.7 Rétro-propagation du gradient (*Back Propagation*)

Pour chaque exemple d'entraînement, l'algorithme *Back Propagation* effectue les opérations suivantes :

- ✓ D'abord, des exemples d'entraînement sont fournis à la couche d'entrée, puis le signal est transmis couche par couche jusqu'à ce que la couche de sortie se produise
- ✓ Et puis on calcule l'erreur de couche de sortie et l'erreur est propagée vers l'arrière aux neurones de la couche cachée
- ✓ Enfin selon l'erreur de couche cachée des neurones on ajuste les valeurs de poids et de seuil
- ✓ Le processus est cyclé jusqu'à ce qu'une condition d'arrêt soit atteinte, par exemple, l'erreur d'entraînement a atteint une petite valeur

3 Classification de séquences

Les séquences n'ont pas de caractéristiques explicites. Même avec des techniques sophistiquées de sélection de caractéristiques, la dimensionnalité des caractéristiques potentielles peut être très élevée et la nature séquentielle des caractéristiques est difficile à capturer. Cela rend la classification des séquences difficile. Et La plupart des classificateurs, tels que l'arbre de décision et le réseau de neurones, ne peuvent prendre des données d'entrée que comme vecteur de caractéristiques.

Une façon de résoudre le problème de la classification des séquences consiste à transformer une séquence en un vecteur de caractéristiques grâce à des sélections de caractéristiques.

Pour garder l'ordre des éléments dans une séquence, une courte segmentation de séquence de k symboles consécutifs, appelé un *k-gram*, est généralement sélectionné comme une caractéristique. Étant donné un ensemble de *k-gram*, une séquence peut être représentée comme un vecteur de la présence et de l'absence des *k-gram* ou comme un vecteur des fréquences des *k-gram*. [5]

4 Réseau de neurones récurrents

"Un réseau de neurones récurrents est un réseau de neurones artificiels présentant des connexions récurrentes. Un réseau de neurones récurrents est constitué d'unités(neurones) interconnectés. Les unités sont reliées par des arcs qui possèdent un poids. La sortie d'un neurone est une combinaison non linéaire de ses entrées." [WWW2] Dans la Figure 4, c'est un schéma d'un réseau de neurones récurrents à une unité reliant l'entrée et la sortie du réseau. A droite est la version dépliée de la structure.

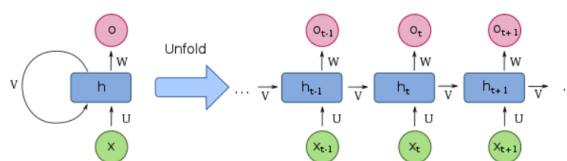


Figure 4 – Schéma d'un réseau de neurones récurrents

5 Long Short-Term Memory

Un réseau *Long Short-Term Memory* (*LSTM*) est un type de réseau de neurones récurrent. En général, une unité *LSTM* est composée d'une cellule, d'une porte d'entrée (*input gate*), d'une porte de sortie (*output gate*) et d'une porte d'oubli (*forget gate*). La cellule est pour mémoriser l'état à long terme.

Le schéma du *LSTM* est dans la Figure 5.

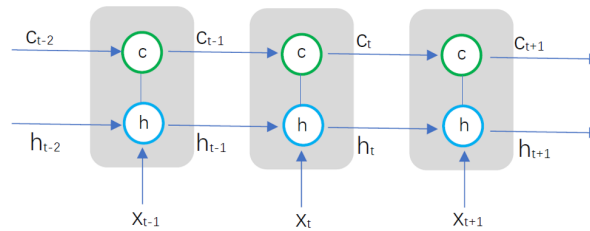


Figure 5 – Schéma du LSTM

A l'instant t , *LSTM* a trois entrées :

- X_t : le vecteur d'entrée
- h_{t-1} : le vecteur de sortie au moment antérieur
- c_{t-1} : le vecteur d'état de cellule au moment antérieur

LSTM utilise deux *gates* pour contrôler l'état de la cellule c . La *forget gate* f est utilisé pour déterminer combien de c_{t-1} sera effacé. *Input gate* i détermine combien de X_t sera entré dans la cellule. *Output gate* o sert à contrôler combien de c_t sera utilisé pour calculer la sortie. W est le poids.

Le schéma d'une cellule du *LSTM* est dans la Figure 6.

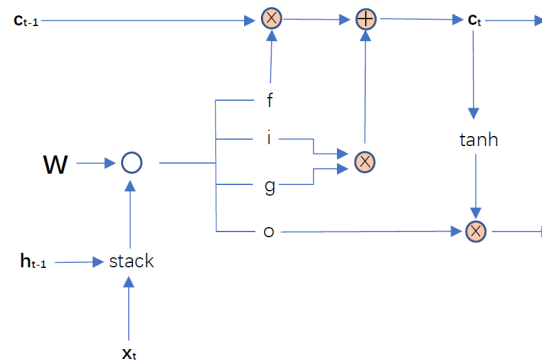


Figure 6 – Schéma d'une cellule du LSTM

4

Analyse et conception

1 Analyse du problème

Le problème de mémorisation intelligente est séparé du problème *SMTT* pour l'optimiser par une méthode de *Machine Learning*. Mais, les deux problèmes sont indépendants.

Dans ce projet, nous nous intéressons qu'en donnant une solution s'elle est utile pour résoudre des autres problèmes. Donc nous définissons ce problème comme un problème de classification de séquence.

2 Analyse de données

Les données pour l'entraînement sont stockées dans les fichiers d'un format *txt*. Elles sont récupérées de la mémoire. Il y a les données d'instance et les données de solutions. On peut dire que dans chaque instance il y a plusieurs solutions. Chaque solution est un échantillon d'entraînement.

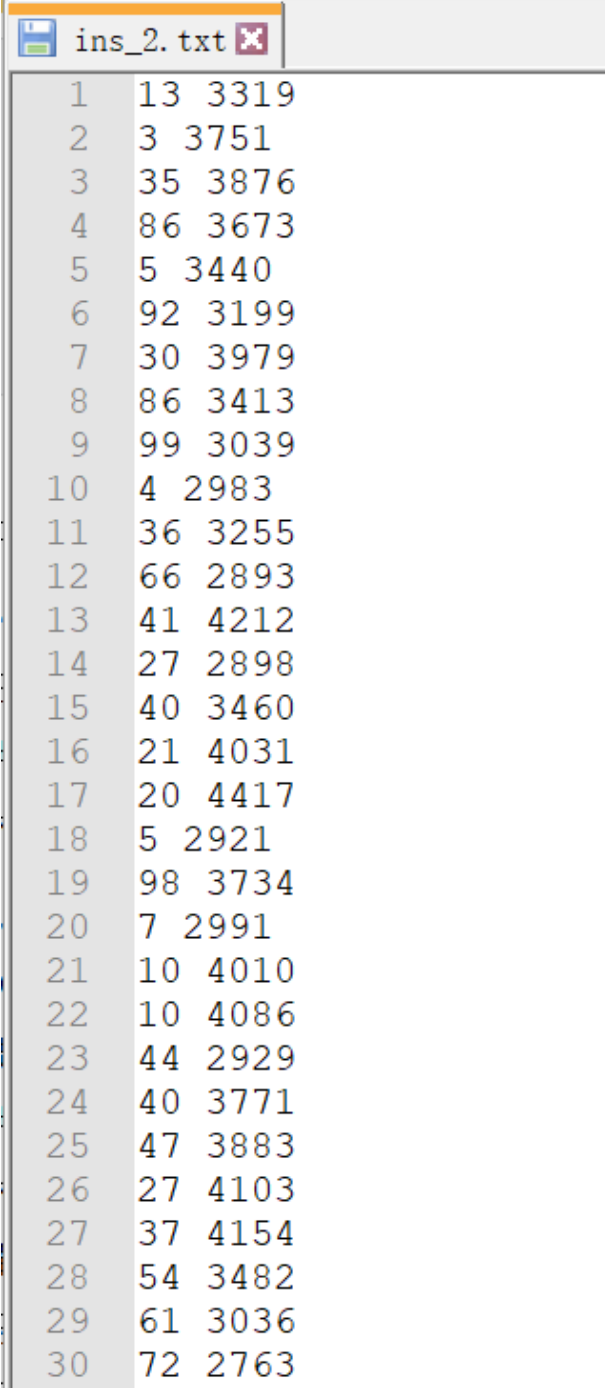
Par exemple, le fichier « *ins_2.txt* » contient les données d'une instance. Chaque ligne représente le *processing time* et le *due date* d'une tâche. Pour la première ligne dans la [Figure 1](#) ci-dessous, « 13 » est le *processing time* du *job* numéro 1 et « 3319 » est le *due date* du *job* numéro 1. Il y a 200 *jobs* dans cette instance et le numéro de *jobs* est en ordre (numéro 1 - numéro 200).

Le fichier « *memog_200_2.txt* » contient les données de solutions. Chaque deux lignes représente une solution d'un sous-problème. Par exemple : pour la première ligne dans la [Figure 2](#), « 0 » est le nombre de fois d'utilisation de cette solution, « 10 » est le nombre de *jobs* dans cette solution, « 3804 » est la date début (*start time* t_j) de cette solution. Pour la deuxième ligne, chaque chiffre représente le numéro d'un *job* dans cette solution.

Il faut noter que chaque solution est de différente taille.

3 Analyse d'entrée/sortie

Nous allons organiser les données dans une séquence comme l'entrée. Les entrées peuvent être une séquence de différentes taille parce que dans chaque solution le nombre de tâches n'est

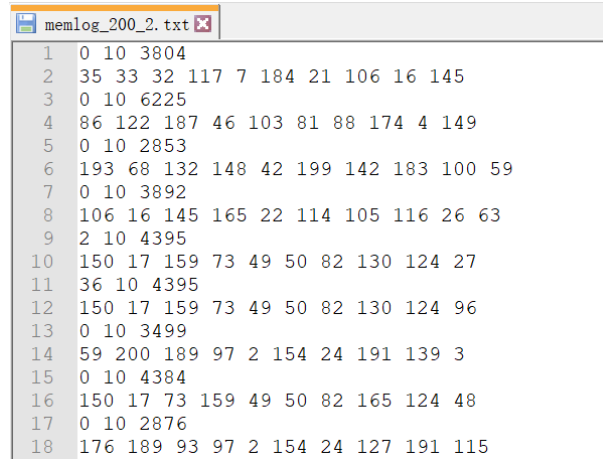


1	13	3319
2	3	3751
3	35	3876
4	86	3673
5	5	3440
6	92	3199
7	30	3979
8	86	3413
9	99	3039
10	4	2983
11	36	3255
12	66	2893
13	41	4212
14	27	2898
15	40	3460
16	21	4031
17	20	4417
18	5	2921
19	98	3734
20	7	2991
21	10	4010
22	10	4086
23	44	2929
24	40	3771
25	47	3883
26	27	4103
27	37	4154
28	54	3482
29	61	3036
30	72	2763

Figure 1 – Données d'une instance

pas pareil. Sur l'organisation des données, nous réfléchissons d'abord utiliser la moyenne de *processing time* et *due date* pour réduire la dimension d'entrée. Mais après la discussion avec encadrant, nous pensons cette méthode ne peut pas bien représenter les caractéristiques des données. Il vaut mieux de saisir chaque *processing time* et chaque *due date* en ordre.

La sortie est un résultat de classification. Il y a deux possibilités, la solution sera utilisée et la solution ne sera pas utilisée.



Index	Data
1	0 10 3804
2	35 33 32 117 7 184 21 106 16 145
3	0 10 6225
4	86 122 187 46 103 81 88 174 4 149
5	0 10 2853
6	193 68 132 148 42 199 142 183 100 59
7	0 10 3892
8	106 16 145 165 22 114 105 116 26 63
9	2 10 4395
10	150 17 159 73 49 50 82 130 124 27
11	36 10 4395
12	150 17 159 73 49 50 82 130 124 96
13	0 10 3499
14	59 200 189 97 2 154 24 191 139 3
15	0 10 4384
16	150 17 73 159 49 50 82 165 124 48
17	0 10 2876
18	176 189 93 97 2 154 24 127 191 115

Figure 2 – Données de solutions

4 Analyse de méthode

En général, quand nous choisissons une méthode de *Machine Learning*, on a plusieurs points à réfléchir, par exemple :

- ✓ Les caractéristiques de données
- ✓ La dimension et l'attribut de données
- ✓ Le temps d'entraînement
- ✓ L'urgence de mission
- ✓ L'utilisation de données
- ✓ La précision

Dans ce projet, les données sont des séquences avec plusieurs éléments. Les séquences n'ont pas de caractéristiques explicites. Même avec des techniques sophistiquées de sélection de caractéristiques, la dimensionnalité des caractéristiques potentielles peut être très élevée et la nature séquentielle des caractéristiques est difficile à capturer. Cela rend la classification de séquences difficile. La dimension de données est grande à cause du nombre de tâches dans une instance.

Cette mission n'est pas très urgente, donc nous n'allons pas de limite du temps d'entraînement. Pour le résultat, il faut une haute précision, sinon il ne peut pas optimiser le problème original.

En l'absence d'algorithmes testés différents, mêmes le développeur de données et d'algorithmes de *Machine Learning* ne peuvent pas dire quel algorithme a une meilleure performance. Nous ne nous attendons pas réussir pour une seule fois, mais nous espérons analyser les méthodes communes de *Machine Learning* pour choisir un algorithme.

Pour la méthode de classification naïve bayésienne, elle est adaptée aux tâches de multi-classification. Elle a une très bonne performance pour un petit volume de données. Mais pour un grand volume de données, la méthode ne marche pas bien. Quand les caractéristiques d'entrée sont dépendantes, il y a aussi des problèmes. Donc, cette méthode n'est pas choisie.

Pour la méthode de k plus proches voisins (KNN), la pensée est simple et la théorie est mûre, la complexité du temps d'entraînement est $O(n)$ et elle a une haute précision. Mais une grande quantité de calculs et beaucoup de mémoire sont nécessaires. Notre objectif consiste à économiser la mémoire. Donc, cette méthode n'est pas adaptée.

Pour la méthode de machine à vecteurs du support, l'idée de la classification est très simple, soit maximiser la distance entre les échantillons d'entraînement et la zone de décision. Et la

classification a un bon résultat. Mais c'est difficile pour cette méthode d'entraîner un gros volume de données. C'est une limite.

Pour la méthode d'arbre de décision, la classification par un seul arbre de décision est faible et elle est facile à sur-ajuster.

Pour la méthode d'arbre décisionnel, la précision est haute est l'entraînement est rapide. Mais la dimension des données ne peut pas être très grande.

Pour la méthode de réseau de neurones, la précision est haut. Et l'extraction automatique des caractéristiques peut être effectuée sans intervention humaine, cela permet de résoudre la difficulté que les séquences n'ont pas de caractéristiques explicites. Cette méthode est adaptée à un grand volume de données d'entraînement. L'inconvénient est que l'entraînement a une longue période et beaucoup de paramètres.

En comparant ces méthodes de *Machine Learning*, la méthode de réseau de neurones semble être un bon choix. Elle permet de traiter des problèmes non structurés dans des domaines très complexes. Il est plus performant que les statistiques ou les arbres de décisions.

Voici un tableau de synthèse **Figure 3**.

Analyse de méthode			
L'extraction automatique des caractéristiques	Adaptée à un gros volume de données	L'entraînement est rapide	Haute précision
Réseau de neurones	Réseau de neurones	Forêt d'arbres décisionnels, KNN	SVM, Réseau de neurones, KNN, Classification naïve bayésienne (petit volume de données)

↓

Réseau de neurones

20

Figure 3 – Tableau de synthèse

5 Conception

Pour les réseaux de neurones classiques, il y a deux limitation :

- n'acceptent que le vecteur de taille fixe.
- ne tiennent pas compte de la nature séquentielle de certaines données (langue, vidéo, série temporelle etc).

On a déjà parlé d'entrée, c'est une séquence de taille variable. Ainsi, les réseaux de neurones classiques ne s'appliquent pas à notre problème.

Mais les réseaux de neurones récurrents surmontent ces limitations. Il est adapté pour des données d'entrée de taille variable. Néanmoins les réseaux de neurones récurrents affrontent le problème de disparition du gradient pour apprendre à mémoriser des événements passés.

Et l'ordre de la séquence est aussi important parce que dans le problème original (*SMTT*), l'objectif est d'organiser les tâches en ordre afin de minimiser le retard total. C'est-à-dire il faut

tenir compte de les tâches antérieures. Donc c'est nécessaire de mémoriser des évènements passés.

Et un réseau *LSTM* est l'architecture particulière de réseau de neurones récurrents la plus utilisé en pratique qui permet de résoudre le problème de disparition de gradient. L'état de cellule du *LSTM* qu'on a mentionné dans l'état de l'art est pour le résoudre. Il peut sauvegarder l'état d'évènement à long terme.

Par conséquent, *LSTM* est notre choix final.

5

Mise en œuvre

1 Outils et librairies utilisées

- Librairies utilisées : *tensorflow* Figure 1, *numpy* Figure 2

Tensorflow est un outil open source d'apprentissage automatique pour le calcul numérique en utilisant des graphes de flux de données.



Figure 1 – Logo du Tensorflow

Numpy est une bibliothèque destinée à manipuler des matrices ou tableaux multidimensionnels ainsi que des fonctions mathématiques opérant sur ces tableaux.



Figure 2 – Logo du Numpy

- Outil de visualisation : *Tensorboard*

Tensorboard peut être utilisé pour visualiser le graphe de *Tensorflow* et aussi les paramètres.

Pour les détails sur les versions exactes et les installations sont en annexe.

2 Implémentation

2.1 Composition du projet

Ce projet est composé comme la **Figure 3** ci-dessous.

- *lstm.py* contient les codes principaux pour réaliser les fonctionnements.
- *test.py* sert à faire les tests unitaires.
- Le dossier "*data*" est pour stocker les données d'entraînement et les fichiers générés pour lancer *Tensorboard*.
- Le dossier "*test data*" est pour stocker les données pour les tests unitaires.
- Le dossier "*MODEL*" sert à sauvegarder le modèle généré par le réseau de neurones.
- *lstm.html* et *test.html* sont respectivement les documentations du *lstm.py* et *html.py*.

.cache	2018/3/24 18:55
.idea	2018/3/26 10:11
__pycache__	2018/3/26 0:00
data	2018/3/25 0:26
MODEL	2018/3/26 9:20
test data	2018/3/25 0:36
lstm	2018/3/26 0:01
lstm	2018/3/26 8:38
README.md	2018/2/21 10:41
tensorboard	2018/3/26 8:47
test	2018/3/25 23:49
test	2018/3/25 23:49

Figure 3 – Composition du projet

2.2 Diagramme de classe

Le diagramme de classes dans la **Figure 4** montre la modélisation du projet :

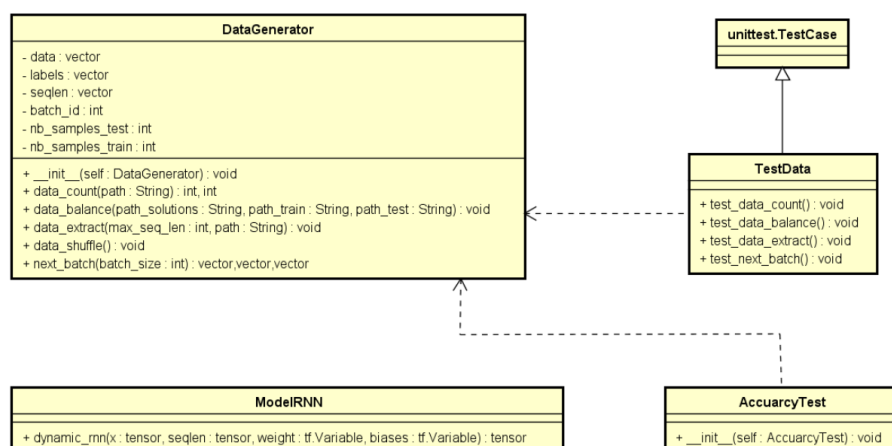


Figure 4 – Diagramme de classes

- La classe *DataGenerator* définit plusieurs méthodes pour traiter des données.
- La classe *ModelRNN* définit la méthode pour construire un réseau de neurones récurrent.
- La classe *AccuracyTest* sert à tester avec des données pour obtenir un pourcentage de précision.
- La classe *TestData* sert à faire des tests unitaires.

2.3 Choix techniques

2.3.1 Paramètres

Ce sont des paramètres que j'ai utilisé.

```
start_learning_rate = 10**-5
training_steps = 2000 # Number of training steps
batch_size = 256 # Number of data used to train each step
display_step = 50 # Print the loss and accuracy each 50 steps

# Network Parameters
seq_max_len = 200 # Sequence max length, Nbjobs
n_hidden = 256 # Hidden layer num of features. Size of weight array in a cell. The
               # capacity of learning ability.
n_classes = 2 # Appeared sequence or not
```

- *starting_learning_rate* : le taux d'apprentissage au début, représente la vitesse d'apprentissage.
- *training_steps* : le nombre de fois d'entraînement d'un *batch* de données. On peut dire « étape » en français.
- *batch_size* : la taille d'un batch, c'est-à-dire le nombre de données qu'on utilise pour faire l'entraînement chaque fois.
- *display_step* : chaque *display_step*, on affiche le résultat.
- *seq_max_len* : la longueur maximum de séquences.
- *n_hidden* : le nombre de couches cachées et aussi la taille de tableau de poids dans une cellule *LSTM*, représente la capacité d'apprentissage.

2.3.2 Définition d'entrées

Pour les données d'entrées, j'ai récupéré ce qu'on a besoin de données originales. On peut calculer le *start time* de chaque tâche par le *start time* et le *processing time* de son *job* antérieur.

Donc, les données d'entrées est un vecteur de 3-dimension. La première dimension représente une instance, la taille est le nombre de solutions dans cette instance. La deuxième dimension représente une solution, la taille est le nombre de *jobs* dans cette solution. La troisième dimension représente un *job*, la taille est 3 y compris le *start time* t_j , le *processing time* p_j et le *due date* d_j de ce *job*.

Il faut noter que la taille de chaque solution n'est pas pareil et j'ai complété chaque solution avec 0 pour que ils soient de la même taille afin de les saisir dans le réseau de neurones.

C'est montré dans la [Figure 5](#).

2.3.3 Définition d'étiquettes (*labels*)

Les *labels* pour l'entraînement est générés par le nombre de fois d'utilisation d'une solution. Si ce nombre est 0, le *label* est $[0., 1.]$, sinon le *label* est $[1., 0.]$. Les *labels* sont utilisés comme les valeurs réelles pour l'apprentissage. Ils seront contrastés avec les valeurs prédictives.

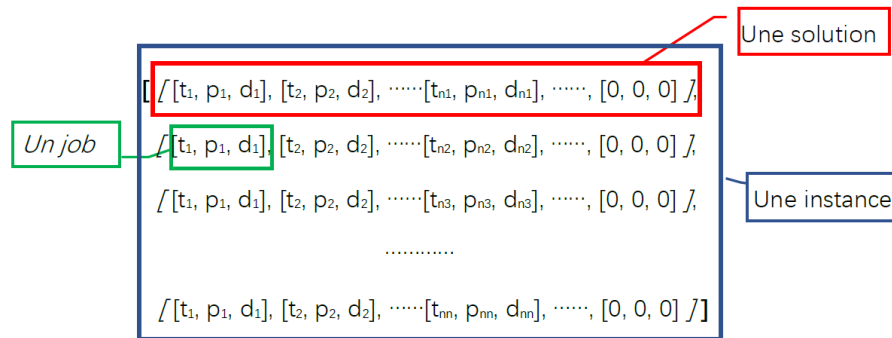


Figure 5 – Données d'entrée

2.3.4 Définition de sorties

Les état final du *LSTM* est utilisé pour calculer la sortie.

```
outputs, state = tf.contrib.rnn.static_rnn(cell=lstm_cell, inputs=x, dtype=tf.
                                float32, sequence_length=seqlen)
state = tf.stack(state[1])
pred_state = tf.matmul(tf.stack(state), weights['out']) + biases['out']
```

2.3.5 Méthode pour construire le réseau

D'abord, j'ai défini une cellule *lstm_cell* avec *tensorflow*. Ensuite, j'ai créé un réseau de neurones récurrents par *lstm_cell*. Le code est au dessous. « *outputs* » est une liste de sorties à chaque étape de temps. « *state* » est l'état final.

```
import tensorflow as tf
x = tf.unstack(value=x, num=seq_max_len, axis=1)
lstm_cell = tf.contrib.rnn.BasicLSTMCell(n_hidden)
outputs, state = tf.contrib.rnn.static_rnn(cell=lstm_cell, inputs=x, dtype=tf.
                                float32, sequence_length=seqlen)
state = tf.stack(state[1])
pred_state = tf.matmul(tf.stack(state), weights['out']) + biases['out']
return pred_state
```

2.3.6 Définition de *loss*

Le *loss* est utilisé pour calculer la différence entre la valeur prédictive du modèle et la valeur réelle.

L'objectif du modèle est de minimiser le *loss*.

Le *loss* est défini comme :

```
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(logits=pred,
                                labels=y)) # Define loss
```

La méthode *tf.nn.softmax_cross_entropy_with_logits_v2(logits=pred, labels=y)* calcule l'entropie croisée softmax entre les *logits* (les valeurs prédictives) et les *labels* (les valeurs réelles). L'idée est de mesurer la différence de la distribution de probabilité entre les *logits* et les *labels*. Après, j'ai utilisé *tf.reduce_mean()* pour calculer la moyenne des entropies croisées *softmax* afin de obtenir une valeur de *loss*.

2.3.7 Définition de précision

Pour définir la précision d'entraînement, j'ai utilisé le code ci-dessous.

```
correct_pred = tf.equal(tf.argmax(pred, 1), tf.argmax(y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))
```

La méthode *tf.equal()* est pour détecter si la valeur prédictive correspond à la valeur réelle cette méthode retourne un tableau de type booléen. Ensuite, il faut calculer la moyenne de ces valeurs booléennes pour obtenir une précision d'entraînement.

2.3.8 Définition d'optimiseur

L'optimiseur sert à implémenter l'algorithme de descente de gradient pour minimiser le *loss*.

```
optimizer = tf.train.GradientDescentOptimizer(learning_rate=start_learning_rate).
            minimize(cost)
```

Compte tenu de la limite de la CPU de mon ordinateur, j'ai choisi les données d'une instance pour l'entraînement. Dans cette instance, il y a 41814 échantillons (solutions). Je les ai divisé en deux parties : 31362 échantillons pour l'entraînement, 10452 échantillons pour le test.

Chaque échantillon a une étiquette (*label*). Donc notre apprentissage contient à l'apprentissage supervisé. C'est une technique d'apprentissage automatique où l'on cherche à produire automatiquement des règles à partir d'une base de données d'apprentissage.

Les poids dans le réseau de neurones sont au préalable initialisés avec des valeurs aléatoires. On considère ensuite un ensemble de données qui vont servir à l'apprentissage.

Le processus est que : En donnant un batch de données comme entrée chaque étape, *LSTM* calcule la sortie du réseau, après il fait une rétro-propagation du gradient pour calculer l'erreur pour chaque neurone d'un réseau de neurones. Enfin, selon les erreurs de neurones, les valeurs de poids sont mises à jour.

2.4 Analyse

2.4.1 Équilibrage de données

Dans les données originales, les échantillons d'entraînement qui appartiennent à la classe [0.,1.] (le nombre de fois d'utilisation est 0) est beaucoup plus que lesquels qui appartiennent à la classe [1.,0.] (le nombre de fois d'utilisation n'est pas 0).

Les données d'entraînement sont équilibrées et cela permet d'améliorer la précision d'entraînement et de test.

2.4.2 Mélange de données

D'abord, je n'ai pas mélangé les données, et l'entraînement n'est pas stable. L'entraînement a une oscillation.

Après le mélange, le résultat est plus stable.

2.4.3 Réorganisation de données

Au début, les données sont organisées en un tableau d'une forme [nombre de solutions, seq_max_len*3, 1]. Après, je l'ai changé à [nombre de solutions, seq_max_len, 3]. Ainsi, la vitesse d'exécution sont améliorées. Et cela permet de rendre la caractéristique d'un job plus apparente.

2.4.4 Ajustement de paramètres

L'ajustement de paramètre comme le taux d'apprentissage, la taille d'un batch, le nombre de couches cachées est aussi important pour améliorer mon modèle.

Limites et risques Les données d'entraînement sont nombreuses. Par exemple, dans une instance, il y a environ 150000 solutions. Donc, une machine doit être aussi puissante pour faire l'entraînement sinon la machine peut être bloquée.

Et le processus pour ajuster les paramètres sont très long et sec. Pendant cet étape, il faut essayer plusieurs fois différents paramètres. Donc il faut être très patient.

3 Évaluation de performance

L'évaluation de performance dépend trois aspects : la tendance de *loss* d'entraînement, la tendance de précision d'entraînement et la précision de test. J'ai utilisé *Tensorboard* pour visualiser la tableau de tendance de *emphloss* d'entraînement et la tableau de tendance de précision d'entraînement.

3.1 Stratégie d'évaluation

Pour justifier le résultat, j'ai fait un tableau **Table 1** de différents modèles que j'ai construit.

Table 1 – Tableau de synthèse

Modèle	Structure
modèle 1	Le modèle sans entraînement
modèle 2	modèle 1 + entraînement
modèle 3	modèle 2+mélange et équilibrage de données+ajustement de taux d'apprentissage
modèle 4	modèle 3 + ajustement de la taille de batch
modèle 5	modèle final

Le tableau **Table 2** est la stratégie d'évaluation.

3.2 Analyse des résultats

Résultat de modèle 1

J'ai testé ce modèle 10 fois sans entraînement avec poids aléatoire. On peut calculer la moyenne pour la comparer avec le modèle qui est avec l'entraînement.

Table 2 – Tableau de stratégie d'évaluation

ID	Condition
1	Évaluation du modèle 1
2	Évaluation du modèle 2
3	Évaluation du modèle 3
4	Évaluation du modèle 4
5	Évaluation du modèle 5

Table 3 – Tableau de résultat de modèle 1

test ID	1-1	1-2	1-3	1-4	1-5
Précision	0.46470	0.50057	0.46450	0.50564	0.49062

Table 4 – Tableau de résultat de modèle 1

test ID	1-6	1-7	1-8	1-9	1-10	Moyenne
Précision	0.49493	0.50000	0.48259	0.50000	0.47426	0.487781

Le résultat est dans le tableau [Table 3](#) et [Table 4](#).

La moyenne de précision de test du modèle 1 est 0.487781.

Résultat de modèle 2

Le mélange et l'équilibrage de données ne sont pas faits dans le modèle 2. J'ai obtenu une précision de test 0.44718716.

Et dans le tableau de tendance de *loss* et de précision d'entraînement [Figure 6](#), les deux variables ne convergent pas du tout.

Résultat de modèle 3

Les résultats de précision de test qui utilise différents taux d'apprentissage sont dans le tableau [Table 5](#).

Table 5 – Tableau de résultat de modèle 3

Learning rate	10^{-7}	10^{-6}	10^{-5}	10^{-4}
Précision	0.6283008	0.6977612	0.63432837	0.6653272

Les tableaux de tendance de *loss* et de précision d'entraînement sont dans la [Figure 7](#), la [Figure 8](#), la [Figure 9](#) et la [Figure 10](#).

On va comparer les résultats par deux aspects : la précision et la convergence de *loss* et de la précision d'entraînement.

Et par comparaison, je pense que 10^{-5} est un bon choix comme le taux d'apprentissage.

Résultat de modèle 4

Les résultats de précision de test qui utilise différents taille de batch sont dans le tableau [Table 6](#).

Les tableaux de tendance de *loss* et de précision d'entraînement sont dans la [Figure 11](#), la [Figure 12](#) et la [Figure 13](#).

Par comparaison, la convergence de *loss* et de précision d'entraînement est meilleur quand la taille de batch égale à 512.



Figure 6 – Pas de mélange et équilibrage de données

Table 6 – Tableau de résultat de modèle 4

Batch size	128	256	512
Précision	0.5164562	0.7090509	0.58802146

Résultat de modèle 5

Le résultat est dans le tableau [Table 7](#) et [Table 8](#).

Table 7 – Tableau de résultat de modèle 5

test ID	5-1	5-2	5-3	5-4	5-5
Précision	0.6625526	0.8034826	0.6472446	0.8104669	0.65059316

Table 8 – Tableau de résultat de modèle 5

test ID	5-6	5-7	5-8	5-9	5-10	Moyenne
Précision	0.7057979	0.66350937	0.7106774	0.8214696	0.6880023	0.71637964

La moyenne de précision du modèle 5 est 0.71637964.

On peut voir que la précision augmente environ 22.86% en comparant avec le modèle sans entraînement. Cela permet de justifier l'efficacité de notre méthode.

Et jusqu'à ce moment, les paramètres du meilleur modèle que j'ai entraîné sont :

Figure 7 – Learning rate= 10^{-7}

```
# Parameters
start_learning_rate = 10**-5
training_steps = 2000 # Number of training steps
batch_size = 512 # Number of data used to train each step

# Network Parameters
seq_max_len = 200 # Sequence max length, Njobs
n_hidden = 256 # Hidden layer num of features. Size of weight array in a cell. The
                # capacity of learning ability.
n_classes = 2 # Appeared sequence or not
```

La précision de test est 0.73488325. Et la tableau de tendance de *loss* et précision d'entraînement est dans la [Figure 14](#).

Remarque : Dans les tableaux de tendance de *loss* et de précision, la ligne avec une couleur foncée est la courbe après le lissage. Et la ligne avec une couleur claire est la courbe originale.

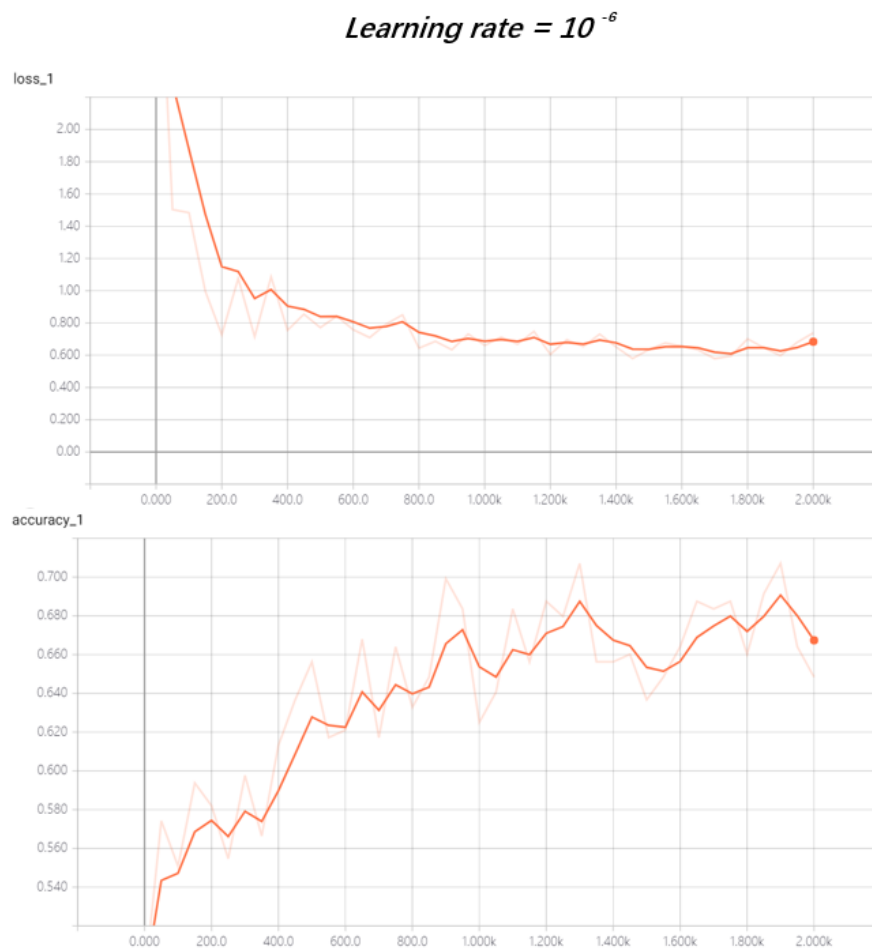


Figure 8 – Learning rate= 10^{-6}

**Figure 9** – Learning rate= 10^{-5}

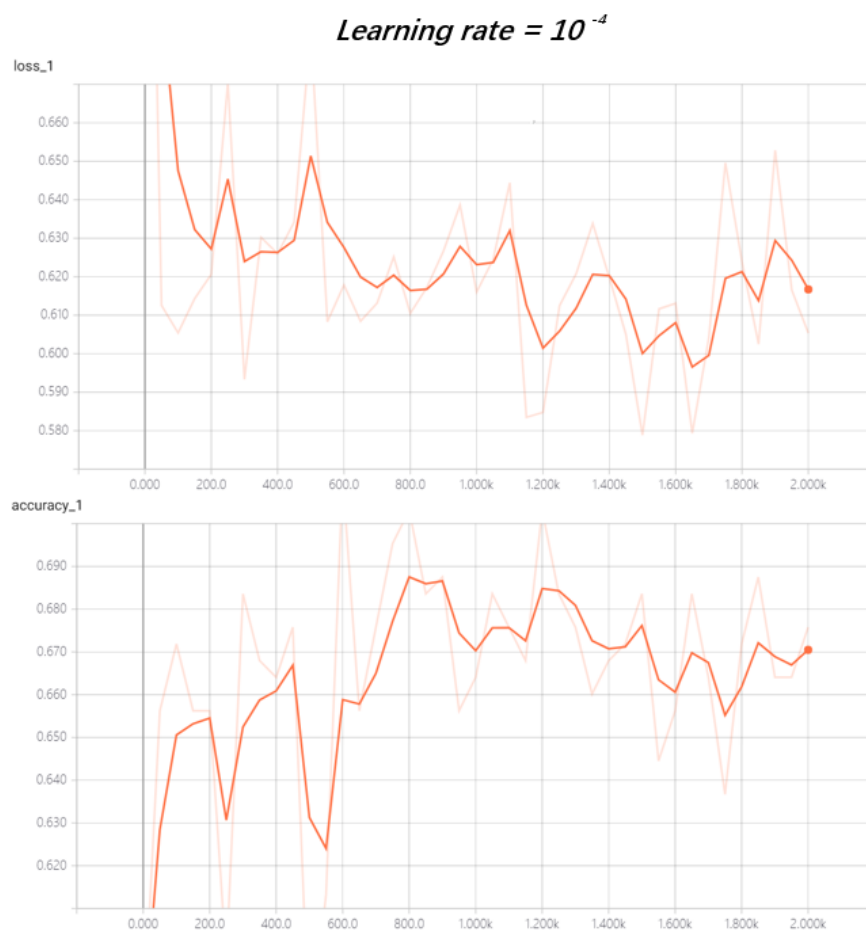


Figure 10 – Learning rate= 10^{-4}

Batch size = 128

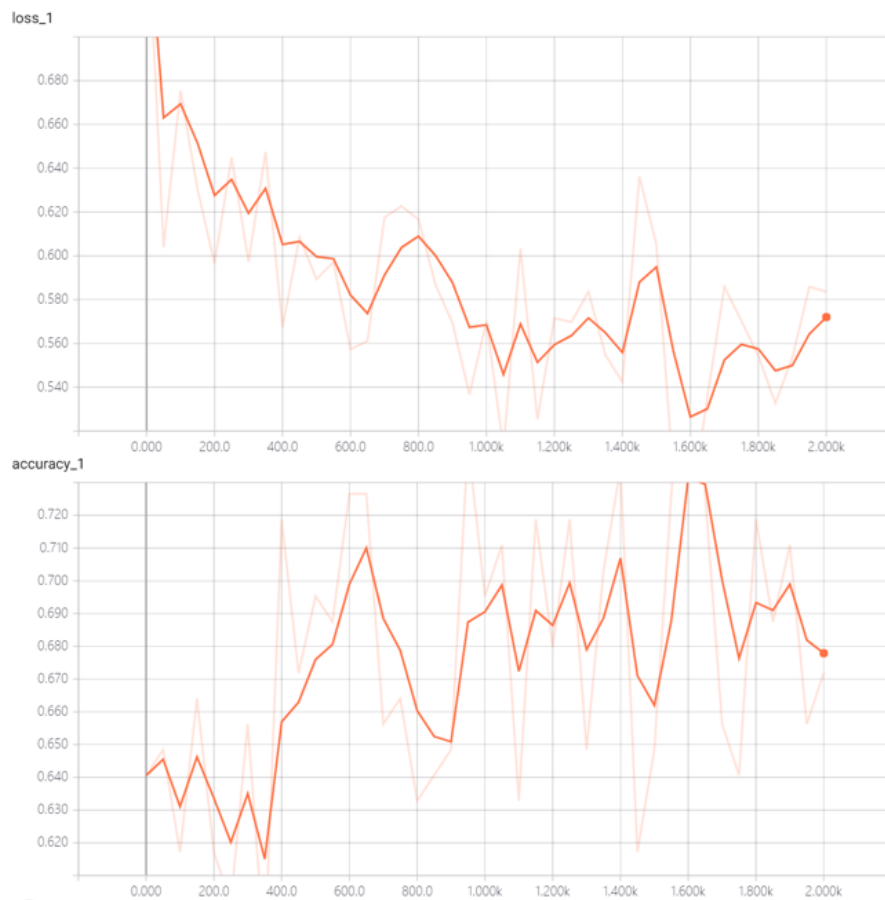


Figure 11 – Batch size=128

Batch size = 256



Figure 12 – Batch size=256

**Figure 13 – Batch size=512**



Figure 14 – Le meilleur modèle rencontré

6

Bilan et conclusion

Dans le premier semestre, j'ai lu quelques littératures pour savoir plus sur le problème et l'organisation de données. Et j'ai étudié plusieurs méthodes de *Machine Learning* et je les ai comparées. Ça me permet d'améliorer la compétence d'auto-apprentissage et d'analyse. Enfin j'ai choisi la méthode de réseau de neurones pour faire ce projet.

Et dans le deuxième semestre, j'ai fait tous mes efforts sur la réalisation. J'ai traité les données et construit le modèle du réseau de neurones récurrents pour faire l'entraînement. Et j'ai essayé plusieurs méthodes pour améliorer le modèle. Enfin j'ai fait nombreux tests sur l'implémentation afin de vérifier le bon fonctionnement du modèle. J'ai appris le processus d'apprentissage profond et j'ai eu une connaissance profonde de réseau de neurones. J'ai beaucoup appris dans ce projet.

J'ai résumé ce que j'ai fait et la perspective.

1 Fait

- 1) Analyse du contexte et du problème
- 2) L'organisation du projet
- 3) Analyse de l'entrée et de la sortie
- 4) Apprentissage des principales méthodes de *Machine Learning*
- 5) Apprentissage la méthode de classification de séquence
- 6) Choix et vérification d'une méthode
- 7) Rédaction du cahier de spécification et du rapport
- 8) Construire un modèle en utilisant la méthode de réseau de neurones
- 9) Entraîner et évaluer le modèle
- 10) Améliorer le modèle
- 11) Modifier le rapport

2 Perspective

- 1) Utiliser la validation croisée pour estimer la fiabilité
- 2) Augmenter le nombre d'échantillons

3) Appliquer le modèle en pratique

3 Bilan sur la qualité

Après les essais de plusieurs méthode pour améliorer le modèle d'apprentissage, la précision augmente environ 22.86% en comparant avec le modèle sans entraînement. C'était pas mal.

Le plus important c'est que dans ce domaine la méthode d'apprentissage profond n'est pas beaucoup utilisée. Le résultat de projet indique que l'on peut continuer la recherche en utilisant la méthode d'apprentissage profond dans ce domaine.

Annexes

A

Spécifications fonctionnelles

La fonctionnalité du système est de prévoir si une solution est utile pour déterminer si elle doit être mémorisée.

L'entrée est une séquence qui contient les données liées d'une solution.

La sortie est un résultat de classification. Il y a deux possibilités, la solution est utile et la solution n'est pas utile.

Donc, il faut construire un classificateur pour réaliser cette fonction.

B

Spécifications non fonctionnelles

1 Contraintes de développement

Il n'y a pas de contrainte de développement liée au matériel. Il faut noter cependant qu'une machine doit être "puissante" pour l'étape d'entraînement.

2 Contraintes de fonctionnement

Il y a quelques contraintes sur les données.

- ✓ La dimensionnalité de données est grande.
- ✓ La taille de données de chaque solution est différente.
- ✓ Une solution (correspond à un nœud dans l'arbre) peut apparaître plusieurs fois dans une instance, mais pour les autres instances, ce n'est pas forcément le cas. Donc il faut considérer la relation entre le nœud et le nœud parent. Ainsi, la dimensionnalité de données doit augmenter.

1 Installation et configuration

Installer Python 3.6.2

Télécharger l'installateur de Python 3.6 (<https://www.python.org/>) et l'installer.

Variable d'environnement

Ajouter le chemin de Python (par exemple : « C:\Python36 ») et le chemin de *pip* (par exemple : « C:\Python36\Lib\site-packages\pip ») à variable d'environnement.

Installer *tensorflow*, *numpy*

Utiliser *pip* pour installer les bibliothèques.

Les commande d'installation du *tensorflow* Figure 1 :

- *pip install --upgrade tensorflow*

Les commande d'installation du *numpy* :

```
D:\PRD\Code\Memorisation>pip install --upgrade tensorflow
Collecting tensorflow
  Downloading tensorflow-1.6.0-cp36-cp36m-win_amd64.whl (32.3MB)
    100% |#####| 32.3MB 40kB/s
```

Figure 1 – Installation de *tensorflow*

- *pip install numpy*

Installer Git

- Télécharger Git sur le site web <https://git-scm.com/downloads>

- Cliquer avec le bouton droit dans votre répertoire, choisir « Git Bash Here » et vous pouvez voyer un terminal Figure 2

- Taper la commande « *git clone https://github.com/unbreakablelzc/Memorisation.git* » pour obtenir le code et vous pouvez voyer différentes versions du code Figure 3.

Lancer Tensorboard

Taper la commande dans votre terminal :

- *tensorboard --logdir path/to/log-directory* Figure 4

Remarque : « path/to/log-directory » est un chemin absolu

```

MINGW64:/c/Users/tospi/Desktop
tospi@DESKTOP-V33D3TJ MINGW64 ~/Desktop
$ |

```

Figure 2 – Lancer Git

```

MINGW64:/c/Users/tospi/Desktop
tospi@DESKTOP-V33D3TJ MINGW64 ~/Desktop
$ git clone https://github.com/unbreakable1zc/Memorisation.git
Cloning into 'Memorisation'...
remote: Counting objects: 102, done.
remote: Compressing objects: 100% (75/75), done.
remote: Total 102 (delta 49), reused 74 (delta 24), pack-reused 0
Receiving objects: 100% (102/102), 20.35 MiB | 2.34 MiB/s, done.
Resolving deltas: 100% (49/49), done.
Checking out files: 100% (25/25), done.
tospi@DESKTOP-V33D3TJ MINGW64 ~/Desktop
$ |

```

Figure 3 – Obtenir code de Github

```

D:\PRD\Code\Memorisation>tensorboard --logdir D:\PRD\Code\Memorisation\data\try
TensorBoard 1.6.0 at http://DESKTOP-V33D3TJ:6006 (Press CTRL+C to quit)

```

Figure 4 – Lancer Tensorboard

- Naviguer votre navigateur web indiqué, ici c'est « <http://DESKTOP-V33D3KTJ:6006> » pour voir l'interface de *TensorBoard* **Figure 5**

2 Lancement du projet

Le projet compile sous « *Pycharm 2017.1.3* » sous « *Windows 10 64 bit* ». Les bibliothèques indiquées ci-dessus doivent être installées. Et la configuration d'environnement est nécessaire. Ensuite, vous pouvez exécuter *lstm.py* en utilisant *Pycharm*. Vous pouvez obtenir :

- Un affichage du processus d'exécution **Figure 6**
- Un fichier d'évènement qui peut être utilisé pour visualiser le graphe et les paramètres de *Tensorflow* **Figure 7**
- Des fichiers de variables qui sert à sauvegarder le modèle d'apprentissage **Figure 8**

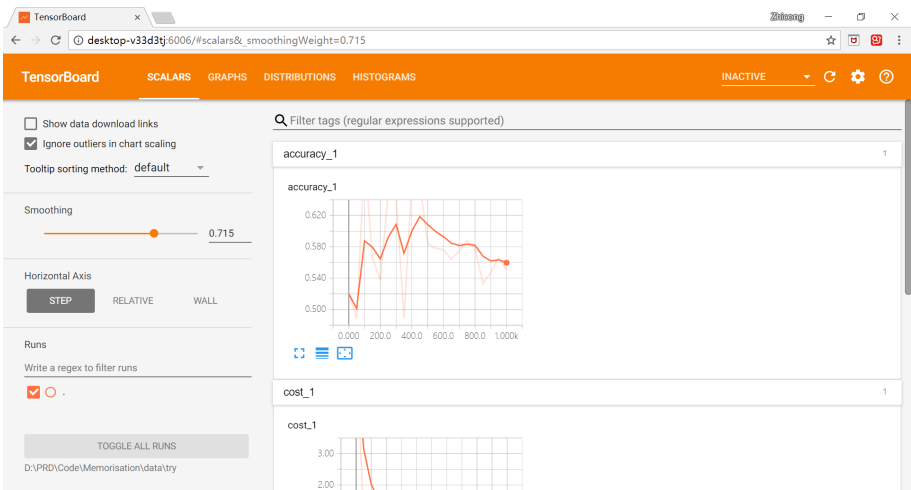


Figure 5 – Variables avec Tensorboard

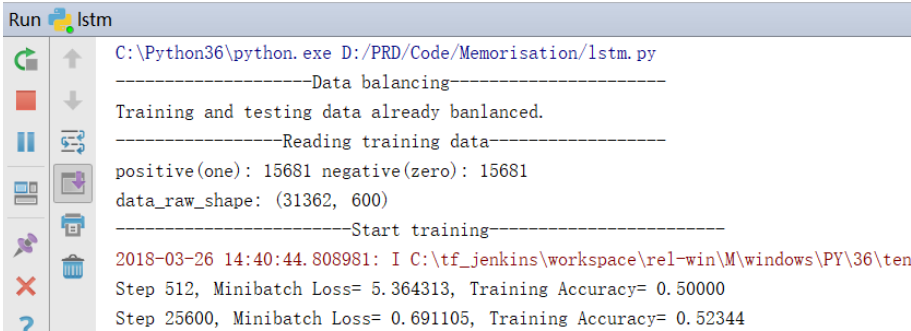


Figure 6 – Exemple de résultat



Figure 7 – Fichier d'évènement

checkpoint	2018/3/25 23:03	文件	1 KB
model.ckpt.data-00000-of-00001	2018/3/25 23:03	DATA-00000-OF-0...	1,043 KB
model.ckpt.index	2018/3/25 23:03	INDEX 文件	1 KB
model.ckpt.meta	2018/3/25 23:04	META 文件	12,831 KB

Figure 8 – Fichiers de variables

D

Plan de développement et organisation

Dans les parties qui suivent, j'ai explicité les différentes tâches principales de notre projet avant de les représenter dans un diagramme de Gantt pour les répartir dans le temps.

1 Tâche 1

Tâche 1 permet de comprendre le projet et d'en définir son périmètre. Tout au long du projet, cette tâche pourra être reprise si besoin (informations supplémentaires, par exemple). Elle est composée de lecture d'articles et de documents concernant notre sujet, mais aussi d'informations fournies par la MOA et de réunions entre la MOA et la MOE.

2 Tâche 2

Tâche 2 consiste à apprendre des principales méthodes de *Machine Learning*. J'ai appris les méthodes qui sont informées dans l'état de l'art. Dans la réunion avec les encadrants, on s'est discuté pour choisir une méthode. Par analyse et comparaison des différentes méthodes, on a choisi la méthode réseau de neurones.

3 Tâche 3

Tâche 3 consiste à collectionner les données et apprendre des librairies ce que je vais utiliser. J'ai appris la librairie *tensorflow* et aussi la librairie *Keras*. *tensorflow* est le back-end de *Keras*, il est plus puissant que *Keras*. Donc finalement j'ai choisi *tensorflow*.

4 Tâche 4

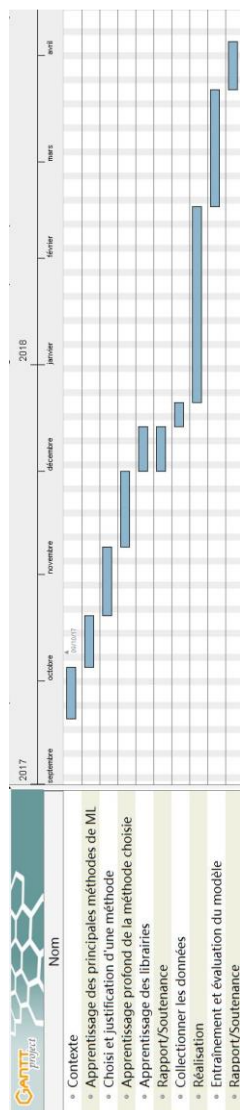
Tâche 4 consiste à la réalisation. Au début, j'ai traité les données et récupéré les données utiles. Et après j'ai construit le modèle du réseau de neurones récurrents pour faire l'entraînement. Et puis j'ai essayé plusieurs méthodes pour améliorer le modèle, y compris l'équilibrage et le mélange de données, l'ajustement de paramètres.

5 Tâche 5

Tâche 5 consiste à évaluer le modèle. Je vais effectuer de nombreux tests sur l'implémentation afin de vérifier le bon fonctionnement du modèle.

6 Diagramme de Gantt

Le diagramme de Gantt est comme **Figure 1**



E

Dossiers de tests

1 Tests unitaires

Généralement, le framework « *unittest* » du *Python* est utilisé pour faire les tests unitaires. « *unittest* » fournit une classe de base « *TestCase* » qui peut être utilisée pour créer des cas de test.

Les tests individuels sont définis par des méthodes dont le nom commence par « test ». Cette convention de nommage informe le *test runner* si une méthode est une méthode de test.

Ce que j'ai utilisé pour faire des tests unitaires est principalement l'appel de la méthode *assertEqual()* afin de vérifier un résultat attendu.

1.1 Test runner

Un *test runner* est un composant qui organise l'exécution de tests et fournit le résultat à l'utilisateur.

1.2 Test case

Un cas de test (*test case*) est l'unité individuelle de test. Il vérifie une réponse spécifique à un ensemble d'entrées particulières.

Voici un tableau de tests unitaires [Figure 1](#), [Figure 2](#).

Le résultat de tests unitaires est dans la [Figure 3](#).

2 Tests fonctionnels

Les calculs comme l'entraînement d'un réseau de neurones profond avec *Tensorflow* peuvent être très compliqués. Pour faciliter la compréhension et optimiser des programmes, on peut utiliser *Tensorboard* pour visualiser le graphe de flux de données et aussi les paramètres. Et on peut aussi l'utiliser pour faire les tests fonctionnels.

ID	Nom	Description	Résultat attendu	Statut
1	<i>test_data_count</i>	Appeler la méthode <i>data_count()</i> dans la classe <i>DataGenerator</i> avec un ensemble de données connues pour tester si le compte de deux classes de données est correct.	<i>True</i> (<i>nb_zero=17</i>); <i>True</i> (<i>nb_one=8</i>);	OK
2	<i>test_data_balance</i>	Appeler la méthode <i>data_balance()</i> dans la classe <i>DataGenerator</i> avec un ensemble de données connues pour tester si les données sont bien équilibrées. Les données de nombre pair et impair sont tout testées.	<i>True</i> (le fichier de données d'entraînement généré est pareil comme le fichier connu) ; <i>True</i> (le fichier de données de test généré est pareil comme le fichier connu) ; <i>True</i> ; <i>True</i> ;	OK
3	<i>test_data_extract</i>	Appeler la méthode <i>data_extract()</i> dans la classe <i>DataGenerator</i> avec un ensemble de données connues pour tester si les données récupérées sont correctes.	<i>True</i> (le vecteur du <i>data</i> généré est pareil comme le vecteur connu) ; <i>True</i> (le vecteur des <i>labels</i> généré est pareil comme le vecteur connu) ;	OK

Figure 1 – Tableau de tests unitaires

			<i>True</i> (le vecteur du <i>seqlen</i> généré est pareil comme le vecteur connu)	
4	<i>test_next_batch</i>	Appeler la méthode <i>next_batch()</i> dans la classe <i>DataGenerator</i> avec un ensemble de données connues pour tester si le <i>batch_id</i> généré est correct.	<i>True</i> ; <i>True</i>	OK

Figure 2 – Tableau de tests unitaires

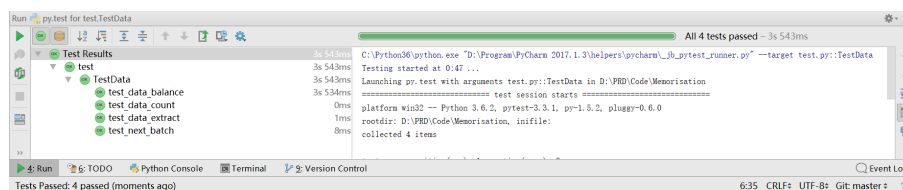
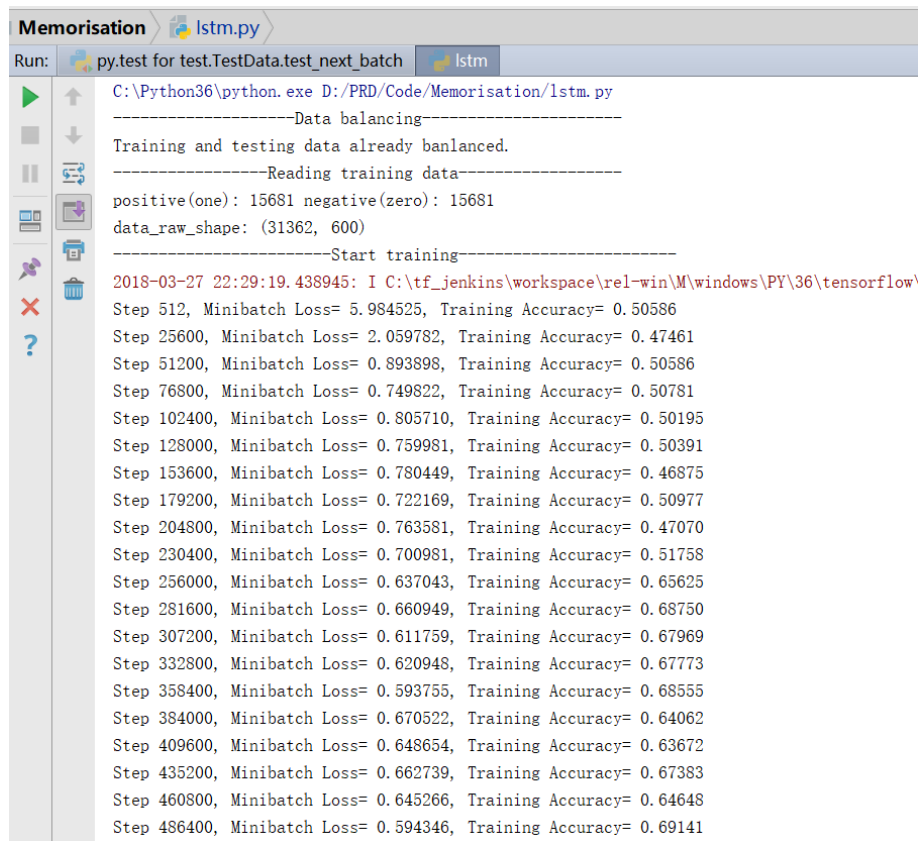


Figure 3 – Résultat de tests unitaires

2.1 Test d'entraînement

Dans le processus d'entraînement, l'objectif est de minimiser le « *Minibatch Loss* » pas à pas de sorte que l'on puisse augmenter le « *Training Accuracy* ».

Pour le tester, j'ai affiché le « *Minibatch Loss* » et le « *Training Accuracy* » chaque 50 étapes. Le résultat est dans la Figure 4, Figure 5.



```

Memorisation > lstm.py
Run: py.test for test.TestData.test_next_batch lstm
C:\Python36\python.exe D:/PRD/Code/Memorisation/lstm.py
-----Data balancing-----
Training and testing data already banlanced.
-----Reading training data-----
positive(one): 15681 negative(zero): 15681
data_raw_shape: (31362, 600)
-----Start training-----
2018-03-27 22:29:19.438945: I C:\tf_jenkins\workspace\rel-win\M\windows\PY\36\tensorflow\
Step 512, Minibatch Loss= 5.984525, Training Accuracy= 0.50586
Step 25600, Minibatch Loss= 2.059782, Training Accuracy= 0.47461
Step 51200, Minibatch Loss= 0.893898, Training Accuracy= 0.50586
Step 76800, Minibatch Loss= 0.749822, Training Accuracy= 0.50781
Step 102400, Minibatch Loss= 0.805710, Training Accuracy= 0.50195
Step 128000, Minibatch Loss= 0.759981, Training Accuracy= 0.50391
Step 153600, Minibatch Loss= 0.780449, Training Accuracy= 0.46875
Step 179200, Minibatch Loss= 0.722169, Training Accuracy= 0.50977
Step 204800, Minibatch Loss= 0.763581, Training Accuracy= 0.47070
Step 230400, Minibatch Loss= 0.700981, Training Accuracy= 0.51758
Step 256000, Minibatch Loss= 0.637043, Training Accuracy= 0.65625
Step 281600, Minibatch Loss= 0.660949, Training Accuracy= 0.68750
Step 307200, Minibatch Loss= 0.611759, Training Accuracy= 0.67969
Step 332800, Minibatch Loss= 0.620948, Training Accuracy= 0.67773
Step 358400, Minibatch Loss= 0.593755, Training Accuracy= 0.68555
Step 384000, Minibatch Loss= 0.670522, Training Accuracy= 0.64062
Step 409600, Minibatch Loss= 0.648654, Training Accuracy= 0.63672
Step 435200, Minibatch Loss= 0.662739, Training Accuracy= 0.67383
Step 460800, Minibatch Loss= 0.645266, Training Accuracy= 0.64648
Step 486400, Minibatch Loss= 0.594346, Training Accuracy= 0.69141

```

Figure 4 – Affiche du résultat

2.2 Test de graphe de Tensorflow

Dans mon code, j'ai ajouté quelques codes pour générer le fichier d'évènement. Et j'ai utilisé ce fichier pour visualiser le graphe de flux de données et le tableau de tendance du « *loss* », « *Accuracy* ».

Voici la figure de graphe de flux de données **Figure 6**. Il est cohérent comme prévu.

Voici la figure du tableau de tendance de « *Loss* » **Figure 7**. On peut voir que le « *Loss* » diminue graduellement. C'est aussi comme prévu.

Voici la figure du tableau de tendance de « *Accurary* » **Figure 8**. On peut voir que le « *Accurary* » augmente graduellement. C'est comme prévu.

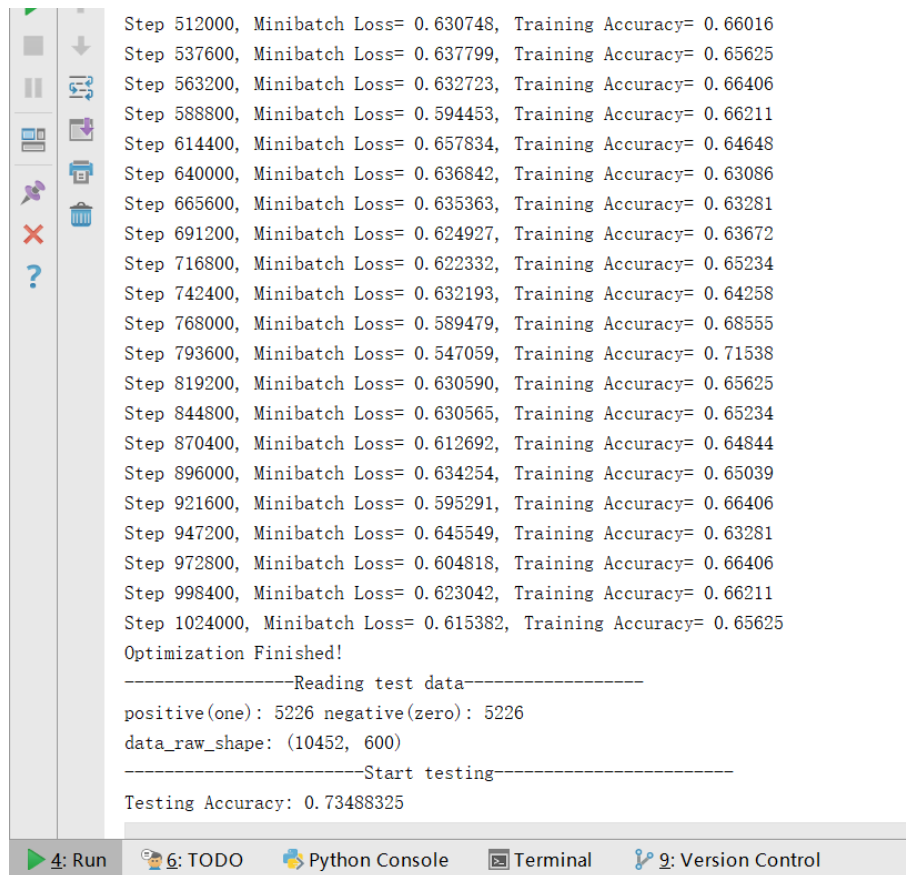


Figure 5 – Affiche du résultat

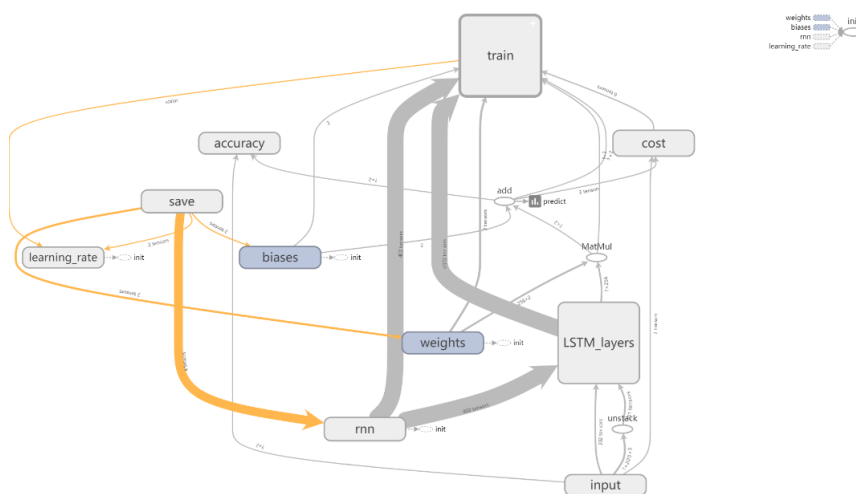


Figure 6 – Graphe de flux de données

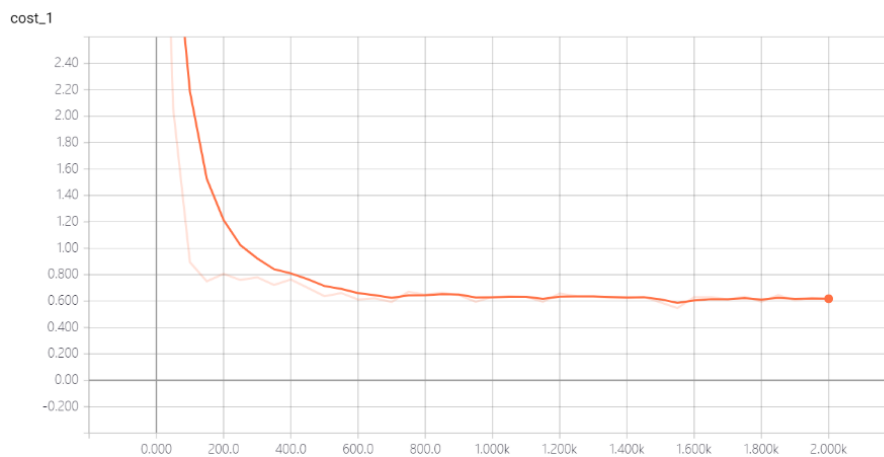


Figure 7 – Tableau de tendance du loss



Figure 8 – Tableau de tendance de précision d'entraînement



Webographie

- [WWW1] *Perceptron*. fr. Page Version ID : 142691175. Nov. 2017. URL : <https://fr.wikipedia.org/w/index.php?title=Perceptron&oldid=142691175> (visité le 07/12/2017).
- [WWW2] *Réseau de neurones récurrents*. fr. Page Version ID : 145972439. Mar. 2018. URL : https://fr.wikipedia.org/w/index.php?title=R%C3%A9seau_de_neurones_r%C3%A9currents&oldid=145972439 (visité le 31/03/2018).
- [WWW3] *Séparation et évaluation* — Wikipédia. URL : https://fr.wikipedia.org/wiki/S%C3%A9paration_et_%C3%A9valuation (visité le 07/12/2017).

Bibliographie

- [1] Eugene L. LAWLER. « A “Pseudopolynomial” Algorithm for Sequencing Jobs to Minimize Total Tardiness**Research supported by National Science Foundation Grant GJ-43227X ». In : *Studies in Integer Programming*. Sous la dir. de P.L. HAMMER, E.L. JOHNSON, B.H. KORTE et G.L. NEMHAUSER. T. 1. Annals of Discrete Mathematics Supplement C. Elsevier, 1977, p. 331–342. DOI : [https://doi.org/10.1016/S0167-5060\(08\)70742-8](https://doi.org/10.1016/S0167-5060(08)70742-8). URL : <http://www.sciencedirect.com/science/article/pii/S0167506008707428>.
- [2] *Machine Learning*. 2016. ISBN : 9787302423287. URL : <https://books.google.fr/books?id=j0G8nQAACAAJ>.
- [3] Lei SHANG, Vincent T’KINDT et Federico DELLA CROCE. « Exact Solution of the Single Machine Total Tardiness Problem : the Power of Memorization ». In : *International Conference on Industrial Engineering and System Management (IESM’17)*. Saarbrücken, Germany, oct. 2017. URL : <https://hal.archives-ouvertes.fr/hal-01550022>.
- [4] Włodzimierz SZWARC, Federico DELLA CROCE et Andrea GROSSO. « Solution of the single machine total tardiness problem ». In : *Journal of Scheduling* 2.2 (1999), p. 55–71. ISSN : 1099-1425. DOI : [10.1002/\(SICI\)1099-1425\(199903/04\)2:2<55::AID-JOS14>3.0.CO;2-5](https://doi.org/10.1002/(SICI)1099-1425(199903/04)2:2<55::AID-JOS14>3.0.CO;2-5). URL : [http://dx.doi.org/10.1002/\(SICI\)1099-1425\(199903/04\)2:2%3C55::AID-JOS14%3E3.0.CO;2-5](http://dx.doi.org/10.1002/(SICI)1099-1425(199903/04)2:2%3C55::AID-JOS14%3E3.0.CO;2-5).
- [5] Zhengzheng XING, Jian PEI et Eamonn KEOGH. « A Brief Survey on Sequence Classification ». In : *SIGKDD Explor. Newsl.* 12.1 (nov. 2010), p. 40–48. ISSN : 1931-0145. DOI : [10.1145/1882471.1882478](https://doi.org/10.1145/1882471.1882478). URL : <http://doi.acm.org/10.1145/1882471.1882478>.

Mémorisation Intelligente:

optimiser par apprentissage automatique

Résumé

Ce projet utilise la méthode d'apprentissage automatique pour optimiser un problème d'ordonnancement classique *SMTT* (*Single Machine Total Tardiness Problem*). Le problème est divisé en un certain nombre de sous-problèmes et en prenant la meilleure solution trouvée, on est assuré d'avoir résolu le problème initial. Pour faciliter on peut mémoriser les solutions de sous-problèmes résolus dans la mémoire. Lorsque ce sous-problème réapparaît, la solution enregistrée sera utilisée pour résoudre ce sous-problème. L'objectif est de prévoir si une solution d'un sous-problème dans *SMTT* sera utilisée pour résoudre d'autres sous-problèmes. Le projet peut être considéré comme un problème de classification de séquences.

Mots-clés

SMTT, Mémorisation, Séquence, Apprentissage Automatique, Réseau de Neurones Artificiels, *LSTM*, *Tensorflow*, *Python*, \LaTeX

Abstract

This project uses methods of machine learning to optimize a classic scheduling problem SMTT (Single Machine Total Tardiness Problem.) The problem is divided into a number of sub-problems and by taking the best solution that was found, we are sure to have solved the initial problem. To facilitate, we can memorize the solutions of sub-problems in the memory. When the sub-problem reappears, the solution which was memorized in the memory will be used to solve it. The goal is to predict whether a solution to a sub-problem in SMTT will be used to solve other sub-problems. The project can be considered as a sequence classification problem.

Keywords

SMTT, *Memorisation*, *Sequence*, *Machine Learning*, *LSTM*, *Tensorflow*, *Python*, \LaTeX

Tuteurs académiques

Lei SHANG

Vincent T'KINDT

Maxime MARTINEAU

Étudiant

Zhicong LIU (DI5)