Université de Montréal

Étude de modèles neuronaux de questions-réponses

par

Jean Archambault

Département d'informatique et de recherche opérationnelle

Faculté des arts et des sciences

Mémoire présenté à la Faculté des études supérieures

en vue de l'obtention du grade de Maître ès sciences (M.Sc.) en informatique

Août 2021

Université de Montréal

Faculté des arts et des sciences

Ce mémoire intitulé

Étude de modèles neuronaux de questions-réponses

présenté par

Jean Archambault

A été évalué par un jury composé des personnes suivantes :

Bang Liu

_____

(Président-rapporteur)

Philippe Langlais

_____

(Directeur de recherche)

Guy Lapalme

_____

(Membre du jury)

Mémoire accepté le :

Août 2021

_____

Résumé

Dans le domaine du traitement automatique du langage naturelle, la tâche question-réponse (Question-Answer (QA)) consistant à développer des systèmes générant une réponse plausible à une question posée en langage naturel par un utilisateur depuis une source d'information, demeure d'actualité. Elle présente de nombreuses applications pratiques dont la recherche affinée d'information sur le web. Aujourd'hui, suite à une requête, les moteurs de recherche actuels retournent des listes classées de documents mais ils ne fournissent pas de réponse à la question de l'utilisateur.

Depuis plus de cinquante ans, différentes approches et technologies ont été développées, qui ont mené à des avancées significatives en QA. Parmi celles-ci, les embeddings de mots, des vecteurs numériques représentant la signification des mots dans des contextes, des ensembles de données modèles et l'utilisation de réseaux neuronaux (RN) ont permis le développement de systèmes QA performants.

Dans ce contexte, ce mémoire a porté sur l'étude du modèle BIDAF (Bidirectional Attention Flow), un des systèmes QA à base de RN les plus performants au début de ces travaux, testé au moyen de SQuAD (Stanford Question Answering Dataset), un des benchmarks les plus populaires en QA. L'étude initiale de BIDAF a démontré un certain nombre de différences structurelles notables entre sa description littéraire et son implémentation. Différentes variantes de BIDAF ont donc été développées et testées auxquelles on a ajouté l'entraînement des embeddings de mots durant l'entraînement du modèle ainsi que la modulation cyclique du taux d'apprentissage.

Les modèles de structures similaires à la description littéraire de BIDAF avec entraînement des embeddings de mots et une version simplifiée ont démontré de meilleures performances, soit de 59.56% en exact match (*EM*) et 67.09% en *F1*, et *EM* = 60.20% et *F1* = 67.69%, respectivement. Cette performance a été améliorée par la modulation du taux d'apprentissage à *EM* = 61.42% et *F1* = 68.46%.

Mots clés : Tâche question-réponse, embeddings, réseaux neuronaux, BIDAF

Summary

In the field of natural language processing, the question-answer (QA) task involving the development of systems generating a plausible answer to a question asked in natural language by a user from an information source, remains a hot topic. It has many practical applications including fine-grained information retrieval on the web. Today, following a query, current search engines return lists of classified documents but they do not provide an answer to the user's question.

Since more than fifty years, a number of approaches and technologies have been developed which have led to significant advances in QA. Among these, word embeddings, numerical vectors representing the meaning of words in contexts, the development of model or benchmark datasets and the use of neural networks (NN) have enabled the development of efficient QA systems.

In this context, this thesis focused on the study of the BIDAF (Bidirectional Attention Flow) model, one of the most efficient NN-based QA systems at the start of this work, tested using SQuAD (Stanford Question Answering Dataset), one of the most popular benchmarks in QA. BIDAF's initial study showed a number of notable structural differences between its literary description and its implementation. Different variants of BIDAF were therefore developed and tested to which were added word embedding training during model training as well as cyclic modulation of the learning rate.

The models of structures similar to the literary description of BIDAF with training of word embeddings and a simplified version showed better performances, i.e. 59.56% in exact match (*EM*) and 67.09% in *F1*, and *EM* = 60.20% and *F1* = 67.69 %, respectively. This performance was improved by modulating the learning rate to *EM* = 61.42% and *F1* = 68.46%.

Key words : Question-answer task, embeddings, neural networks, BIDAF

Table des matières

Liste des tableaux

Liste des figures

Liste des sigles

n:	Pour l'ensemble de données selQA, nombre de phrases d'un ensemble de phrases candidates qui répondent à une question
id :	Pour l'ensemble SQuAD, identifiant unique de chaque paire question-réponse
N_{dfc} :	Nombre de fois que le système prédit, comme réponse, un segment de texte dans le contexte qui correspond exactement au segment de texte de la bonne réponse associée à la question
N_q :	Nombre de questions de l'ensemble de données
N_{mi} :	Nombre de mots identiques entre ces deux segments de textes, soit d'une réponse exacte et d'une réponse prédite
$N_{mi\text{-}max}$:	Valeur maximale de N_{mi} parmi les 5 bonnes réponses possibles de l'ensemble de développement
$prec$:	Précision de prédiction calculée selon l'équation 14
rec:	Recall calculé selon l'équation 15
F1 :	Taux du nombre de mots partagés entre les réponses prédites et les bonnes réponses aux questions calculé selon l'équation 16
L_p :	Longueur, en nombre de tokens, du segment de texte prédit et de la réponse exacte
L_{re} :	Longueur, en nombre de tokens, du segment de la réponse exacte
$\sum_{i=1}^{N_q}$:	Somme de i=1 à i=N_q
EM :	Taux de correspondance exacte entre les réponses prédites et les bonnes réponses aux questions calculé selon l'équation 13
Nx :	Nombre de couches consécutives identiques interconnectées, chacune composée d'une sous-couche d'auto-attention à multiples têtes d'une unité Transformer selon la Figure 2
L :	Nombre de couches ou blocs Transformer du Modèle BERT (Tableau 3)
H :	Dimension cachée, longueur d'embedding d'un mot au Modèle BERT (Tableau 3)

A :	Nombre de têtes d'auto-attention du Modèle BERT (Tableau 3)
S :	Vecteurs de paramètres appris à l'ajustement du Modèle BERT appliqué à SQuAD (Figure 4)
$\in$:	Sigle représentant l'appartenance
$\mathbb{R}$:	Sigle représentant un ensemble de nombres réels
$\mathbb{R}^H$:	Vecteur de nombre réels de dimension *H*
H :	Pour le Modèle BERT appliqué à SQuAD (Figure 4), longueur maximale du paragraphe
E:	Vecteurs de paramètres appris à l'ajustement du Modèle BERT appliqué à SQuAD (Figure 4)
T_i:	Vecteur caché final provenant de BERT pour le mot *i* du paragraphe
q2c$_i$:	Vecteur d'auto-attention du Modèle BIDAF de la question au contexte pour la paire contexte-question *i*
c2q$_i$:	Vecteur d'auto-attention du Modèle BIDAF du contexte à la question pour la paire contexte-question *i*
$\{x_1, \dots x_T\}$	Mots tokénisés du contexte
$\{q_1, , \dots q_J\}$	Mots tokénisés de la question
y :	Sortie du réseau autoroute selon l'équation 1 selon BIDAF
x :	Entrée du réseau autoroute selon l'équation 1 selon BIDAF
MLP_1 :	Multilayer Perceptron 1 de l'équation 1 selon BIDAF
$\overline{MLP_2}$:	Multilayer Perceptron 2 de l'équation 1 selon BIDAF
sigmoid :	Fonction d'activation de type sigmoïde à l'équation 1 selon BIDAF
relu :	Fonction d'activation de type relu à l'équation 1 selon BIDAF
$X \in \mathbb{R}^{T \times d}$:	Représentations finales du contexte de dimension Txd selon BIDAF
$Q \in \mathbb{R}^{j \times d}$:	Représentations finales de la question de dimension jxd selon BIDAF
d :	Dimension cachée = 100 selon BIDAF
T :	Longueur en mots du contexte
j :	Longueur en mots de la question
$H \in \mathbb{R}^{T \times 2d}$:	Représentation contextuelle des vecteurs des mots du contexte *X* selon BIDAF

$U \in \mathbb{R}^{j \times 2d}$: Représentation contextuelle des vecteurs des mots de la question Q selon BIDAF

$S \in \mathbb{R}^{T \times J}$: Matrice de similarité partagée entre les embeddings contextuels du contexte et de la question selon BIDAF et l'équation 2

α : Paramètre appris de l'équation 2 selon BIDAF

$w_{(S)}^{\mathrm{T}} \in \mathbb{R}^{6d}$: Vecteur de poids appris de l'équation 3 selon BIDAF

$a_t \in \mathbb{R}^{J}$: Poids d'attention des mots de la question pour le t^e mot du contexte calculés par l'équation 4 selon BIDAF

$\widetilde{U} \in \mathbb{R}^{2d \times T}$: C2Q calculé par l'équation 5 selon BIDAF

$b \in \mathbb{R}^{T}$: Poids des mots du contexte pour Q2C calculés par l'équation 6 selon BIDAF

$\widetilde{H} \in \mathbb{R}^{2d \times T}$: Q2C calculé par l'équation 7 selon BIDAF

G : Résultat de la combinaison des embeddings contextuels du contexte H et les vecteurs d'attention C2Q et Q2C par les équations 8 et 9 selon BIDAF

$\beta()$: Fonction vecteur entraînée qui fusionne les trois vecteurs d'entrée H, C2Q et Q2C pour calculer G par l'équation 9 selon BIDAF

$M \in \mathbb{R}^{2d}$: Sortie de la couche de modélisation de BIDAF qui utilise deux RNR consécutifs de type BILSTM, chacun de dimension cachée = d avec concaténation interne, le tout depuis G obtenu par l'équation 9

p^1: Distribution des probabilités que la position de chaque mot du contexte soit le début de la réponse à la question calculée par l'équation 10 selon BIDAF

$w_{(p^1)}^{\mathrm{T}} \in \mathbb{R}^{10d}$: Poids appris au MLP appliqué à la concaténation de G et M selon l'équation 10 et BIDAF

$M^2 \in \mathbb{R}^{2d}$: Sortie d'un RNR de type BILSTM de dimension cachée = d avec concaténation interne appliqué à M pour le calcul de p^2

p^2: Distribution des probabilités que la position de chaque mot du contexte soit la fin de la réponse à la question calculée par l'équation 11 selon BIDAF

$w_{(p^2)}^{\mathrm{T}} \in \mathbb{R}^{10d}$	Poids appris au MLP appliqué à la concaténation de G et M^2 selon l'équation 11 et BIDAF
k:	Position de début de la réponse dans le contexte
l:	Position de fin de la réponse dans le contexte
$\mathcal{L}(\theta)$:	Perte d'entraînement du Modèle calculé selon l'équation 12
θ:	Poids appris du modèle
N:	Nombre d'exemples de l'ensemble de données
y_i^1:	Vrai indice de début de la réponse dans le contexte de l'exemple i
y_j^2:	Vrai indice de fin de la réponse dans le contexte de l'exemple i
p_k:	k^e valeur du vecteur p
(x, y):	Pairs de mots utilisé pour pair2vec [37]
c:	Mots du contexte pour [37]
$c\_ew \in \mathbb{R}^{n,mcl,100}$	Cube de données avec embeddings de mots pour les contextes avec *n* = nombre d'exemples
$q\_ew \in \mathbb{R}^{n,mq,100}$	Cube de données avec embeddings de mots pour les questions avec *n* = nombre d'exemples
mcl :	Longueur maximale en mots des contextes
mq :	Longueur maximale en mots des questions
$maxlenC$:	Longueur maximale en termes de caractères des contextes calculée selon l'équation 17
$maxlenQ$:	Longueurs maximales en termes de caractères des questions calculée selon l'équation 18
$C^{n,maxlenC}$:	Matrice de caractères des contextes
$Q^{n,maxlenQ}$:	Matrice de caractères des questions
$vocCA^{69,8}$:	Matrice de poids aléatoires des embeddings indexés d'un vocabulaire de caractères
$c\_ewCH \in \mathbb{R}^{n,maxlenC,8}$	Cube de données avec embeddings de caractères pour les contextes avec *n* = nombre d'exemples
$q\_ewCH \in \mathbb{R}^{n,maxlenQ,8}$.	Cube de données avec embeddings de caractères pour les questions avec *n* = nombre d'exemples

$c\_ewCHF \in \mathbb{R}^{n,mcl,100}$ Cube de données avec embeddings de mots à base de caractères pour les contextes avec *n* = nombre d'exemples

$q\_ewCHF \in \mathbb{R}^{n,mq,100}$. Cube de données avec embeddings de mots à base de caractères pour les questions avec *n* = nombre d'exemples

qa : Représentation à la sortie de la couche d'embeddings contextuels des questions selon BIDAF

ca : Représentation à la sortie de la couche d'embeddings contextuels des contextes selon BIDAF

ceEX : Tenseur expansé de dimension (n,300,30,200) des embeddings contextuels des contextes ca (Figure 12)

qeEX : Tenseur expansé de dimension (n,300,30,200) des embeddings contextuels des questions qa (Figure 12)

cqM : Résultat de la multiplication de ceEX par qeEX (Figure 12)

cqConcat : Résultat de la concaténation de ceEX, qeEX et cqM (Figure 12)

S0 : Sortie du MLP appliqué à cqConcat (Figure 12)

Sbar : Sortie du softmax appliqué à S, la matrice de similarité (Figure 12)

SbE : Expansion à un tenseur de dimension (n,300,30,1) de Sbar (Figure 12)

c2q0 : Résultat de la multiplication de SbE avec qeEX (Figure 12)

c2q : Calcul final de C2Q (Figure 12)

ecC2q : Résultat de la multiplication de ca avec c2q (Figure 12)

b0 : Résultat d'un max sur l'axe 2 de S, la matrice de similarité (Figure 12)

b1 : Résultat de l'application d'un softmax sur b0 (Figure 12)

b2 : Résultat de l'expansion et la répétition de b1 (Figure 12)

b3 : Résultat de l'expansion de b2 sur l'axe 2 (Figure 12)

q2c0 : Résultat de la multiplication de b3 par ceEX (Figure 12)

q2c : Calcul final de Q2c (Figure 12)

ecQ2c : Résultat de la multiplication de ca, les embeddings contextuels des contextes par q2c (Figure 12)

h1 : Dimension de la couche cachée des 2 RNR de type BILSTM produisant le tenseur M depuis G (Figure 13)

g0 : Résultat du premier BILSTM appliqué à G (Figures 13 et 14)

G_M : Résultat de la concaténation de G et M (Figures 13 et 14)

G_MDO : Résultat de l'application d'un dropout de 0.2 à G_M (Figures 13 et 14)

G_M1 : Résultat de l'application d'un MLP à G_MDO (Figures 13 et 14)

G_M1S : Redimensionnement de G_M1 (Figures 13 et 14)

P_{yS} : Résultat de l'application d'un softmax à G_M1S pour donner la distribution de probabilités sur les mots des contextes des positions de début des réponses (Figures 13 et 14)

M2 : Résultat du BILSTM appliqué à M (Figure 13)

G_M2 : Résultat de la concaténation de G et M2 (Figures 13 et 14)

G_M2DO : Résultat de l'application d'un dropout de 0.2 à G_M2 (Figures 13 et 14)

G_M22 : Résultat de l'application d'un MLP à G_M2DO (Figures 13 et 14)

G_M2S : Redimensionnement de G_M22 (Figures 13 et 14)

P_{yE} : Résultat de l'application d'un softmax à G_M2S pour donner la distribution de probabilités sur les mots des contextes des positions de fin des réponses (Figures 13 et 14)

Span(k,l) Pour chaque exemple, prédiction des positions de début (k) et de fin (l) de la réponse dans le contexte par un max appliqué au produit de P_{yS} par P_{yE} (Figures 13 et 14)

ySE : Résultat de l'expansion de yS (Figure 14)

ySES : Résultat de l'application d'un softmax sur ySE (Figure 14)

ySES_M : Résultat de la multiplication de ySES par M (Figure 14)

a1i : Résultat d'une somme sur l'axe 2 de ySES_M (Figure 14)

a2i : Répétition et expansion de a1i sur l'axe 2 (Figure 14)

MXa2i : Résultat de la multiplication de M par a2i (Figure 14)

GMam : Résultat de la concaténation de G,M,a2i et MXa2i (Figure 14)

M2 : Résultat de l'application d'un BILSTM sur GMam (Figure 14)

TA :	Taux d'apprentissage
base_lr :	Valeur minimale de modulation du TA
max_lr :	Valeur maximale de modulation du TA
mode ou $scale\_fn$:	Fonction de modulation du TA possible, soit triangulaire, triangulaire dégressive (équation 23) et exponentielle (équation 24)
step_size ou TP:	Taille de pas de modulation du TA, soit le nombre d'itérations (ou de mini-batchs) d'un demi cycle de modulation complet entre les valeurs minimale et maximale du TA spécifiées
scale_mode :	{'cycle','iteration'}) détermine si la fonction de modulation est calculée en fonction du nombre de cycles ou d'itérations selon les équations 21 (cycle) et 22 (itération) selon la fonction de modulation utilisée (triangulaires : itération et exponentielle : cycle)
TA_t:	TA à l'itération (ou mini-batch) *t* calculé selon l'équation 19 ou 20
x:	Paramètre des équations 19, 20, 23 et 24 calculé selon l'équation 22
$cycle$:	Paramètre des équations 19 et 22 calculé selon l'équation 21
$gamma$:	Paramètre de l'équation 24 nécessaire au calcul de la fonction de modulation du TA exponentielle
trainable:	Paramètre de la couche d'embeddings de Keras indiquant si les embeddings sont entraînés (=*True*) ou non (=*False*) durant l'entraînement du modèle

Liste des abréviations

5ySE :	Précisions de prédictions par comparaison aux 5 réponses possibles de l'ensemble de développement des prédictions simultanément correctes de yS et yE
AI :	Artificial Intelligence
API :	Application Programming Interface
AR :	Auto régressif (XLNet)
ASKMSR :	Système QA à étapes multiples [9]
BASEBALL :	Premier système QA publié en 1961 [2]
BC :	Base de connaissances
BERT :	Bidirectional Encoder Representations from Transformers: système QA utilisant des réseaux neuronaux de type Transformer [20]
BIDAF :	Bidirectional Attention Flow : système QA utilisant des réseaux neuronaux de type BILSTM [19]
BIGRU :	Réseau neuronal qui comprend deux unités GRU, l'une prenant l'entrée vers l'avant et l'autre vers l'arrière
BILSTM :	Réseau neuronal qui comprend deux unités LSTM, l'une prenant l'entrée vers l'avant et l'autre vers l'arrière [48]
C2Q :	Attention bidirectionnelle contexte à question de BIDAF
ca :	Embeddings contextuels du contexte
Callbacks :	Objet Keras où peuvent être effectuées certaines actions et fonctions durant l'entraînement d'un modèle
CMU :	Carnagie Melon University
DO :	Dropout : paramètre de certaines couches de réseaux neuronaux qui permet d'ignorer aléatoirement une certaine portion d'unités neuronales d'une couche [54]
EE :	Entraînement des embeddings
ELMo :	Embeddings from Langage Models : représentation de type embeddings de mots qui dépend de toute la phrase d'où provient le mot [51]

EM :	Exact Match : taux de correspondance exacte entre les réponses prédites et les bonnes réponses aux questions
ema :	Exponential Moving Average
Entrainement yS :	Position du début de la réponse dans le contexte auquel est associée la question prédite par la méthode model.evaluate() pour l'ensemble d'entraînement
Entrainement yE :	Position de la fin de la réponse dans le contexte auquel est associée la question prédite par la méthode model.evaluate() pour l'ensemble d'entraînement
Ep :	Epoch
epoch :	Le nombre d'epochs est un hyperparamètre qui définit le nombre de fois que l'algorithme du modèle sera entraîné sur l'ensemble de données d'apprentissage. Un epoch signifie que chaque échantillon de l'ensemble de données d'apprentissage a eu la possibilité de mettre à jour les paramètres du modèle.
F1 :	Paramètre mesurant le taux des nombres de mots partagés entre les réponses prédites et les bonnes réponses aux questions
FAIR :	Facebook AI Research
G :	Giga (10^9)
GloVe :	Global Vectors for Word Representation: Type d'embeddings de mots [18]
GPU :	Graphic Processing Unit
GRU :	Gated Recurrent Unit : type de réseau neuronal récurrent [49]
IBM :	International Business Machines
json :	Format de fichier standard ouvert utilisé pour stocké des données
Keras :	Bibliothèque de logiciels open source qui fournit une interface Python (API) pour l'apprentissage machine et Tensorflow
KT-NET :	Knowledge and Text fusion NET : système QA utilisant BERT auquel on ajoute des bases de connaissances [29]
LSTM :	Long short-term memory : type de réseau neuronal récurrent [41]
LUNAR :	Système QA publié en 1973-1978 [3]

M:	Million
MLP:	Multilayer Perceptron: réseau de neurones artificiels de type feedforward
MS Marco :	MicroSoft MAchine Reading COmprehension dataset : ensemble de données utilisé pour l'entraînement de systèmes QA à domaine ouvert [24]
MySE :	Précisions de prédictions selon la multiplication des matrices de probabilités de yS et yE pour des prédictions simultanément correctes de yS et yE pour l'ensemble d'entraînement et de développement
ND :	Non déterminé
NewsQA :	Ensemble de données utilisé pour l'entraînement de systèmes QA à domaine ouvert [23]
NLP :	Natural Langage Processing
NUDT :	National University of Defense Technology
pair2vec	Word pair vectors: modèle pré-entraîné d'embeddings représentant une relation entre deux mots [37]
PLM :	Permutation Langage Modeling : unité utilisé par XLNet
Pytorch :	Bibliothèque de logiciels open source basée sur la bibliothèque Torch principalement développée par Facebook pour l'apprentissage machine
Q2C :	Attention bidirectionnelle question à contexte de BIDAF
qa :	Embeddings contextuels de la question
QA :	Question Answering
QANet :	Question Answering network : système QA utilisant des réseaux neuronaux de type convolution [31]
qas :	Étiquette de l'ensemble de données SQuAD contenant des paires réponse-question
R-NET :	Système QA utilisant des réseaux neuronaux de type GRU [32,34]
RDA :	Renforcement dynamique d'apprentissage utilisé par le système QA RMR [33,36]

ReLU : Rectified Linear Unit : fonction d'activation de réseau neuronal dont la sortie égale l'entrée si cette dernière est positive sinon zéro.

RMR : Reinforced Mnemonic Reader : système QA utilisant des réseaux neuronaux de type BILSTM [33,36]

RN: Réseau neuronal

RNR : Réseau neuronal récurrent, soit avec une boucle de retour interne

SAN : Stochastic Answer Network : système QA utilisant des réseaux neuronaux de type BILSTM et GRU [30]

S : Matrice de similarité

SEDT : Structural Embedding of dependency Tree : embeddings de type syntactique structural [38]

SelQA : Selection based Question Answering : ensemble de données utilisé pour l'entraînement de systèmes QA à domaine ouvert [25]

SEMABC : Sans embeddings de mots à base de caractères

SGD : Stochastic gradient descent : optimiseur

SLQA : Semantic Learning for Question Answering : système QA utilisant un réseau d'attention hiérarchique [35]

Softmax : Fonction d'activation qui affecte des probabilités décimales à chaque classe d'un problème multi-classe de telle sorte qu'elles somment à 1.0.

SpanBERT : Version du système BERT utilisant des séquences de mots masquées [29]

SQuAD : Stanford Question Answering Dataset : ensemble de données utilisé pour l'entraînement de systèmes QA à domaine ouvert [21]

START : Premier système QA à domaine ouvert [5]

step-decay : Algorithme de modulation du taux d'apprentissage [56]

T2 : triangular2 : Mode de modulation cyclique du taux d'apprentissage selon [57]

TA : Taux d'apprentissage

TALN : Traitement automatique de langage naturel

Tensorflow : Bibliothèque de logiciels open source développée par Google pour l'apprentissage machine

Test 5yS : Position du début de la réponse dans le contexte auquel est associée la question prédite par comparaison aux 5 réponses possibles de l'ensemble de développement

Test 5yE : Position de la fin de la réponse dans le contexte auquel est associée la question prédite par comparaison aux 5 réponses possibles de l'ensemble de développement

Test MyS : Position du début de la réponse dans le contexte auquel est associée la question prédite selon la multiplication des matrices de probabilités de yS et yE pour l'ensemble d'entraînement et de développement

Test MyE : Position de la fin de la réponse dans le contexte auquel est associée la question prédite selon la multiplication des matrices de probabilités de yS et yE pour l'ensemble de développement

Test SyS : Position du début de la réponse dans le contexte auquel est associée la question prédite par la méthode model.evaluate() pour l'ensemble de développement

Test SyE : Position de la fin de la réponse dans le contexte auquel est associée la question prédite par la méthode model.evaluate() pour l'ensemble de développement

TP : Taille de pas

TPU: Tensor Processing Unit: unité physique de calcul de tenseurs

trainable : Paramètre de la couche d'embeddings de Keras permettant (= *True*) ou non (=*False*) l'entraînement des embeddings durant l'entraînement du modèle

TREC : Text REtrieval Conference

WikiQA : Ensemble de données utilisé pour l'entraînement de systèmes QA à domaine ouvert [22]

Word2vec : Outil plus générique d'entraînement et d'utilisation d'embeddings pré-entraînés [16,17]

XLNet : Système QA, modèle de langage utilisant des réseaux neuronaux de type Transformer comme BERT [26]

YodaQA : Système QA à étapes multiples [10]

yE : Position de la fin de la réponse dans le contexte auquel est associée la question

yS : Position du début de la réponse dans le contexte auquel est associée la question

Remerciements

En premier lieu, j'aimerais remercier mon directeur de recherche, le Professeur Philippe Langlais, pour son aide et sa patience dans la réalisation de ces travaux et de ce mémoire. J'aimerais également remercier mon épouse, Mme Louise Biron, et ma fille, Li-Anne Archambault, pour leur support et leur patience constants dans une autre de mes aventures un peu spéciales. Finalement, je voudrais remercier les deux autres membres du jury, les Professeurs Bang Liu et Guy Lapalme, pour le temps et les efforts qu'ils ont consacrés à lire et corriger ce mémoire.

Chapitre 1 Introduction

Dans le domaine du traitement automatique du langage naturel (TALN), la tâche question-réponse (Question Answering (QA)) consiste à développer un système automatique qui produit une réponse plausible à une question posée en langage naturel par un utilisateur depuis une source d'information plus ou moins spécifique [1].

Cette tâche est caractérisée par cinq éléments principaux, soit :

1. Le genre de question posée, par exemple
   a. Une question factuelle qui commence typiquement par une expression interrogative ('qu'est-ce, quoi, quand, qui, où, combien' etc.) et qui requiert généralement comme réponse un fait relativement précis sous la forme d'un court texte dont l'exactitude est facile à mesurer, ou
   b. Une question plus générique ou qualitative, par exemple qui débute par 'comment', qui requiert une réponse plus nuancée pas nécessairement factuelle dont l'exactitude est difficile à mesurer;
2. La façon dont est produite la réponse, ainsi on qualifie
   a. Un système QA d'extractif lorsqu'il est conçu pour choisir un segment de texte, un mot ou une entité d'un texte comme réponse, ou
   b. Un système QA de génératif s'il réécrit la réponse afin qu'elle ait du sens sans le contexte de la question et/ou du texte dont elle provient;
3. Le genre de source d'information utilisée pour trouver la réponse, soit structurée, comme les bases de connaissances (BC), semi-structurées, comme les listes et les tableaux d'information, et non-structurée comme les textes en langage naturel;
4. Le domaine qu'elle cible, soit
   a. À domaine fermé lorsque le système s'adresse à un domaine particulier (la loi, la médecine etc.) et qu'il peut y exploiter des connaissances spécifiques; dans ce cas, généralement, le nombre de types de questions et les textes d'où peuvent être extraites les réponses sont limités, ou

b. À domaine ouvert lorsque les questions posées visent des domaines larges et que le système réfère à des sources d'information larges et ouvertes comme internet ou Wikipedia;

5. La méthodologie, l'approche utilisée pour trouver une réponse, soit
    a. Basée sur la recherche d'information qui retourne principalement comme réponse à la question un segment pertinent extrait d'un texte,
    b. Basée sur une approche de TALN dont l'objectif est d'extraire des chaines de mots candidates du contexte et de les ordonner par correspondance sémantique avec la question, et
    c. Basée sur l'approche BC qui élabore une représentation de la question et la transforme en un énoncé de prédicat pour le graphe de connaissances de la base.

La tâche QA présente de nombreuses applications pratiques dont la recherche affinée d'information sur le web. Aujourd'hui, on peine à naviguer dans la quantité d'information disponible en ligne. Suite à une requête quelconque, les moteurs de recherche actuels retournent des listes classées de documents mais ils ne fournissent pas de réponse à la question de l'utilisateur. On a besoin de systèmes de type QA auxquels un utilisateur pose une question en langage naturel et qui lui livrent rapidement une réponse concise avec un contexte pour valider cette réponse.

1.1 Bref historique de l'étude des systèmes QA

L'étude des systèmes QA remonte à environ 60 ans et accompagne l'apparition du concept d'intelligence artificielle. Depuis, les technologies reliées au QA ont constamment progressé dans le domaine du TALN. Les premiers travaux portaient sur la mise au point de systèmes de règles syntactiques élaborées manuellement pour répondre à des questions simples exprimées en langage naturel à partir d'un ensemble de faits tabulés et codés, le tout, en autres, à cause des ressources computationnelles limitées dans les années 60 et 70.

Ainsi BASEBALL [2], un premier système QA fermé publié en 1961, permettait de répondre à des questions factuelles simples posées en langage naturel sur le baseball de la ligue américaine durant une saison. Ce système déterminait la structure syntaxique de la question, convertissait ses composantes en paires ‘attribut = valeur’ puis tentait d’identifier le fait manquant dans sa BC comme réponse à la question. BASEBALL était limité à un domaine et à une base de connaissances structurées.

Un autre système QA marquant fut LUNAR [3] publié en 1973-1978 conçu pour aider les géologues lors de l’analyse d’échantillons lunaires rapportés par les missions Apollo. Encore ici, le système déterminait la structure syntaxique de la question posée en langage naturel dont il précisait le sens par des règles de dérivation sémantique. Par la suite, avec ces informations, il parcourait sa base de données de 13,000 informations pour identifier une réponse compatible. Il atteignait une performance de 78% de précision à prédire une bonne réponse sur un ensemble de questions test.

Ces deux systèmes ainsi que d’autres [4] furent les précurseurs du développement, vers les années 2000, de systèmes élaborés dits à BC qui collectaient, structuraient et emmagasinaient une foule d’information (des connaissances) dans des domaines spécifiques. On leur ajoutait des algorithmes de recherche d’information ajustés à la structure des connaissances de la BC interrogée pour en faire des systèmes QA. Parmi ceux-ci, on trouve les systèmes experts, des systèmes informatiques tentant d’imiter le processus de décision d’experts humains dans un domaine particulier. Ceux-ci étaient généralement conçus pour solutionner des problèmes complexes par raisonnement en parcourant une BC spécifique principalement structurée sous la forme de règles ‘si…. alors…’. Ce genre de système comprend un engin d’inférence, qui applique des règles à des faits connus pour en déduire de nouveaux, et une BC qui contient des faits et des règles.

Cependant, tous ces systèmes de QA, malgré leur complexité croissante, demeuraient à domaine fermé. Or, en 1993, apparait START [5] considéré comme le premier système de QA à domaine ouvert disponible en ligne. START utilise le modèle de données *objet-*

propriété-valeur estimé capable de capturer le contenu sémantique d'une multitude de questions réelles posées par des utilisateurs. START utilise ce modèle pour 1) comprendre et traduire la question posée par l'utilisateur en une requête *objet-propriété* puis 2) trouver la *valeur* associée. Sa composante Omnibase, une base de données virtuelle, extrait de l'internet les sources d'information reliées au sujet de la requête, transforme ces informations structurées, semi-structurées et non structurées en triplets *objet-propriété-valeur* et y cherche la *valeur* associée à la requête de départ pour répondre à la question de l'utilisateur. En 2002, suite à des millions de questions posées à START, sa performance atteignait 71% à trouver une bonne réponse à une question posée. START est toujours disponible pour consultation à http://start.csail.mit.edu/index.php.

Au début des années 2000, entre autres aux 8^{e} compétitions TREC (Text REtrieval Conference) [6] qui présentaient un premier volet en QA, la recherche sur les systèmes QA à domaine ouvert devient populaire. Ainsi, avec le développement de BC structurées plus génériques comme Freebase [7], une base de connaissances sous la forme de triplets *sujet-prédicat-objet* contenant aujourd'hui 1.9G (G : 10^9) unités et disponibles à https://developers.google.com/freebase, un certain nombre de systèmes de QA performants comme WebQuestions [8] ont été développés pour répondre à des questions simples. WebQuestions projette les questions vers des réponses en générant des prédicats logiques par alignement des mots clés de la question avec les entités de Freebase et d'un corpus de textes puis en affinant cet ensemble au moyen de prédicats compatibles proches entre eux par une opération de jointure. WebQuestions atteignait 62% de précision à prédire une bonne réponse à une question posée. Ces systèmes, cependant, sont limités à l'ontologie, la forme (type, classement, relations etc.) sous laquelle les connaissances de la BC sont structurées.

Afin d'augmenter leur portée, d'autres systèmes de QA à étapes multiples ont été développés utilisant à la fois des corpus de textes non structurés et des BC structurées. Ainsi ASKMSR [9] reformule la question sous plusieurs formes simples qu'il transforme en requêtes soumises à un engin de recherche d'information pour identifier des textes

pertinents. Les sommaires des documents ainsi collectés sont extraits pour leurs uni-, bi- et trigrammes qui sont filtrés en fonction des reformulations de la question pour donner une réponse plus probable. Ce système atteignait 61% de précision. De même YodaQA [10] comporte des étapes 1) d'analyse de question, soit l'extraction de ses caractéristiques principales (mots clés, objet de la question etc.), 2) de recherche de réponses candidates selon ces caractéristiques dans la version anglaise de wikimedia et dans les BC structurées DBpedia, Freebase et WordNet, 3) d'analyse de ces réponses candidates, 4) de fusion et de pointage des meilleures réponses candidates en fonction de la question et 5) d'identification de la réponse la plus probable. La performance de ce système était de 34% de précision que la première réponse candidate soit la bonne réponse à la question.

Le point culminant de cette approche fut le succès du système WATSON de IBM [12] qui gagna le jeu télévisé Jeopardy! en 2011. Ce système QA complexe utilisait une approche hybride de technologies de recherche d'information, de TALN et de BC.

Toutes ces approches, qu'on pourrait qualifier de traditionnelles, étaient basées sur l'analyse syntaxique, logique, modélisée etc. de la question, la collection d'éléments de BC et de textes non structurés pertinents aux caractéristiques de la question et l'identification de réponses candidates selon différentes techniques d'alignement et d'affinage.

Évidemment, l'essentiel de la tâche QA réside dans

1. La 'quantification' du sens de la question et de ses mots dans son contexte,
2. La recherche, à partir de cette 'quantification' de la question, d'un meilleur texte pouvant contenir la réponse à la question et
3. L'identification de la réponse dans ce texte toujours selon la signification quantifiée de la question.

Or la première faiblesse de toutes les approches précédentes était justement cette quantification du sens de la question très rudimentaire sinon inexistante. Vers les années

2000, un certain nombre de modèles de textes, comme le Latent Dirichlet Allocation [12] et le Latent Semantic Analysis [13], ont été développés pour estimer des représentations continues des mots. En 2003, Bengio et al. [14] présentaient une nouvelle approche de représentation vectorielle du sens des mots dans un contexte, appelée embeddings de mots, considérée comme un développement très important en TALN. Cette représentation des mots est apprise au moyen d'un modèle neuronal de langue entraîné sans supervision. D'un corpus de textes, un vocabulaire de mots est extrait, à chacun desquels on associe un vecteur numérique. Ces textes ainsi numérisés sont alimentés à un réseau neuronal, se terminant par une fonction softmax, entraîné sans supervision pour ses paramètres qui maximisent le log de la vraisemblance du corpus de textes. La sortie est composée des embeddings de chaque mot du vocabulaire, le vecteur numérique initial qui lui a été affecté, affiné par le corpus de textes utilisé et l'entraînement du réseau neuronal, ce qui donne un vecteur de caractéristiques ('features') distribuées apprises de chaque mot.

Par la suite, Collobert et Weston [15] en 2008-2011 démontraient la très grande utilité des embeddings de mots pré-entraînés pour plusieurs tâches de TALN. Puis en 2013, Mikolov et al. [16,17] présentaient word2vec, un outil plus générique d'entraînement et d'utilisation d'embeddings pré-entraînés alors que Pennington et al. [18] introduisaient GloVe, un autre genre d'embeddings de mots. Word2vec apprend les embeddings en reliant les mots à leur contexte sans tenir compte de leur fréquence alors que GloVe les calcule en tenant compte de la probabilité de leur cooccurrence dans le corpus de textes. Ces deux types d'embeddings contextuels de mots sont parmi les plus populaires utilisés aujourd'hui en TALN comparativement aux embeddings dits statiques, c'est-à-dire non contextuels.

Dans le contexte de la tâche QA, l'utilisation d'embeddings de mots représente donc une approche particulièrement intéressante pour

1. Numériser le sens des mots de la question, et donc de la question,

2. Puis identifier, parmi un corpus de textes également numérisés au moyen d'embeddings de mots ceux ayant une plus haute affinité à la question par de méthodes d'appariement numériques,
3. Puis analyser ces derniers au moyen d'autres méthodes d'appariement numérique afin d'y trouver une réponse plausible à la question.

En fait, cette approche est aujourd'hui particulièrement populaire pour le développement de systèmes de QA très performants à partir de modèles neuronaux d'apprentissage profond comme BIDAF [19] et BERT [20] qui extraient, depuis les représentations numériques de la question et du contexte combinées de multiples façons, des caractéristiques (features) plus ou moins interprétables qui permettent d'identifier dans ce même contexte une réponse probable à la question.

Un certain nombre d'ensembles de données (ou *benchmarks*) ont par ailleurs été élaborés comprenant des questions et des textes non structurés contenant les réponses à ces questions (présentés à la section 1.2) pour le développement de ces systèmes.

C'est dans ce contexte que se situent les travaux du présent mémoire où on a étudié le modèle neuronal BIDAF [19] et plusieurs variantes testés sur l'ensemble de données SQuAD [21], le tout utilisant des embeddings de mots et de caractères.

1.2 Ensembles de données – Benchmarks

Différents ensembles de données ont été développés et utilisés selon une variété de systèmes pour prédire une réponse à une question depuis un texte. Dans le cadre de ce mémoire, on cible les ensembles de données de forme paragraphe ou contexte, question et réponse dont les principaux sont présentés ci-après.

WikiQA (2015) [22]

Cet ensemble relativement limité comporte 3,047 questions échantillonnées des journaux de requêtes de l'engin de recherche Bing, chacune associée au sommaire d'une page de Wikipédia sur le sujet de la question. La réponse exacte est annotée par un humain.

Toutes les phrases du sommaire sont considérées comme des réponses possibles, chacune annotée si elle est ou non la réponse à la question. Certaines questions n'ont pas de réponses correctes. L'ensemble de données contient 29,258 phrases, dont 1,473 sont étiquetées comme réponse à leur question correspondante. L'objectif est de prédire la phrase qui contient la réponse à la question.

NewsQA (2016/8) [23]

Cet ensemble compte 119,633 paires de question-réponse générées par un humain basées sur un ensemble de 12,744 textes de nouvelles publiés par CNN choisis aléatoirement. Les réponses sont des segments continus de caractères de longueurs variables contenus dans le texte correspondant. Des questions peuvent ne pas avoir de réponse dans le texte correspondant. L'objectif est de prédire la réponse contenue dans le texte correspondant à la question.

MS Marco (2016/8) [24]

Cet ensemble comporte 1,010,916 questions (avec ou sans réponse) échantillonnées des journaux de requêtes de l'engin de recherche Bing, chacune avec une réponse de forme variable générée par un humain, dont 182,669 ont été réécrites par un humain. L'ensemble de données contient 8,841,823 textes extraits de 3,563,535 documents web obtenus de Bing. À chaque question, sont associés en moyenne 10 textes qui peuvent contenir une réponse à la question. L'objectif est de prédire si le(s) texte(s) qui lui sont associés peut/peuvent répondre à la question.

SelQA (2016) [25]

Cet ensemble relativement limité a été développé pour deux tâches de sélection de réponses à une question, soit 1) la sélection d'une phrase réponse et 2) le déclenchement d'une réponse. Cette dernière tâche est définie "comme la sélection d'un nombre quelconque (n> = 0) de phrases d'un ensemble de phrases candidates qui répondent à une question où l'ensemble de phrases candidates peut ou non contenir des phrases qui répondent à la question" [25].

SelQA compte 8,481 sections de texte obtenues de 486 publications sur les 10 sujets les plus populaires de Wikipedia, soit les arts, la nature, les aliments, les événements historiques, les films, la musique, la science, le sport, les voyages et la télévision. Ces sections ont été utilisées par un humain pour générer 7,904 questions de faits dont les réponses sont des phrases contenues dans les sections de texte correspondantes.

SQuAD (2016) [21]

Cet ensemble compte 23,215 paragraphes d'au moins 500 mots obtenus de 536 textes de Wikipédia couvrant une multitude de sujets. De chaque texte, jusqu'à 5 paires de question-réponse ont été générées par des humains. Chaque réponse, étiquetée manuellement, est un segment de phrase contenu dans le texte associé, indiqué par sa position numérique, en caractères et espaces, de début dans le paragraphe. L'ensemble SQuAD est divisé (sélection aléatoire) en un sous-ensemble d'entraînement de 18,896 paragraphes auxquels sont associés 87,599 paires de question-réponse et deux sous-ensembles de 2,067 paragraphes auxquels sont associés 10,600 paires de question-réponse pour les essais de développement et de test. Le sous-ensemble de test n'est pas publié puisqu'il sert au classement des différents systèmes soumis au site de SQuAD (voir ci-après). L'objectif est de prédire la position de début et de fin de la réponse à la question dans le paragraphe correspondant.

Puisque SQuAD s'est avéré être l'ensemble de données le plus populaire pour l'étude des systèmes de QA, nous avons concentrés nos efforts sur celui-ci (version 1.1), dont un exemple est présenté au Tableau 1. Cet exemple comporte :

1. Un texte ("context") continu d'un maximum de 651 mots composé de phrases de longueurs variables et
2. L'étiquette "qas" qui contient plusieurs paires "answers"-"question", chacune indiquant :
   a. Pour "answers" :
      i. La position de début d'une réponse en caractères et espaces dans le paragraphe ("answer_start"),
      ii. Le texte de cette réponse ("text"),

iii. L’identifiant ("id") unique de cette paire question-réponse et

b. Pour "question", la question associée à cette réponse.

La position de fin de la réponse est évidemment obtenue en ajoutant à la position de début la longueur (en caractères et espaces) de la réponse.

Tableau 1. Donnée typique contexte, questions et réponses de l’ensemble SQuAD 1.1


```
{"data":[{"paragraphs":[{"context":"Architecturally, the school has a Catholic character. Atop the Main Building's gold dome is a golden statue of the Virgin Mary. Immediately in front of the Main Building and facing it, is a copper statue of Christ with arms upraised with the legend \"Venite Ad Me Omnes\". Next to the Main Building is the Basilica of the Sacred Heart. Immediately behind the basilica is the Grotto, a Marian place of prayer and reflection. It is a replica of the grotto at Lourdes, France where the Virgin Mary reputedly appeared to Saint Bernadette Soubirous in 1858. At the end of the main drive (and in a direct line that connects through 3 statues and the Gold Dome), is a simple, modern stone statue of Mary.",
"qas":[{"answers":[{"answer_start":515,"text":"SaintBernadette Soubirous"}],
"id":"5733be284776f41900661182",
"question":"To whom did the Virgin Mary allegedly appear in 1858 in Lourdes France?"},
{"answers":[{"answer_start":187,"text":" a copper statue of Christ"}],
"id":"5733be284776f4190066117f",
"question":"What is in front of the Notre Dame Main Building?"},
{"answers":[{"answer_start":279,"text":"the Main Building"}],
"id":"5733be284776f41900661180",
"question":"The Basilica of the Sacred heart at Notre Dame is beside to which structure?"},
{"answers":[{"answer_start":381,"text":"a Marian place of prayer and reflection"}],"id":"5733be284776f41900661181",
"question":"What is the Grotto at Notre Dame?"},
{"answers":[{"answer_start":92,"text":"a golden statue of the Virgin Mary"}],"id":"5733be284776f4190066117e",
"question":"What sits on top of the Main Building at Notre Dame?"}]},
```

Les exemples de l'ensemble SQuAD 1.1 de développement et de test sont de la même forme. Cependant, ces exemples comportent jusqu'à cinq bonnes réponses possibles à une même question.

L'ensemble SQuAD se distingue comme un des ensembles les plus utilisés pour la tâche QA avec un tableau de classement de performance de modèles d'apprentissage profonds de 66 scores sur l'ensemble SQuAD 1.1 (https://rajpurkar.github.io/SQuAD-explorer) au 26 mars 2020. La taille de SQuAD est significative et donc se prête bien à des approches d'apprentissage profond. Au début de la rédaction de ce mémoire (mars 2020), les meilleures performances (en correspondance exacte ('exact match') sur SQuAD 1.1) rapportées sont de 82.30% et 88.95% pour l'humain et pour le réseau XLNet [26], respectivement. L'ensemble SQuAD 1.1 a été choisi au début de ce projet, en février 2017, pour les travaux présentés dans ce mémoire[1].

La suite de ce mémoire se présente comme suit. Le Chapitre 2 porte sur une revue des plus récents travaux publiés sur la tâche question-réponse depuis un texte, soit les modèles neuronaux les plus performants dans ce domaine sur l'ensemble de données choisi, SQuAD. Le Chapitre 3 résume la méthodologie utilisée pour réaliser les travaux de cette étude. Le Chapitre 4 présente et discute des principaux résultats obtenus lors de cette étude, soit les différents modèles testés selon leurs hyper-paramètres afin de maximiser leur généralisation. Finalement, le Chapitre 5 conclut ce mémoire.

[1] On notera que les travaux de ce mémoire ont été réalisés à mi-temps.

Chapitre 2 Modèles neuronaux développés en QA utilisant SQuAD

Ce chapitre porte sur une revue ciblée des plus récents modèles neuronaux performants développés pour la tâche de QA utilisant l'ensemble de données SQuAD. À la section 2.1, les systèmes simples élaborés par les développeurs de l'ensemble SQuAD [21] sont discutés. Par la suite, à la section 2.2, les systèmes neuronaux les plus performants sur SQuAD 1.1 sont présentés à l'exception du Modèle BIDAF [19] qui est discuté plus en détails à la section 2.3 puisqu'il est l'objet des présents travaux.

On notera que, dans le domaine QA, les principales métriques utilisées pour mesurer la performance d'un modèle à prédire les bonnes réponses aux questions posées par rapport à un contexte de référence sont la correspondance exacte ('exact match' (EM)) soit le taux de correspondance exacte entre les réponses prédites et les bonnes réponses aux questions, et le paramètre *F1* qui mesure le taux du nombre de mots partagés entre les réponses prédites et les bonnes réponses aux questions. Le calcul de EM et *F1* est présenté à la section 3.4.

2.1 Systèmes originaux utilisant SQuAD

Les développeurs de l'ensemble de données SQuAD, Rajpurkar et al. [21], ont publié leurs travaux en octobre 2016. Ils ont développé et entraîné un modèle de type régression logistique utilisant 180 million de traits (features), principalement de type lexical et d'arbres de dépendance, extraits des réponses, sur l'ensemble d'entrainement et l'ont testé sur les ensembles de test et de développement de SQuAD. Leurs meilleurs résultats sur ces deux derniers ensembles ont été de 40.0% et 40.4% en EM et de 51.0% et 51.0% en *F1*. Dans ce contexte, il est intéressant de souligner les résultats de Elbaz [27] qui, au moyen d'un modèle de régression logistique plus simple de 8,725 traits non lexicaux, obtient de meilleures performances, soit de 48.4% et 57.5% en EM et *F1*, respectivement, sur l'ensemble de développement. En comparaison, les résultats obtenus par des valideurs humains sur ces deux ensembles sont de 80.3% et 77.0% en EM et 90.5% et 86.8% en pointage *F1*.

2.2 Principaux modèles à réseaux neuronaux testés sur SQuAD 1.1

Le Tableau 2 présente, par rang et type de modèle, les réseaux neuronaux, ayant démontré les meilleures performances sur SQuAD 1.1 extraits du site de classement de cet ensemble de données.

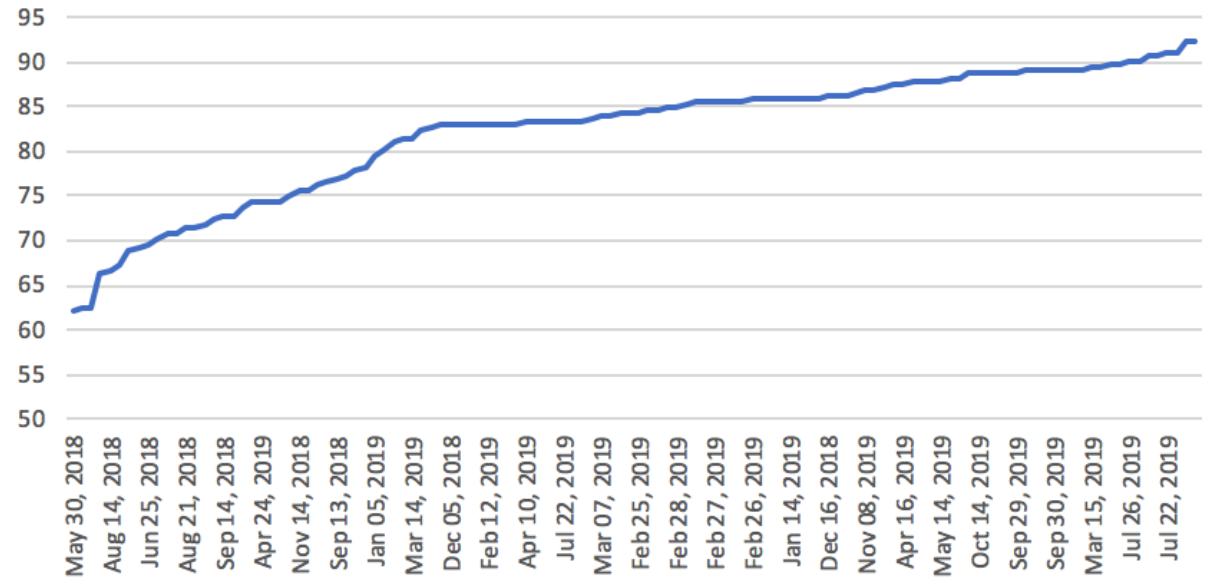
Tableau 2. Meilleurs modèles neuronaux testés sur SQuAD1.1 regroupés et publiés en date du 26 mars 2020

Rang (date: an/mois)	Modèle	Groupe auteur	EM (%)	*F1* (%)	Référence
(16/10)	Performance humaine	Stanford University	82.304	91.221	[21]
1 (19/05)	XLNet (modèle unique)	Google Brain & Carnagie Mellon Univ. (CMU)	**89.898**	**95.080**	[26]
3 (19/07)	SpanBERT (modèle unique)	Facebook AI Research (FAIR) & UW	88.839	94.635	[28]
6 (18/10)	BERT (modèle ensemble)	Google AI Language	87.433	93.160	[20]
9 (19/09)	KT-NET (modèle unique)	Baidu NLP	85.944	92.425	[29]
24 (17/12)	SAN (modèle ensemble)	Microsoft Business AI Solutions Team	79.608	86.496	[30]
11 (18/07)	QANet (modèle ensemble)	Google Brain & CMU	84.454	90.490	[31]
12 (18/07)	R-NET (modèle ensemble)	Microsoft Research Asia	84.003	90.147	[32]
16 (18/02)	Reinforced Mnemonic Reader + A2D (modèle ensemble)	Microsoft Research Asia & NUDT	82.849	88.764	[33,36]
16 (18/01)	R-NET+ (modèle ensemble)	Microsoft Research Asia	82.650	88.493	[34]
16 (18/01)	SLQA+ (modèle ensemble)	Alibaba iDST NLP	82.440	88.607	[35]
19 (17/11)	BIDAF + Self Attention + ELMo (ensemble)	Allen Institute for AI	81.003	87.432	[36]
46 (17/09)	BIDAF + Self Attention (modèle unique)		72.139	81.048	[37]
55 (17/09)	AllenNLP BIDAF (modèle unique)		67.618	77.151	
45 (17/02)	BIDAF (modèle ensemble)	Allen Institute for AI & University of Washington	73.639	81.931	[19]
54 (16/11)	BIDAF (modèle unique)		67.974	77.323	[19]
45 (17/04)	SEDT+BIDAF (modèle ensemble)	CMU	73.723	81.530	[38]

Par ailleurs, la Figure 1 illustre la performance, depuis mai 2018, des différents modèles testés sur SQuAD 1.1 selon Simon Hughes (http://simonhughes.blogspot.com/2019/11/

ai-reading-comprehension-scores.html). On observe qu'au début des travaux du présent mémoire, le modèle BIDAF (Bi-directional Attention Flow) démontrait les meilleures performances sur SQuAD 1.1. Le modèle BIDAF a donc été choisi comme sujet d'étude du présent mémoire.

A : Précision (EM) en fonction de la date de soumission du modèle



B : Tableau des données avec modèles du 20 mai 2018 au 24 avril 2019

Date	Model	Score
May 30, 2018	BiDAF-No-Answer (single model) University of Washington [modified by Stanford]	62.093
May 30, 2018	BiDAF + Self Attention (single model) Allen Institute for Artificial Intelligence [modified by Stanford]	62.305
Nov 27, 2018	Tree-LSTM + BiDAF + ELMo (single model) Carnegie Mellon University	62.341
May 30, 2018	BiDAF + Self Attention + ELMo (single model) Allen Institute for Artificial Intelligence [modified by Stanford]	66.251
Aug 14, 2018	eeAttNet (single model) BBD NLP Team	66.633
Jun 27, 2018	BSAE AddText (single model) reciTAL.ai	67.422
Sep 13, 2018	BiDAF++ (single model) UW and FAIR	68.866

Figure 1. Précisions de prédictions de différents modèles sur SQuAD 1.1 depuis mai 2018
(http://simon-hughes.blogspot.com/2019/11/ai-reading-comprehension-scores.html)

Les meilleurs modèles testés et référencés sur SQuAD 1.1 du Tableau 2 qui présentent, depuis 2018, des performances supérieures à BIDAF, soit XLNet, BERT, KT-NET, SAN, QANet, R-NET, Reinforced Mnemonic Reader et SQLA, sont brièvement discutés ci-après. À la section 2.3, on discutera plus en détails du modèle BIDAF.

1. BERT [20]

Le modèle BERT (Bidirectional Encoder Representations from Transformers) utilise une architecture récente (2017) de réseau neuronal (RN) appelée *Transformer* [39] qui est basée sur un mécanisme d'auto-attention ('self-attention') qu'on dit bien adapté à la compréhension de textes.

Une unité *Transformer* typique représentée à la Figure 2 comporte deux sections, soit l'encodage (partie de gauche) et le décodage. BERT n'utilise que la section encodage de l'unité *Transformer*.

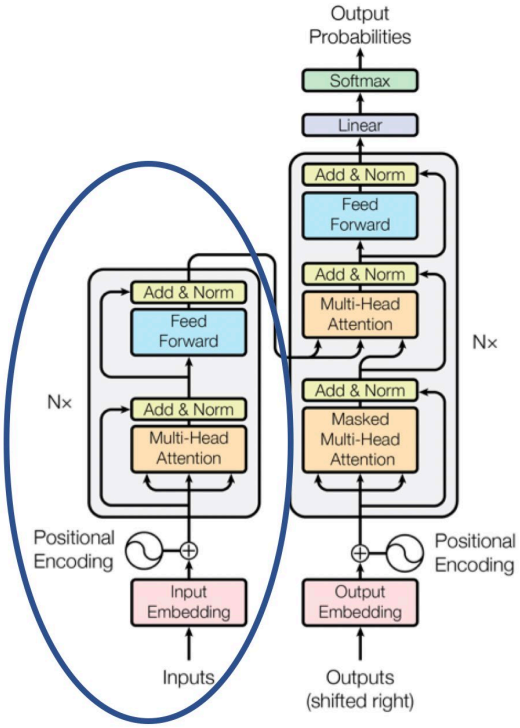


Figure 2. Structure d'une unité *Transformer* typique d'après Vaswami et al. [39]

Dans une unité *Transformer*, l'entrée du réseau est un texte de mots tokénisés qu'on numérise en embeddings et auquel on additionne un encodage de position du mot dans le texte afin de permettre au modèle d'utiliser l'ordre dans la séquence de mots. Cet encodage de position est de même dimension que celle de l'embedding du mot et additionné à ce dernier.

La partie encodage d'une unité *Transformer* comporte six (Figure 2 : Nx = 6) couches consécutives identiques interconnectées, chacune composée d'une sous-couche d'auto-attention à multiples têtes (multi-head) (Figure 3) et d'un réseau de neurones artificiels de type feedforward (multilayer perceptron (MLP)). Comme indiqué à la Figure 2, la sortie de chaque sous-couche d'auto-attention est additionnée à son entrée. Le résultat est normalisé [40] avant d'entrer à la sous-couche MLP dont la sortie est aussi additionnée à son entrée puis normalisée avant d'entrer dans la couche suivante d'encodage. Chaque sous-couche MLP comporte deux transformations linéaires séparées par une activation de type ReLU.

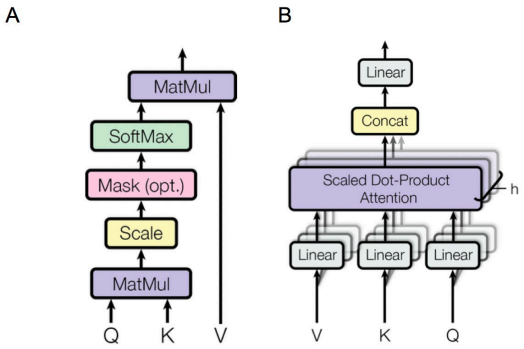


Figure 3. Calcul d'auto-attention: A: à produit scalaire simple, B: à multiples entrées

On notera que BERT utilise une unité *Transformer* plutôt que des RN récurrents (RNR) comme le 'long short-term memory' (LSTM) [41] couramment utilisé en modélisation neuronale en TALN, notamment dans le domaine QA. Un RNR de type LSTM [41], contrairement à un réseau neuronal MLP standard, possède des connexions de rétroaction. Il peut donc traiter à la fois des points de données uniques, mais également des séquences entières de données comme un texte de mots numérisés au moyen d'embeddings. Une unité LSTM commune est composée d'une cellule, d'une porte d'entrée, d'une porte de sortie et d'une porte d'oubli. La cellule se souvient des valeurs sur des intervalles de longueurs arbitraires et les trois portes régulent le flux d'information entrant et sortant de la cellule. Les réseaux LSTM peuvent ainsi classer, traiter et faire des prédictions basées sur des données de séries chronologiques, entre autres lorsqu'il y a des décalages de durée inconnus entre des événements importants d'une série chronologique.

L'unité *Transformer* possède au moins deux avantages importants pour la modélisation de textes par rapport aux RNR comme le LSTM. Premièrement, elle traite un segment de texte en parallèle, ce qui permet l'apprentissage de dépendances sur de plus longues distances, comparativement au traitement séquentiel réalisé par les RNR. Deuxièmement, elle n'implique que des multiplications de matrices, quelques couches neuronales entièrement connectées à action directe (MLP) et permet le calcul parallèle, ce qui est beaucoup plus rapide que les calculs séquentiels et récurrents des RNR qui souffrent en plus de problèmes de disparition et d'explosion de gradients [42].

BERT [20] est un modèle de langage pré-entrainé, c'est-à-dire qu'une version générique de BERT de compréhension de langage est pré-entrainée de manière non supervisée sur un large corpus de texte puis utilisée et ajustée pour différentes tâches de traitement de langage naturel, par exemple la tâche de QA dans le présent cas. Il est pré-entraîné au moyen de deux tâches non supervisées sur un corpus qui résulte de la concaténation des ensembles BooksCorpus (800 millions (M) de mots) [43] et des passages de textes de Wikipedia (2,500M de mots).

À l'utilisation de BERT pour la tâche QA sur SQuAD 1.1, tel qu'illustré à la Figure 4, les données d'embeddings d'entrée de la question (séquence A) et du paragraphe (séquence B) sont assemblées dans une seule séquence. Les seuls nouveaux paramètres appris durant l'ajustement de BERT pour cette tâche sont les vecteurs $S \in \mathbb{R}^H$ et $E \in \mathbb{R}^H$ où *H* est la longueur maximale du paragraphe. Sachant que le vecteur caché final provenant de BERT pour le mot *i* du paragraphe est $T_i \in \mathbb{R}^H$, alors la probabilité que le mot *i* soit le début de la réponse est obtenu par le produit scalaire de T_i et *S* suivi par une fonction softmax sur tous les mots du paragraphe. La même approche est utilisée pour la prédiction de la position de fin de la réponse dans le paragraphe. Le modèle est entrainé sur 3 epochs avec une taille de lots de 32.

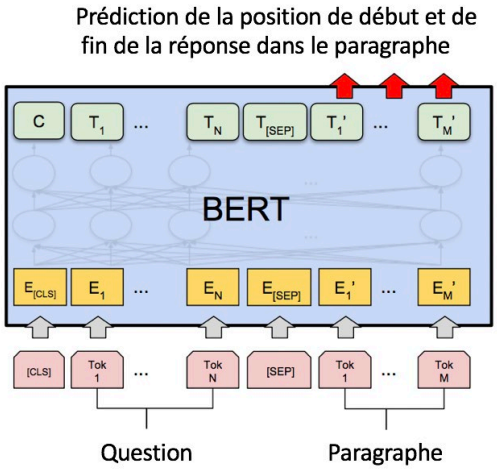


Figure 4. Application de BERT à la tâche QA sur SQuAD 1.1 d'après Devlin et al. [20]

BERT, au pré-entrainement, exige beaucoup de ressources (16 et 64 puces tensor processing units (TPU) et 4 jours de calculs), est lourd (110M et 340M de paramètres pour les versions de base et large, respectivement) et fait appel à beaucoup de ressources textuelles externes. À l'entraînement sur SQuAD 1.1, seuls les résultats obtenus avec BERT de base peuvent être reproduits au moyen d'une unité GPU de 12 à 16G de mémoire à cause de la lourdeur du modèle (https://github.com/google-research/bert/blob /master/README.md). Les performances sur SQuAD 1.1 sont impressionnantes : EM = 85.1% et *F1* = 91.8% pour le modèle de base et EM = 87.4% et *F1* = 93.1% pour le modèle large [20].

2. XLNet [26]

Tout comme BERT, XLNet est un modèle de langage qui utilise des unités *Transformer* et qui nécessite un pré-entraînement intensif avant son implémentation pour des tâches particulières. Plus spécifiquement, XLNet est un modèle autorégressif (AR) généralisé pour la compréhension de langage naturel où chaque mot dépend de tous les mots précédents. Il est généralisé parce qu'il capture le contexte de façon bidirectionnelle au moyen d'un mécanisme de modélisation par permutation de langage ('permutation language modeling' (PLM)) en entrainant un modèle AR sur toutes les permutations possibles de mots dans une phrase tout en conservant l'ordre des mots dans la séquence originale. XLNet maximise la vraisemblance logarithmique attendue sur toutes les

permutations possibles de la séquence de mots. Chaque position apprend alors à utiliser l'information contextuelle de toutes les positions, ce qui capture le contexte bidirectionnel.

Tout comme BERT, les corpus BooksCorpus [44] et la version anglaise de Wikipédia ont été utilisés pour pré-entrainer XLNet auxquels Giga [45], ClueWeb 2012-B [46] et Common Crawl [47] ont été ajoutés. Après l'utilisation d'heuristiques et de filtres, ce pré-entrainement est réalisé sur 32G mots, soit des ressources textuelles externes encore plus lourdes que pour BERT.

Les modèles XLNet de base et large ont les mêmes hyper paramètres d'architecture que BERT, ce qui donne des modèles de même grandeur. Le pré-entrainement de XLNet large requiert 512 TPU pour 500,000 étapes avec une taille de lot de 2,048 et 2.5 jours, ce qui est environ 5 fois plus intensif que BERT large.

Tout comme pour BERT, l'application de XLNet à SQuAD (2.0) présente de sérieuses difficultés de capacité de calcul et de mémoire. Les résultats présentés à la publication [26] ont été obtenus au moyen d'unités TPU qui ont plus de mémoire que les unités GPU. Il est donc très difficile de reproduire les résultats sur une unité GPU de 16G qui ne peut contenir qu'une séquence d'une longueur de 512 mots pour XLNet large. On doit alors limiter la longueur de séquence et la taille de lots par exemple à 256 mots pour des tailles de lot de 24 et 2 pour XLNet de base et large, respectivement. Les performances de XLNet sur SQuAD 1.1 sont impressionnantes : EM = 89.8% et *F1* = 95.0% pour le modèle large [26], ce qui surpasse la version large de BERT (EM = 87.4% et *F1* = 93.1%).

3. KT-NET [29]

L'objectif du modèle KT-NET (pour Knowledge and Text fusion NET) est d'améliorer les approches à modèles de langage, BERT dans ce cas-ci, avec l'ajout de connaissances extraites de BC structurées. En résumé, KT-NET utilise BERT large pré-entrainé et lui ajoute des ressources externes additionnelles, soit les BC WordNet et NELL, afin d'enrichir la représentation obtenue de BERT au moyen d'embeddings de connaissances associées aux mots du paragraphe et de la question. La performance de KT-NET sur

SQuAD 1.1 est de EM = 85.9% et *F1* = 92.4%, ce qui est légèrement inférieur aux résultats obtenus avec BERT malgré les nombreuses étapes ajoutées pour KT-NET. On peut donc supposer les difficultés pratiques des essais de KT-NET suite à la discussion précédente sur ce sujet à propos de BERT.

4. SAN [30]

Le modèle SAN (stochastic answer network) utilise l'approche plus traditionnelle des RNR de type LSTM [41] (BILSTM [48]) et 'gated recurrent unit' (GRU) [49].

Une unité BILSTM est un RNR qui comprend deux unités LSTM, l'une prenant l'entrée vers l'avant et l'autre vers l'arrière. La sortie d'une unité BILSTM peut être la multiplication, la somme, la concaténation ou la moyenne des sorties de chaque unité LSTM ou les sorties des deux unités LSTM sous la forme d'une liste. Une unité GRU est une variation d'une unité LSTM où deux portes ('update gate' et 'reset gate') sont utilisées plutôt que trois pour le LSTM et la couche cachée est simplifiée.

Tel qu'illustré à la Figure 5, le modèle SAN est composé de 5 couches consécutives, soit une couche d'encodage d'embeddings, une couche d'encodage contextuel, une couche d'attention, une couche de génération de mémoire et une couche de réponse.

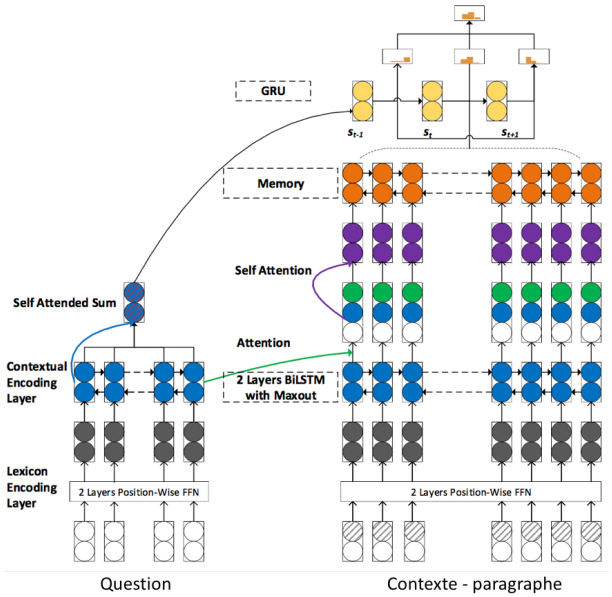


Figure 5. Architecture du modèle SAN d'après Liu et al. [30]

Les essais sur SQuAD1.1 du modèle SAN, avec une taille de lots de 32, l'optimiseur Adamax et un taux d'apprentissage de 0.002 diminué de moitié à tous les 10 epochs, donnent EM = 79.6% et *F1* = 86.4%, ce qui est évidemment moindre que les résultats obtenus au moyen des modèles précédents mais à une utilisation de ressources physiques très inférieure, soit un temps de calcul de 22 minutes/epoch sur un GPU Titan X.

5. QANet [31]

Le modèle QANet (question answering network) fait appel à des unités neuronales de type convolution [50] et d'auto-attention plutôt que des RNR, ce qui lui permet d'être beaucoup plus rapide (~4 à 13 fois) que ces derniers.

Un réseau neuronal convolutif est un réseau acyclique qui comporte trois couches successives, soit

- Une couche de convolution où les données d'entrée sont multipliées par un filtre, une matrice de poids partagés ajustés à l'entraînement, afin de les mapper à une matrice de caractéristiques,
- Une couche dite de détection où une activation non linéaire de type MLP est appliquée à cette matrice de caractéristiques et
- Une couche de type *pooling* où une fonction de sommaire statistique, par exemple de moyenne ou de valeur maximale, est appliquée à la sortie de la couche précédente pour rendre la représentation finale du réseau approximativement invariable à de petites variations de l'entrée.

Comme illustré à la Figure 6, ce réseau relativement complexe comporte six composantes principales, soit une couche d'embeddings, un bloc d'encodage d'embeddings, soit la partie B de la Figure 6 qui inclue un réseau convolutif, une couche d'attention contexte-question, une série de trois blocs d'encodage (partie B de la Figure 5) consécutifs et, comme réseau de sortie, deux séries parallèles de couches de

concaténation, de MLP avec activation linéaire et de softmax pour la prédiction de la position de début et de fin de la réponse, respectivement.

Les essais sur SQuAD1.1 du modèle QANet donnent EM = 84.4%% et *F1* = 90.4%, ce qui le place immédiatement après les trois modèles de langage discutés précédemment mais sans l'utilisation de ressources textuelles externes et en utilisant beaucoup moins de ressources physiques. En fait, malgré l'utilisation de 46 réseaux à convolution, les auteurs indiquent que QANet est 5 fois plus rapide que BIDAF [19], pour une performance en *F1* de 77.0% comparable à celle de BIDAF, ce dernier utilisant une série de RNR de type BILSTM [48] comme discuté à la section 2.3.

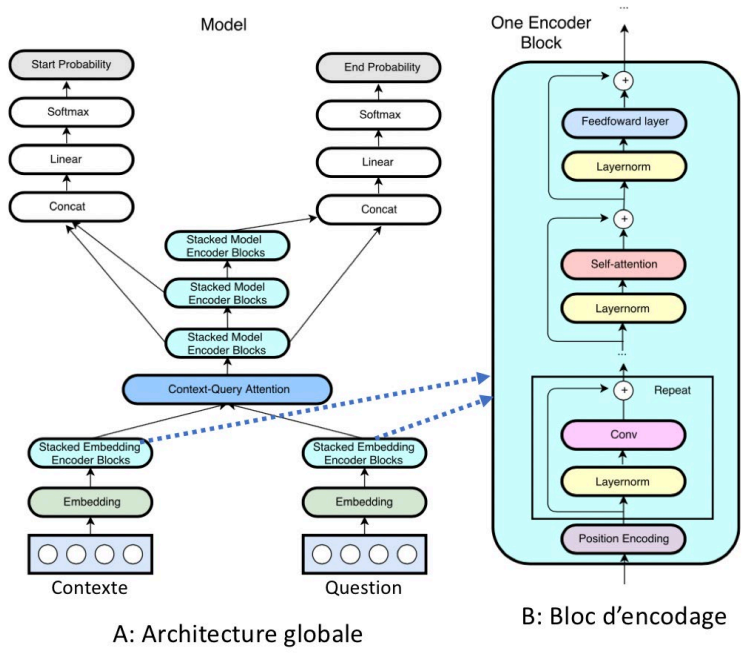


Figure 6. Architecture du modèle QANet d'après Yu et al. [31]

6. R-NET [32]

Le modèle R-NET utilise l'approche plus traditionnelle des RNR, et plus spécifiquement des unités GRU [49] qui sont une variante des réseaux LSTM. Tel qu'illustré à la Figure 7, le modèle R-NET comporte quatre composantes, soit un encodeur de type RNR pour construire la représentation des contextes et des questions séparément, une couche à RNR de type GRU pour représenter la correspondance entre la question et le contexte,

une couche d'auto-correspondance pour représenter l'information de l'ensemble du contexte et finalement une couche de prédiction des limites de la réponse basée sur un réseau de type pointeur.

Le concept d'un réseau pointeur implique que la représentation finale du réseau est utilisée pour pointer la position pertinente sur l'encodage de la séquence d'entrée. Pour ce faire, il utilise la représentation finale du contexte additionnée à sa propre sortie précédente qui est alimentée à un réseau MLP avec activation de type tangente hyperbolique, et le tout est normalisé par une fonction softmax afin d'estimer la position la plus probable du début et de la fin de la réponse dans le contexte.

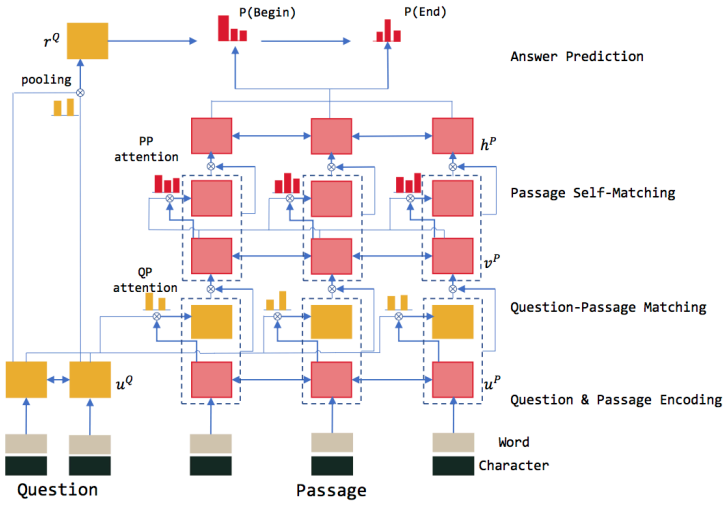


Figure 7. Architecture du modèle R-NET d'après [32]

Au moyen d'un ensemble de modèles (20 essais), ce modèle atteint EM = 84.0% et *F1* = 90.1%, ce qui le place à quasi égalité avec le modèle QANET et sous les autres modèles précédents.

7. Reinforced Mnemonic Reader [33,36]

Le modèle Reinforced Mnemonic Reader (RMR) présente un mécanisme différent de ré-attention qui mémorise temporairement les attentions passées qu'il utilise pour raffiner les attentions courantes dans une structure d'alignement à plusieurs étapes. Tel qu'illustré à la Figure 8, le modèle RMR comporte trois composantes principales, soit un encodeur qui construit des représentations contextuelles de la question et du contexte

séparément, un aligneur itératif qui calcule des alignements multiples entre la question et le contexte au moyen du mécanisme de ré-attention et un pointeur de réponse similaire à celui utilisé par le modèle R-NET qui prédit séquentiellement la chaîne de mots réponse dans le contexte.

Ce modèle utilise des unités RNR de type BILSTM [48] avec une couche cachée principale de dimension 100 et une méthode d'entraînement combinant l'approche standard de vraisemblance maximale et une approche de renforcement dynamique d'apprentissage (RDA) [33]. Ce modèle atteint EM = 82.8% et *F1* = 88.7%, ce qui est sous les modèles précédents.

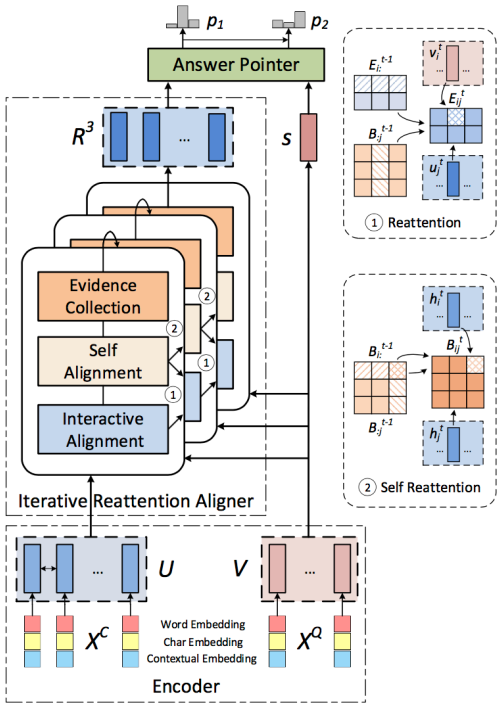


Figure 8. Architecture du modèle RMR d'après Hu et al. [33]

8.SLQA [35]

Le modèle SQLA (Semantic Learning for Question Answering) est décrit comme un réseau d'attention hiérarchique. Tel qu'illustré à la Figure 9, il comporte quatre couches principales, soit une couche d'encodage, une couche d'attention et de fusion hiérarchique, une couche de correspondance et une couche de sortie.

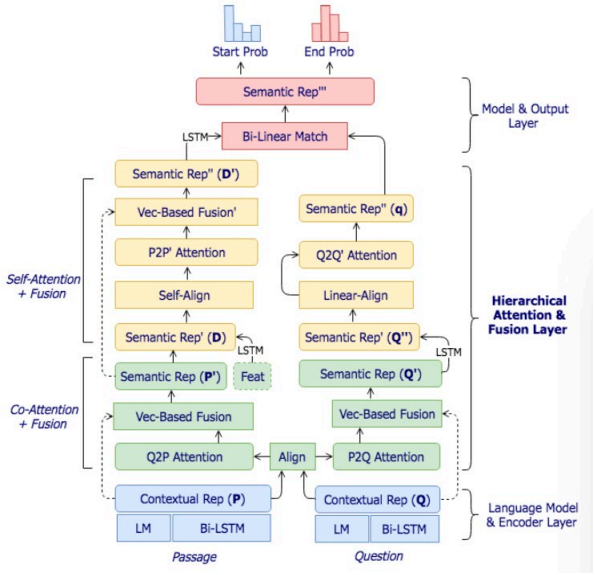


Figure 9. Architecture du modèle SQLA d'après Peters et al. [35]

Ce modèle a été entraîné en mode ensemble, c'est-à-dire qu'il a été entraîné sur 15 essais avec les mêmes hyperparamètres. À l'étape d'essai avec l'ensemble de développement, la réponse présentant la somme la plus élevée des scores de probabilité obtenus avec les 15 modèles d'entraînement est choisie comme réponse la plus probable à la question associée. Ce modèle atteint un EM = 82.44% et un *F1* = 88.60% sur SQuAD 1.1 pour sa version ensemble, ce qui le place au même niveau que le modèle RMR et inférieurs aux précédents.

Le Tableau 3 résume les caractéristiques principales des modèles discutés plus haut. On observe que les modèles les plus performants, soit les trois premiers,

1. Utilisent la même architecture de base, soit des unités Transformers,
2. Des embeddings à l'entrée de hautes dimensions,
3. Requièrent un pré-entrainement intensif au moyen de ressources externes très importantes et
4. Sont très lourds à entraîner et à implémenter pour des applications spécifiques.

Il n'est donc pas étonnant de constater que ces efforts sont principalement supportés par les Google, Facebook et Microsoft de ce monde. Par ailleurs, les approches plus

‘traditionnelles’ (si on peut parler de traditionnel en apprentissage profond) qui font principalement appel à des réseaux RNR ou à convolution et à différents mécanismes d’attention ainsi qu’à des embeddings d’entrée plus ou moins complexes, présentent quand même des performances (EM ~ 82-84%) proches des premiers, mais en utilisant beaucoup moins de ressources physiques, donc plus à la portée de tous, et ne requièrent pas de ressources externes importantes.

C’est dans ce contexte qu’au début de ce projet, en février 2017, l’approche de modèles à base de RNR plus légers et plus accessibles a été choisie pour les travaux présentés dans ce mémoire ciblant, en autre, le modèle BIDAF [19] qui était à l’époque (Figure 1) un des plus performants dans le domaine QA appliqué à SQuAD 1.1.

Tableau 3A. Comparaison des modèles les plus performants sur SQuAD 1.1

Modèle	Type	Performance		Principales originalités	Embeddings (emb.) d'entrée	Commentaires
		EM (%)	*F1* (%)			
BERT	Langage pré-entrainé	Base : 85.1 Large : 87.4	Base : 91.8 Large : 93.1	*Encodage avec unités Transformer bidirectionnelles *Pré-entrainement avec tâches non supervisées sur BookCorpus et Wikipedia.	*De mots (WordPiece), de segments et de positions, additionnés de dimension (dim.) 768 (de base) et 1,024 (large)	*Ressources externes importantes *Très lourd au pré-entraînement et à l'utilisation
XLNet	Langage pré-entrainé autorégressif	**89.8**	**95.1**	*Comme BERT avec pré-entraînement sur BookCorpus, Wikipedia, GIGA, ClueWeb et Common Crawl	Comme BERT avec encodage de position relatif	*Ressources externes plus importantes que BERT *Très lourd
KT-NET	BERT	85.9	92.4	*Utilise BERT pré-entraîné à l'encodage *Ajout d'emb. de connaissances de BC *Ajout d'une couche d'auto-correspondance	Comme BERT large avec concaténation (concat.) d'emb. de BC de dim. 100	*Lourdeur d'utilisation comme BERT
SAN	RNR de type LSTM & GRU	79.6	86.4	*Embeddings de mots, Lexicon et CoVe *Utilisation d'une couche mémoire *Fonction bilinéaire à la sortie	De mots (GloVe 300) et linguistiques (pour P, 300), réduits à 300 par MLP puis concat. avec emb. CoVe (600) des mots	
QANet	Réseaux à convolution	84.4	90.4	*Attention de type Transformer *Réseaux à convolution	De mots (GloVe 300) et de caractères (GloVe 300; valeurs maximales) concat. avec ajout d'une couche autoroute	Réseaux à convolutions beaucoup plus rapides (~5X) que RNR

Tableau 3B. Comparaison des modèles les plus performants sur SQuAD 1.1

Modèle	Type	Performance		Principales originalités	Embeddings (emb.) d'entrée	Commentaires
		EM (%)	*F1* (%)			
R-NET	RNR de type GRU	84.0	90.1	*Couche de correspondance P-Q *Couche d'auto-correspondance sur P *Couche de sortie de type pointeur	De mot (GloVe 300) et de caractères (GloVe 300), alimentés à un BIGRU de couche cachée de 100; Emb. concat. puis alimentés à BIGRU de dim. 2*75	
RMR	RNR de type BILSTM	82.8	88.7	*3 blocs d'aligneurs itératifs de ré-attention avec une fonction de fusion heuristique *Couche finale de pointeur de prédiction de réponse	De mot (GloVe 100 et ELMo 1024 concat [51]), de caractères (GloVe 300 à BILSTM de sortie 100) et lexicaux (30), tous concat. et alimentés à un BILSTM de sortie 200	
SQLA	Attention/ fusion hiérarchique + RNR de type BILSTM à l'entrée et à la sortie	82.4	88.6	*Couche d'attention et de fusion hiérarchique *Fonction bilinéaire à la sortie	De mot (GloVe 100) et de caractères (ELMo [51] de dim 1,024 [36] à l'entrée et de 20 à la sortie) puis concaténation de ces derniers, BILSTM (100) puis concat. avec emb. de caractères Et linguistiques 40 à la fin de la couche attention-fusion	

2.3 Modèle BIDAF

Dans cette section, le modèle BIDAF [19] est présenté en détails puisqu'il est celui utilisé et modifié dans les travaux de ce mémoire. La publication originale du modèle BIDAF (Bidirectional Attention Flow) appliqué à SQuAD 1.1 date de 2016 et a été révisée en 2018 [19].

La principale originalité de BIDAF porte sur l'utilisation d'un mécanisme bidirectionnel de flux d'attention qui permet d'obtenir une représentation du contexte dépendante ('aware') de la question. Ce mécanisme est appliqué dans les deux directions, soit de la question au contexte (Q2C) et du contexte à la question (C2Q), ce qui fournit de l'information complémentaire à chacun d'eux. Une attention est calculée pour chaque paire contexte-question d'indice *i*. Cette attention n'est fonction que de la question et du contexte en cours et ne dépend pas directement de l'attention de l'étape précédente. Les vecteurs résultants $q2c_i$ et $c2q_i$ sont concaténés avec les représentations des embeddings contextuels de la question et du contexte obtenues de la couche précédente, et le tout est alimenté à la couche de modélisation subséquente. Cette approche est sensée forcer la couche d'attention à se concentrer sur l'apprentissage de l'attention entre la question et le contexte, et la couche de modélisation à se concentrer sur l'apprentissage de l'interaction dans la représentation du contexte dépendante de la question, soit la sortie de la couche d'attention. Ceci permet à l'attention à chaque étape de ne pas être affectée par des relations incorrectes des étapes précédentes.

Tel qu'illustré à la Figure 10, le modèle BIDAF comporte cinq couches principales, soit une couche d'encodage d'embeddings, une couche d'embeddings contextuels, une couche d'attention bidirectionnelle, une couche de modélisation et une couche de sortie.

À la couche d'encodage d'embeddings, les mots tokénisés du contexte $\{x_1, x_2, \dots x_T\}$ et de la question $\{q_1, q_2, \dots q_J\}$ sont encodés au moyen des embeddings pré-entrainés de mots et de mots à base de caractères. Les embeddings de mots sont obtenus des embeddings pré-entraînés GloVe [18] de dimension 100. Les embeddings de mots à

base de caractères sont calculés selon l'approche de [52]. Aux caractères d'un vocabulaire sont associés des embeddings de dimension 8 initialisés aléatoirement. Les embeddings de caractères de chaque mot du contexte et de la question sont obtenus depuis ce vocabulaire, et ceux-ci sont soumis à un réseau de convolution pour obtenir, pour chaque mot, un embedding de mot à base de caractères de dimension 100.

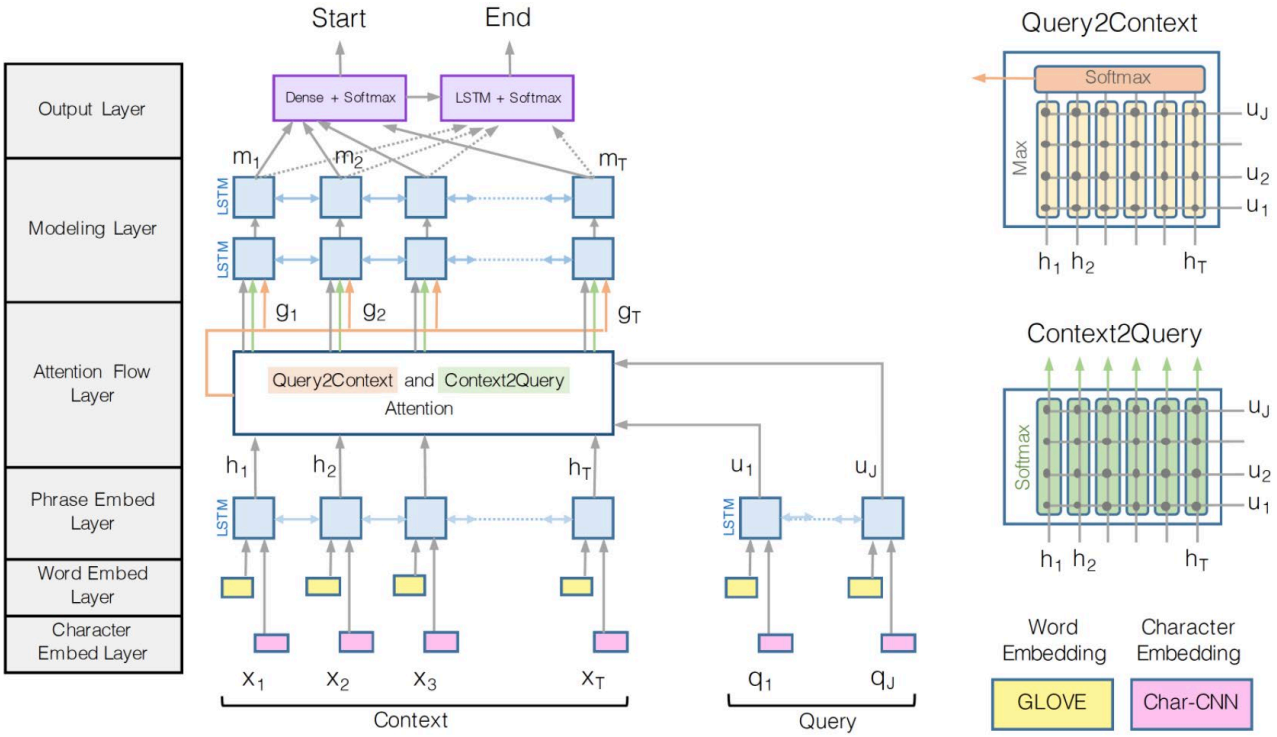


Figure 10. Architecture du modèle BIDAF d'après Seo et al. [19]

Par la suite, les embeddings GloVe et de mots à base de caractères de chaque mot sont concaténés puis soumis à un réseau autoroute ('highway') défini selon l'équation 1 [53] :

$$y = MLP_1(sigmoid)x \times MLP_2(relu)x + (1 - MLP_1 x)x. \qquad (1)$$

Soit avec des activations de type sigmoid et relu, respectivement.

On obtient alors des représentations finales du contexte $X \in \mathbb{R}^{T \times d}$ et de la question $Q \in \mathbb{R}^{j \times d}$ où d = 100 et T et j sont, respectivement, la longueur en mots du contexte et de la question.

À la couche d'embeddings contextuels, les représentations X et Q sont alimentées séparément à deux réseaux parallèles BILSTM (avec concaténation interne des LSTM unidirectionnels) avec dimension de sortie égale à 2d pour modéliser les interactions temporelles entre les mots du contexte et de la question, respectivement. On obtient alors une représentation contextuelle $H \in \mathbb{R}^{T \times 2d}$ des vecteurs des mots du contexte X et $U \in \mathbb{R}^{j \times 2d}$ des vecteurs de mot de la question Q.

L'objectif de la couche d'attention bidirectionnelle est de lier et fusionner l'information des mots du contexte et de la question. On y calcule les attentions bidirectionnelles C2Q et Q2C depuis la matrice de similarité partagée $S \in \mathbb{R}^{T \times J}$ entre H et U, les embeddings contextuels du contexte et de la question, où S_{tj} représente la similarité entre le t^e mot du contexte et le j^e mot de la question. Cette matrice est calculée selon l'équation 2.

$$S_{tj} = \alpha(H_{:t}, U_{:j}) \in \mathbb{R} \tag{2}$$

Où α est un paramètre appris et la fonction $\alpha(H_{:t}, U_{:j})$ choisie est donnée par l'équation 3.

$$\alpha(h, u) = w_{(S)}^{\mathrm{T}}[h; u; h \circ u] \tag{3}$$

Où $w_{(S)}^{\mathrm{T}} \in \mathbb{R}^{6d}$ est un vecteur de poids appris, $\circ$ dénote une multiplication d'élément à élément et [;] indique la concaténation le long des lignes. Cette matrice de similarité $\boldsymbol{S}$ est utilisée pour calculer les attentions C2Q et Q2C comme décrit ci-après.

L'attention C2Q représente quels mots de la question sont les plus pertinents pour chaque mot du contexte. Soit $a_t \in \mathbb{R}^J$ les poids d'attention des mots de la question pour le t^e mot du contexte, avec $\sum a_{tj} = 1$, ceux-ci sont calculés selon l'équation 4.

$$a_t = softmax(S_{t:}) \in \mathbb{R}^J \tag{4}$$

Par la suite, C2Q (où $\widetilde{U} \in \mathbb{R}^{2d \times T}$) est calculé par l'équation 5.

$$\widetilde{U}_{:t} = \sum_j a_{tj} U_{:j} \tag{5}$$

L'attention Q2C signifie quels mots du contexte ont la plus proche similarité à un des mots de la question et donc sont les plus critiques pour répondre à la question. On calcule les poids d'attention des mots du contexte selon l'équation 6.

$$b = softmax\big(max_{col}(S)\big) \in \mathbb{R}^T \tag{6}$$

Puis on calcule Q2C (où $\widetilde{H} \in \mathbb{R}^{2d \times T}$) selon l'équation 7 où $\tilde{h}$ est répété T fois sur la colonne pour donner $\widetilde{H}$.

$$\tilde{h} = \sum_t b_t H_{:t} \in \mathbb{R}^{2d} \tag{7}$$

Finalement, les embeddings contextuels du contexte H et les vecteurs d'attention C2Q et Q2C sont combinés pour obtenir G, selon l'équation 8, où chaque vecteur colonne est la représentation de chaque mot du contexte par rapport à la question.

$$G_{:t} = \beta\big(H_{:t}, \widetilde{U}_{:t}, \widetilde{H}_{:t}\big) \in \mathbb{R}^{d_G} \tag{8}$$

Où $G_{:t}$ est le t^e vecteur colonne correspondant au t^e mot du contexte et $\beta()$ est une fonction vecteur entraînée qui fusionne les trois vecteurs d'entrée. La fonction $\beta()$ est donnée par l'équation 9, soit une simple concaténation, qui a été trouvée expérimentalement suffisamment performante par les auteurs.

$$\beta\big(h, \tilde{u}, \tilde{h}\big) = \big[h; \tilde{u}; h \circ \tilde{u}; h \circ \tilde{h}\big] \in \mathbb{R}^{8d} \tag{9}$$

À la couche de modélisation, on capture l'interaction des mots du contexte conditionnés par la question. Pour ce faire, on utilise deux RNR consécutifs de type BILSTM, chacun de dimension cachée = d avec concaténation interne, avec la sortie du second dénotée $M \in \mathbb{R}^{2d}$. On assume que chaque colonne vecteur de M contient l'information contextuelle du mot par rapport à tout le texte du contexte et de la question.

À la couche de sortie, on calcule la distribution des probabilités que la position de chaque mot du contexte soit le début de la réponse à la question p^1 selon l'équation 10.

$$p^1 = softmax\left(w_{(p^1)}^{\mathrm{T}}[G; M]\right) \tag{10}$$

Où $w_{(p^1)}^{\mathrm{T}} \in \mathbb{R}^{10d}$.

Pour obtenir la distribution des probabilités que la position de chaque mot du contexte soit la fin de la réponse p^2, on passe M dans un autre RNR de type BILSTM de dimension cachée = d avec concaténation interne dont la sortie est dénotée $M^2 \in \mathbb{R}^{2d}$. Par la suite, on calcule p^2 au moyen de l'équation 11.

$$p^2 = softmax\left(w_{(p^2)}^{\mathrm{T}}[G; M^2]\right) \tag{11}$$

Où $w_{(p^2)}^{\mathrm{T}} \in \mathbb{R}^{10d}$.

La plage de réponse dans le contexte (k: position de début de la réponse dans le contexte et l: position de fin de la réponse dans le contexte) est choisie selon la valeur maximale du produit des probabilités $p_k^1 p_l^2$ avec la restriction que $k \leq l$.

La perte d'entraînement, à minimiser, est définie comme la somme, pondérée sur tous les exemples, des probabilités log négatives des vrais indices de début et de fin de la réponse dans le contexte par rapport aux distributions prédites de positions de début p^1 et de fin p^2 de la réponse dans le contexte, selon l'équation 12.

$$\mathcal{L}(\theta) = -\frac{1}{N}\sum_{j=1}^{N}\left(\log\left(p_{y_j^1}^1\right) + \log\left(p_{y_j^2}^2\right)\right) \tag{12}$$

Où θ sont les poids appris du modèle, N, le nombre d'exemples de l'ensemble de données, y_i^1 et y_i^2 sont les vrais indices de début et de fin de la réponse dans le contexte de l'exemple i et p_k est la k^e valeur du vecteur $\boldsymbol{p}$.

L'entraînement de BIDAF est fait au moyen de l'optimiseur Adadelta avec un taux d'apprentissage de 0.5 et une taille de lots de 60. Le temps d'entraînement est d'environ 20 heures sur une unité GPU Titan X. Les auteurs ont également entraîné un modèle ensembliste comportant 12 essais d'entraînement avec les mêmes hyper paramètres et la même architecture. À l'étape de test, la réponse avec la plus haute somme des scores de confiance parmi les 12 essais effectués était choisie pour chaque question. Les résultats obtenus (Tableau 2) sont de EM = 67.7% et *F1* = 77.3% et EM = 72.6% et *F1* = 80.7% pour les modèles unique et ensembliste, respectivement, sur l'ensemble de

développement et EM = 67.9% et *F1* = 77.3% et EM = 73.6% et *F1* = 81.9% pour les modèles unique et ensemble, respectivement, sur l'ensemble test.

En résumé, BIDAF comporte un encodage initial de mots à partir d'embeddings de mots pré-entrainés et de mots à base de caractères, une couche d'encodage contextuel utilisant un réseau BILSTM, un mécanisme d'attention bidirectionnel, une couche de modélisation utilisant des réseaux BILSTM et une couche de sortie MLP-softmax. En évaluant son architecture par rapport à celle des modèles précédents (Tableau 5, autres que les modèles de langage pré-entraînés), on observe que seule sa couche d'encodage contextuelle se compare à celle du modèle SAN et que le modèle QAnet lui emprunte son mécanisme d'attention bidirectionnel. Autrement, BIDAF diffère de tous ces autres modèles.

Un certain nombre de variantes de BIDAF ont été publiées dont les plus performantes sont présentées au Tableau 2.

1.BiDAF + ELMo + Auto-attention [36]
Cette variante utilise à l'entrée des embeddings GloVe 300 [18] concaténés à des embeddings de type ELMo [51] de dimension 512 des mots du contexte et de la question. Les embeddings ELMo diffèrent des autres types d'embeddings en ce sens qu'un mot se voit attribuer une représentation qui dépend de toute la phrase d'où il provient. Ces embeddings sont obtenus des états internes de réseaux LSTM bidirectionnels profonds qui sont entraînés sur un grand corpus de texte (voir section 3.2).

Les embeddings ELMo remplacent les embeddings de mot à base de caractères de la version originale de BIDAF. De plus, ces embeddings ELMo de mots sont concaténés avec la sortie de la couche d'encodage contextuel et les unités RNR de type LSTM sont remplacées par des unités GRU. Une couche d'auto-attention serait ajoutée à la sortie de la couche d'encodage contextuelle. Or les indications sur la structure de ce modèle sont peu précises concernant cette dernière qui, en fait, pourrait n'être que la dernière concaténation $[H; X_{emb}^{ELMo}]$ et $[U; Q_{emb}^{ELMo}]$ (https://github.com/allenai/allennlp/issues/2595).

Au Tableau 2, on observe que cette variante atteint des performances de EM = 81.0% et *F1* = 87.4%, ce qui est nettement supérieur à celles de BIDAF original (67.7% et 77.3%). Ces résultats suggèrent l'importance de l'encodage d'embeddings sur la performance du modèle BiDAF.

On notera que la variante BIDAF++ [36,37] (renommée DialogQA) indiquée au Tableau 2 serait la même que BiDAF + ELMo + Auto-attention. Or BIDAF++ présente des performances de EM = 77.5% et *F1* = 84.8%, ce qui est inférieur à celles de la variante précédente BIDAF + ELMo + Auto-attention.

Une autre variante de BIDAF++ avec l'ajout d'une unité pair2vec relativement complexe [37] a été publiée. L'objectif de pair2vec est de représenter les relations entre les pairs de mots (x, y) d'un contexte c en modélisant la cooccurrence à trois voies entre ces paires et le contexte c qui les relie, ce qui donne des embeddings basés sur les pairs qui peuvent être utilisés pour améliorer la performance de systèmes QA, dans ce cas-ci BIDAF++. Dans les faits, cette dernière variante présente des performances de EM = 78.2% et *F1* = 85.5% inférieures à celles du modèle BIDAF++ original.

2.BIDAF + Auto-attention [37]

Cette variante utilise la même architecture que celle de BIDAF originale sauf pour les deux modifications suivantes. Des unités RNR de type GRU remplacent les unités LSTM et une couche d'auto-attention est introduite entre la couche d'attention bidirectionnelle et la couche de modélisation.

Les couches de modélisation et de sortie de ce modèle sont identiques à celle de BIDAF sauf pour l'utilisation d'unités BIGRU plutôt que BILSTM.

Tel qu'indiqué au Tableau 2, les résultats obtenus pour ce modèle sont de EM = 72.1% et *F1* = 81.0%, ce qui est supérieur à ceux obtenus pour le modèle BIDAF unique (EM = 67.7% et *F1* = 77.3%).

3.SEDT + BIDAF [38]

Cette variante alimente au modèle BIDAF original des embeddings de mot pré-entraînés Glove 100 et de mot à base de caractères (comme pour BIDAF) concaténés à des embeddings syntactiques structuraux (Structural Embedding of Dependency Trees (SEDT)) des mots du contexte et de la question. Tel qu'indiqué au Tableau 2, les résultats obtenus pour ce modèle sont de EM = 73.7% et *F1* = 81.5%, ce qui est supérieur à ceux obtenus pour le modèle BIDAF unique (EM = 67.7% et *F1* = 77.3%).

Dans le cadre des travaux du présent mémoire, on a développé et testé différentes variantes plus basiques de BIDAF, soit au niveau des différences de structure observées entre la version décrite plus haut selon la publication originale de BIDAF [19] et son implémentation disponible à https://github.com/allenai/bi-att-flow. En plus, on a testé l'utilisation des embeddings de type ELMo comme [36] et le remplacement des unités RNR de type LSTM par des unités de type GRU.

Chapitre 3 Méthodologie

3.1 Métriques

3.1.1 *EM* et *F1*

Le calcul de *EM* est fait selon l'équation 13 :

$$EM = 100 \times {}^{N_{dfc}}\!/_{N_q} \tag{13}$$

où N_{dfc} est le nombre de fois où le système prédit des positions de début et de fin d'un segment de texte dans le contexte qui correspond exactement au segment de texte de la bonne réponse associée à la question, et N_q, le nombre de questions de l'ensemble de données.

Pour l'ensemble de développement, qui contient jusqu'à cinq bonnes réponses possibles à la question (voir la description des données de l'ensemble SQuAD à la section 1.2), un segment de texte prédit qui correspond exactement au segment d'une des bonnes réponses est considéré comme une bonne réponse prédite.

Le calcul de *F1* est réalisé comme suit.

1. Pour chaque question, le segment de texte tokénisé prédit par le modèle est comparé, token à token, au segment de texte tokénisé de la réponse exacte d'où on obtient le nombre de mots identiques entre ces deux segments, soit N_{mi}. Pour l'ensemble de développement, qui contient jusqu'à 5 bonnes réponses possibles à la question, on choisit la valeur maximale de N_{mi} ($N_{mi\text{-}max}$) parmi ces tests de comparaison pour le calcul de *F1*.
2. Pour chaque réponse à une question, on calcule une précision ($prec$) et un *recall* (*rec*) selon les équations 14 et 15 suivantes depuis les longueurs, en nombre de tokens, du segment de texte prédit (L_p) et de la réponse exacte (L_{re}):

$$prec = {}^{N_{mi-max}}\!/_{L_p} \tag{14}$$

$$rec = {}^{N_{mi-max}}\!/_{L_{re}} \tag{15}$$

3. Puis on calcule *F1* selon l'équation 16 :

$$F1 = 100 \times \left(\sum_{i=1}^{N_q} \left({2 \times prec_i \times rec_i}/{(prec_i + rec_i)} \right) \right) \Big/ N_q \qquad (16)$$

3.1.2 Calculs des pertes et des précisions de prédictions durant l'entraînement

Durant l'entrainement d'un modèle, Keras permet de suivre à la fin de chaque epoch les pertes et le calcul de précision des prédictions de la position de début (yS) et de fin (yE) de la réponse dans le contexte au moyen de la fonction *history* de Keras pour l'ensemble d'entraînement.

De même, au moyen de l'objet Callbacks de Keras, on peut calculer, à la fin de chaque epoch, les pertes et trois mesures de précision de prédictions du modèle pour l'ensemble de développement. Ces trois mesures de précision de prédictions sont calculées selon les trois méthodes suivantes:

1. Pour les pertes et les prédictions de prédiction de yS et de yE de la réponse dans le contexte[2] par la fonction *model.evaluate()* de Keras appliquée aux exemples de l'ensemble de développement,
2. Pour les prédictions de yS et yE comparées aux cinq réponses correctes possibles de l'ensemble de développement (section 1.2 – SquAD et Tableau 1)[3], par la fonction *model.predict()* appliquée aux exemples de l'ensemble de développement qui sont comparées aux 5 réponses possibles aux questions de l'ensemble de développement et
3. Selon la publication originale de BIDAF [19] comme indiqué après l'équation 11 de la section 2.3 par la multiplication des matrices de probabilités yS et yE pour trouver la plage de réponse prédite dans le contexte (k, l) de valeur maximale du produit des probabilités $yS_k yE_l$ avec la restriction que $k \leq l$ pour chaque exemple[4].

[2] Figure A1 précision de prédictions: Test SyS et Test SyE
[3] Figure A1 : Test 5yS et Test 5yE
[4] Figure A1 : Test MyS et Test MyE

Par la suite, pour chacune de ces trois mesures, on calcule les précisions de prédiction simultanément correctes pour le début et la fin de la réponse dans le contexte, soit MySE pour l'ensemble d'entraînement, et Test 5ySE, depuis Test 5yS et Test 5yE, et Test MySE, depuis Test MyS et Test MyE.

À la fin de l'essai d'un modèle, on utilise une fonction qui donne un fichier .json des segments de texte des réponses prédites reconstruits à partir des positions yS et yE prédites par le modèle pour évaluer sa performance au moyen de l'application *evaluate.py* fournie par l'équipe de SQuAD (https://github.com/allenai/bi-att-flow/blob/master/SQuAD/evaluate-v1.1.py) pour calculer *EM* et *F1* selon les équations 13 et 16, respectivement.

3.2 Implémentation de BIDAF

3.2.1 Introduction

L'implémentation originale de BIDAF [19] en Tensorflow de 2017 et révisée jusqu'en janvier 2018 est disponible à https://github.com/allenai/bi-att-flow. Cette implémentation et la description du modèle BIDAF [19] ont été étudiées pour développer une nouvelle version de BIDAF en API Keras avec Tensorflow en backend lors des travaux de ce mémoire. Dans le texte qui suit on se référera à cette implémentation en Tensorflow de 2017 comme à "l'implémentation originale" de BIDAF. Trois autres implémentations ont été trouvées et étudiées, soit

1. L'implémentation de Jojonki en Pytorch disponible à https://github.com/jojonki/BiDAFc de 2017 pour laquelle aucun résultat n'a pu être trouvé sauf dans les discussions à EM ~ 40% sur l'ensemble de développement,
2. L'implémentation de Kadam Parikh en Keras mais non sous la forme API disponible à https://github.com/ParikhKadam/bidaf-keras/tree/master/bidaf en 2018/9 pour laquelle aucun résultat n'a pu être trouvé et
3. L'implémentation de Allen NLP en Pytorch disponible à https://github.com/allenai/allennlp-readingcomprehension/blob/master/allennlp_rc/models/bidaf.py de 2019 dont les résultats sont présentés au Tableau 2 pour le

modèle unique (EM = 67.6% et *F1* = 77.1%), ce qui se compare à ceux de la publication de BIDAF [19] (EM = 67.7% et *F1* = 77.3%).

Il est intéressant de noter au moins les cinq différences importantes suivantes du modèle BiDAF entre son implémentation originale et la description présentée à la section 2.3 de son architecture tirée de la publication [19].

1. À l'implémentation, les paramètres des sous-couches autoroutes (équation 1), au-dessus de l'encodage en embeddings des mots du contexte et de la question, sont partagés entre l'encodage du contexte et de la question, ce qui n'est pas indiqué à la publication [19].
2. De même, à l'implémentation, les paramètres des RNR d'encodage contextuel BILSTM sont partagés entre l'encodage du contexte et de la question, avec la question précédant le contexte, ce qui n'est pas indiqué à la publication [19].
3. Au calcul de la matrice de similarité (équations 2 et 3), le réseau MLP ($w_{(S)}^{\mathrm{T}}$) sans fonction d'activation (équation 3) est d'une dimension cachée de 1 à l'implémentation contrairement à *6d* indiquée à la publication [19].
4. À la couche de sortie, le réseau MLP interne sans activation précédent l'activation softmax ($w_{(p^1)}^{\mathrm{T}}$ (équation 10) et $w_{(p^2)}^{\mathrm{T}}$ (équation 11)) est d'une dimension cachée de 1 contrairement à *10d* indiquée à la publication [19].
5. Le calcul de la probabilité de la position de la fin de la réponse dans le contexte p^2 est différent et beaucoup plus complexe à l'implémentation original de BIDAF que l'équation 11 et que le calcul de la probabilité de la position du début de la réponse dans le contexte p^1 selon l'équation 10, soit selon la publication de BIDAF [19].

L'objectif des sous-sections suivantes est de résumer l'implémentation originale de BIDAF par rapport à sa description présentée à la section 2.3 selon sa publication [19] afin d'en produire une version sous la forme Keras API avec Tensorflow en backend.

3.2.2 Traitement des données

L'ensemble SQuAD 1.1 utilisé pour ce projet a été décrit à la section 1.2. Les sous-ensembles d'entraînement et de développement ont été traités séparément au moyen de

certaines composantes de l'application https://github.com/strin/counter-SQuAD/blob/master/data.py. Cette application inclut une fonction qui transforme la position de début de la réponse dans le paragraphe, en termes de caractères et d'espaces (voir la description des données de l'ensemble SQuAD à la section 1.2), en termes de mots, l'approche utilisée par BIDAF et les autres modèles décrits à la section 2.2.

Pour les embeddings de mots, un vocabulaire de mots uniques des ensembles d'entrainement et de développement a été créé au moyen d'une fonction obtenue de l'application https://github.com/strin/counter-SQuAD/blob/master/data.py, ce qui a produit un vocabulaire de 81,983 mots uniques pour l'ensemble d'entraînement et un vocabulaire de 23,931 mots uniques pour l'ensemble de développement. En comparant ces deux vocabulaires, on trouve 4,869 mots du second qui ne sont pas présents dans le premier. On ajoute donc ces 4,869 mots au vocabulaire de l'ensemble d'entrainement pour obtenir un vocabulaire indexé de 86,852 mots uniques des deux ensembles.

Tout comme pour BIDAF [19], on utilise les embeddings pré-entrainés GloVe 100 [18]. Depuis le vocabulaire indexé et les embeddings GloVe pré-entrainés de dimension 100 de glove.6B.100d.txt de 400,000 mots (https://nlp.stanford.edu/projects/glove/), on crée une matrice indexée de poids/embeddings de 86,852 X 100 qui contient les embeddings des mots indexés du vocabulaire. Cette matrice est sauvegardée pour utilisation subséquente.

Cette opération permet d'associer à 69,274 mots du vocabulaire original (79.76%) des embeddings pré-entrainés 100 de GloVe. Un vecteur d'embeddings zéro est associé aux 17,579 mots du vocabulaire qui n'ont aucun embedding GloVe. Le Tableau 4 présente les 240 premiers mots n'ayant pas d'embeddings GloVe 100. Ceux-ci comprennent des noms propres (*kaddafi, beyonceae*), des noms mal orthographiés (*aeronatical, tempations, advetisments, popularion, prarie* etc.), des expressions (*kofc, wsnd, hteir, mntn* etc.), des mots rares (*netbase, polymixins, lincosamides, lipopeptides*), des mots

résultant de l'omission d'une ponctuation (*postniminals* etc.) etc. Un vecteur d'embedding zéro est associé à ces mots.

Tableau 4. Exemples de mots sans embeddings GloVe 100

venite	beyonceae	jorgi	prontosil	biaobocki	tronchet
aeronatical	cspinet	popovi	domagk	matuszyski	orlans
lobund	unhealthiness	ljubia	tyrothricin	witwicki	gestirne
gnotobiotics	netbase	krsti	tyrocidine	gadkowska	funbre
postnominals	advetisments	miloevi	lysates	gladkowska	triomphale
kofc	gatefive	lazarevo	reactogenic	jarocki	clsinger
klavern	bayonce	delbrck	toxoids	radziwi	floriani
cavanugh	aspiro	activitty	pipelin	posenhimself	charactersa
blantz	agnz	polymyxins	coukell	reviewsin	healthcould
sienko	topshops	rifamycins	frdric	darem	franchomme
mestrovic	charitybuzz	lipiarmycins	franois	composisition	plitical
lafun	mntn	lincosamides	wodziska	zdzisaw	bozzolini
chroust	montaa	lipopeptides	tudes	jachimecki	prosseda
metrovi	popularion	glycylcyclines	scherzos	geographicla	blanchar
notrdem	prarie	tigecycline	elazowa	eugne	biogrpahies
daym	raillink	oxazolidinones	fridericus	grzymaa	chopiniana
snite	morony	fidaxomicin	krzyanowska	michaowski	orchestrationsfrom
clestine	koocanusa	anitibiotics	skarbeks	ftis	tcherepninfor
hailandire	lonepine	sprectrum	brochw	appearancesfew	yszczyski
wsnd	jurisdictionscity	antibacterialsfor	chopins	hexameron	obreskoff
wvfi	mtis	oxazolidinonesare	ywny	dsseldorf	vendme
jerian	kutenai	penicilin	belweder	bendemann	mthode
bijnse	monatanas	slufonamides	ursyn	wodziskis	cruveilhier
beyinc	honyockers	semisytetic	niemcewicz	wodziski	lablache
darlette	scissorbills	revolutinzed	nasze	baillot	lefbure
hopera	honyocker	microorgainsim	przebiegi	chausse	wly
tempations	hunyak	microorgansim	wrfel	htel	fioriture
disban	icty8	photodermatitis	jzef	liszy	euorpeans
lades	cppcg	vulvo	eolomelodicon	agoult	extramusical
recordson	ritualcide	subtherapeutic	ename	perforemed	kobylaska
kaddafi	caedere	chlortetracycline	szafarnia	biogrpahers	hrtel
rowearning	factorsinform	adipogenesis	dziewanowski	amantine	ekier
lifeandtimes	jonassohn	whhat	krasiski	whre	oshow
tedxeuston	bjrnson	counterindication	przedmiecie	soire	zywny
irreemplazable	politicides	hje	salonik	flicien	polonasises
montina	condonation	nontherapeutic	chopinw	mallefille	compositons
moniqu	signatoriesnamely	2562	ambroy	valldemossa	perfomers
beyontourage	yugoslaviasigned	antibiosis	mieroszewski	compalin	playingfor
hteir	cppc	salvarsan	woyciechowski	canuts	relude
scaptia	genocidenamely	arsphenamine	nepomucen	nohant	appoggiaturas

3.2.3 Longueurs maximales du contexte et de la question

Deux paramètres importants du modèle à déterminer sont la longueur maximale, en termes de mots-tokens, du contexte par rapport aux positions de début et de fin de la réponse dans celui-ci, et la longueur maximale de la question à considérer. Puisque cette

tâche est traitée comme un problème de classification, la longueur maximale du contexte choisie pour le modèle détermine sa dimension. Les plus longs contextes et questions de l'ensemble d'entrainement comportent 677 et 40 mots, et pour l'ensemble de développement, 634 et 33 mots. Par ailleurs, on observe que les positions de début et de fin des réponses dans les contextes sont principalement inférieures à 300 (99.9%) et même à 200 (99.1%) et 150 (96.5%) pour l'ensemble d'entrainement comme illustré à la Figure 11, et de même pour l'ensemble de développement (<300 :99.7%; <200 :98.7%; < 150 : 95.9%).

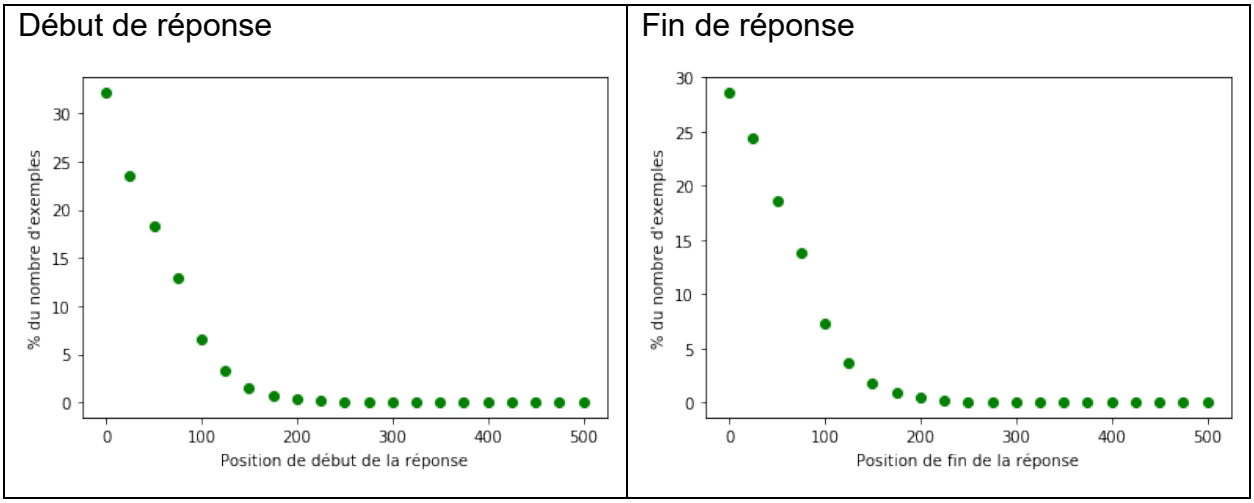


Figure 11. Distribution des positions de début et de fin de réponses pour l'ensemble d'entrainement

À ce sujet, le Tableau 5 présente les longueurs maximales de contexte et de question utilisées pour les modèles discutés précédemment selon leur implémentation. À l'exception de BIDAF, on observe des longueurs largement supérieures à 300 (et même pour la question, supérieures à 33 et 40 mots). Dans ce contexte, les longueurs maximales des contextes et des questions ont été fixées à 300 et 30, respectivement, pour ce projet.

À la couche d'encodage d'embeddings, les mots tokénisés du contexte et de la question sont encodés pour des embeddings de mots pré-entrainés et de mots à base de caractères comme suit.

Tableau 5. Longueurs maximales du contexte et de la question pour différents modèles

Modèle	Longueur maximale en mots du contexte	Longueur maximale en mots de la question
BERT [20]	384	64
XLNet [26]	512	64
KT-NET [29]	512	64
QANet [31]	400	50
R-NET [32]	400	50
BIDAF [19]	256	30

3.2.4 Couche d'encodage d'embeddings

Les embeddings des mots des contextes et des questions sont obtenus à partir des embeddings pré-entraînés GloVe de dimension 100 [18] au moyen de la matrice de poids des embeddings indexés du vocabulaire, tel qu'indiqué à la section 3.2.2, et de la couche Keras 'Embedding'. On notera que cette dernière comporte le paramètre *trainable*, dont les valeurs sont *True* ou *False*, qui permet l'entraînement ou non des embeddings de mots durant l'entraînement du modèle. L'effet de cette option de la couche Keras 'Embedding' sur la performance de certains modèles a été testé dans les travaux discutés au Chapitre 4.

On obtient alors, pour l'ensemble d'entrainement, les cubes de données $c\_ew \in \mathbb{R}^{n,mcl,100}$ pour les contextes et $q\_ew \in \mathbb{R}^{n,mq,100}$ pour les questions où *n* est le nombre d'exemples, *mcl,* la longueur maximale des contextes (=300), et *mq,* la longueur maximale des questions (=30). On obtient des cubes équivalents pour l'ensemble de développement.

Le calcul des embeddings de mots à base de caractères est effectué selon BIDAF [19]. En résumé, les contextes et les questions tokénisés sont traités afin d'obtenir un texte continu de mots. Des longueurs maximales en termes de caractères des contextes ($maxlenC$) et des questions ($maxlenQ$) sont définis selon les équations 17 et 18.

$$maxlenC = 16 * (mcl + 1) \quad (17)$$

$$maxlenQ = 16 * (mq + 1) \quad (18)$$

Où le facteur 16 est la longueur maximale en caractères d'un mot, choisie selon [19], *mcl* = 300, la longueur maximale d'un contexte en mots tokénisés, auquel on ajoute 1 pour

obtenir $maxlenC = 4816$, mq = 30 , la longueur maximale d'une question en mots tokénisés, à laquelle on ajoute 1 pour obtenir $maxlenQ = 496$. Ces valeurs de $maxlenC$ et $maxlenQ$ sont requises afin d'obtenir à la sortie de la couche d'embeddings de mots à base de caractères selon BIDAF[19] des cubes d'embeddings c_ewCHF et q_ewCHF de dimensions identiques à celles des cubes de données des embeddings de mots $c\_ew \in \mathbb{R}^{n,mcl,100}$ et $q\_ew \in \mathbb{R}^{n,mq,100}$ décrits plus haut.

Les embeddings de caractères des contextes et des questions sont obtenus à partir de leurs matrices de caractères $C^{n,maxlenC}$ et $Q^{n,maxlenQ}$ au moyen d'une matrice de poids aléatoires des embeddings indexés d'un vocabulaire de caractères $vocCA^{69,8}$ comme BIDAF[19] et de la couche Keras 'Embedding'. On notera que cette dernière comporte le paramètre *trainable*, dont les valeurs sont *True* ou *False*, qui permet l'entraînement ou nom des embeddings de caractères durant l'entraînement du modèle. L'effet de cette option de la couche Keras 'Embedding' a été testé dans les travaux discutés au Chapitre 4. Les embeddings de caractères des contextes et des questions ainsi obtenus se présentent sous la forme de cubes de dimensions $c\_ewCH \in \mathbb{R}^{n,maxlenC,8}$ et $q\_ewCH \in \mathbb{R}^{n,maxlenQ,8}$.

Tout comme selon la publication de BIDAF [19], ces embeddings de caractères des contextes et des questions sont alimentés à une sous-couche à réseaux de convolution (4 couches de convolution, chacune avec 100 filtres de largeur 5, une activation de type *relu* et une couche MaxPooling d'une grandeur de 16) dont les sorties sont concaténées puis ceci est alimenté à un MLP sans activation de couche cachée de 100 pour les transformer en embeddings de mots à base de caractères de dimensions $c\_ewCHF \in \mathbb{R}^{n,mcl,100}$ et $q\_ewCHF \in \mathbb{R}^{n,mq,100}$.

Selon BIDAF [19], les paramètres de cette sous-couche à réseaux de convolution sont partagés entre le contexte et la question. Pour ce faire, on a testé deux stratégies soit

1. Celle selon l'implémentation de BIDAF avec les énoncés

```
with tf.variable_scope('char'):
    tf.get_variable_scope().reuse_variables()
```

entre les couches *c_ewCHF* et *q_ewCHF*, et

2. Celle selon Keras où une même couche sharedchEmb est définie et utilisée pour les contextes et les questions.

Finalement, les embeddings de mots obtenus de ceux pré-entrainés GloVe et ceux à base de caractères des contextes (c_ew et c_ewCHF) et des questions (q_ew et q_ewCHF) sont concaténés puis alimentés à un réseau autoroute à paramètres partagés (équation 1). Tout comme pour la sous-couche à réseaux de convolution précédente, deux stratégies de partage de paramètres ont été testées pour le réseau autoroute entre le contexte et la question.

3.2.5 Couche d'embeddings contextuels

Cette couche, alimentée depuis la sortie des couches autoroutes précédentes pour le contexte et la question, comporte deux réseaux parallèles BILSTM, le premier pour la question et le second pour le contexte selon l'ordre d'implémentation de BIDAF, à paramètres partagés entre la question et le contexte avec une dimension cachée de 100, un dropout (DO) [54] interne de 0.2 et la concaténation interne des sorties des deux directions. On notera que, selon l'implémentation de BIDAF, le premier RNR cible la question et comporte un dropout externe alors que le second est pour le contexte sans dropout. Leurs sorties sont de formes $H \in \mathbb{R}^{T \times 2d}$ des vecteurs de mots du contexte X (avec T = 300) et $U \in \mathbb{R}^{j \times 2d}$ des vecteurs de mots de la question Q (avec j = 30). Encore ici, tout comme pour les couches à paramètres partagés pour les embeddings de mots à base de caractères et pour les couches autoroutes à paramètres partagés, deux stratégies ont été expérimentées pour le partage des paramètres, soit

1. Celle selon l'implémentation de BIDAF mentionnée au début de la section 3.2 avec les énoncés
```
with tf.variable_scope('char'):
    tf.get_variable_scope().reuse_variables()
```
entre les couches *qa* et *ca*, et
2. Celle selon Keras où une même couche sharedBILTSM est définie et utilisée pour les contextes et les questions.

3.2.6 Couche d'attention bidirectionnelle

L'implémentation de cette couche, depuis la sortie de la couche précédente jusqu'à *G* selon l'équation 7, est particulièrement complexe tel qu'illustrée à la Figure 12.

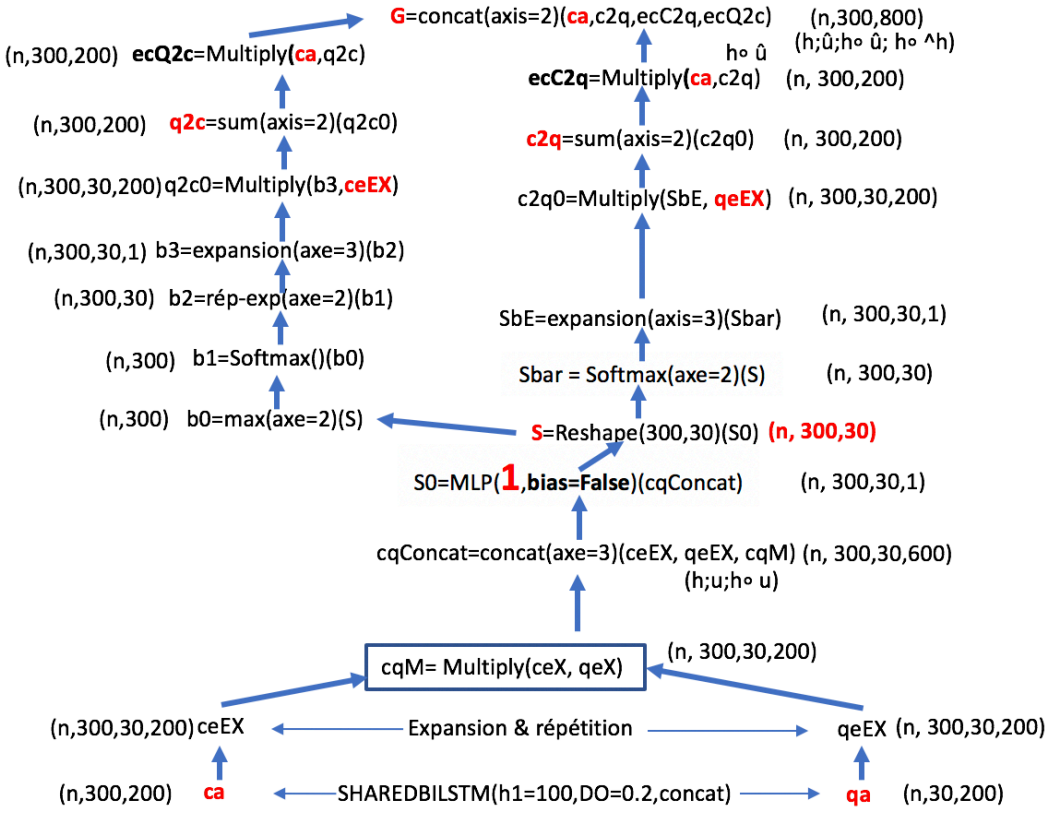


Figure 12. Architecture de la couche d'attention bidirectionnelle

3.2.7 Couches de modélisation et de sortie

Les couches de modélisation et de sortie selon BIDAF [19] décrites à la section 2.3 et son implémentation diffèrent significativement, en particulier pour le calcul de la position

de fin du segment de la réponse dans le contexte, comme l'illustrent les architectures des couches de modélisation et de sortie présentées aux Figures 13 et 14.

Selon BIDAF [19], la couche de modélisation (Figure 13) implique traiter le tenseur *G* obtenu de la couche d'attention bidirectionnelle précédente au moyen de deux RNR de type BILSTM consécutifs identiques, chacun de couche cachée h1 = 100 avec un dropout [54] d'un taux de 0.2 et une concaténation des sorties de chaque direction. Le résultat est *M* de dimensions (n, 300, 200) qui servira à la prédiction de la position de début de la réponse. Ce tenseur *M* est alimenté à un autre réseau RNR de type BILSTM identique aux deux précédents pour donner *M2* de dimensions (n, 300, 200) qui servira à la prédiction de la position de fin de la réponse. Par la suite, les tenseurs *M* et *M2* sont traités de façon identique, selon les équations 10 et 11, respectivement, pour prédire les positions de début et de fin de la réponse. La plage de la réponse dans le contexte (positions de début et de fin *k* et *l*) est choisie selon la valeur maximale du produit des matrices de probabilités $P_{ys}^{k} P_{ye}^{l}$ avec la restriction que $k \leq l$.

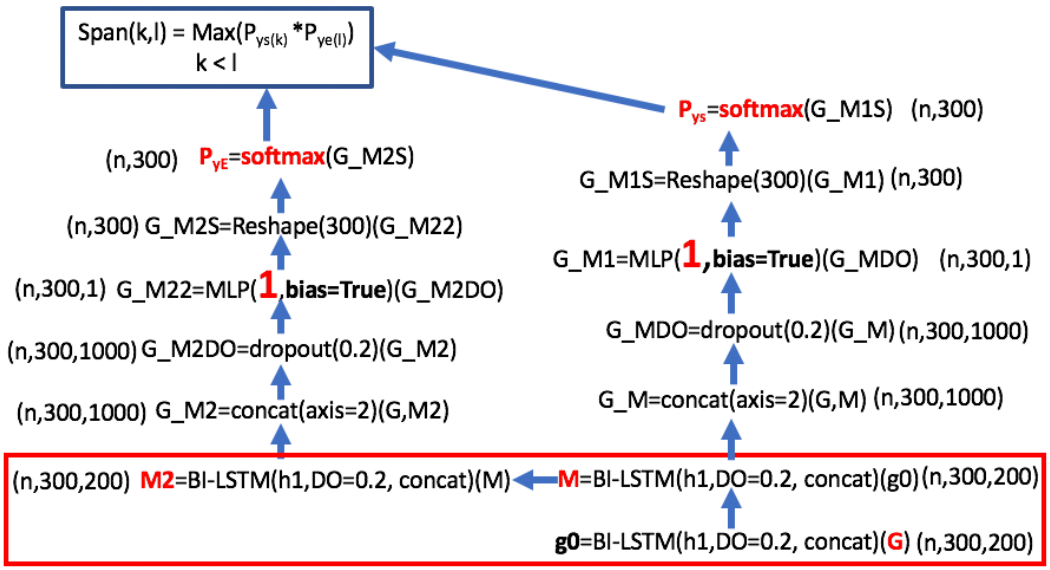


Figure 13. Architecture des couches de modélisation et de sortie selon [19]

Les couches de modélisation et de sortie de l'implémentation originale de BIDAF (Figure 13) diffèrent des précédentes essentiellement à la prédiction de la position de fin de la réponse dans le contexte. Ainsi, le réseau de prédiction de la position de début de la

réponse (Figure 14 partie de droite) est identique à celui selon BIDAF [19] (partie de droite de la Figure 13) alors que la prédiction de la position de fin de la réponse comporte les plus nombreuses étapes illustrées à la partie gauche de la Figure 14.

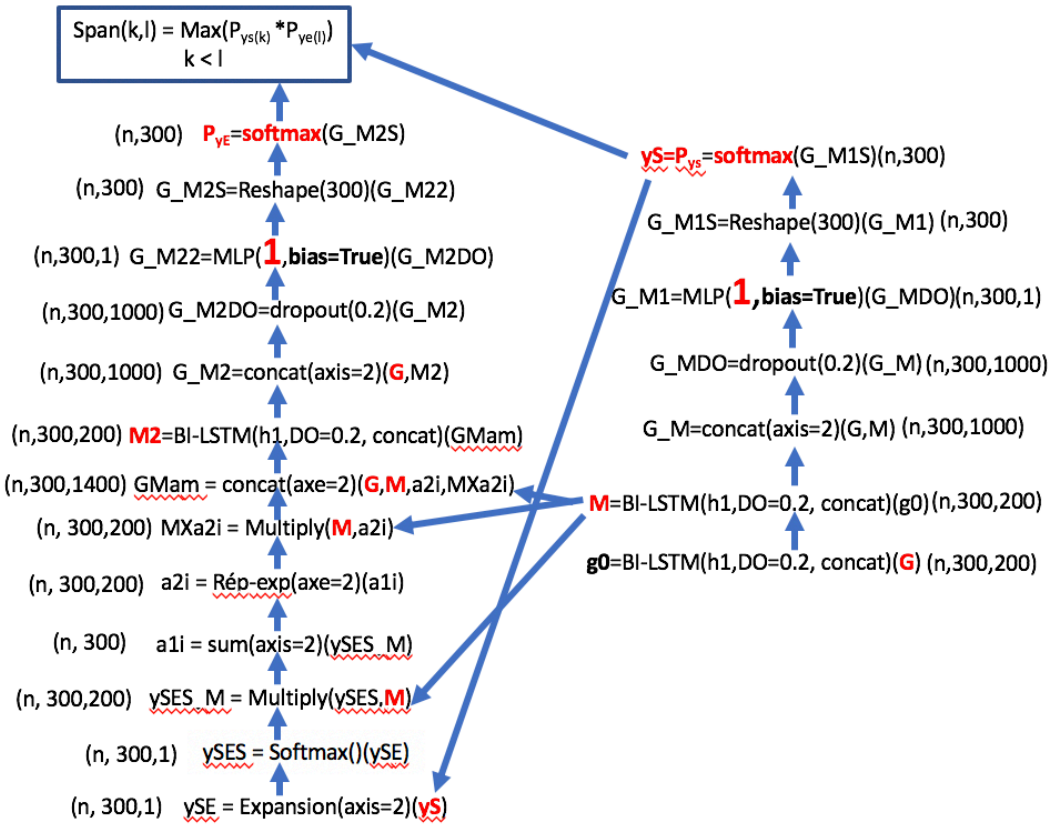


Figure 14. Architecture des couches de modélisation et de sortie selon l'implémentation de BIDAF

3.2.8 Détails additionnels d'implémentation

Tout comme pour BIDAF [19],

1. La dimension de la couche cachée du modèle (BILSTM d'encodage contextuel et de la couche de modélisation) est de 100,
2. Un dropout [54] d'un taux de 0.2 est ajouté au début de chaque réseau de convolution, aux réseaux BILSTM et avant les MLP précédents les softmax des prédictions des positions de réponse,
3. On utilise l'optimiseur Adadelta avec un taux d'apprentissage de 0.5, une taille de lots de 30 (plutôt que 60 comme BIDAF) et la minimisation des pertes par entropie croisée catégorique et

4. Pendant l'entraînement, les moyennes mobiles de tous les poids du modèle sont maintenues avec un taux de décroissance exponentielle de 0.999 (ema : exponential moving average); au moment du test, les moyennes mobiles au lieu des poids bruts sont utilisées.

3.2.9 Variantes du modèle BIDAF

Dans le contexte de la reproduction de BIDAF, trois variantes structurelles importantes ont été identifiées aux sous-sections 3.2.4 à 3.2.8 entre la description de BIDAF selon la publication [19] et son implémentation originale soit

1. Le partage ou non de paramètres, selon les stratégies tensorflow ou keras, entre les représentations de la question et du contexte aux couches d'embeddings de mots à base de caractères, de réseaux autoroutes et à la couche d'embeddings contextuels,
2. L'utilisation de dropouts externes ou internes aux couches d'embeddings contextuels et
3. Les deux structures possible de la couche de sortie, soit de la prédiction des positions de début et de fin de la réponse dans le contexte, telles illustrées aux Figures 13 et 14.

Ces trois variantes, ainsi que l'entraînement des embeddings de mots obtenus des embeddings pré-entraînés GloVe et de mots à base de caractères, ont été étudiés dans le cadre des travaux de ce mémoire portant sur la reproduction du modèle BIDAF.

3.3 Modulation cyclique du taux d'apprentissage

La méthode de modulation du taux d'apprentissage utilisée dans ce travail est celle présentée par Smith [57]. Elle comporte deux étapes, soit l'estimé d'une meilleure plage de modulation du taux d'apprentissage (TA) et l'algorithme de modulation du TA.

Smith présente brièvement une méthode d'estimation d'une meilleure plage de modulation du TA selon le modèle testé qui a été reprise et mieux élaborée à https://www.jeremyjordan.me/nn-learning-rate/. Cette méthode consiste à réaliser une

expérience simple où le TA est progressivement augmenté de façon linéaire ou exponentielle après chaque mini-batch et la perte d'entraînement est enregistrée à la fin de chaque mini-batch. On obtient alors un profile de pertes en fonction du TA tel que présenté à la Figure 15 (modifiée de https://www.jeremyjordan.me/nn-learning-rate/). Ce profile comporte trois parties distinctes : la partie de gauche où le TA est trop faible et la perte ne diminue pas, la meilleure plage centrale du TA qui permet une diminution rapide des pertes et la partie de droite où le TA est trop élevé, ce qui mène éventuellement à la divergence des pertes.

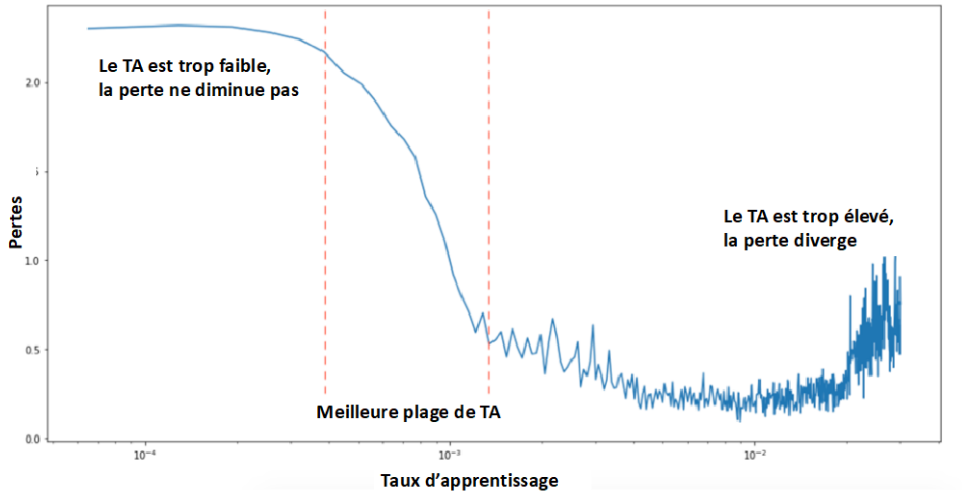


Figure 15. Estimation de la meilleure plage de taux d'apprentissage

L'algorithme de modulation du TA consiste à le moduler entre une valeur minimale (base_lr) et une valeur maximale (max_lr) à la fin de chaque mini-batch selon une fonction (mode), une taille de pas (step_size) et un mode de modulation (scale_mode), ces paramètres étant spécifiés à l'appel de la classe CyclicLR(). La taille de pas (TP) est le nombre d'itérations (ou de mini-batchs) d'un demi cycle de modulation complet entre les valeurs minimale et maximale du TA spécifiées. Le mode de modulation (scale_mode = {'cycle','iteration'}) détermine si la fonction de modulation est calculée en fonction du nombre de cycles ou d'itérations selon les équations 19 (cycle) et 20 (itération) selon la fonction de modulation utilisée (triangulaires : itération et exponentielle : cycle):

$$TA_t = base\_lr + (\max\_lr - base\_lr) * max\big(0,(1-x)\big) * scale\_fn(cycle) \quad (19)$$

$$TA_t = base_{lr} + (\max\_lr - base\_lr) * max\big(0,(1-x)\big) * scale\_fn(iteration) \quad (20)$$

Où TA_t est le TA à l'itération (ou mini-batch) *t*,

$scale\_fn(\ \ )$ est la fonction de modulation et

les variables $cycle$ et x sont calculées selon les équations 21 et 22 :

$$cycle = \lfloor 1 + itération/(2 * TP) \rfloor \quad (21)$$

$$x = \left| \frac{itération}{TP} - 2 * cycle + 1 \right| \quad (22)$$

Trois fonctions de modulation sont disponibles, soit

1. Une fonction triangulaire constante égale à 1,
2. Une fonction triangulaire dégressive (triangular2 (T2)) qui diminue de moitié la valeur maximale du TA à chaque cycle de modulation définie selon l'équation 23 :

$$scale\_fn\ = 1/\left(2.0^{(x-1)}\right) \quad (23)$$

3. Et une fonction exponentielle qui diminue par un facteur gamma exposant x la valeur maximale du TA à chaque itération selon l'équation 24 :

$$scale\_fn = gamma^x \quad (24)$$

La classe CyclicLR permet également utiliser d'autres fonctions de modulation spécifiques définies à un paramètre scale_fn à l'appel de la classe.

3.4 Implications pratiques

En résumé, ce chapitre présente les détails d'implémentation utilisés, très souvent non explicités aux sources d'information consultées, pour développer la version Keras API du modèle BIDAF des travaux de ce mémoire ainsi que ses variantes testées, dont les résultats sont discutés au Chapitre 4. Ainsi, on notera la logique du choix des paramètres de longueurs du contexte et de la question discutée à la section 3.2.3 par rapport à la réalité statistique des données et comparativement aux choix faits pour les modèles compétitifs (Tableau 5).

De même, le calcul des embeddings de mots à base de caractères n'est pas explicité à la publication de BIDAF [19] ni à son implémentation originale. Dans ce contexte, la méthodologie décrite à la section 3.2.4 a été développée et est utilisée pour la version Keras API de BIDAF des présents travaux. Également, l'entraînement des embeddings

de mots pré-entraînés GloVe et des embeddings de mots à base de caractères durant l'entraînement du modèle a été ajouté comme variante de BIDAF, ce qui n'a jamais été testé, selon les informations disponibles, pour le modèle BIDAF utilisant l'ensemble de données SQuAD 1.1.

Par ailleurs, parmi les cinq anomalies d'implémentation notées à la section 3.2.1 entre la description de BIDAF présentée à la section 2.3 et son implémentation originale, le partage ou non de paramètres, selon les approches Keras et Tenssorflow, entre les représentations du contexte et de la question aux couches autoroutes et d'encodage contextuel est étudié comme variantes de BIDAF. De même, les deux méthodes de calcul de la probabilité p^2 de prédiction de la position de la fin de la réponse dans le contexte (Figures 13 et 14) sont étudiées comme variantes de BIDAF.

Finalement, la méthodologie de modulation cyclique du taux d'apprentissage développée par Smith [57] et décrite à la section 3.3 a été testée afin d'augmenter la performance des meilleurs modèles identifiés au Chapitre 4.

Chapitre 4 Résultats et discussions

Dans ce chapitre, les résultats les plus importants des essais de reproduction du modèle BIDAF testé sur SQuAD 1.1 sont résumés. La section 4.1 discute d'essais de reproduction du modèle original de BIDAF selon la version trouvée sur le web. À la section 4.2, on résume les résultats des essais de différentes configurations de BIDAF décrites au Chapitre 3. Par la suite, la section 4.3 présente les essais de quelques variantes simplifiées de BIDAF alors que la section 4.4 discute de l'effet de la modulation du taux d'apprentissage sur la performance de certains de ces modèles.

On notera que les meilleures performances de précision de prédictions des modèles discutés dans les sections suivantes ont atteintes, au mieux, *EM* = 61.4% et *F1* = 68.4%, ce qui est inférieur à celles de BIDAF [19] de *EM* = 68.0% et *F1* = 77.3%.

4.1 Essai de l'implémentation originale de BIDAF

L'implémentation originale indiquée à la section 3.1.1 n'a pu être testée à cause des nombreux (61) problèmes rapportés (https://github.com/allenai/bi-att-flow/issues?page=1&q=is%3Aissue+is%3Aopen) ainsi que la version plus évoluée de Tensorflow disponible sur les ordinateurs utilisés dans le cadre des présents travaux. Cependant, depuis https://github.com/allenai/bi-att-flow/issues/41, une version compatible, tf1.8, a été trouvée à https://github.com/allenai/bi-att-flow/compare/master...Vimos:tf1.8 disponible en juillet 2018. L'essai de cette version a donné *EM* = 64.9% et *F1* = 75.1% avec les embeddings de mots à base de caractères et *EM* = 61.0% et *F1* = 72.3% sans ces derniers.

4.2 Essai de reproduction du modèle BIDAF

Dans cette section, on discute de l'essai des variantes de BIDAF identifiées à la section 3.2.9 dont les combinaisons de structure testées sont présentées au Tableau 6. Ainsi, par exemple, le modèle A1 comporte

1. Des couches d'embeddings de mots à base de caractères, de réseaux autoroutes et d'embeddings contextuels qui ne partagent pas leurs paramètres entre la représentation de la question (qa) et du contexte (ca),

Tableau 6. Combinaisons de variantes de structures des modèles testés

Modèle	Partage de paramètres - Stratégie	Dropout à la couche d'embeddings contextuels	Couche de modélisation et de sortie	Entraînement des embeddings
A1	Non	Interne à qa et ca	BIDAF [19]	Non
A2	Non	Interne à qa et ca	Impl. de BIDAF	Non
B1	Non	Interne à qa et ca	BIDAF [19]	Oui
B2	Non	Interne à qa et ca	Impl. de BIDAF	Oui
C1	Oui Tensorflow	Externe à qa	BIDAF [19]	Non
C2	Oui Tensorflow	Interne à qa et ca	BIDAF [19]	Non
D1	Oui Tensorflow	Externe à qa	Impl. de BIDAF	Non
D2	Oui Tensorflow	Interne à qa et ca	Impl. de BIDAF	Non
E1	Oui Tensorflow	Externe à qa	Impl. de BIDAF	Oui
E2	Oui Tensorflow	Interne à qa et ca	Impl. de BIDAF	Oui
E3	Oui Tensorflow	Externe à qa	BIDAF [19]	Oui
E4	Oui Tensorflow	Interne à qa et ca	BIDAF [19]	Oui
F1	Oui Keras	Externe à qa	BIDAF [19]	Non
F2	Oui Keras	Interne à qa et ca	BIDAF [19]	Non
G1	Oui Keras	Interne à qa et ca	Impl. de BIDAF	Non
H1	Oui Keras	Interne à qa et ca	BIDAF [19]	Oui
H2	Oui Keras	Interne à qa et ca	Impl. de BIDAF	Oui

2. Des couches d'embeddings contextuels qui possèdent un dropout interne (de 0.2) aux réseaux BILSTM pour la question et le contexte,
3. Une couche de sortie selon la publication de BIDAF [19], tel qu'illustré à la Figure 13, et
4. Des embeddings de mots pré-entraînés GloVe et de mots à base de caractères non entraînés durant l'entraînement du modèle.

Par ailleurs, au modèle E1,

1. Les couches d'embeddings de mots à base de caractères, de réseaux autoroutes et d'embeddings contextuels partagent leurs paramètres entre la représentation de la question (qa) et du contexte (ca) selon la stratégie Tensorflow,
2. Un dropout externe (de 0.2) est appliqué à la représentation de la question uniquement avant son alimentation à la couche d'embeddings contextuels,
3. La couche de sortie est selon l'implémentation de BIDAF, tel qu'illustré à la Figure 14, et
4. Les embeddings de mots pré-entraînés GloVe et de mots à base de caractères sont entraînés durant l'entraînement du modèle.

Les résultats détaillés de l'essai de ces modèles sont présentés à l'Appendice A aux sections A.1 à A.3. Ils sont résumés au Tableau 7. On notera que la mesure MySE représente la précision de prédictions simultanément correctes des positions de début, yS, et de fin, yE, de la réponse dans le contexte selon la méthode de multiplication des matrices de probabilités de yS et yE pour trouver la plage de réponse prédite dans le contexte (k, l) de valeur maximale du produit des probabilités $yS_k yE_l$ avec la restriction que $k \leq l$ pour chaque exemple tel que décrit à la section 3.1.2.

Les modèles A1, E2, E4, F2, G1 et H1 présentent les meilleures performances sur l'ensemble de développement. La seule constance entre ces six modèles est l'utilisation de dropouts internes aux réseaux RNR BILSTM des couches d'embeddings contextuels.

Tableau 7. Résumé des résultats obtenus au moyen des modèles A1 à H2 testés

Modèle	Variantes de structure				À pertes minimales Ens. développement			À la fin de l'entraînement				
								Ens. Entraîn.	Ensemble de développement			
	EE[1]	Partage de paramètres[2]	DO à emb. contextuels	Couche de modélisation et de sortie	Pertes	Ep	MySE (%)	MySE (%)	Pertes	MySE (%)	*EM* (%)	*F1* (%)
A1	Non	Non	Int. à qa & ca	BIDAF [19]	1.5333	15	54.35	63.47	1.5599	55.92	**58.78**	**66.09**
A2	Non	Non	Int. à qa & ca	Impl. de BIDAF	1.9455	20	44.30	50.50	1.9455	44.30	46.70	55.26
B1	Oui	Non	Int. à qa & ca	BIDAF [19]	1.8817	18	47.28	64.11	1.9031	47.09	49.64	58.55
B2	Oui	Non	Int. à qa & ca	Impl. de BIDAF	1.8453	19	46.79	61.12	1.8966	46.35	48.77	58.21
C1	Non	Oui Tensorflow	Externe à qa	BIDAF [19]	2.0449	16	42.46	58.44	2.0912	42.95	45.20	53.79
C2	Non	Oui Tensorflow	Int. à qa & ca	BIDAF [19]	1.8440	20	46.93	55.56	1.8440	46.93	49.54	57.96
D1	Non	Oui Tensorflow	Externe à qa	Impl. de BIDAF	2.0060	16	41.99	55.61	2.0252	43.54	45.85	55.17
D2	Non	Oui Tensorflow	Int. à qa & ca	Impl. de BIDAF	1.9016	20	44.87	51.34	1.9016	44.87	47.25	56.83
E1	Oui	Oui Tensorflow	Externe à qa	Impl. de BIDAF	1.7296	12	47.25	**72.22**	1.8862	49.53	52.14	61.09
E2	Oui	Oui Tensorflow	Int. à qa & ca	Impl. de BIDAF	1.5480	18	53.85	63.88	1.5763	55.01	**57.64**	**65.89**
E3	Oui	Oui Tensorflow	Externe à qa	BIDAF [19]	1.6298	13	52.14	**75.48**	1.8930	51.84	54.70	63.20
E4	Oui	Oui Tensorflow	Int. à qa & ca	BIDAF [19]	1.5205	13	55.20	68.81	1.5593	56.58	**59.39**	**67.36**
F1	Non	Oui Keras	Externe à qa	BIDAF [19]	1.8236	13	47.31	65.10	1.9040	48.00	50.40	59.41
F2	Non	Oui Keras	Int. à qa & ca	BIDAF [19]	1.5606	20	54.58	60.38	1.5606	54.57	**57.06**	**64.81**
G1	Non	Oui Keras	Int. à qa & ca	Impl. de BIDAF	1.5181	15	55.06	62.59	1.5415	55.53	**58.23**	**66.14**
H1	Oui	Oui Keras	Int. à qa & ca	BIDAF [19]	1.5148	15	55.93	66.46	1.5488	56.88	**59.56**	**67.09**
H2	Oui	Oui Keras	Int. à qa & ca	Impl. de BIDAF	1.7403	19	50.74	63.32	1.7833	50.63	53.16	61.94

(1) EE : Entraînement des embeddings (2) Couches d'embeddings, de réseaux autoroutes & d'embeddings contextuels

En fait, on notera l'effet régularisateur (prévention de la surmodélisation (*overfitting*) à l'entraînement et augmentation des précisions de prédiction avec l'ensemble de développement) de ces dropouts internes [54] aux couches d'embeddings contextuels des modèles E2 et E4 comparativement aux modèles E1 et E3, respectivement, ces derniers ne possédant qu'un seul dropout externe aux couches d'embeddings contextuels.

Depuis cette constance (dropouts internes aux embeddings contextuels), on observe les points suivants.

1. La couche de sortie selon l'implémentation de BIDAF (Figure 14) n'améliore pas la performance du modèle A2 comparativement à celle du modèle A1 qui comporte une couche de sortie selon BIDAF [19]. On note le même effet pour le modèle H2 comparativement au modèle H1. Cependant, ceci dépend des autres composantes du modèle comme noté ci-après pour les modèles E2 et E4, et F2 et G1.
2. Aux modèles B1 et B2, on note que l'entraînement des embeddings durant l'entraînement du modèle sans partage de paramètres ne suffit pas à améliorer la performance du modèle.
3. De même, le partage de paramètres selon Tensorflow sans entraînement des embeddings n'améliore pas la performance des modèles C2 et D2.
4. Cependant le partage de paramètres selon Keras, avec (modèle H1) ou sans entraînement des embeddings (modèles F2 et G1, donc quel que soit la couche de sortie) améliore leurs performances.
5. L'entraînement des embeddings combiné au partage de paramètres selon Tensorflow, quel que soit la couche de sortie, améliore la performance des modèles E2 et E4.

En résumé, le partage de paramètres, selon Tensorflow avec entraînement des embeddings durant l'entraînement du modèle, et selon Keras, avec ou sans entraînement des embeddings dans ce cas, combiné à l'effet régularisateur des dropouts internes aux

réseaux RNR BILSTM des embeddings contextuels donnent des modèles présentant les meilleures performances.

Ces résultats peuvent s'expliquer, au moins en partie, en considérant les deux points suivants. En premier lieu, l'entraînement des embeddings de mots durant l'entraînement du modèle permet d'associer aux mots du vocabulaire ne possédant pas d'embeddings pré-entraînés GloVe (section 3.2.4 : 20.24% des mots du vocabulaire) présents dans le contexte et/ou la question, des embeddings d'entrainement, ce qui augmenterait la capacité du modèle. En fait, on notera qu'à la revue des modèles les plus performants testés sur SQuAD au Chapitre 2, incluant BIDAF, l'entraînement des embeddings durant l'entraînement des modèles ne semble pas avoir été étudié.

Par ailleurs, le partage de paramètres aux couches d'embeddings de mots à base de caractères, de réseaux autoroutes et d'embeddings contextuels entre les représentations de la question et du contexte favoriserait l'appariement de ces deux représentations aux couches subséquentes et donc l'amélioration de la capacité et de la performance du modèle.

4.3 Pertes et performances des modèles testés

Tous ces modèles présentent des dynamiques plus ou moins différentes. Il est intéressant de comparer leurs valeurs minimales des pertes et maximales des MySE, respectivement, pour l'ensemble de développement tel qu'illustré au Tableau 8 classées selon la valeur de MySE (ou *EM*) de l'ensemble de développement en fin d'entraînement.

On observe que, pour la majorité des modèles, les valeurs maximales et en fin d'entraînement de MySE sont très comparables. L'exception du modèle E3 provient de ses pertes en fin d'entraînement (1.8930) plus élevées que les valeurs minimales notées à MySE maximales (1.6978) indiquant sa divergence en fin d'entraînement.

La Figure 16 illustre une relation linéaire entre les précisions de prédiction MySE, à pertes minimales (Figure 16A), et MySE maximales (Figure 16B) en fonction des pertes. En fait,

on observe cette relation linéaire entre MySE et les pertes pour tous les modèles précédents, et discutés plus loin, durant l'entraînement comme le présente la Figure 17 pour les Modèles H1, C1 et E1. Évidemment, les modèles possédant les meilleures capacités atteignent de plus hautes performances de précision de prédictions, ce qui entraînent des pertes finales moindres.

Tableau 8. Pertes minimales et MySE maximales sur l'ensemble de développement des modèles A1 à H2

Modèle	Pertes minimales			MySE maximale			Valeurs finales - développement			
	Pertes	Ep	MySE (%)	MySE (%)	Ep	Pertes	MySE (%)	Pertes	EM (%)	*F1* (%)
H1	1.5148	15	55.93	56.89	20	1.5488	56.89	1.5488	**59.56**	**67.09**
E4	1.5205	13	55.20	56.88	18	1.5488	56.58	1.5593	**59.39**	**67.36**
A1	1.5333	15	54.35	55.92	20	1.5599	55.92	1.5599	**58.78**	**66.09**
G1	1.5181	15	55.06	55.53	20	1.5415	55.53	1.5415	**58.23**	**66.14**
E2	1.5480	18	53.85	55.01	20	1.5763	55.01	1.5763	57.64	65.89
F2	1.5606	20	54.58	54.58	20	1.5606	54.58	1.5606	57.06	64.81
E3	1.6298	13	52.14	52.37	15	1.6978	51.84	1.8930	54.70	63.20
H2	1.7403	19	50.74	50.74	19	1.7403	50.63	1.7833	53.16	61.94
E1	1.7296	12	47.25	49.90	16	1.7573	49.54	1.8862	52.14	61.09
F1	1.8236	13	47.31	48.69	18	1.8906	48.00	1.9040	50.40	59.41
B1	1.8817	18	47.28	47.48	19	1.8975	47.10	1.9031	49.64	58.55
C2	1.8440	20	46.93	46.93	20	1.844	46.93	1.8440	49.54	57.96
B2	1.8453	19	46.79	46.79	19	1.8453	46.36	1.8966	48.77	58.21
D2	1.9016	20	44.87	44.87	20	1.9016	44.87	1.9016	47.25	56.83
A2	1.9455	20	44.30	44.30	20	1.9455	44.30	1.9455	46.70	55.26
D1	2.0060	16	41.99	43.55	20	2.0252	43.55	2.0252	45.85	55.17
C1	2.0449	16	42.46	42.95	20	2.0912	42.95	2.0912	45.20	53.79

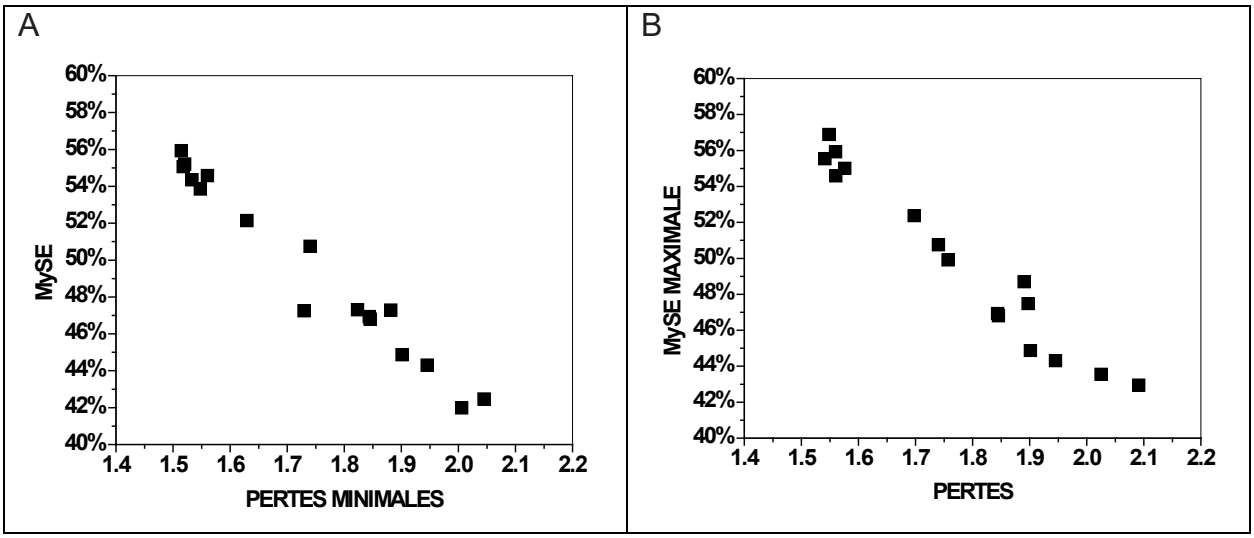


Figure 16. Précisions à pertes minimales et maximales en fonction des pertes des modèles A1 à H2

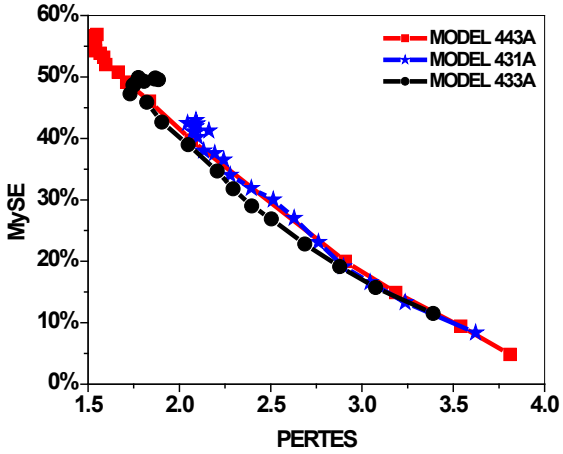


Figure 17. Précisions de prédictions en fonction des pertes des modèles H1, C1 et E1

4.4 Modèles modifiés

Il est possible d'apporter un très grand nombre de modifications aux meilleurs modèles présentés à la section précédente afin d'essayer d'augmenter leur performance. Dans cette section, on discute de l'effet de quelques modifications apportées à certains de ceux-ci sur leur performance.

4.4.1 Utilisation d'unités RNR de type GRU

Certaines études discutées aux sections 2.2 et 2.3 soulignent l'effet positif de substituer des unités RNR de type GRU aux unités LSTM sur la performance de certains modèles,

dont SAN [30], R-NET [32] et BIDAF avec ELMo [36] et BIDAF avec auto-attention [37]. Dans ce contexte, cette stratégie a été expérimentée pour les modèles J1 et J2 qui sont le modèle G1 auxquels on a substitué des unités RNR de type GRU à la couche d'embeddings contextuels uniquement, pour le modèle J1, ainsi qu'aux unités BILSTM de la couche d'attention bidirectionnelle de BIDAF (section 3.1.6) pour le modèle J2.

Tel que résumé au Tableau 9, on observe que le Modèle J1 présente une performance similaire à celle du modèle G1 alors que l'ajout d'unités GRU aux couches d'embeddings contextuels et d'attention bidirectionnelle au modèle J2 diminue significativement sa capacité.

Tableau 9. Résumé des résultats des modèles J1 à J4

Modèle	EE[1]	Partage de param.	Couche de modélisation et de sortie	Modifi-cation	À la fin de l'entraînement (%)			
					Ens. d'entraî-nement	Ensemble de développement		
					MySE	MyS E	*EM*	*F1*
G1	Non	Oui Keras	Implé. de BIDAF		62.59	55.53	**58.23**	**66.14**
J1 (G1)	Non	Oui Keras	Implé. de BIDAF	GRU à emb. context.	62.87	54.67	**57.28**	**65.52**
J2 (G1)	Non	Oui Keras	Implé. de BIDAF	GRU à emb. context. & modél.	49.03	45.30	47.90	57.63
A1	Non	Non	BIDAF [19]		63.47	55.92	**58.78**	**66.09**
J3 (A1)	Non	Non	BIDAF [19]	SEMABC[2]	60.25	53.90	56.79	65.46
H1	Oui	Oui Keras	BIDAF [19]		66.46	56.88	**59.56**	**67.09**
J4 (H1)	Oui	Oui Keras	BIDAF [19]	SEMABC[2]	67.72	57.38	**60.20**	**67.69**

(1) EE : Entraînement des embeddings
(2) SEMABC : sans embeddings de mots à base de caractères

4.4.2 Effet des embeddings de mots à base de caractères

Il est intéressant de tester l'effet des embeddings de mots à base de caractères sur certains des meilleurs modèles présentés à la section 4.2. Ainsi les modèles J3 et J4 sans embeddings de mots à base de caractères sont les modèles A1 et H1, respectivement.

Dans le premier cas (modèles J3 vs A1), on observe au Tableau 9 une certaine diminution de la capacité du modèle J3 avec l'absence des embeddings de mots à base de caractères qui se traduit par une perte de 2% de précisions de prédictions sur l'ensemble de développement à la fin de l'entraînement. Dans le second cas (modèles J4 vs H1), cependant, l'absence d'embeddings de mots à base de caractères, semble légèrement favorable à la performance du modèle J4 par rapport à celle du modèle H1. Il est possible que cet effet provienne de l'entraînement des embeddings de mots GloVe durant l'entraînement du modèle J4 qui compenserait pour l'absence d'embeddings de mots à base de caractères.

4.4.3 Modèles simplifiés

Une des étapes les plus lourdes, en termes d'intensité de manipulation de tenseurs, du modèle BIDAF est le calcul de la matrice de similarité S (section 2.3 : équation 2), tel que décrit à la section 3.2.6, depuis la sortie de la couche d'embeddings contextuels (section 3.2.5). Tel que discuté à la section 2.3, l'objectif de S, jusqu'aux attentions C2Q et Q2C, est de fusionner et de lier l'information des mots du contexte et de la question. Tel qu'illustré à la Figure 12, ce calcul de S implique :

1. L'expansion des embeddings contextuels ca et qa à des tenseurs de dimensions (n,300,30,200),
2. La multiplication de ces tenseurs pour donner cqM de dimensions (n,300,30,200),
3. La concaténation des trois tenseurs ca et qa expansés et cqM pour donner un tenseur cqConcat de dimensions (n,300, 30,600),
4. L'alimentation de ce dernier tenseur à un MLP sans activation de dimension cachée de 1 pour donner le tenseur S0 de dimensions (n,300,30,1) et le redimensionnement de S0 à S de dimensions (n,300,30).

Dans la présente section, on étudie l'effet sur la capacité de certains modèles présentés à la section 4.2 (partage de paramètres selon la stratégie Keras) de simplifier cette opération en calculant directement S par la multiplication des embeddings contextuels ca et qa (transposée).

Les quatre modèles K1 à K4 testés sont, respectivement, les modèles F2 (sans entraînement des embeddings et avec couche de sortie selon BIDAF [19]), H1 (F2 avec entraînement d'embeddings), G1 (sans entraînement des embeddings et avec couche de sortie selon l'implémentation de BIDAF) et H2 (G1 avec entraînement d'embeddings) simplifiés par rapport au calcul de S. Leurs performances sont comparées à celles des modèles non simplifiés au Tableau 10.

Tableau 10. Résumé des résultats des modèles KX avec partage de paramètres selon Keras

Modèle	EE[1]	Couche de modélisation et de sortie	Modification	À la fin de l'entraînement (%)			
				Ens. d'entraînement	Ens. de développement		
				MySE	MySE	*EM*	*F1*
F2	Non	BIDAF [19]		60.38	54.57	57.06	64.81
K1	Non	BIDAF [19]	Calcul de S simplifié (CSS)	60.48	55.57	**58.23**	**65.84**
H1	Oui	BIDAF [19]		66.46	56.88	**59.56**	**67.09**
K2	Oui	BIDAF [19]	CSS	66.60	56.83	**59.55**	**67.26**
G1	Non	Implémentation de BIDAF		62.59	55.53	**58.23**	**66.14**
K3	Non	Implémentation de BIDAF	CSS	58.41	53.96	56.58	65.18
H2	Oui	Implémentation de BIDAF		63.32	50.63	53.16	61.94
K4	Oui	Implémentation de BIDAF	CSS	64.72	56.32	**59.12**	**66.60**

(1) : Entraînement des embeddings

Tous les modèles Ki simplifiés présentent, à l'epoch 1, des pertes 30% plus faibles et des précisions de prédictions deux fois supérieures à celles obtenues au moyen de tous les modèles discutés précédemment. La capacité initiale des modèles KX est donc significativement améliorée par la simplification du calcul de S par rapport aux modèles plus compliqués discutés aux sections précédentes. Cette accélération initiale des modèles KX, cependant, n'améliore généralement pas leurs performances finales.

De façon surprenante, cette simplification, combinée à l'entraînement des embeddings et à la structure de la couche de modélisation et de sortie selon l'implémentation de BIDAF (Figure 14), provoque, pour le modèle K4, une augmentation significative (+6%) de sa capacité comparativement au modèle H2 non simplifié.

Il est intéressant de souligner que les modèles K2 et K4, qui présentent les meilleures performances avec l'ensemble de développement comparables aux meilleures performances des modèles du Tableau 7, comportent l'entraînement des embeddings. Par ailleurs, on notera que, sur un même ordinateur et dans tous les cas, les modèles simplifiés tournent deux fois plus rapidement que ceux non simplifiés, ce qui, évidemment, provient de la simplification du calcul de la matrice de similarité S.

4.5 Modulation du taux d'apprentissage

Tous les modèles précédents ont été testés au moyen de l'optimiseur Adadelta dont le taux d'apprentissage (TA) a été fixé à 0.5 selon la publication originale du modèle BIDAF [19]. De nombreux auteurs [56-57 et autres] soulignent la très grande importance de cet hyper paramètre à l'optimisation de modèles d'apprentissage machine. Dans cette dernière section du Chapitre 4, on évalue l'effet de quelques stratégies de modulation du TA sur la performance de certains des meilleurs modèles identifiés aux sections précédentes.

Un certain nombre d'optimiseurs ont été testés à différents TA (Adadelta : 0.1, 0.3, 0.5, 0.9, 2.0, 3.0, 5.0 et 8.0; stochastic gradient descent (SGD) : 0.01, 0.1 et 0.2; Adams : 0.0002, 0.0005, 0.001, 0.002 et 0.004) avec quelques-uns des modèles précédents sans

obtenir de performances remarquables (résultats non présentés). De même quelques stratégies simples (diminution du TA à des taux de 0.93 à 0.99 à l'atteinte d'une valeur minimale des pertes de l'ensemble de développement) et plus élaborées (step-decay [56]) ont été testées avec l'optimiseur Adadelta à un TA initial de 0.5 également sans grand succès (résultats non présentés).

La stratégie de variation cyclique du TA [57] présentée à la section 3.3, cependant, a permis d'obtenir quelques résultats intéressants lorsqu'utilisée avec deux des modèles ayant démontré les meilleures performances aux sections précédentes, soit

1. Le modèle simplifié K2 (partage de paramètres selon Keras, entraînement des embeddings et couche de sortie selon la publication originale de BIDAF[19]) (Tableau 10 : *EM* = 59.55% et *F1* = 67.26%) et
2. Le modèle H1 (Tableau 7 : *EM* = 59.56% et *F1* = 67.09%).

Tel que discuté à la section 3.3, cette stratégie implique le choix de valeurs appropriées d'un certain nombre de paramètres, soit des TA minimal et maximal, de la taille de pas (TP), du mode ou de la fonction de modulation (triangulaire, triangulaire dégressive (triangulaire2) et exponentielle décroissante). Évidemment le choix de meilleures valeurs de ces paramètres est vaste.

Dans ce contexte, la méthode présentée à première partie de la section 3.3 a été utilisée pour estimer de meilleures plages de variation cyclique du TA pour ces modèles tel que discuté à la section A.4.1. Pour la TP, on a utilisé des multiples du nombre d'itérations par epoch (dans le cas présent : 87,599/30 = 2920 itérations par epoch, 87,599 étant le nombre d'exemples de l'ensemble d'entraînement et 30, la taille d'un lot) tel que suggéré par Smith [57].

Selon cette approche et la Figure A4, on observe une meilleure plage de modulation du TA pour le modèle K2 de 0.11 à 1.2. De même, depuis la Figure A5, on observe une meilleure plage de modulation du TA pour le modèle H1 de 0.14 à 1.2.

4.5.1 Effet de variations cycliques du TA sur le modèle K2

À la première série d'essais (S1.1 à S1.3), le mode de modulation triangular2 (équation 23) (T2) a été testé sur 20 à 24 epochs avec des cycles multiples de modulation du TA selon les paramètres présentés au Tableau 11, soit des valeurs minimale et maximale du TA selon la section précédente et des valeurs de TP, tel que suggéré à [57], fixées à 3 à 4 fois le nombre d'itérations par epoch (2920). Les principaux résultats de ces essais sur l'ensemble de développement sont illustrés à la Figure 18, qui inclue ceux obtenus avec le modèle original K2, et résumés au Tableau 11.

Aux Figures 18B et 18D, on observe l'effet positif de la modulation du TA sur la performance du modèle, soit la plus rapide diminution des pertes et augmentation des niveaux de précision des prédictions en comparaison du modèle K2 comme le rapporte Smith [57]. Ceci se traduit par de faibles augmentations des précisions de prédiction à pertes minimales, à valeur maximale de MySE et en fin d'entraînement tel qu'illustré au Tableau 11. Par ailleurs, on note à la fin des essais S1.2 et S1.3 des augmentations non négligeables des pertes comparativement aux valeurs minimales pour ces essais ainsi qu'aux pertes finales obtenues à l'essai du modèle K2, ce qui indique la divergence du modèle provoquée par la modulation du TA et un effet plateau des précisions de prédiction pour S1.2 et S1.3.

En fait, si on regarde de plus près les profils de précisions de prédictions au-delà de l'epoch 4 selon les cycles de modulation du TA utilisés, tel qu'illustré à la Figure 19, on observe que les meilleures précisions de prédictions ne coïncident pas toujours avec la fin d'un cycle contrairement à ce que rapporte Smith [57]. Ces résultats suggèrent que, pour le modèle K2, les cycles multiples utilisés, entre autres aux essais S1.2 et S1.3, seraient possiblement trop courts.

Tableau 11. Paramètres et résultats des séries d'essais de modulation du TA du modèle K2

Essai	Paramètres					Pertes minimales Ens. de développement			Valeur max. de MySE Ens. de développement			À la fin de l'entraînement Ens. de développement			
	TA min	TA max	TP		Nb epochs	Pertes	Epoch	MySE (%)	MySE (%)	Epoch	Pertes	Pertes	MySE (%)	*EM* (%)	*F1* (%)
				X2920											
K2	TA = 0.5				20	1.5050	18	56.49	57.24	19	1.5082	1.5292	56.83	**59.55**	**67.26**
Première série d'essais															
S1.1	0.10	1.0	8760	3	20	1.4997	16	56.75	57.24	20	1.5200	1.5200	57.24	**60.07**	**67.45**
S1.2	0.15	1.2	11680	4	20	1.5045	10	56.55	57.99	16	1.5387	1.5733	57.80	**60.28**	**67.77**
S1.3	0.15	1.2	11680	4	24	1.5007	15	56.91	57.55	17	1.5089	1.5657	57.50	**60.30**	**67.76**
Deuxième série d'essais															
S2.1	0.15	1.2	10220	3.5	7	1.5117	7	55.24	55.24	7	1.5117	1.5117	55.24	58.00	66.40
S2.2	0.15	1.2	11680	4	8	1.5120	7	55.09	56.02	8	1.5131	1.5131	56.02	58.95	66.60
S2.3	0.15	1.2	16060	5.5	11	1.5160	11	57.33	57.33	11	1.5160	1.5160	57.33	ND	ND
S2.4	0.15	1.2	17520	6	12	1.4847	8	56.30	58.36	12	1.4954	1.4954	**58.36**	**61.18**	**68.19**
S2.5	0.15	1.2	18980	6.5	13	1.4882	12	58.07	58.07	12	1.4882	1.5106	57.71	**60.45**	**68.07**
S2.6	0.15	1.2	23360	8	16	1.4804	12	57.44	58.33	14	1.5251	1.5299	58.13	**60.86**	**68.28**
S2.7	0.15	1.2	26280	9	18	1.4778	12	56.65	58.09	18	1.5701	1.5701	58.09	**60.90**	**68.07**
S2.8[1]	0.15	1.8	17520	6	12	1.4875	10	57.03	57.34	12	1.4948	1.4948	57.34	**60.31**	**67.88**
S2.9[1]	0.3	2.0	17520	6	12	1.4767	8	56.12	58.11	12	1.5027	1.5027	58.11	**60.94**	**67.96**
S2.4e	0.10	4.0	17520	6	12	1.4924	8	56.32	58.28	12	1.5186	1.5186	**58.28**	**61.03**	**68.33**
S2.4f	0.10	5.0	17520	6	12	1.4989	9	57.29	58.70	12	1.5315	1.5315	**58.70**	**61.42**	**68.46**
S2.4g	0.10	6.0	17520	6	12	1.5038	9	56.55	57.70	11	1.5289	1.5492	57.37	**60.16**	**67.91**

(1) Mode exponentiel

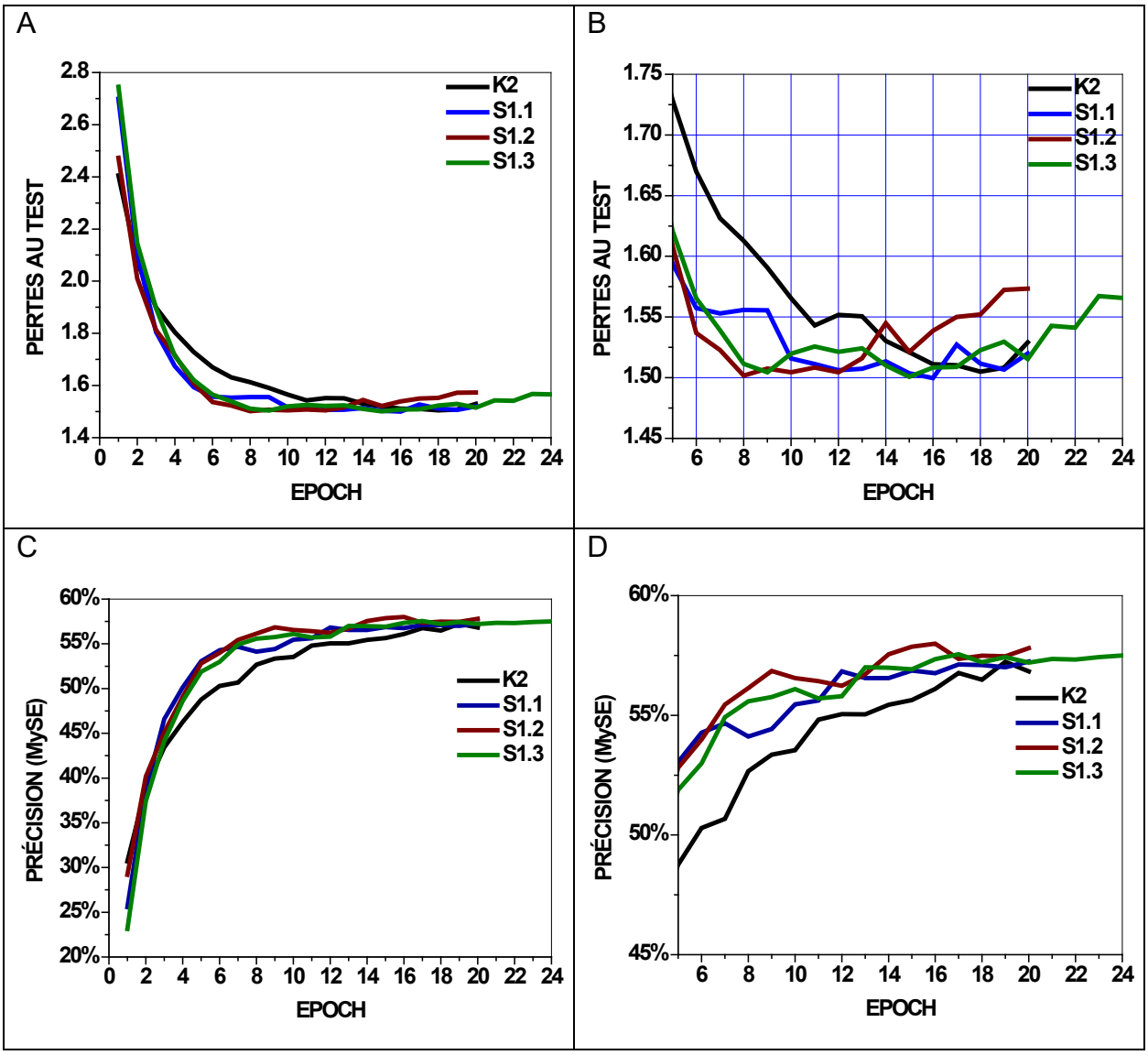


Figure 18. Résultats de la 1er série d'essais du modèle K2 avec modulation du TA

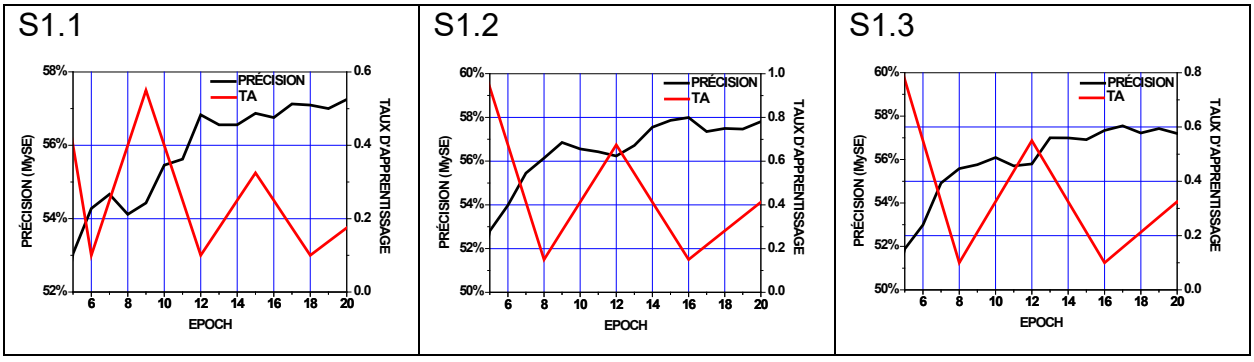


Figure 19. Précisions de prédictions et modulation du TA pour les essais S1.1 à S1.3

Dans ce contexte, une seconde série d'essais a été réalisée en mode T2, avec des TA minimal et maximal de 0.15 et 1.2 identiques mais sur un seul cycle de modulation du TA avec augmentation de la TP d'un essai à l'autre. De plus, deux essais (S2.8 et S2.9) ont été réalisés en mode exponentiel (équation 24; gamma = 0.99997 [57]) à une TP de 17520, et des TA minimal et maximal de 0.15 et 1.8 et 0.3 et 2.0, puisque la fonction exponentielle atténue le TA maximal, ce que donnait des TA maximaux effectifs de 1.12 et 1.30 à l'epoch 6. Les paramètres de ces essais et leurs résultats sont présentés au Tableau 11, dont le nombre d'epochs de l'essai qui correspond à la fin du cycle de modulation selon la valeur de la TP.

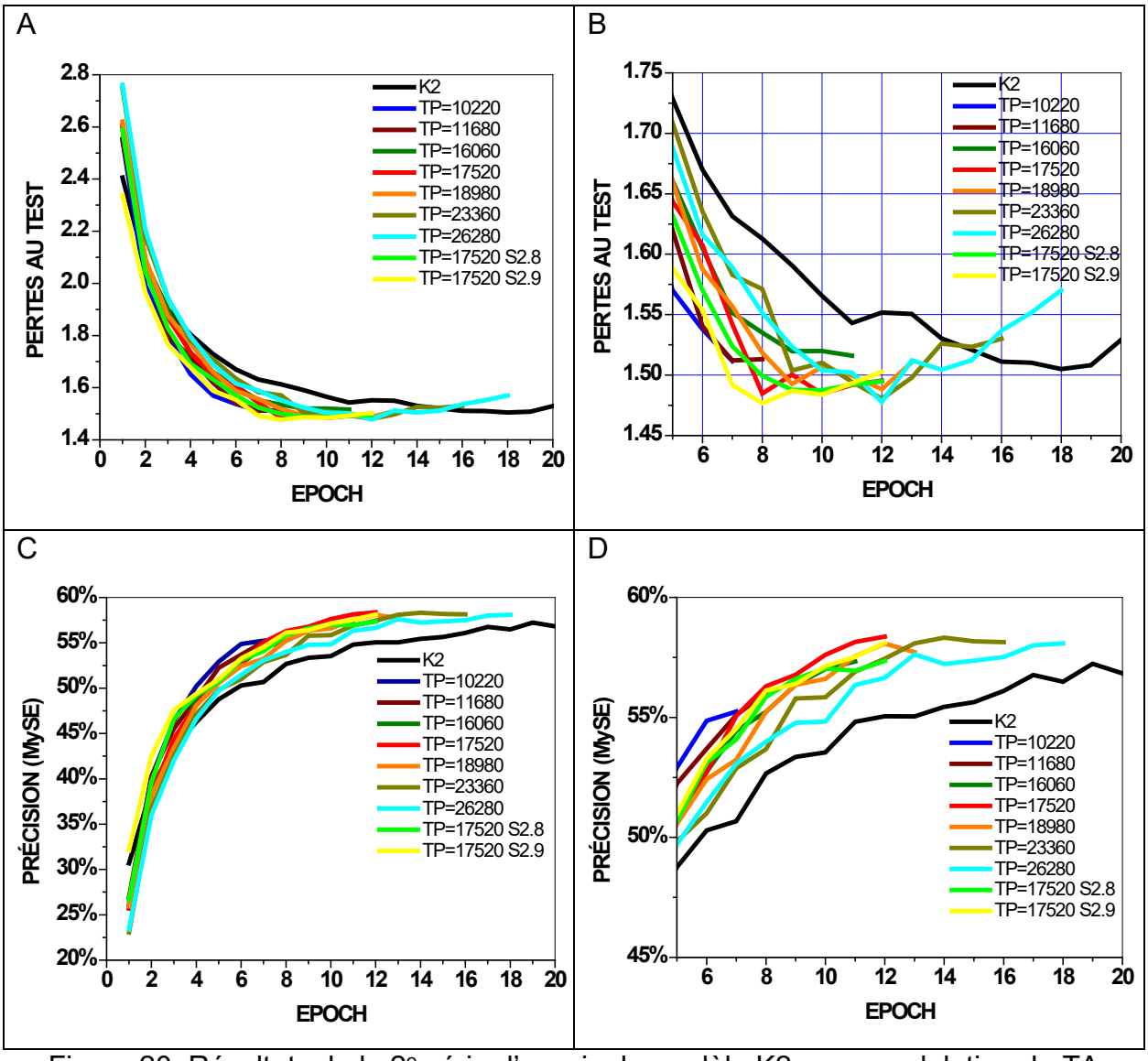


Figure 20. Résultats de la 2e série d'essais du modèle K2 avec modulation du TA

Encore ici, on note aux Figures 20B et 20C, l'effet positif de la modulation du TA sur la performance du modèle, soit une plus rapide et plus significative diminution des pertes et augmentation des niveaux de précisions des prédictions en comparaison du modèle K2. Dans ce cas-ci, cependant, les augmentations de précisions de prédictions en fin d'entraînement sont non négligeables, en particulier pour l'essai S2.4 à une TP de 17520.

Les essais réalisés au moyen du mode exponentiel avec les paramètres utilisés donnent des résultats similaires, mais un peu plus faibles en termes de précision de prédictions, que ceux faits avec le mode T2 à une TP de 17520 (S2.4). Par ailleurs, on notera que les profils des pertes de ces trois derniers essais (S2.4, S2.8 et S2.9), ainsi que leurs pertes minimales et finales, sont plus faibles et à plus faibles epochs (8-10) que celles des autres essais de cette série incluant K2, ce qui suggère une plus faible divergence du modèle à ces conditions. On tiendra compte de ces résultats à la série d'essais S3.X discutée à la section A.4.3.

On observe que, pour les essais réalisés à des TP supérieures à 17520 (S2.5 à S2.7), les pertes finales augmentent au niveau de celles du modèle K2 original et au-delà à une TP de 26280. On notera qu'augmenter le nombre d'epochs à une TP donnée n'améliore pas la performance (voir l'essai S2.4a au Tableau A1) puisque ceci provoque évidemment le début d'un nouveau cycle de modulation du TA, et il est rapporté [57] que de meilleurs résultats sont obtenus en fin de cycle. Ces résultats diffèrent de ceux de Smith [57] qui indique que la performance des réseaux qu'il a testés au moyen de la méthode de modulation du TA était peu affectée par la valeur de la TP.

Aux valeurs des paramètres utilisées, ces essais à un seul cycle, en particulier à une TP de 17520, présentent donc de meilleures performances que ceux à cycles multiples. Quelques autres essais autour de ce dernier point ont été réalisés pour tenter d'améliorer la performance du modèle, dont les résultats sur l'ensemble de développement sont résumés à la section A.4.2. Ces essais n'ont pas permis d'améliorer cette performance.

Cependant, d'autres essais subséquents ont été réalisés toujours à une TP de 17520 mais en mode exponentiel, à un TA minimal de 0.1 et à plus hauts TA maximaux, soit 2.4e: 4.0 (TA maximal effectif de 2.40), 2.4f : 5.0 (TA maximal effectif de 2.99) et 2.4g (TA maximal effectif de 3.58) dont les résultats sont présentés au Tableau 11. Ces trois derniers essais présentent des taux de diminution des pertes et d'augmentation de précision de prédiction encore plus élevés que l'essai S2.4 mais des précisions de prédiction finales de l'ordre de celles de l'essai S2.4.

En résumé, aux valeurs des paramètres utilisées, les résultats du Tableau 11 suggèrent qu'une modulation à des TP de 8760 à 11680 ne permet pas une amélioration significative de la performance du modèle K2 alors que les essais à une TP de 17520, présentent une certaine amélioration mais limitée.

Afin de tenter de briser ce plafond de précisions de prédictions, une dernière stratégie de modulation à deux et trois étapes a donc été expérimentée dont les paramètres et les résultats pour l'ensemble de développement sont présentés à la section A.4.3 et résumés au Tableau A2. Ces derniers essais présentent des performances comparables à celles obtenues aux essais S2.4 et S2.4f mais au moyen de stratégies de modulation du TA plus compliquées et en un nombre d'epochs plus élevé.

4.5.2 Effet de variations cycliques du TA sur le modèle H1

Des résultats obtenus à la section précédente, les premiers essais du modèle H1 effectués avec modulation du TA l'ont été avec des profils monocycles dont les paramètres et les résultats sont présentés au Tableau 12 et illustrés à la Figure 21.

Tout comme pour tous les essais avec le modèle K2 et tel qu'illustré aux Figures 21A à 21D, la modulation monocycle du TA accélère l'apprentissage et améliore la performance du modèle H1 aux conditions utilisées particulièrement en mode T2, à des TA minimal et maximal de 0.15 et 1.0 et à une TP de 26280, incluant la précision des prédictions dont l'augmentation, cependant, demeure encore limitée à 1.8%.

Tableau 12. Paramètres et résultats de la 1ère série d'essais de modulation du TA du modèle H1

Essai	Paramètres						Pertes minimales Ens. de développement			Valeur max. de MySE Ens. de développement			À la fin de l'entraînement Ens. de développement			
	TA min	TA max	TP	X 2920	Mode	Nb ep.	Pertes	Ep.	MySE (%)	MySE (%)	Ep.	Pertes	Pertes	MySE (%)	EM (%)	*F1* (%)
443A	TA = 0.5					20	1.5148	15	55.93	56.88	20	1.5488	1.5488	56.88	**59.56**	**67.09**
S4.1	0.15	1.0	1.4738	12	T2	12	1.4739	12	57.67	57.67	12	1.4739	1.4739	57.67	**60.40**	**68.21**
S4.2	0.15	1.0	1.4724	13	T2	14	1.4724	13	57.27	57.88	13	1.5006	1.5006	57.88	**60.81**	**68.26**
S4.3	0.15	1.0	1.4649	12	T2	16	1.4650	12	57.41	58.20	16	1.4897	1.4897	58.20	**60.99**	**68.26**
S4.4	0.15	1.0	1.4735	13	T2	18	1.4735	13	57.16	58.69	17	1.4965	1.5189	58.58	**61.40**	**68.32**
S4.5	0.15	1.2	1.4720	15	T2	18	1.4721	15	57.61	58.05	17	1.5243	1.5264	57.84	**60.67**	**67.96**
S4.6	0.15	1.0	1.4860	16	T2	20	1.4861	16	57.58	57.88	20	1.5518	1.5518	57.88	**60.61**	**68.01**
S4.7	0.15	2.0[1]	1.4623	12	Exp	16	1.4624	12	57.31	58.35	15	1.5113	1.5144	58.20	**60.99**	**68.38**
S4.8	0.15	2.2[2]	1.4889	10	Exp	18	1.4889	10	56.62	58.07	15	1.5053	1.5502	58.05	**60.89**	**68.25**

(1) TA maximal effectif = 1.067
(2) TA maximal effectif = 1.081

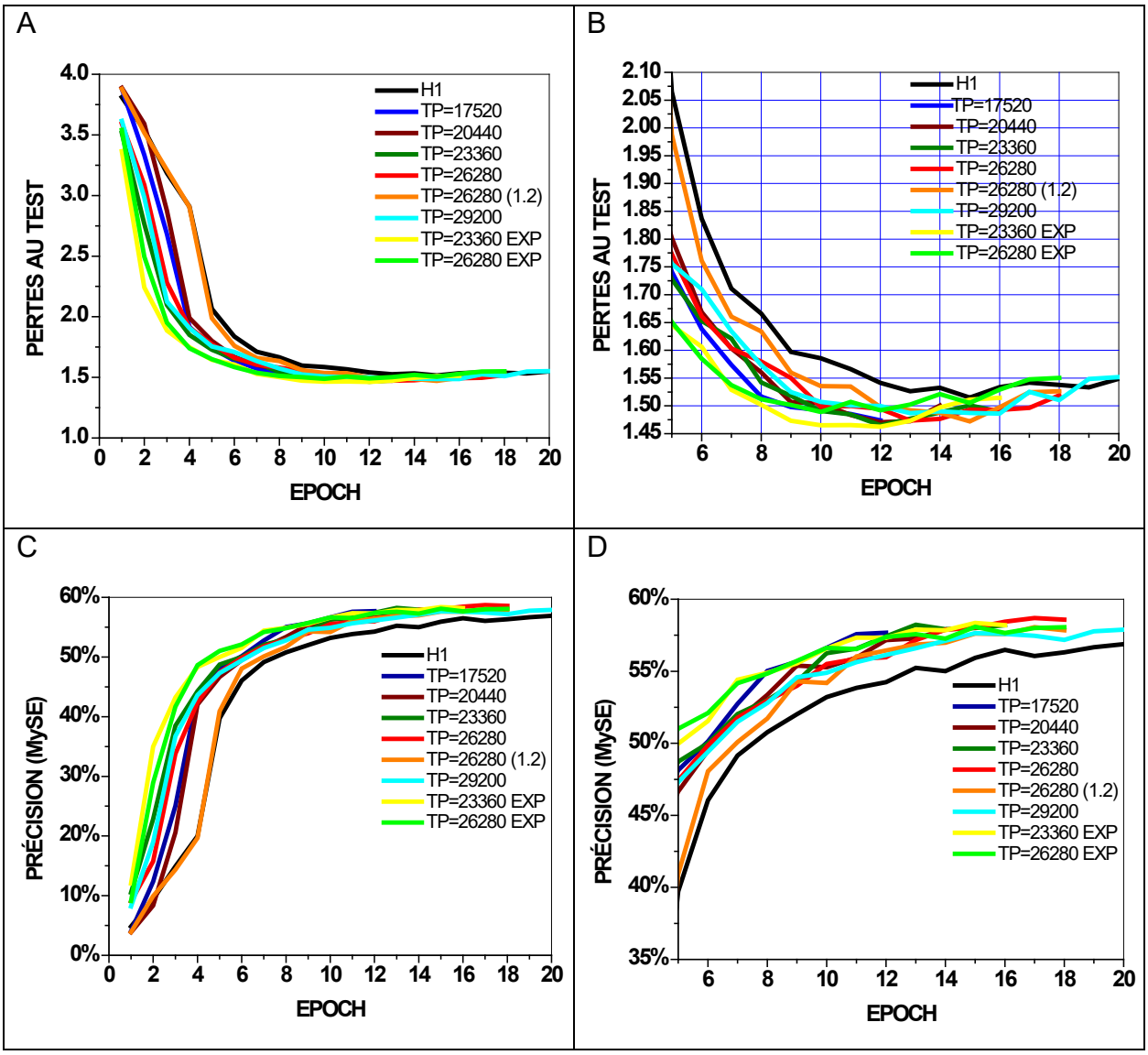


Figure 21. Résultats de la 1e série d'essais du modèle H1 avec modulation du TA

Le modèle H1 est plus complexe que le modèle K2 comme indiqué à la section 4.4.3. Ceci se traduit pour le premier, entre autres, par une dynamique plus lente (Figures 21A à 21D) comparativement au Modèle K2 (Figures 20A à 20D), incluant à l'epoch 1 avec des pertes significativement plus élevées (3.8 vs 2.7) et des précisions de prédictions (5% vs 25%) significativement plus basses que pour le modèle K2 comme l'illustre la Figure 20. Cette dynamique initiale plus lente du modèle H1 peut, au moins en partie, expliquer les plus hautes TP requises pour améliorer sa performance comparativement au modèle K2.

D’un autre côté, les performances des deux modèles avec modulation du TA, en ce qui concerne les profils des pertes et des augmentations de précisions de prédictions (Figures 20 et 21) ainsi que les valeurs minimales des pertes et maximales des précisions de prédictions (Tableaux 11 et 12) sont comparables. Ainsi, on notera que le mode exponentiel (essais S4.7 et S4.8) permet d’obtenir des profils de pertes (Figure 21B) plus rapides et plus faibles pour le modèle H1 tout comme pour le modèle K2 (Figure 20B, essai S2.9).

Tout comme pour le modèle K2, afin de tenter de briser ce plafond de précisions de prédictions, une dernière stratégie de modulation à deux étapes a donc été expérimentée dont les paramètres et les résultats sont présentés à la section A.4.4. Ces derniers essais présentent également, des performances comparables à celles obtenues aux essais S4.4 et S2.4f mais au moyen de stratégies de modulation du TA plus compliquées et en un nombre d’epochs plus élevé.

4.6 Comparaison des modèles testés avec les résultats publiés pour BIDAF

4.6.1 Modèles avec variances structurelles

Tel que discuté à la section 4.2, les variantes structurelles de BIDAF développées et testées incluaient 1) le partage des paramètres des couches d’embeddings de mots à base de caractères, de réseaux autoroutes et d’embeddings contextuels, 2) l’ajout de dropouts aux représentations de la question (qa) et du contexte (ca) à la couche d’embeddings contextuels et 3) la couche de modélisation et de sortie selon la publication de BIDAF[19] et son implémentation. À celles-ci l’entraînement des embeddings, durant l’entraînement du modèle, a été ajouté.

Le Tableau 7 résume les résultats de ces travaux. Tel que présenté au Tableau 13, on observe que les meilleures performances obtenues sur l’ensemble de développement sont celles des modèles H1, E4 et, quelque peu moindre, du modèle A1 qui tous comportaient des dropouts internes aux représentations de la question et du contexte à la couche d’embeddings contextuels. Ces résultats obtenus avec entraînement des

embeddings pour H1 et E4, cependant, demeurent plus faibles que ceux de la version compatible tf1.8 de BIDAF original trouvée à https://github.com/allenai/bi-att-flow/compare/master...Vimos:tf1.8 discutée à la section 4.1 et que ceux rapportés à la publication originale de BIDAF[19].

Ces différences proviennent très probablement de subtilités plus ou moins évidentes du modèle BIDAF[19] qui n'ont pu être reproduites dans le présent travail.

Tableau 13. Comparaison des meilleures performances des modèles testés à celles de BIDAF original

Modèle	Structure			*EM* (%)	*F1* (%)
	Entraînement des embeddings	Stratégie de partage de paramètres	Couches de modélisation et de sortie		
H1	Oui	Keras	Selon BIDAF[19]	59.6	67.1
E4	Oui	Tensorflow		59.4	67.4
A1	Non	Aucun		58.8	66.1
Version compatible tf1.8 (section 4.1) de BIDAF				64.9	75.1
Version originale de BIDAF[19]				68.0	77.3

4.6.2 Essai d'optimisation par modifications structurelles des modèles développés

Par la suite, à la section 4.4, un certain nombre de modifications à certains de ces modèles ont été investiguées afin d'améliorer leur performance. Tel que présenté aux Tableaux 9 et 10, les modèles résultants ont permis d'accélérer significativement les profils d'augmentation de précisions de prédictions et de diminution des pertes sur l'ensemble de développement sans toutefois améliorer leur performance au-delà des résultats obtenus pour les meilleurs modèles développés précédemment présentés à la section 4.2 tel que résumé au Tableau 14.

Tableau 14. Comparaison des meilleures performances des modèles modifiés à celles de BIDAF original

Modèle	Structure				EM (%)	*F1* (%)
	Entraînement des embeddings	Stratégie de partage de paramètres	Couches de modélisation et de sortie	Autre		
K2	Oui	Keras	Selon BIDAF[19]	Calcul de S simplifié	59.5	67.3
Version compatible tf1.8 (section 4.1) de BIDAF					64.9	75.1
Version originale de BIDAF[19]					68.0	77.3
Sans embeddings de mots à base de caractères						
J4	Oui	Keras	Selon BIDAF[19]		60.2	67.7
J3	Non	Aucun			56.8	65.5
Version compatible tf1.8 (section 4.1) de BIDAF sans embeddings de mots à base de caractères					61.0	72.3
Version originale de BIDAF[19] sans embeddings de mots à base de caractères					65.0	75.4

4.6.3 Modèles avec modulation du TA

Finalement, à la section 4.5, on a testé l'effet de la modulation du TA sur la performance du modèle simplifié K2 et du modèle H1 afin d'améliorer leur performance. Tel que présenté aux Tableaux 11 à 12, plusieurs stratégies de modulation du TA ont été investiguées, qui ont permis, encore une fois, d'accélérer significativement les profils d'augmentation de précisions de prédictions et de diminution des pertes sur l'ensemble de développement et, cette fois-ci, d'améliorer, de façon limitée, leur performance comparativement aux résultats obtenus pour les meilleurs modèles présentés à la section 4.2 tel que résumé au Tableau 15.

Ainsi, la modulation du TA à l'essai des modèles K2 (essai S2.4f) et H1 (essai S4.4), a permis d'augmenter *EM* de ~1.9% en 12 epochs, dans le premier cas, et en 18 epochs dans le second cas, pour des performances finales, cependant, qui demeurent plus faibles que celles obtenues de la version tf1.8 et de BIDAF original [19].

Tableau 15. Comparaison des meilleures performances des modèles testés avec modulation du TA à celles de BIDAF original

Essai	Stratégie de modulation							EM (%)	*F1* (%)
	Étape	Mode	TP	PC	TA min	TA max	Nb ep		
K2	TA = 0.5							59.5	67.3
S2.4f	1	Exp.	17520	NA	0.1	5.0	12	61.4	68.5
S2.4	1	T2	17520	NA	0.15	1.2	12	61.2	68.2
H1	TA = 0.5							59.6	67.1
S4.4	1	T2	26280	NA	0.15	1.0	18	61.4	68.3
Version compatible tf1.8 (section 4.1) de BIDAF							12	64.9	75.1
Version originale de BIDAF[19]							12	68.0	77.3

En somme, on peut dire que les modèles développés et testés dans le cadre du présent travail présentant les meilleures performances ont atteint, aux paramètres et conditions utilisés, leurs capacités maximales qui demeurent inférieures à celles de la version tf1.8 et de BIDAF original [19]. Il n'a pas été possible d'identifier les raisons de ces différences.

Chapitre 5 Conclusion

5.1 Sur la performance de la version du modèle BIDAF développé et testé

L'objectif de ces travaux était la reproduction, au moyen de la librairie python – Keras avec Tensorflow en backend, du modèle BIDAF testé sur l'ensemble SQuAD 1.1 [19] puisque ce modèle était le plus performant rapporté au début ces travaux, soit en février 2017. Une seule approche similaire (utilisant la librairie python – Keras) a été identifiée dont les résultats de performance n'ont pas été trouvés. Or, à l'étude de ce modèle et de son implémentation, on a noté des différences importantes entre sa description selon la publication [19] et son implémentation rapportée à la section 3.1.1 (https://github.com/allenai/bi-att-flow). Dans une première étape (section 4.2) on a donc développé et testé une série de modèles explorant les différences structurelles principales rapportées, soit :

1. Le partage ou non des paramètres des couches d'embeddings de mots à base de caractères, de réseaux autoroutes et d'embeddings contextuels selon les librairies Tensorflow et Keras,
2. L'ajout d'un dropout externe uniquement sur qa et interne sur qa et ca à la couche d'embeddings contextuels et
3. La couche de modélisation et de sortie selon la publication de BIDAF[19] et son implémentation

auxquelles on a ajouté l'entraînement des embeddings de mots pré-entraînés GloVe et de mots à base de caractères durant l'entraînement du modèle.

Il s'est avéré que les modèles présentant les meilleures performances (*EM* = 59.5% et *F1* = 67.2%), soit les modèles H1 et E4 ainsi que la version simplifiée de H1 (au calcul de la représentation de la couche de similarité entre la question et le contexte), soit K2, comportaient

1. L'entraînement des embeddings de mots pré-entraînés GloVe et de mots à base de caractères durant l'entraînement du modèle,

2. Le partage des paramètres des couches d'embeddings de mots à base de caractères, de réseaux autoroutes et d'embeddings contextuels selon les librairies Tensorflow ou Keras,
3. L'ajout d'un dropout interne sur qa et ca à la couche d'embeddings contextuels et
4. La couche de modélisation et de sortie selon la publication de BIDAF[19].

On notera que l'entraînement des embeddings durant l'entraînement du modèle et la modulation du taux d'apprentissage n'ont jamais été rapportés ou testés sur le modèle BIDAF original. Il est donc possible que l'utilisation de ces deux approches au modèle original BIDAF améliore sa performance.

Par ailleurs, la modulation monocyclique du taux d'apprentissage aux modèles H1 et K2 a permis d'améliorer quelque peu cette performance à *EM* = 61.4% et *F1* = 68.4%.

Ces résultats demeurent inférieurs à ceux rapportés originalement pour BIDAF de *EM* = 68.0% et *F1* = 77.3% [19].

Il est difficile d'identifier pourquoi le modèle BIDAF développé et testé dans le cadre des présents travaux présente des performances inférieures à celles du modèle original BIDAF. Cependant, il nous semble que la combinaison des différences d'implémentation suivantes peuvent, en grande partie, expliquer cette plus faible performance de notre version de BIDAF, soit :

1. La méthodologie de calcul des embeddings de mots à base de caractères utilisée et décrite à la section 3.2.4 très probablement différente de celle utilisée à l'implémentation originale du modèle BIDAF,
2. L'absence de masques de notre version de BIDAF contrairement à l'implémentation originale du modèle BIDAF où ceux-ci sont présents à la couche d'attention depuis le calcul du tenseur de similarité S (équation 2) jusqu'au calcul des attentions C2Q (équation 6) et Q2C (équation 7) ainsi qu'aux MLP précédents l'activation finale softmax pour le calcul de p^1 (équation 10) et de p^2 (équation 11),

3. Un certain nombre de détails de l'implémentation originale de BIDAF en Tensorflow implémentés de façon inappropriée lorsque traduit à notre version en KERAS API avec backend en Tensorflow et
4. L'implémentation probablement inappropriée, durant l'entraînement du modèle, des moyennes mobiles de ses poids avec un taux de décroissance exponentielle de 0.999 (ema : exponential moving average) et l'utilisation, au test, des moyennes mobiles au lieu des poids bruts.

5.2 Apprentissages personnels

Finalement ces travaux et ce mémoire représentent les premières aventures de recherche de l'auteur dans les domaines du TALN, et en particulier du QA, et en apprentissage profond (AP). Dans le premier cas, nous avons, évidemment, beaucoup appris sur la pratique du traitement des langues en informatique et sur l'utilisation des embeddings de mots pré-entrainés et de mots à base de caractères pour numériser leur signification dans le cadre d'une tâche de TALN à résoudre au moyen, dans le cas présent, de modèles neuronaux d'AP.

En ce qui concerne les connaissances acquises en AP, elles sont nombreuses, depuis des détails pratiques d'implémentation aux approches de plus haut niveau.

Par exemple, on a noté l'évidente lourdeur du modèle BIDAF en KERAS-Tensorflow qui requiert l'utilisation d'ordinateurs à GPU récents avec suffisamment de mémoire ($\geq 6G$) pour rouler dans un temps raisonnable (~40 minutes par epoch). L'auteur a du acquérir son propre ordinateur avec unité GPU pour compléter les travaux de ce mémoire. Par ailleurs, à la suite de ces travaux, nous avons appris la bibliothèque d'AP Pytorch que nous avons appliquée à BIDAF. Il est intéressant d'observer que cette version de BIDAF en Pytorch tourne 10 fois plus rapidement que celle développée et testée dans le cadre des travaux de ce mémoire. Pytorch représente donc une plateforme d'AP beaucoup plus performante que KERAS-Tensorflow.

Les modèles d'AP à réseaux neuronaux demeurent des boîtes noires dont le comportement est difficile à visualiser, à modifier durant l'entraînement et à prévoir. Les interactions entre leurs composantes sont complexes et difficiles à expliquer, et d'un modèle à un autre, une même composante peut se comporter de façon différente. L'optimisation de ces réseaux est donc longue et pénible, et demande un bon niveau d'expérience pratique. L'auteur, dans le cadre de ses activités professionnelles, a contribué, au début de 2020, au démarrage de l'entreprise Zetane Systems Inc. (https://zetane.com) qui a développé une plateforme informatique innovante permettant de visualiser le comportement de modèles à réseaux neuronaux (en KERAS, Tensorflow etc.) durant l'entraînement ainsi que de modifier leurs paramètres durant l'entraînement, ce qui permet d'accélérer leur optimisation. Zetane™ est présentement utilisée avec succès par un certain nombre de joueurs du domaine comme Thales et BlueWave~AI. Si ce n'est pas déjà fait, il serait intéressant pour les chercheurs en AP du DIRO d'investiguer cette plateforme. Zetane™ était disponible à l'auteur en juin 2020 mais n'a pu être utilisée par manque de temps.

Un autre apprentissage important et évident des présents travaux porte sur les difficultés de reproduction d'un modèle d'AP à réseaux neuronaux depuis sa publication et son implémentation, en autre considérant 1) le passage d'un environnement informatique à un autre plus récent, 2) les mises à niveaux continuelles des plateformes d'AP qui entraînent des modifications d'implémentation pas toujours évidentes et 3) la 'traduction' de l'implémentation d'un modèle d'une plateforme d'AP à une autre, ici de Tensorflow à KERAS avec Tensorflow en backend. Un des avantages de la plateforme KERAS API est d'obtenir un séquencement informatique plus facilement représentatif et compréhensible du modèle mathématique et plus facile à débugger.

Parmi les autres apprentissages réalisés, on notera :

1. L'utilité de l'étude plus approfondie de modèles compétitifs afin d'identifier de nouvelles approches et de nouvelles idées d'essais d'optimisation du modèle de réseaux neuronaux ciblé,

2. L’identification et l’investigation d’anomalies entre la description littéraire d’un modèle et son implémentation,
3. L’étude comparative de différentes implémentations d’un modèle pour mieux le comprendre et identifier des composantes implémentées au potentiel d’optimisation plus élevé,
4. L’effet positif surprenant de la modulation cyclique du taux d’apprentissage sur le modèle BIDAF,
5. D’un point de vue pratique,
    a. L’utilité de la couche Keras lambda avec fonction en Tensorflow lorsque la couche équivalente en Keras n’est pas compatible avec le tenseur qui lui est alimenté et
    b. L’incompatibilité de la sauvegarde en-line checkpoints de Keras avec certains types de couche neuronale.

Sources documentaires

[1] Huang Z., Xu S., Hu M., Wang X., Qiu J., Fu Y., Zhao Y., Peng Y., & Wang C., "Recent Trends in Deep Learning Based Open-Domain Textual Question Answering Systems", IEEE Access 8, 94341-94356, 2020.

[2] Green B.F., Wolf A.K., Carol Chomsky C., & Laughery K.. "Baseball: an automatic question-answerer", In Papers presented at the May 9-11, 1961, western joint IRE-AIEE-ACM computer conference, pages 219–224. ACM, 1961.

[3] Woods W.A., "Progress in natural language understanding: an application to lunar geology", In Proceedings of the June 4-8, 1973, national computer conference and exposition, pages 441–450. ACM, 1973.

[4] Hirschman L. & Gaizauskas R., "Natural language question answering: the view from here", Natural Language Engineering 7 (4): 275-300, 2001.

[5] Katz B., Felshin S., Lin J. & Marton G.. "Viewing the Web as a Virtual Database for Question Answering", In Mark T. Maybury, editor, New Directions in Question Answering. Cambridge, Massachusetts: MIT Press, pp. 215-226, 2004.

[6] Voorhees E.M., "The TREC-8 question answering track report" in Proc. Text Retr. Conf. (TREC), pp. 77-82, 1999.

[7] Bollacker K., Evans C., Paritosh P., T. Sturge T. & Taylor J., "Freebase: A collaboratively created graph database for structuring human knowledge" in Proc. ACM SIGMOD Int. Conf. Manage. Data Aug. pp. 1247_1250, 2008.

[8] J. Berant J., Chou A., R. Frostig R. & Liang P. S., "Semantic parsing on freebase from question-answer pairs," in Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP), pp. 1533-1544, 2013.

[9] E. Brill E., S. Dumais S. & Banko M., "An analysis of the AskMSR question answering system", in Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP), pp. 257-264, 2002.

[10] Baudis P., "YodaQA: A modular question answering system pipeline", in Proc. Int. Student Conf. Electr. Eng. POSTER, p. 8, 2015.

[11] Ferrucci D., Brown E., Chu-Carroll J., Fan J., Gondek D., Kalyanpur A. A., Lally A., Murdock J. W., Nyberg E., Prager J., Schlaefer N. & Welty C., "Building Watson: An overview of the DeepQA project", AI Mag., 31 (3), 59, 2010.

[12] Blei D.M., Ng A.Y. & Jordan M.I., "Latent Dirichlet Allocation", Journal of Machine Learning Research 3, 993-1022, 2003.

[13] Wiemer-Hastings P., "Latent Semantic Analysis", In book Encyclopedia of Language and Linguistics, 2e ed. Publisher: Elsevier, January 2004.

[14] Bengio Y., Ducharme R., Vincent P. & Jauvin C., "A Neural Probabilistic Language Model", Journal of Machine Learning Research 3, 1137–1155, 2003.

[15] Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., & Kuksa, P., "Natural Language Processing (almost) from Scratch", Journal of Machine Learning Research, 12 (Aug), 2493–2537, 2011, http://arxiv.org/abs/1103.0398.

[16] Mikolov, T., Corrado, G., Chen, K., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. Proceedings of the International Conference on Learning Representations (ICLR 2013), 1–12.

[17] Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Distributed Representations of Words and Phrases and their Compositionality. NIPS, 1–9.

[18] Pennington, J., Socher, R., & Manning, C. D. (2014). Glove: Global Vectors for Word Representation. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, 1532–1543. http://doi.org/10.3115/v1/D14-1162

[19] Seo M., Kemnhavi A., Farhadi A. & Hajishirzi H., "Bi-Directional Attention Flow for Machine Comprehension", arXiv:1611.01603v6 [cs.CL] 21 Jun 2018.

[20] Devlin J., Chang M.W., Lee K. & Toutanova K., "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", arXiv:1810.04805v2 [cs.CL] 24 May 2019.

[21] Rajpurkar P., Zhang J., Lopyrev K., & Liang P., "SQuAD: 100,000+ Questions for Machine Comprehension of Text", arXiv:1606.05250v3 [cs.CL] 11 Oct, 2016.

[22] Yang Y. & Meek W. Y. C., "WIKIQA: A Challenge Dataset for Open-Domain Question Answering", Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, Sept 2015 pp 2013-2018, https://www.aclweb.org/anthology/D15-1237.pdf.

[23] Trischler A., Wang T., Yuan X., Harris J., Sordoni A., Bachman P. & Suleman K., "NEWSQA: A Machine Comprehension Dataset", arXiv:1611.09830v3 [cs.CL] 7 Feb 2017.

[24] Bajaj P., Campos D., Craswell N., Deng L., Gao J., Liu X., Majumder R., McNamara A., Mitra B., Nguyen T., Rosenberg M., Song X., Stoica A., Tiwary S., & Tong Wang T., "MS MARCO: A Human Generated Machine Reading Comprehension Dataset", arXiv:1611.09268v3 [cs.CL] 31 Oct 2018.

[25] Jurczyk T., Zhai M. & Choi J.D., "SelQA: A New Benchmark for Selection-based Question Answering", arXiv:1606.08513v3 [cs.CL] 28 Oct 2016.

[26] Yang Z., Dai Z., Yang Y., Carbonell J., Salakhutdinov R., Quoc V. Le Q.V., "XLNet: Generalized Autoregressive Pretraining for Language Understanding", arXiv:1906.08237v2 [cs.CL] 2 Jan 2020.

[27] Elbaz, Ilan, "Un système de question-réponse simple appliqué à SQuAD", Thèse de Maîtrise, Université de Montréal, Mars 2020

[28] Joshi M., Danqi Chen D., Liux Y., Weldy D.S., Zettlemoyeryx L. & Levyx O., "SpanBERT: Improving Pre-training by Representing and Predicting Spans", arXiv:1907.10529v3 [cs.CL] 18 Jan 2020.

[29] An Yang A., Wang Q., Liu J., Liu K., Lyu Y., Wu U., She Q. & Sujian Li, "Enhancing Pre-Trained Language Representations with Rich Knowledge for Machine Reading Comprehension", Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, pages 2346–2357, Florence, Italy, July 28 - August 2, 2019.

[30] Rezaeinia S.M., Ghodsi A. & Rahmani R., "Improving the Accuracy of Pre-trained Word Embeddings for Sentiment Analysis", https://arxiv.org/pdf/1711.08609.pdf.

[31] Yu A. W., Dohan D., Luong M. T., Zhao R., Chen K., Norouzi M. 7 Le Q. V., "QANET : Combining Local Convolution with Global Self-Attention for reading Comprehension", arXiv:1804.09541v1 [cs.CL] 23 Apr. 2018.

[32] Natural Language Computing Group, Microsoft Research Asia, "R-NET: Machine Reading Comprehension with Self-Matching Networks", https://www.microsoft.com/en-us/research/wp-content/uploads/2017/05/r-net.pdf

[33] Huy M., Pengy Y., Huangy Z., Qiuz X., Weix F. & Zhoux M., "Reinforced Mnemonic Reader for Machine Reading Comprehension", arXiv:1705.02798v6 [cs.CL] 6 Jun 2018.

[34] Wang W., Yang N., Wei F., Chang B. & Zhou M., "Gated Self-Matching Networks for Reading Comprehension and Question Answering", Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, pages 189–198Vancouver, Canada, July 30 - August 4, 2017.
[35] Wang W., Yan M. & Wu C., "Multi-Granularity Hierarchical Attention Fusion Networks for Reading Comprehension and Question Answering", Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Long Papers), pages 1705–1714, Melbourne, Australia, July 15 - 20, 2018.
[36] He G., "Contextual Question Answering with Improved Embedding Models", https://georgehe.me/coqa.pdf.
[37] Joshiy M., Choiy E., Levyz O., Weldy D.S. & Zettlemoyer L., "pair2vec: Compositional Word-Pair Embeddings for Cross-Sentence Inference", arXiv:1810.08854v2 [cs.CL] 5 Apr 2019.
[38] Rui Liu R., Junjie Hu J., Wei W., Yang Z., Eric Nyberg "Structural Embedding of Syntactic Trees for Machine Comprehension", arXiv:1703.00572v3 [cs.CL] 31 Aug 2017.
[39] Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A.N., Kaiser L., & Polosukhin I., "Attention Is All You Need", arXiv:1706.03762v5 [cs.CL] 6 Dec 2017.
[40] Ba J.L., Kiros J.R., & Geoffrey E Hinton G.E., "Layer normalization", arXiv preprint arXiv:1607.06450, 2016.
[41] Hochreiter S. & Schmidhuber J., "Long short-term memory", Neural computation, 9(8):1735-80, 1997.
[42] Bengio, Y., Simard, P., & Frasconi, P., "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 5(2), 157–166, 1994.
[43] Zhu Y., Kiros R., Zemel R., Salakhutdinov R., Urtasun R., Torralba A. & Fidler S., "Aligning Books and Movies: Towards Story-like Visual Explanations by Watching Movies and Reading Books", arXiv:1506.06724v1 [cs.CV] 22 Jun 2015.
[44] Joshiy M., Eunsol Choiy E., Weldy D.S. & Luke Zettlemoyeryz L., "TriviaQA: A Large Scale Distantly Supervised Challenge Dataset for Reading Comprehension", arXiv: 1705.03551v2 [cs.CL] 13 May 2017

[45] Parker R., Graff D., Kong J., Chen K., & Maeda K., "English gigaword", fifth edition, Technical Report. Linguistic Data Consortium, Philadelphia, Tech. Rep., 2011.
[46] Callan J., Hoy M., Yoo C. & Le Zhao L., "Clueweb09 data set", 2009.
[47] Common Crawl. Common crawl. URl: http://http://commoncrawl. org.
[48] Graves A. & Schmidhuber J., "Framewise phoneme classification with bidirectional LSTM and other neural network architectures", Neural Networks 18(5-6):602-10, 2005.
[49] Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau D., Bougares F., Schwenk H. & Bengio, Y. (2014). "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". arXiv:1406.1078.
[50] Chollet F., "Xception: Deep learning with depthwise separable convolutions", arXiv: 1610.02357v3 [cs.CV] 4 Apr 2017.
[51] Petersy M.E., Neumanny M., Iyyery M., Gardnery M., Clark C., Lee K. & Luke Zettlemoyery L., "Deep contextualized word representations", arXiv:1802.05365v2 [cs.CL] 22 Mar 2018.
[52] Kim Y., "Convolutional Neural Networks for Sentence Classification", Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 1746–1751,October 25-29, 2014, Doha, Qatar.
[53] Srivastava R.K., Klaus Greff K., & Schmidhuber J.. "Highway networks", *CoRR*, abs/1505.00387, 2015. URL http://arxiv.org/abs/1505.00387.
[54] Srivastava N., Hinton G., Krizhevsky A., Sutskever I. & Salakhutdinov R., "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", Journal of Machine Learning Research 15 (2014) 1929-1958.
[55] Chelba C., Mikolov T., Schuster M., Ge Q., Brants T., Koehn P. & Robinson T., "One BillionWord Benchmark for Measuring Progress in Statistical Language Modeling" arXiv:1312.3005v3 [cs.CL] 4 Mar 2014.
[56] Ge R., Kakade S.M., Kidambi R. & Netrapalli P., "The Step Decay Schedule: A Near Optimal, Geometrically Decaying Learning Rate Procedure For Least Squares" arXiv:1904.12838v2 [cs.LG] 29 Oct 2019.
[57] Smith L.N., "Cyclical Learning Rates for Training Neural Network", arXiv:1506.01186v6 [cs.CV] 4 Apr 2017.

Appendice A. Résultats expérimentaux

A.1 Essais de modèles avec couches sans partage de paramètres

Dans cette section, on discute des modèles A1 et A2 dont les couches d'embeddings de mots à base de caractères, de réseaux autoroutes et d'embeddings contextuels ne partagent pas leurs paramètres entre le contexte et la question tel que discuté aux sections 3.2.4 et 3.2.5 et les embeddings de mots pré-entraînés GloVe et de mots à base de caractères ne sont pas entraînés durant l'entraînement du modèle. On notera qu'un dropout interne de 0.2 est inclus aux couches BILSTM d'embeddings contextuels du contexte et de la question comme pour BIDAF [19].

A.1.1 Avec couche de sortie selon BIDAF

Le modèle A1 possède une couche de sortie selon la publication originale de BIDAF [19] tel qu'illustrée à la Figure 13, soit que les réseaux des couches de sortie de prédiction des positions de début et de fin de la réponse sont de structures similaires. Il compte 1.68M paramètres entrainables.

Les résultats obtenus au moyen de ce modèle sont présentés aux Figures A1 et A2. À la Figure A1, on observe que les profils des pertes de prédiction en fonction du nombre d'epochs pour les positions de début (yS) et de fin (yE) de la réponse dans le contexte sont pratiquement identiques pour les ensembles d'entrainement et de développement (Test), respectivement. Ce comportement est commun à tous les essais effectués dans ces travaux. Dans ce contexte, tous les résultats de pertes discutés pour les différents modèles testés ne porteront que sur ceux pour les prédictions de yS.

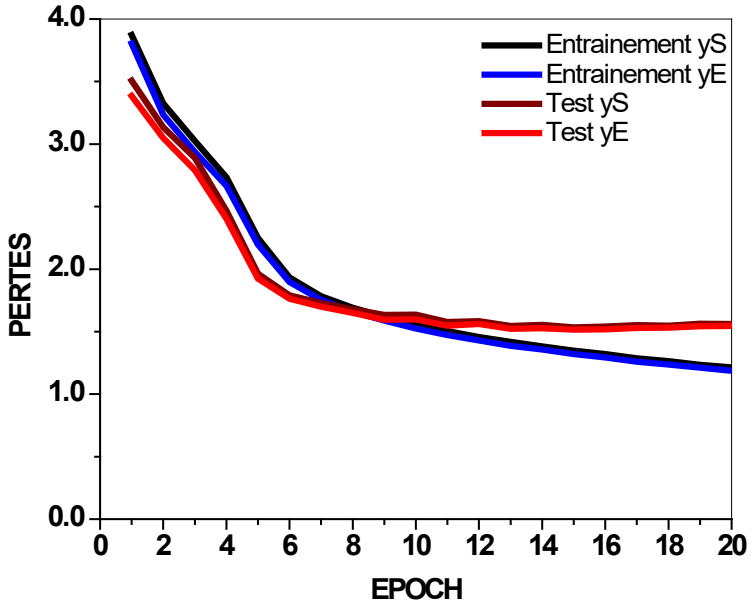


Figure A1. Pertes par entropie croisée catégorique du modèle A1

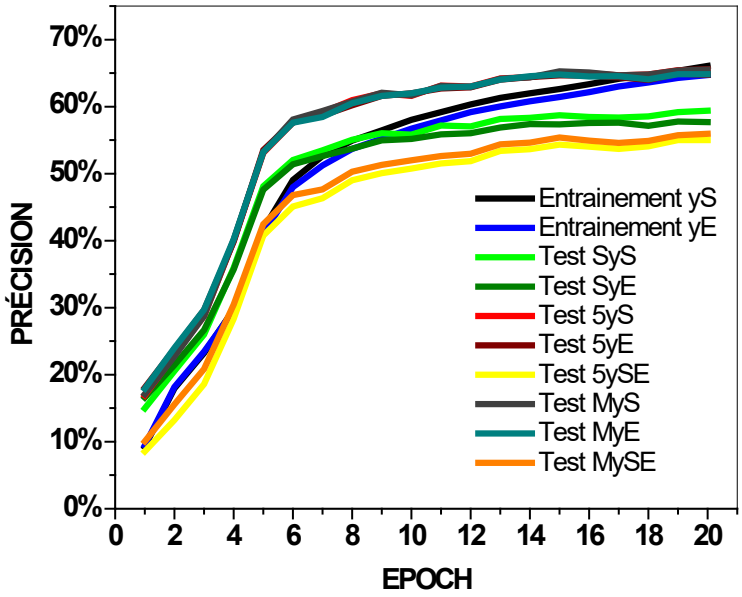


Figure A2. Performance de précisions de prédictions du modèle A1

À la Figure A2, on présente les profils de précisions de prédictions (calculées selon la section 3.1.2) durant l'entraînement du modèle A1

a. De yS et yE pour l'ensemble d'entraînement,
b. De yS et yE pour l'ensemble de développement (Test) selon
    a. La fonction *model.evaluate()* de Keras (Test SyS et Test SyE),

b. Les cinq réponses correctes possibles de l'ensemble de développement (Test 5yS et Test 5yE) et
c. La multiplication des matrices de probabilités yS et yE (Test MyS et Test MyE), et

c. De Test 5ySE, soit de la prédiction simultanément correcte de yS et yE depuis Test 5yS et Test 5yE et
d. De Test MySE, soit de la prédiction simultanément correcte de yS et yE depuis Test MyS et Test MyE.

On observe que les profils de précisions de prédictions de yS et de yE en fonction du nombre d'epochs sont pratiquement identiques,

1. Pour l'ensemble d'entrainement (Entrainement yS et Entraînement yE), et
2. Pour l'ensemble de développement, obtenus selon les trois méthodes de mesure utilisées (section 3.1.2), soit
   a. Pour Test SyS et Test SyE,
   b. Pour Test 5yS et Test 5yE et
   c. Pour Test MyS et Test MyE.

De plus, on notera que les profils des quatre dernières mesures (Test 5yS et Test 5yE et Test MyS et Test MyE) se confondent et sont supérieures à celles de Test SyS et Test SyE.

De même, les résultats Test 5ySE et Test MySE sont aussi pratiquement identiques mais avec ceux de Test MySE toujours légèrement supérieurs à ceux de Test 5ySE.

Ces comportements sont communs à tous les modèles testés dans ces travaux. Dans ce contexte, tous les résultats de précision de prédiction présentés des différents modèles testés ne porteront que sur MySE, la précision de prédiction de yS et yE simultanément correctes selon la multiplication des matrices de probabilités de yS et yE.

Pour le modèle A1, les précisions de prédiction en fin d'entraînement obtenues pour l'ensemble de développement sont de 55.92% pour MySE et de *EM* = 58.78% et *F1* =

66.09%, ce qui représente une des meilleures performances d'un modèle dans ce travail. La MySE de l'ensemble d'entrainement à la fin de l'entraînement était de 63.47%. Cette performance du modèle A1, tout comme pour tous les autres modèles discutés dans les sections suivantes, a été répétée avec des résultats similaires.

A.1.2 Avec couche de sortie selon l'implémentation de BIDAF

Le modèle A2 possède une couche de sortie selon l'implémentation de BIDAF tel qu'illustrée à la Figure 14, soit que le réseau de prédiction de yE débute à partir de la prédiction de yS et est plus complexe que selon la publication sur BIDAF [19]. Tout comme le modèle A1, il ne comporte pas d'entraînement d'embeddings (section 3.2.4) ni de partage de paramètres. Il compte 2.64M paramètres entrainables. Les précisions de prédictions finales obtenues pour l'ensemble de développement sont de 44.30% pour MySE et de *EM* = 46.70% et *F1* = 55.26%. La MySE de l'ensemble d'entrainement à la fin de l'entraînement était de 50.50%. Ces résultats indiquent donc que la structure du réseau de sortie selon l'implémentation de BIDAF utilisée au modèle A2 a un effet négatif sur sa performance comparativement à celle du modèle A1.

A.1.3 Avec embeddings de mots et de mots à base de caractères entraînés

Le modèle B1 est identique au modèle A1 (couche de sortie selon la publication originale de BIDAF [19] tel qu'illustrée à la Figure 14) sauf qu'il comporte l'entrainement des embeddings de mots pré-entrainés GloVe 100 et de mots à base de caractères selon l'option de la couche embedding de Keras *trainable* = *True* (voir section 3.2.4). Il compte 10.37M paramètres entrainables.

Les précisions de prédictions finales obtenues pour l'ensemble de développement sont de 47.09% pour MySE et de *EM* = 49.64% et *F1* = 58.55%. La MySE de l'ensemble d'entrainement à la fin de l'entraînement était de 64.11%. L'entrainement des embeddings de mots à base de caractères et de mots pré-entrainés GloVe 100 diminue donc la performance du modèle B1 comparativement à celle du modèle A1 sur l'ensemble de développement.

Curieusement, la performance du modèle B1, avec entraînement des embeddings, sur l'ensemble d'entraînement est pratiquement la même que celle du modèle A1 (MySE = 63.47%).

À la revue des modèles les plus performants testés sur SQuAD au Chapitre 2, incluant BIDAF, l'entraînement des embeddings durant l'entraînement des modèles ne semble pas avoir été étudié. L'entraînement des embeddings devrait théoriquement augmenter la capacité d'un modèle, et donc sa performance puisque ceci permet, au moins d'estimer des embeddings pour les mots ne possédant pas d'embeddings GloVe pré-entraînés, comme souligné à la section 3.2.2, soit 20.24% des mots du vocabulaire. Cette augmentation de capacité devrait au moins se refléter sur la performance du modèle avec l'ensemble d'entraînement. Or les résultats pratiquement identiques obtenus pour les modèles B1 et A1 (avec et sans entraînement des embeddings, respectivement) sur l'ensemble d'entraînement indiquent, au moins pour cette configuration, que d'autres composantes structurelles des modèles limitent plus leur capacité comme discuté dans les sous-sections suivantes.

Le modèle B2 est identique au modèle A2 (couche de sortie selon l'implémentation de BIDAF tel qu'illustrée à la Figure 14) sauf qu'il comporte l'entrainement des embeddings de mots GloVe 100 et de mots à base de caractères selon l'option de la couche embedding de Keras *trainable* = *True*. Il compte 11.33M paramètres entrainables.

Les précisions de prédictions finales obtenues pour l'ensemble de développement sont de 46.35% pour MySE et de *EM* = 48.77% et *F1* = 58.21%. La MySE de l'ensemble d'entrainement à la fin de l'entraînement était de 61.12%. Dans ce cas, on observe donc que l'entraînement des embeddings augmente la capacité du modèle B2 comparativement à celle du modèle A2, tel qu'anticipé, ce qui se reflète par une meilleure performance de +11% sur l'ensemble d'entraînement et de +2% sur l'ensemble de développement. Comme suggéré plus haut, l'effet théoriquement positif de l'entraînement des embeddings sur la capacité d'un modèle dépend donc de ses autres composantes structurelles.

A.2 Essais de modèles avec couches à paramètres partagés selon la stratégie Tensorflow

Dans cette section, on discute des modèles C1 à E4 dont les couches d'embeddings de mots à base de caractères, de réseaux autoroutes et d'embeddings contextuels partagent leurs paramètres entre le contexte et la question selon la stratégie 1, soit utilisant les énoncés

```
with tf.variable_scope('char'):
    tf.get_variable_scope().reuse_variables()
```

selon Tensorflow tel que discuté aux sections 3.2.4 et 3.2.5. Dans ce cas, deux options ont été investiguées, soit A) avec dropout externe de 0.2 sur les embeddings contextuels de la question qa seulement comme pour l'implémentation de BIDAF et B) avec dropout interne de 0.2 au réseau RNR BILSTM appliqué à qa et à ca comme pour BIDAF [19].

A.2.1 Avec couche de sortie selon BIDAF

Le modèle C1 possède une couche de sortie selon la publication originale de BIDAF [19] tel qu'illustrée à la Figure 13, soit que les réseaux de prédiction des positions de début et de fin de la réponse sont de structures similaires, et un dropout externe appliqué uniquement sur qa. Il compte 1.68M paramètres entrainables.

Les précisions de prédictions en fin d'entraînement obtenues pour l'ensemble de développement sont de 42.95% pour MySE et de *EM* = 45.20% et *F1* = 53.79%. La MySE de l'ensemble d'entrainement à la fin était de 58.44%.

Le modèle C2 possède la même structure que le modèle C1 sauf pour un dropout interne de 0.2 au réseau RNR BILSTM d'embeddings contextuels appliqué à qa et à ca. Il compte 1.68M paramètres entrainables.

Les précisions de prédictions finales ainsi obtenues pour l'ensemble de développement sont 46.93% pour MySE et de *EM* = 49.54% et *F1* = 57.96%. La MySE de l'ensemble d'entrainement à la fin était de 55.56%. En comparaison du modèles A1, ces résultats indiquent que, pour le Modèle C2, le partage de paramètres selon la stratégie 1 (Tensorflow) n'améliore pas sa performance. Cependant, on observe une meilleure

performance (+ 5%) du modèle C2 comparativement à C1. Donc l'ajout de dropouts internes au BILSTM des embeddings contextuels améliore la performance du modèle C2 comparativement à celle du modèle C1.

A.2.2 Avec couche de sortie selon l'implémentation de BIDAF

Le modèle D1 possède une couche de sortie selon l'implémentation de BIDAF tel qu'illustrée à la Figure 14, soit que le réseau de prédiction de yE débute à partir de la prédiction de yS et est plus complexe que pour la prédiction de yS et un dropout externe est appliqué uniquement sur qa. Il compte 2.64M paramètres entrainables.

Les précisions de prédictions finales ainsi obtenues pour l'ensemble de développement sont de 43.54% pour MySE et de *EM* = 45.85% et *F1* = 55.17%. La MySE de l'ensemble d'entrainement à la fin était de 55.61%. En comparaison du Modèle C1 (couche de sortie selon BIDAF [19]), celui-ci présente des performances marginalement plus élevées sur l'ensemble de développement.

Le modèle D2 possède la même structure que le modèle D1 sauf pour l'application d'un dropout interne au réseau RNR BILSTM d'embeddings contextuels appliqué à qa et ca. Il compte 2.64M paramètres entrainables.

Les précisions de prédictions finales ainsi obtenues pour l'ensemble de développement sont de 44.87% pour MySE et de *EM* = 47.25% et *F1* = 56.83%. La MySE de l'ensemble d'entrainement à la fin était de 51.34%. En comparaison du Modèle D1 (un seul dropout aux embeddings contextuels), le modèle D2 (deux dropouts internes) présente une meilleure performance (+ 2%) mais moindre que C2 (couche de sortie selon BIDAF, deux dropout internes; *EM* = 49.54%) par rapport à C1 (couche de sortie selon BIDAF, un dropout externe; *EM* = 45.20%). On note donc des interactions complexes entre les différentes composantes des modèles difficiles à interpréter.

A.2.3 Avec embeddings de mots et de mots à base de caractères entraînés

Le modèle E1 est identique au modèle D2 (couche de sortie selon l'implémentation de BIDAF tel qu'illustrée à la Figure 14 et dropout sur qa seulement) sauf qu'il comporte

l'entrainement des embeddings de mots pré-entrainés GloVe 100 et de mots à base de caractères selon l'option de la couche embedding de Keras *trainable* = *True*. Il compte 11.33M paramètres entrainables.

Les précisions de prédictions finales obtenues pour l'ensemble de développement sont de 49.53% pour MySE et de *EM* = 52.14% et *F1* = 61.09%. La MySE de l'ensemble d'entrainement à la fin était de 72.22%. En comparaison du modèle D1 (MySE de l'ensemble d'entraînement = 55.61%; ensemble de développement : *EM* = 45.85% et F1 = 55.17%), on observe que l'entraînement des embeddings augmente considérablement la capacité du modèle E1 tel qu'indiqué par sa performance sur les ensembles d'entraînement (MySE = 77.22% ou +17%) et de développement (+6%) tel qu'anticipé.

Le modèle E2 est identique au modèle d2 (couche de sortie selon l'implémentation de BIDAF tel qu'illustrée à la Figure 14 et dropouts internes sur ca et qa au réseau RNR BILSTM des embeddings contextuels) sauf qu'il comporte l'entrainement des embeddings de mots pré-entrainés GloVe 100 et de mots à base de caractères selon l'option de la couche embedding de Keras *trainable* = *True*. Il compte 11.33M paramètres entrainables.

Les précisions de prédictions finales obtenues pour l'ensemble de développement sont de 55.01% pour MySE et de *EM* = 57.64% et *F1* = 65.89%. La MySE de l'ensemble d'entrainement à la fin était de 63.88%. Encore ici et tel qu'anticipé, l'entraînement des embeddings permet également d'augmenter considérablement la capacité du modèle E2 comparativement au modèle D1 (MySE = 51.34% pour l'ensemble d'entraînement et *EM* = 47.25% et *F1* = 56.83% pour l'ensemble de développement). En fait, le modèle E2 présente une performance nettement supérieure à celle des modèles C1 à E1 précédents, et celle-ci se compare à celle du modèle A1, les deux se classant parmi les meilleurs résultats obtenus dans le cadre des présents travaux.

En fait, on notera que l'utilisation de dropouts internes à qa et ca au modèle E2, semble diminuer la sur-modélisation (overfitting) par régularisation [54] sur l'ensemble

d'entraînement (MySE = 63.88% vs 72.22% pour le modèle E1), ce qui résulte en une meilleure performance sur l'ensemble de développement (*EM* = 57.64% vs 52.14% pour le modèle E1).

Le modèle E3 est identique au modèle c1 (couche de sortie selon la publication originale de BIDAF [19] tel qu'illustrée à la Figure 13 et dropout sur qa seulement) sauf qu'il comporte l'entrainement des embeddings de mots pré-entrainés GloVe 100 et de mots à base de caractères selon l'option de la couche embedding de Keras *trainable* = *True*. Il compte 10.37M paramètres entrainables.

Les précisions de prédictions finales ainsi obtenues pour l'ensemble de développement sont de 51.84% pour MySE et de *EM* = 54.70% et *F1* = 63.20%. La MySE de l'ensemble d'entrainement à la fin était de 75.48%. Encore ici, on observe une augmentation importante de la performance du modèle E3, incluant une surmodélisation possible sur l'ensemble d'entraînement, causée par l'entraînement des embeddings comparativement au modèle C1 (MySE = 58.44% pour l'ensemble d'entraînement et *EM* = 45.20% sur l'ensemble de développement).

Le modèle E4 est identique au modèle C2 (couche de sortie selon la publication originale de BIDAF [19] tel qu'illustrée à la Figure 13 et dropout interne sur ça et qa au BILSTM des embeddings contextuels) sauf qu'il comporte l'entrainement des embeddings de mots pré-entrainés GloVe 100 et de mots à base de caractères selon l'option de la couche embedding de Keras *trainable* = *True*. Il compte 10.37M paramètres entrainables.

Les précisions de prédictions finales obtenues pour l'ensemble de développement sont de 56.58% pour MySE et de *EM* = 59.39% et *F1* = 67.36%. La MySE de l'ensemble d'entrainement à la fin était de 68.81%. Dans ce cas, la combinaison de l'entraînement des embeddings, du partage de paramètres, de dropouts internes sur qa et ca à la couche d'embeddings contextuels (qui semble limiter l'overfitting possible sur l'ensemble d'entraînement par régularisation [54] comme noté pour le modèle E2 comparativement au modèle E1) et de la couche de sortie selon BIDAF [19] donne une meilleure performance sur l'ensemble de développement comparativement à tous les modèles de

la série C1 à E3, qui se compare avantageusement aux meilleures performances présentées précédemment, soit pour le modèle A1.

A.3 Essais de modèles avec couches à paramètres partagés selon la stratégie Keras

Dans cette section, on discute des modèles F1 À H2 dont les couches d'embeddings de mots à base de caractères, de réseaux autoroutes et d'embeddings contextuels, tel que discuté aux sections 3.2.4 et 3.2.5, partagent leurs paramètres entre le contexte et la question selon la stratégie 2, soit selon Keras.

A.3.1 Avec couche de sortie selon BIDAF

Le modèle F1 possède une couche de sortie selon la publication originale de BIDAF [19] tel qu'illustrée à la Figure 13, soit que les réseaux de prédiction des positions de début et de fin de la réponse sont de structures similaires, et un dropout externe appliqué uniquement sur qa. Il compte 1.44M paramètres entrainables.

Les précisions de prédictions finales obtenues pour l'ensemble de développement sont de 48.00% pour MySE et de *EM* = 50.40% et *F1* = 59.41%. La MySE de l'ensemble d'entrainement à la fin était de 65.10%. Ces résultats suggèrent que, pour le modèle F1, le partage de paramètres selon la stratégie 2 (Keras) présente une performance de 5% supérieure à celle du modèle C1 de même structure mais utilisant la stratégie selon Tensorflow.

Le modèle F2 est identique au modèle F1 sauf pour des dropouts internes au BILSTM des embeddings contextuels appliqués sur qa et ca. Il compte 1.44M paramètres entrainables.

Les précisions de prédictions finales ainsi obtenues pour l'ensemble de développement sont de 54.57% pour MySE et de *EM* = 57.06% et *F1* = 64.81%. La MySE de l'ensemble d'entrainement à la fin était de 60.38%.

Encore ici, on observe l'effet régularisateur des dropouts internes sur qa et ca au BILSTM des embeddings contextuels qui, comparativement au modèle F1, diminue la MySE de l'ensemble d'entraînement, de 65.10% (F1) à 60.38% et, pour l'ensemble de développement, augmente *EM* de 50.40% (F1) à 57.06%.

A.3.2 Avec couche de sortie selon l'implémentation de BIDAF

Le modèle G1 possède une couche de sortie selon l'implémentation de BIDAF tel qu'illustrée à la Figure 14, soit que le réseau de prédiction de yE débute à partir de la prédiction de yS et est plus complexe que pour la prédiction de yS et des dropouts internes au BILSTM des embeddings contextuels sont appliqué sur qa et ca. Il compte 2.40M paramètres entrainables.

Les précisions de prédictions finales ainsi obtenues pour l'ensemble de développement sont de 55.53% pour MySE et de *EM* = 58.23% et *F1* = 66.14%. La MySE de l'ensemble d'entrainement à la fin était de 62.59%.

On notera que ces bonnes performances des modèles F2 et G1 ont été obtenus sans entraînement des embeddings, comparativement aux modèles E4 et E2 dont le partage de paramètres est fait selon la stratégie Tensorflow. Il semble donc que la stratégie de partage de paramètres selon Keras augmente significativement plus la capacité du modèle comparativement à Tensorflow.

La Figure A3 présente les résultats pour ce même modèle G1 entraîné sur 20 et 30 epochs. On observe des profils pratiquement identiques jusqu'à 20 epochs avec plafonnement par la suite. Pour l'essai à 30 epochs, les précisions de prédictions finales obtenues pour l'ensemble de développement sont de 55.75% pour MySE et de *EM* = 58.53% et *F1* = 66.39%, soit pratiquement les mêmes que celles de l'essai à 20 epochs. La MySE de l'ensemble d'entrainement à la fin était de 68.28% comparativement à 62.59% à 20 epochs, le modèle ne généralisant plus au-delà de 20 epochs. L'augmentation du nombre d'epochs confirme le plafonnement de la performance du Modèle G1 pour l'ensemble de développement.

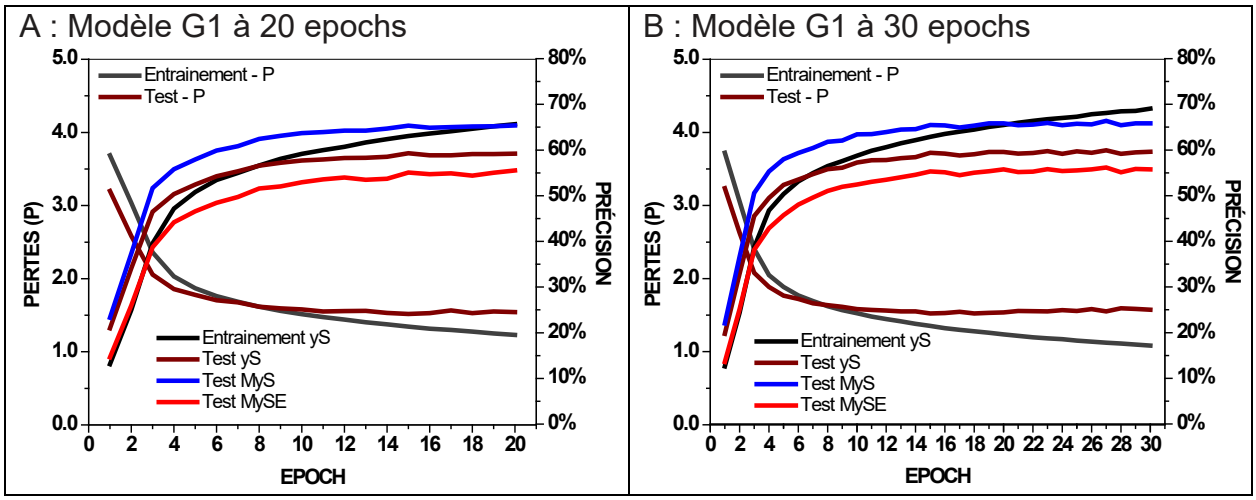


Figure A3. Pertes et précisions de prédictions du modèle G1

A.3.3 Avec embeddings de mots et de mots à base de caractères entraînés

Le modèle H1 est identique au modèle F2 (couche de sortie selon la publication originale de BIDAF [19] tel qu'illustrée à la Figure 13 et dropouts internes sur qa et ca à la couche d'embeddings contextuels) sauf qu'il comporte l'entrainement des embeddings de mots pré-entrainés GloVe 100 et de mots à base de caractères selon l'option de la couche embedding de Keras *trainable* = *True*. Il compte 10.13M paramètres entrainables.

Les précisions de prédictions finales ainsi obtenues pour l'ensemble de développement sont de 56.88% pour MySE et de *EM* = 59.56% et *F1* = 67.09%. La MySE de l'ensemble d'entrainement à la fin était de 66.46%. Comparativement au modèle F2 donc, l'entraînement des embeddings augmente de 2.5% la performance du modèle H1 sur l'ensemble de développement.

Le modèle H2 est identique au modèle G1 (couche de sortie selon l'implémentation de BIDAF tel qu'illustrée à la Figure 14 et dropouts internes sur ca et qa au BILSTM des embeddings contextuels) sauf qu'il comporte l'entrainement des embeddings de mots pré-entrainés GloVe 100 et de mots à base de caractères selon l'option de la couche embedding de Keras *trainable* = *True*. Il compte 11.09M paramètres entrainables.

Les précisions de prédictions finales ainsi obtenues pour l'ensemble de développement sont de 50.63% pour MySE et de *EM* = 53.16% et *F1* = 61.94%. La MyS de l'ensemble d'entrainement à la fin était de 63.32%. De façon surprenante, ces résultats indiquent que l'entraînement des embeddings au modèle H2 (couche de sortie selon l'implémentation de BIDAF) diminue significativement (-5%) sa performance comparativement au modèle G1 (*EM* = 58.23%) sur l'ensemble de développement, ce qui, cependant n'affecte pas la performance sur l'ensemble d'entraînement (442 : MySE = 62.59%). Ceci suggère que, pour cette configuration, l'entraînement des embeddings diminuerait la capacité du modèle H2, tout comme il avait été observé pour le modèle B1 par rapport au modèle A1.

A.4 Modulation du taux d'apprentissage

La méthode présentée à première partie de la section 3.3 a été utilisée pour estimer de meilleures plages de variation cyclique du TA pour ces modèles tel que discuté à la section A.4.1. Pour la TP, on a utilisé des multiples du nombre d'itérations par epoch (dans le cas présent : 87,599/30 = 2920 itérations par epoch, 87,599 étant le nombre d'exemples de l'ensemble d'entraînement et 30, la taille d'un lot) tel que suggéré par Smith [57]. Les sections A.4.2 et A.4.3 présentent les résultats de l'implémentation de cette stratégie de modulation du TA sur les modèles K2 et H1.

A.4.1 Estimés de meilleures plages de variation cyclique du TA

Les modèles K2 et H1 ont été testés selon les paramètres suivants :

- Grandeur d'epoch = 87,599 exemples de l'ensemble d'entraînement,
- Nombre d'itérations par epoch = 2,920,
- Valeurs minimale et maximale du TA de 0.001 et 1.5,
- Nombre d'epochs = 5 et
- Augmentation linéaire du TA de 1.027 par 10,000 epochs.

Les Figures A4.A (pertes brutes) et A4.B (pertes suite à un lissage simple) présentent les pertes à l'entraînement du modèle K2 en fonction du TA lorsque testé selon la méthode présentée à la section 3.3. On notera que ces pertes sont la somme des pertes des prédictions des positions de début (yS) et de la fin (yE) de la réponse dans le contexte

qui sont pratiquement égales tel qu'illustré à la Figure A4. Selon cette approche et la Figure A4, on observe une meilleure plage de modulation du TA pour le modèle K2 de 0.11 à 1.2.

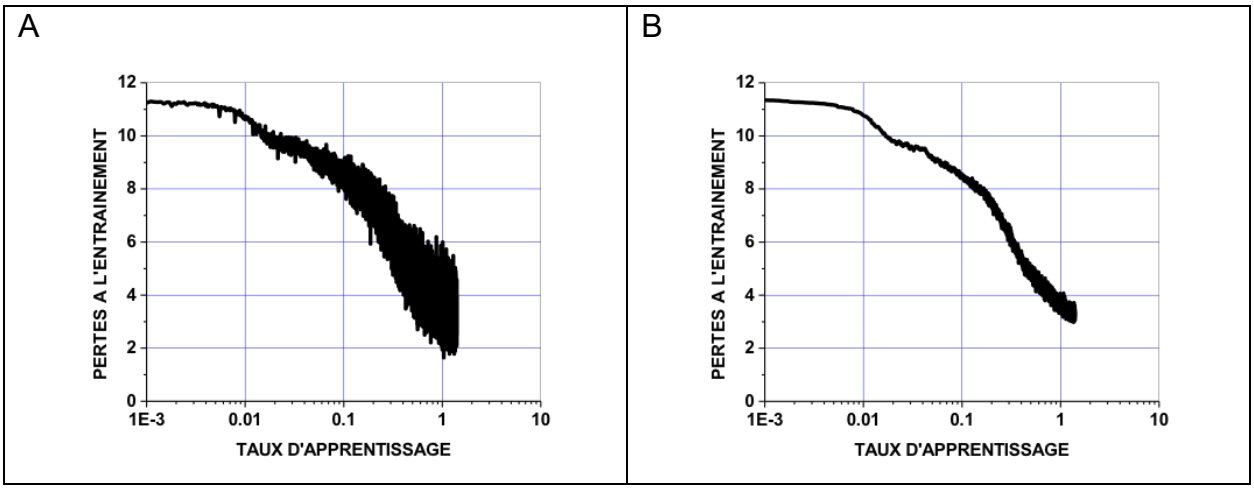


Figure A4. Pertes à l'entraînement en fonction du TA pour le modèle K2

De même, la Figure A5 présente les pertes à l'entraînement du modèle H1 en fonction du TA. Dans ce cas, on observe une meilleure plage de modulation du TA pour le modèle 443A de 0.14 à 1.2.

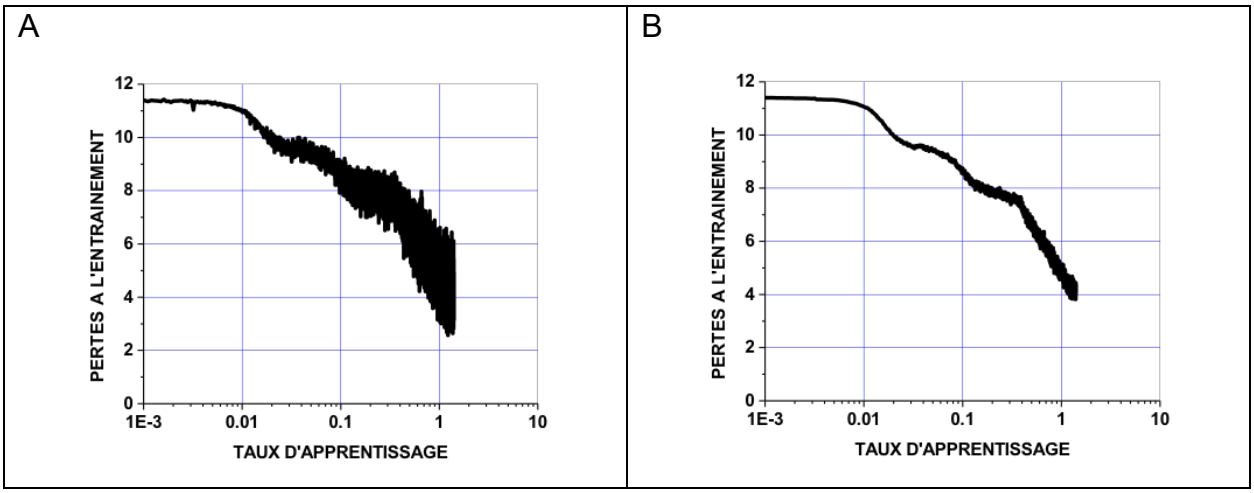


Figure A5. Pertes à l'entraînement en fonction du TA pour le modèle H1

A.4.2 Effet de variations de paramètres à une TP de 17520 sur le modèle K2

Aux valeurs des paramètres utilisées tel que décrit à la section , les essais à un seul cycle à une TP de 17520 présentent donc de meilleures performances que ceux à cycles multiples. Quelques autres essais (2.4a à 2.4b) autour de ce dernier point ont été réalisés pour tenter d'améliorer la performance du modèle, dont les résultats sur l'ensemble de développement sont résumés au Tableau A1. Ces essais n'ont pas permis d'améliorer cette performance.

A.4.3 Effet de stratégies multiples de modulation du TA sur le modèle K2

Afin de tenter de briser ce plafond de précisions de prédictions, une dernière stratégie de modulation à deux et trois étapes a donc été expérimentée dont les paramètres et les résultats pour l'ensemble de développement sont résumés au Tableau A2. À la première étape, on utilise

1. Une fonction de modulation T2, à TA minimal et maximal de 0.15 et 1.2 et à une TP de 17520 (Profil A), soit les essais S3.1 à S3.4, ou
2. Une fonction de modulation exponentielle, à TA minimal de 0.15 (Profil C) et 0.3 (Profil B) et maximal de 2.0 (TA maximaux effectifs de 1.24 et de 1.30 pour les Profiles B et C), à un gamma de 0.99997 et à une TP de 17520, soit les essais S3.10 à S3.34.

Ces fonctions et ces paramètres ont été choisis puisqu'ils présentent des profils de pertes minimales du modèle K2 avec modulation dès l'epoch 8-10 tel qu'illustré à la Figure 20B et soulignés plus haut (essais S2.4 et S2.9).

À la deuxième étape, pour les essais S3.1 à S3.19, à un point de commutation (PC), un second profil de modulation est appliqué sur 2 à 12 epochs. Ces PC ont été choisis afin de correspondre approximativement à des minimums de pertes sur l'ensemble de développement selon les essais S2.4 et S2.9 du Tableau 11. Les profils de modulation testés ont été choisis selon les résultats d'autres essais non présentés. On notera que le

Tableau A1. Paramètres et résultats d'essais de modulation du TA du modèle K2 à une TP de 17520

Essai	Paramètres			Pertes minimales Ens. de développement			Valeur max. de MySE Ens. de développement			À la fin de l'entraînement Ens. de développement			
	TA min	TA max	Nb epochs	Pertes	Epoch	MySE (%)	MySE (%)	Epoch	Pertes	Pertes	MySE (%)	EM (%)	*F1* (%)
K2	NA (TA = 0.5)		20	1.5050	18	56.49	57.24	19	1.5082	1.5292	56.83	59.55	67.26
S2.4	0.15	1.2	12	1.4847	8	56.30	58.36	12	1.4954	1.4954	**58.36**	**61.18**	**68.19**
S2.4a	0.15	1.2	15	1.4780	11	57.57	58.02	14	1.5066	1.5177	57.73	60.45	68.15
S2.4b	0.15	1.0	12	1.4839	10	56.10	57.40	12	1.4952	1.4952	57.40	60.06	67.70
S2.4c	0.15	1.5	12	1.5180	8	55.39	57.23	12	1.5465	1.5465	57.23	60.00	67.20
S2.4d	0.08	1.2	12	1.4950	10	56.76	57.29	11	1.5355	1.5313	57.11	ND	ND

(*) : Mode exponentiel

Tableau A2. Profils de modulation du TA testés avec le modèle K2 et résultats sur l'ensemble de développement

Essai	Profile à l'étape 1	PC (epoch)	Pertes au PC	Profile de TA après le PC				Nb total d'epochs	À la fin de l'entraînement			
				Mode	TA min	TA max	TP		Pertes	MySE (%)	EM (%)	*F1* (%)
H1	NA							20	1.5292	56.83	59.55	67.26
S2.4	A	NA						12	1.4953	**58.36**	**61.18**	**68.19**
S2.4f	E1 : Exp., TA min=0.1, TA max=5.0, TP=17520							12	1.5315	**58.70**	**61.42**	**68.46**
S3.1	A	11	1.4979	T2	0.10	1.0	2920	11+2	1.4782	57.18	**60.07**	**67.73**
S3.2	A	8	1.5177	T2	0.10	1.0	2920	8+8	1.5136	57.90	**60.77**	**68.08**
S3.3	A	12	1.4851	T2	0.10	0.3	2920	12+5	1.5367	57.73	**60.52**	**67.69**
S3.4	A	12	1.4915	T2	0.10	0.4	5840	12+4	1.5181	57.85	**60.85**	**68.18**
S3.10	B	8	1.4926	Exp.	0.15	0.3	1460	8+8	1.4997	**58.14**	**60.97**	**68.37**
S3.11	B	8	1.5059	Exp.	0.15	0.4	1460	8+10	1.5231	57.89	**60.65**	**67.96**
S3.12	B	8	1.4979	Exp.	0.15	0.4	2920	8+8	1.4923	**58.29**	**60.98**	**68.16**
S3.13	B	8	1.4994	Exp.	0.20	1.0	2920	8+12	1.5945	57.47	**60.33**	**68.11**
S3.14	B	8	1.4925	Exp.	0.10	0.8	8760	8+4	1.5097	57.74	**60.51**	**67.90**
S3.15	C	8	1.4870	Exp.	0.10	0.5	1460	8+8	1.5007	**58.18**	**60.92**	**68.10**
S3.16	C	8	1.5042	Exp.	0.15	0.4	2920	8+12	1.5548	57.65	**60.46**	**67.95**
S3.17	C	8	1.4691	Exp.	0.10	0.5	2920	8+10	1.4913	58.03	**60.80**	**68.26**
S3.18	C	8	1.5130	Exp.	0.10	0.6	2920	8+10	1.5551	57.19	**60.03**	**67.66**
S3.19	C	8	1.4863	Exp.	0.10	0.7	2920	8+10	1.5311	58.09	**60.73**	**68.28**
S3.30	C	8	1.5062	Exp.	0.15	2.0	1460	8+2	1.4788	57.24	ND	ND
				Exp.	0.15	0.4	2920	8+2+6	1.5078	**58.26**	**61.08**	**68.23**
S3.31	C	8	1.4979	Exp.	0.15	2.0	2920	8+2	1.4727	57.00	ND	ND
				Exp.	0.15	0.4	2920	8+2+10	1.5418	**58.25**	**61.24**	**68.44**
S3.32	C	8	1.4872	Exp.	0.10	3.0	2920	8+2	1.4616	56.54	ND	ND
				Exp.	0.10	0.5	2920	8+2+10	1.5185	**58.65**	**61.41**	**68.44**
S3.33	C	8	1.4967	Exp.	0.10	5.0	2920	8+2	1.4895	56.33	ND	ND
				Exp.	0.10	0.5	2920	8+2+10	1.5102	**58.16**	**61.12**	**68.20**
S3.34	C	8	1.4989	Exp.	0.10	3.0	1460	8+2	1.4977	56.78	ND	ND
				Exp.	0.10	0.5	2920	8+2+10	1.5511	**58.60**	**61.05**	**68.27**

nombre total d'epochs de ces essais diffèrent puisque chacun correspond à la fin du cycle associée à la TP de l'étape 2 (ou 3 pour S3.30 à S3.34) de l'essai.

En ce qui concerne les essais S3.30 à S3.34, ils comportent trois étapes (Tableau A2), soit :

1. Une première étape de Profil C comme décrit plus haut jusqu'à un PC correspondant à l'epoch 8,
2. Une seconde étape où un TA maximal élevé (2.0, 3.0 ou 5.0) est appliqué sur un ou deux epochs afin de tenter de sortir d'un minimum local et
3. Une troisième étape où un profil de TP et de TA minimal et maximal modérés est appliqué sur 6 à 12 epochs.

Des résultats présentés au Tableau A2, on peut distinguer les points suivants.

1. Les pertes minimales et en fin d'entraînement d'une majorité de ces essais, tout comme pour tous les essais précédents de cette section (Figures 26C et 26D, et 28C et 28D), sont inférieures et atteintes plus rapidement que celles du modèle originale 463B, ce qui souligne encore une fois l'effet bénéfique de la modulation du TA sur la dynamique du modèle 463B. Ceci se traduit par des augmentations modérées des précisions de prédiction en fin d'entraînement.
2. Tous ces essais, incluant ceux à pertes finales supérieures à celles du modèle 463B original, présentent des précisions de prédiction finale supérieures à celles de ce dernier.
3. Les essais S3.1 à S3.34 présentent, malheureusement, des performances marginalement inférieures ou comparables à celles des essais S2.4 et S2.4f obtenues plus rapidement, soit en 12 vs 13 à 20 epochs.

À l'exception des essais S3.2 (1.5177) et S3.18 (1.5130), tous ces essais présentent des pertes au PC approximativement de même niveau, soit 1.49-1.50.

A.4.4 Effet de stratégies multiples de modulation du TA sur le modèle H1

Tout comme pour le modèle K2, afin de tenter de briser ce plafond de précisions de prédictions, une dernière stratégie de modulation à deux étapes a donc été expérimentée. À la première étape, différents profils monocycles sont utilisés tel que résumé au Tableau A3. Ces fonctions et ces paramètres ont été choisis puisqu'ils présentent des profils de pertes minimaux du modèle H1 dès l'epoch 12 tel qu'illustrés à la Figure 21B.

À la deuxième étape, au point de commutation correspondant à l'epoch 12 (18 pour l'essai S5.4), un second profil de modulation est appliqué tel que présenté au Tableau A3. On notera que les nombres d'epochs totaux de ces essais diffèrent puisqu'ils correspondent à la fin du cycle associée à la TP de l'étape 2 de l'essai.

Les résultats présentés au Tableau A3 indiquent les points suivants.

1. Tous ces essais, à l'exception de S5.4, présentent des pertes minimales inférieures et obtenues plus rapidement que celles du modèle original H1, mais comparables à celles de l'essai S4.4.
2. Les meilleurs précisions de prédictions se comparent à celles de l'essai S4.4 à l'exception des plus hautes pertes en fin d'essai pour S5.4 à S5.8.

En résumé, ces derniers essais présentent donc, également, des performances comparables à celles obtenues aux essais S4.4 et S2.4f mais au moyen de stratégies de modulation du TA plus compliquées et en un nombre d'epochs plus élevé.

Tableau A3. Paramètres et résultats sur l'ensemble de développement de la 2e série d'essais de modulation du TA du modèle H1

Essai	Profil de modulation avant le PC (=ep 12) (TA min=0.15)			Pertes au PC	Profil de modulation après le PC (mode = étape 1; TP=2920)		Nb total d'ep.	Pertes minimales			Valeur maximale de MySE			À la fin de l'entraînement			
	TA max	TP	Mode		TA min	TA max		Pert.	Ep.	MySE (%)	MySE (%)	Ep.	Pert.	Pert.	MySE (%)	EM (%)	F1 (%)
H1	TA = 0.5			NA			20	1.5148	15	55.93	56.88	20	1.5488	1.5488	56.88	**59.56**	**67.09**
S4.4	1.0	26280	T2	NA			18	1.4735	13	57.16	58.69	17	1.4965	1.5189	58.58	**61.40**	**68.32**
S5.1	1.0	23360	T2	1.5033	0.10	0.5	12+4	1.4896	14	57.42	57.49	16	1.4940	1.4940	57.49	**60.20**	**67.42**
S5.2				1.4726	0.10	1.0	12+4	1.4795	14	56.67	57.67	14	1.4795	1.4910	58.33	**61.33**	**68.30**
S5.3				1.4845	0.10	1.0	12+8	1.4866	14	56.85	58.03	18	1.5030	1.5174	57.97	**60.82**	**68.35**
S5.4[1]	1.0	29200	T2	1.5640	0.20	0.5	18+2	1.5257	19	57.89	58.30	20	1.5369	1.5368	58.30	**60.94**	**68.11**
S5.5	2.0	26280	Exp	1.4957	0.15	0.5	12+8	1.4948	14	58.22	58.61	16	1.4996	1.5428	58.36	**61.07**	**68.43**
S5.6	2.0	23360	Exp	1.4807	0.15	0.7	12+8	1.4849	14	58.32	58.48	20	1.5246	1.5246	58.48	**61.32**	**68.44**
S5.7				1.5015	0.15	1.0	12+8	1.4863	15	57.03	58.51	18	1.5280	1.5448	58.22	**61.10**	**68.10**
S5.8	1.6	23360	Exp	1.4846	0.20	0.8	18+2	1.5060	13	57.03	58.40	16	1.5250	1.5680	58.23	**60.82**	**68.18**
S5.9[2]	1.6	17520	Exp	1.4865	0.10	1.0	12+4	1.4884	13	58.14	58.14	13	1.4884	1.4983	57.92	**60.63**	**67.90**

(1) PC à l'epoch 18
(2) TP à l'étape 2 = 1460