Guide de l'étudiant

APP3– S7

Réseaux de neurones récurrents

Faculté de génie
Université de Sherbrooke

Hiver 2021

Note : En vue d'alléger le texte, le masculin est utilisé pour désigner les femmes et les hommes.

Document guide
Rédigé par François Grondin, Jean-Samuel Lauzon et Jonathan Vincent, Hiver 2021

Table des matières

1 Activités pédagogiques et compétences

GRO722: Réseaux de neurones récurrents

1. Concevoir et mettre en œuvre des réseaux de neurones récurrents.
2. Mettre en oeuvre un réseau récurrent pour une application de séquences de symboles.

2 Synthèse de l'évaluation

La note attribuée aux activités pédagogiques de l'APP est une note individuelle, sauf pour la validation et le rapport d'APP qui est une note par équipe de deux. **Tous les membres de l'équipe doivent contribuer à la résolution de la problématique.** En cas de disparité importante dans les contributions, une note individuelle pourrait être attribuée durant la validation. **Notez également que copier partiellement ou entièrement des sections de code trouvées sur Internet constitue du plagiat.** L'évaluation porte sur les compétences figurant dans la description des activités pédagogiques de l'APP à la section 1. Ces compétences, ainsi que la pondération de chacune d'entre elles dans l'évaluation de l'unité, sont :

Activité pédagogique	**GRO722**	
Compétences	**C1**	**C2**
Rapport d'APP et validation	60	60
Évaluation sommative	120	120
Évaluation finale	120	120

Tableau 1: Pondération des points

Le pointage obtenu est ensuite utilisé pour attribuer des cotes selon cette grille :

E	D	D+	C-	C	C+	B-	B	B+	A-	A	A+
<50%	50%	53.5%	57%	60.5%	64%	67.5%	71%	74.5%	78%	81.5%	85%

Tableau 2: Grille de notes selon le pointage

3 Qualités de l'ingénieur

Les qualités de l'ingénieur visées par cette unité d'APP sont les suivantes. D'autres qualités peuvent être présentes sans être visées ou évaluées dans cette unité d'APP.

	Q01	Q02	Q03	Q04	Q05	Q06	Q07	Q08	Q09	Q10	Q11	Q12
Touchée	×	×	×		×							
Évaluée	×	×	×		×							

Tableau 3: Qualités de l'ingénieur visées

Les qualités de l'ingénieur sont les suivantes. Pour une description détaillée des qualités et leur provenance, consultez le lien suivant : `http://www.usherbrooke.ca/genie/etudiants-actuels/au-baccalaureat/bcapg/`.

Qualité	**Libellé**
Q01	Connaissances en génie
Q02	Analyse de problèmes
Q03	Investigation
Q04	Conception
Q05	Utilisation d'outils d'ingénierie
Q06	Travail individuel et en équipe
Q07	Communication
Q08	Professionnalisme
Q09	Impact du génie sur la société et l'environnement
Q10	Déontologie et équité
Q11	Économie et gestion de projets
Q12	Apprentissage continu

Tableau 4: Liste des qualités de l'ingénieur

4 Énoncé de la problématique

Vous êtes stagiaire dans une entreprise oeuvrant dans le domaine de la réalité augmentée. On vous affecte à un récent projet d'immersion où l'utilisateur pourra tracer des commandes et écrire avec seulement des gestes de la main.

Votre chargé de projet vous explique que la première itération du projet utilisera les données d'accélération provenant d'une centrale inertielle d'une montre intelligente afin de les convertir en séquences de symboles (lettres) pouvant être traitées par un ordinateur.

Pour faire une preuve de concept, un collègue a déjà fait des ensembles de données en écrivant à la main (en un seul trait continu) plusieurs centaines de mots en anglais. Ce dernier a aussi déjà prétraité les données de la centrale inertielle afin d'obtenir des séquences de coordonnées distribuées dans le temps ($\mathbf{x} \in \mathbb{R}^{2\times T}$). La figure 1 présente un exemple d'échantillon de l'ensemble de données en question.

Figure 1: Échantillon de l'ensemble de données (cible: neuron)

L'ensemble de données qui vous est fourni contient des mots anglais uniques composés d'une à cinq lettres minuscules. Seuls les symboles des 26 lettres de l'alphabet (sans accent ni variation) sont utilisés. Le tracé à la main correspondant à chaque mot est sous la forme d'une séquence de coordonnées réelles 2D prises à intervalles réguliers dans le temps. Certains tracés peuvent avoir une séquence de plusieurs centaines d'éléments.

L'ensemble de données est difficile à utiliser avec des techniques d'apprentissage machine classique. Votre chargé de projet a déjà fait une revue sommaire des différentes techniques qui pourraient s'appliquer à cette situation et vous demande d'utiliser des réseaux de neurones récurrents (e.g. Elman, LSTM, GRU). Les mécanismes d'attention ont également suscité son intérêt. Toutefois, il n'est pas convaincu et vous laisse juger de la pertinence de leur utilisation. Il vous demande également d'utiliser la bibliothèque logicielle PyTorch pour concevoir et entraîner votre réseau de neurones. Cette bibliothèque logicielle a été choisie puisqu'elle est déjà utilisée pour d'autres projets au sein de l'entreprise. Elle est également facile d'utilisation et adéquate pour du prototypage.

Vous aurez à créer une classe `Dataset` qui permettra de préparer les séquences d'entrées réelles et les séquences de symboles cibles. Afin d'accélérer l'entraînement, votre chargé de projet vous indique qu'il serait judicieux d'ajouter du remplissage (*padding*) aux séquences afin de les traiter en lots. Il vous fait aussi part de l'existence de symboles spéciaux lorsqu'il est question de séquences de symboles (un symbole de début, de fin et de remplissage). L'ensemble des symboles devront pouvoir facilement être convertis en représentation 1 parmi N (*one-hot*) et vice-versa à l'aide de dictionnaires Python.

Afin de valider les performances du système, il vous est demandé d'utiliser la distance d'édition de séquences de symboles (distance de Lavenshtein). L'entreprise désire conserver l'implémentation de cette méthode pour d'autres projets. Vous devez donc implémenter votre propre fonction qui doit être indépendante de l'application. De plus, pour déterminer quels caractères sont difficiles à reconnaître, il vous est demandé d'implémenter et de présenter la matrice de confusion des 26 lettres sur l'ensemble de test.

Pour compléter votre travail, vous devez compléter les fichiers `metrics.py`, `dataset.py`, `model.py` et `main.py` qui vous sont fournis. Le fichier `metric.py` devra contenir votre algorithme de calcul de la distance d'édition de séquences de symboles ainsi que votre fonction de calcul de matrice de confusion. Le fichier `dataset.py` devra être complété pour faire une base de données (objet *dataset*) à partir des fichiers de données binaires `data_train.p` et `data_test.p` qui vous sont fournis. Le fichier `model.py` devra contenir votre classe d'architecture du réseau de neurones récurrent. Finalement, le fichier `main.py` devra contenir votre boucle d'entraînement, de validation et de test. Vous devrez afficher les courbes d'entraînements et fournir des exemples de tests.

Dans une phase ultérieure du projet, le modèle devra fonctionner sur un micro-ordinateur limité en puissance de calcul et en mémoire. Pour ce faire, votre chargé de projet vous indique que votre modèle devrait avoir au maximum 35 000 paramètres. Il vous mentionne également que vous devrez choisir (avec justifications) l'architecture, la fonction de coût, l'optimisateur et le type de couches récurrentes qui selon vous répondent le mieux à la tâche. Il vous demande aussi de valider si l'utilisation d'une couche d'attention et/ou l'utilisation de couches récurrentes bidirectionnelles seraient bénéfiques.

5 Connaissances nouvelles

Connaissances déclaratives: Quoi

- Dépliement des paramètres
- Représentation des entrées/sorties
- Disparition et explosion des gradients
- Architecture d'un réseau récurrent simple (RNN)
- Architecture d'un réseau récurrent à mémoire court et long terme (LSTM)
- Architecture d'un réseau récurrent à portes (GRU)
- Mécanisme d'attention

Connaissances procédurales: Comment

- Définir l'architecture d'un réseau de neurones avec un cadre logiciel
- Charger un ensemble de données grâce à un cadre logiciel
- Calculer automatiquement la propagation des gradients à l'aide d'un cadre logiciel
- Effectuer l'entraînement d'un réseau de neurones avec un processeur dédié
- Calculer la distance d'édition

Connaissances conditionnelles: Quand

- Choisir entre un RNN, LSTM et GRU
- Choisir une tâche de génération, prédiction, annotation et traduction

6 Guide de lecture

6.1 Références obligatoires

Voici les références qui sera à l'étude pour cet APP :

The Hundred-Page Machine Learning Book de Andriy Burkov.
Ce livre est disponible en version française, et a été utilisé pour GRO-720 et GRO-721.

Des notes de cours sont également disponibles à la fin de ce guide à la section 15 et des ressources en ligne seront suggérées dans la séquence de lecture.

6.2 Séquence de lecture suggérée

6.2.1 Préparation au 1er prodécural

- Livre – Chapitre 6 : Réseaux de neurones et apprentissage profond, pp. 69–81
- Notes – Section 15.1 : Traitement de séquences
- Notes – Section 15.2: Types de réseaux récurrents

6.2.2 Préparation au 1er laboratoire

- Réseaux récurrents: `https://pytorch.org/docs/stable/nn.html#recurrent-layers`
- Couche RNN (Elman): `https://pytorch.org/docs/stable/generated/torch.nn.RNN.html#torch.nn.RNN`
- Couche LSTM : `https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html#torch.nn.LSTM`
- Couche GRU : `https://pytorch.org/docs/stable/generated/torch.nn.GRU.html#torch.nn.GRU`

6.2.3 Préparation au 2e procédural

- Section 15.3: Réseaux récurrents bidirectionnels
- Section 15.4: Remplissage
- Section 15.5: Module/couche d'attention
- Section 15.6: Métrique pour séquences
- Section 15.7: Extraction de symboles vers jetons (*tokenization*)
- Section 15.8: Couche de vectorisation (*embedding layer*)
- Section 15.9: Stratégies

6.2.4 Préparation au 2e laboratoire

- Réseaux récurrents: `https://pytorch.org/docs/stable/nn.html#recurrent-layers`
- Couche GRU : `https://pytorch.org/docs/stable/generated/torch.nn.GRU.html#torch.nn.GRU`
- Couche de vectorisation : `https://pytorch.org/docs/stable/generated/torch.nn.Embedding.html?highlight=embeding`
- Multiplication matricielle de lots : `https://pytorch.org/docs/stable/generated/torch.bmm.html?highlight=bmm#torch.bmm`
- Copie de tenseur : `https://pytorch.org/docs/stable/tensors.html?highlight=repeat#torch.Tensor.repeat`
- Fonction Softmax : `https://pytorch.org/docs/stable/nn.functional.html?highlight=softmax#torch.nn.functional.softmax`

- Fonction Sommation : `https://pytorch.org/docs/stable/generated/torch.sum.html?highlight=sum#torch.sum`

7 Logiciels

Pour cet APP, vous aurez besoin d'installer l'environnement de Python 3, disponible gratuitement en ligne pour tous les systèmes d'exploitation (`www.python.org`). Il est également suggéré d'utiliser le gestionnaire de progiciel Pip pour installer Pytorch (`https://pytorch.org/`). Il est recommandé d'utiliser l'IDE PyCharm (`https://www.jetbrains.com/fr-fr/pycharm/`).

8 Sommaire des activités liées à l'unité

- Tutorat d'ouverture
- Études personnelles et préparation à la première activité procédurale
- Activité procédurale 1
- Activité en laboratoire 1
- Activité procédurale 2
- Activité en laboratoire 2
- Validation de la problématique
- Tutorat de fermeture
- Évaluation sommative

9 Productions à remettre

- Validation pratique de la solution en laboratoire. Vous devrez présenter votre solution en équipe lors de cette rencontre. **La présence des deux membres de l'équipe est obligatoire.**
- Rapport d'APP à remettre avant 9h00 le jour du deuxième tutorat. Les consignes de réaction de l'unité d'APP sont données à la section 9.2.

Tout retard sur la remise de livrable entraîne une pénalité de 20% par jour.

9.1 Formation des équipes

La taille des équipes pour l'APP est fixée à 2. Aucune exception ne sera accordée. Veuillez utiliser le formulaire disponible sur le site de la session pour inscrire votre équipe.

9.2 Livrables

Votre solution à la problématique sera évaluée lors d'une séance de validation, mais surtout par un rapport qui décrira vos choix technologiques et le code que vous aurez développé. On attend donc une paire de livrables (rapport et code) par équipe de 2 étudiants.

Rapport

Le rapport, de 10 pages **au maximum**, devra contenir :

- Une page de présentation
- Une introduction pour expliquer la problématique
- Une description de la tâche à accomplir (traduction)
- Une justification de l'architecture choisie
- Un schéma-bloc des différentes couches du réseau
- Le nombre total de paramètres à entraîner dans le réseau
- Une description des hyperparamètres utilisés pour l'entraînement : fonction de coût, taux d'apprentissage, taille des lots, etc.
- Un graphique et interprétation de l'erreur en fonction des époques pour les sous-ensembles d'entraînement et de validation
- Un graphique et interprétation de la distance d'édition moyenne en fonction des époques pour les sous-ensembles d'entraînement et de validation
- Une représentation et interprétation de la matrice de confusion des différents symboles
- Une représentation de l'attention sur les coordonnées (facultatif)
- Un exemple de traduction avec le sous-ensemble de test
- Une conclusion qui fait un retour sur les performances obtenues avec les architectures proposées

Fichiers de code

Vous devez également remettre le code complet de votre solution. Vous pouvez inclure seulement les fichiers que vous avez modifiés (ou ajoutés) aux fichiers fournis pour l'APP. Votre code devra être commenté et organisé clairement.

Procédure de dépôt

Vous devrez combiner en une seule archive ZIP votre rapport **en format PDF** et vos fichiers de code pour le dépôt. Le lien vers la plateforme de dépôt sera disponible sur le site web de la session.

10 Évaluations

10.1 Rapport d'APP

Le contenu du rapport sera évalué selon cette pondération :

Activité pédagogique	**GRO722**	
Compétences	**C1**	**C2**
Rapport	**40**	**40**
Description de la tâche à accomplir	10	–
Justification de l'architecture	15	–
Schéma-bloc du réseau, incluant le nombre de paramètres	15	–
Description des hyperparamètres	–	10
Graphique et interprétation de la fonction de coût en fonction des époques	–	10
Graphique et interprétation de la distance d'édition en fonction des époques	–	5
Représentation et interprétation de la matrice de confusion	–	5
Exemples de résultats avec le sous-ensemble de test	–	10
Validation	**20**	**20**
Total	**60**	**60**

Tableau 5: Pondération des points

10.2 Validation

Votre solution sera également validée lors de la séance de validation. Elle sera évaluée de façon critériée. Vous n'avez rien à ajouter au rapport en lien avec la validation, mais le pointage résultant de l'évaluation y sera ajouté lors de la correction. Le tableau 6 présente la grille d'évaluation pour la validation.

10.3 Évaluation sommative

L'évaluation sommative porte sur tous les objectifs d'apprentissage de l'unité. C'est un examen à la fois théorique et pratique qui se fera en laboratoire et **sans documentation** (à l'exception de la documentation PyTorch pertinente). Des informations pertinentes seront ajoutées en annexe à l'examen.

10.4 Évaluation finale

L'évaluation finale se fera par activité pédagogique et portera sur tous les objectifs d'apprentissage de cette activité pédagogique. C'est un examen à la fois théorique et pratique qui se fera en laboratoire et **sans documentation** (à l'exception de la documentation PyTorch pertinente). Des informations pertinentes seront ajoutées en annexe à l'examen.

Qualité	**Q01**	**Q02**	**Q03**	**Q05**
Compétence	**C1**	**C2**	**C2**	**C1**
Pondération	**10**	**10**	**10**	**10**
Excellent (4) – 100%	Connaît adéquatement les architectures de réseaux de neurones récurrents	Applique efficacement la procédure d'entraînement de réseaux de neurones récurrents et analyse parfaitement les résultats	Investigue parfaitement l'impact des hyperparamètres sur les performances du réseau	Les modules pour constituer le réseau de neurones sont parfaitement implémentés
Cible (3) – 85%	Connaît la plupart des architectures de réseaux de neurones récurrents	Applique la procédure d'entraînement de réseaux de neurones récurrents et analyse correctement les résultats	Investigue correctement l'impact des hyperparamètres sur les performances du réseau	Les modules pour constituer le réseau de neurones sont correctement implémentés
Seuil (2) – 60%	Connaît les architectures essentielles des réseaux de neurones récurrents	Applique en bonne partie la procédure d'entraînement de réseaux de neurones récurrents et analyse les résultats avec quelques erreurs mineures	Investigue partiellement l'impact des hyperparamètres sur les performances du réseau avec des erreurs mineures	Les modules pour constituer le réseau de neurones sont partiellement implémentés (les fonctionnalités essentielles sont présentes)
Insuffisant (1) – 25%	Connaît superficiellement les architectures de réseaux de neurones récurrents	Applique minimalement la procédure d'entraînement de réseaux de neurones et analyse les résultats avec plusieurs erreurs importantes	Investigue minimalement l'impact des hyperparamètres sur les performances du réseau avec des erreurs majeures	Les modules pour constituer le réseau de neurones sont partiellement implémentés (les fonctionnalités essentielles sont absentes)
Non initié (0) – 0%	Ne connaît pas les architectures des réseaux de neurones récurrents	N'applique pas la procédure d'entraînement de réseaux de neurones récurrents et n'est pas en mesure d'analyser les résultats	N'investigue pas l'impact des hyperparamètres sur les performances du réseau	Les modules pour constituer le réseau de neurones ne sont pas implémentés

Tableau 6: Grille de validation

11 Formation à la pratique procédurale 1

11.1 Buts de l'activité

- Identifier les tâches de traitement de séquences de symboles.
- Se familiariser avec les architectures de réseaux de neurones récurrents.

11.2 Guide de lecture

The Hundred-Page Machine Learning Book

- Chapitre 6 : Réseaux de neurones et apprentissage profond, pp. 69–81

Notes de cours

- Section 15.1: Traitement de séquences
- Section 15.2: Types de réseaux récurrents

11.3 Contenu

Question 1 – Indiquez s'il s'agit d'une tâche de génération, prédiction, annotation ou traduction.

1. Un spectrogramme est composé de plusieurs trames dans le domaine de la fréquence (obtenues via une *Short Time Fourier Transform*), et le réseau doit prédire un masque comportant le même nombre de trames avec des valeurs comprises entre 0 et 1 pour indiquer si chaque zone de temps-fréquence est dominée par un signal de parole ou de bruit.
2. Lorsque vous entrez des lettres dans une zone de texte de votre téléphone intelligent, celui-ci vous suggère plusieurs mots qui pourraient débuter par cette séquence.
3. Une séquence vidéo est présentée à un réseau de neurones, et ce dernier doit déterminer si la séquence présente ou pas une personne qui se déplace en courant.
4. Vous présentez au réseau de neurones un trajet en voiture composé d'une liste de consignes routières (e.g., tourner à droite, avancer jusqu'à une intersection) effectuées pour se rendre du point A et du point B, et le réseau doit générer une nouvelle liste pour retourner du point B au point A.

Question 2 – Soit un réseau de neurones récurrent simple (Elman) avec les paramètres suivants:

$$\mathbf{W}_h = \begin{bmatrix} +2 & -1 \\ +0 & -2 \\ -1 & +3 \end{bmatrix} \quad \mathbf{U}_h = \begin{bmatrix} +1 & +0 & -1 \\ +3 & -1 & +2 \\ -2 & +1 & +1 \end{bmatrix} \quad \mathbf{b}_h = \begin{bmatrix} +1 \\ +0 \\ -2 \end{bmatrix} \quad \mathbf{W}_y = \begin{bmatrix} +2 & -1 & +3 \end{bmatrix} \quad \mathbf{b}_y = \begin{bmatrix} +5 \end{bmatrix}$$

Le réseau doit effectuer une tâche de prédiction avec la séquence suivante:

$$\mathbf{x}[0] = \begin{bmatrix} +1 \\ -1 \end{bmatrix} \quad \mathbf{x}[1] = \begin{bmatrix} +3 \\ +1 \end{bmatrix} \quad \mathbf{x}[2] = \begin{bmatrix} +0 \\ -2 \end{bmatrix}$$

Avec l'état initial suivant:

$$\mathbf{h}[-1] = \begin{bmatrix} +2 \\ +0 \\ +1 \end{bmatrix}$$

Et la fonction d'activation σ_h est une ReLU, et σ_y est une sigmoïde. Quelle est la prédiction $y[2]$?

Question 3 – Soit un réseau de neurones LSTM avec les paramètres suivants:

$$\mathbf{U}_f = [+1],\ \mathbf{W}_f = [-1],\ \mathbf{b}_f = [+0]\ \mathbf{U}_i = [+1],\ \mathbf{W}_i = [+1],\ \mathbf{b}_i = [+0]\ \mathbf{U}_o = [+1],\ \mathbf{W}_o = [+1],\ \mathbf{b}_i = [+0]$$

Que se passe-t-il au niveau de la mémoire à long terme lorsque:

1. $x[t] \gg h[t-1] > 1.0$
2. $x[t] \ll h[t-1] < -1.0$

Question 4 – Soit deux réseaux de neurones, LSTM et GRU, avec les vecteurs d'entrées et sorties suivants: $\mathbf{h}[t] \in \mathbb{R}^{512}$ et $\mathbf{x}[t] \in \mathbb{R}^{128}$. Combien y a-t-il de paramètres à apprendre pour des réseaux LSTM et GRU à un seul niveau? Que constatez-vous?

Question 5 – Un réseau de neurones récurrent de type LSTM est utilisé pour détecter l'activité vocale dans un signal sonore. Le réseau reçoit en entrée des trames d'un spectrogramme, et génère pour chaque trame une sortie scalaire (0 signifie aucun signal de voix dans la trame, et 1 signifie un signal de voix dans la trame). Le réseau est entraîné avec des signaux d'une durée de 5 secondes, généralement composés d'un silence, un signal de parole avec de courtes pauses entre les mots, puis un deuxième silence. Une fois le réseau entraîné, vous l'utilisez sur des signaux audio de différentes longueurs, et réalisez que le réseau a tendance à sous-estimer la présence de voix pour de longs signaux sonores de 10 secondes et plus. Pourquoi?

12 Formation à la pratique en laboratoire 1

12.1 Buts de l'activité

- Se familiariser avec la bibliothèque logicielle PyTorch pour le traitement de séquences (définition de modèles, chargement d'ensemble de données, entrainements, etc.).

12.2 Guide de lecture

Aide de la bibliothèque logicielle PyTorch

- Réseaux récurrents: `https://pytorch.org/docs/stable/nn.html#recurrent-layers`
- Couche RNN (Elman): `https://pytorch.org/docs/stable/generated/torch.nn.RNN.html#torch.nn.RNN`
- Couche LSTM : `https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html#torch.nn.LSTM`
- Couche GRU : `https://pytorch.org/docs/stable/generated/torch.nn.GRU.html#torch.nn.GRU`

12.3 Contenu

Dans ce laboratoire, vous travaillerez avec un ensemble de signaux périodiques simulés. Les séquences ont une seule dimension de nombres réels entre -1 et 1. Vous devrez comparer dans un premier temps différents types de réseaux de neurones récurrents pour accomplir une tâche de d'annotation (prédire la valeur du signal au prochain pas de temps). Ensuite, à partir de la moitié du signal d'entrée, vous devrez générer le reste du signal. Le dossier distribution contient les fichiers `main.py` , `dataset.py`, `models.py` et `data.p` qui devront être complétés.

Question 1 – Préparation de l'ensemble de données

Complétez la classe *SignauxDataset* du fichier `dataset.py`. Dans la fonction d'initialisation, on vient lire et décoder le fichier binaire `data.p` pour le charger dans la variable *self.data*. Afin de compléter la classe, vous devez effectuer les étapes suivantes:

- Complétez la fonction membre *len* qui renvoie le nombre d'échantillons de l'ensemble de données.
- Complétez la fonction membre *getitem* qui renvoie l'échantillon à un indice passé en paramètre. La sortie devrait être un tuple de deux séquences (entrée et cible) sous la forme de tenseur Pytorch.

En interprétant ce fichier python après avoir fait les modifications nécessaires, vous devriez obtenir un graphique présentant les deux signaux. Le signal d'entrée devrait être plus bruité que le signal cible.

Question 2 – Réseau récurrent (Elman)

Complétez la classe *Model* du fichier `models.py`. Vous devez effectuer les étapes suivantes:

- Complétez la fonction d'initialisation. Le modèle devrait utiliser un réseau récurrent de type **Elman**. Il est recommandé de mettre le paramètre *batch_first* à `Vrai` afin de réserver la première dimension des tenseurs d'entrée pour les échantillons du lot.
- Complétez la fonction membre *forward* qui effectue la passe avant du modèle et renvoie une séquence de même taille que l'entrée. De plus vous devrez gérer le cas où le contexte est passé en paramètre (h=`None`). Nous vous conseillons d'utiliser l'outil de *debug* afin de valider les dimensions de vos tenseurs.
- Quelle fonction d'activation serait la plus appropriée selon vous?

Vous devriez être capable de lancer ce fichier python après avoir fait les modifications nécessaires.

Question 3 – Entraînement du modèle

Complétez la boucle d'entraînement et de validation du fichier `main.py`. Portez une attention particulière aux formes de vos tenseurs. Après avoir fait les modifications, vous devriez être en mesure de voir les courbes d'apprentissage.

Question 4 – Annotation

Complétez la section permettant d'évaluer et d'afficher les résultats d'annotations sur des échantillons aléatoires de l'ensemble de validation. Vous devrez mettre la variable d'affichage au début du fichier à `Vrai`. Une fois cette section complétée vous devriez être en mesure de visualiser la sortie de votre modèle en contraste avec la séquence cible. Une fois le code complété, vous devriez obtenir un résultat semblable à la figure 2.

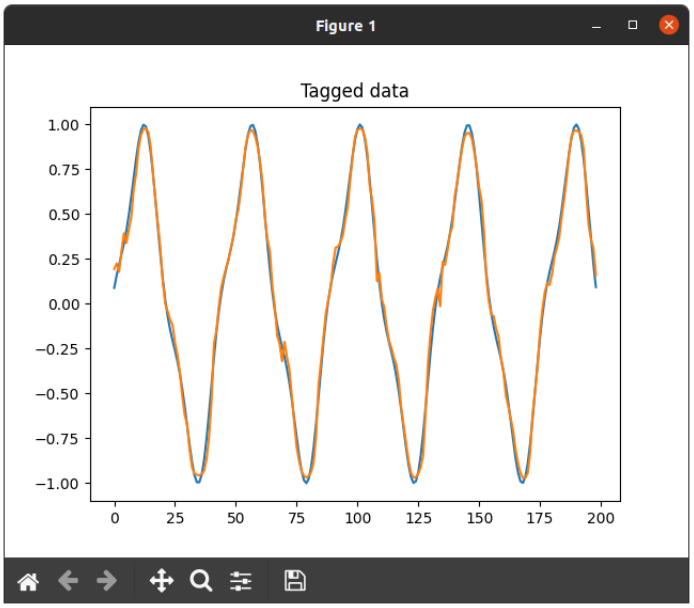


Figure 2: Exemple de sortie du numéro 4

Question 5 – Génération

Complétez la section permettant d'évaluer et d'afficher les résultats de génération sur des échantillons aléatoires de l'ensemble de validation. Vous devrez mettre la variable d'affichage au début de fichier à `Vrai`. Pour ce numéro, vous aurez à évaluer la moitié de la séquence (annotation) puis garder uniquement le dernier contexte et la dernière sortie. Ensuite, de façon itérative, vous devez générer une sortie à la fois puis copier le contexte et la sortie pour l'itération suivante. La figure 3 illustre la tâche demandée. Une fois le code complété, vous devriez obtenir un résultat semblable à la figure 4

Question 6 – LSTM et GRU

Modifiez la classe *Model* afin d'utiliser des unités LSTM ou GRU. Réentrainez le modèle et comparez les séquences générées. Que remarquez vous? Pourquoi?

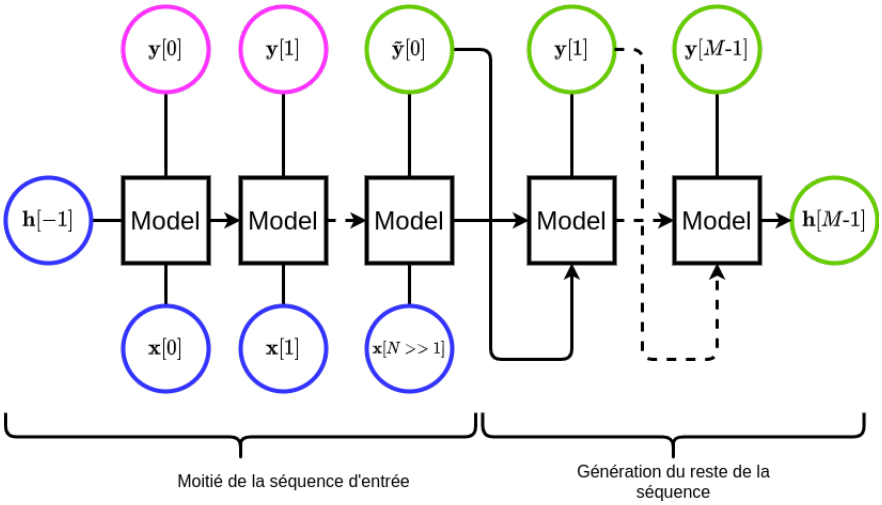


Figure 3: Modèle du laboratoire1 utilisé pour de la génération

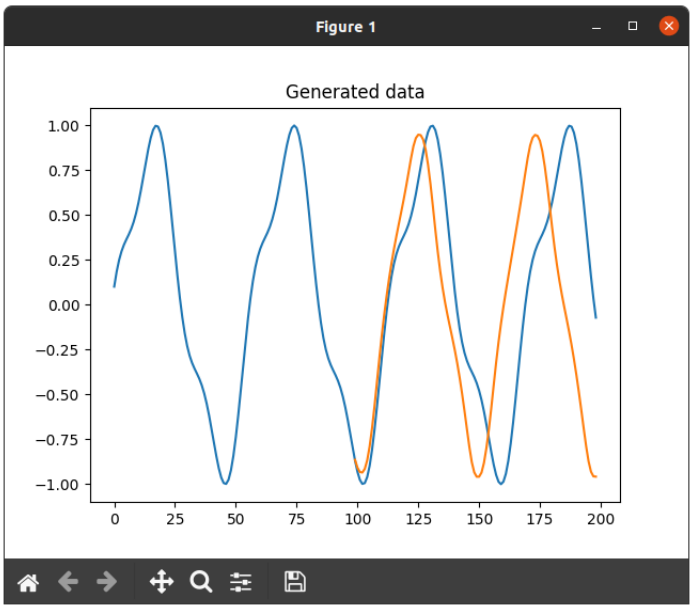


Figure 4: Exemple de sortie du numéro 5

13 Formation à la pratique procédurale 2

13.1 Buts de l'activité

- Utiliser une métrique pour comparer des séquences de symboles de tailles variables.
- Utiliser du remplissage (*padding*) dans un contexte de réseaux récurrents.
- Se familiariser avec les architectures plus avancées de réseaux de neurones récurrents (biRNN, Seq2Seq, module d'attention).

13.2 Guide de lecture

The Hundred-Page Machine Learning Book

- Chapitre 6 : Réseaux de neurones et apprentissage profond, pp. 69–81

Notes de cours

- Section 15.3: Réseaux récurrents bidirectionnels
- Section 15.4: Remplissage
- Section 15.5: Module/couche d'attention
- Section 15.6: Métrique pour séquences
- Section 15.7: Extraction de symboles vers jetons (*tokenization*)
- Section 15.8: Couche de vectorisation (*embedding layer*)
- Section 15.9: Stratégies

13.3 Contenu

Question 1 – En utilisant l'algorithme de distance d'édition avec matrice, déterminez la distance entre les séquences suivantes:

1. $x = \{2, 1, 1, 2\}$ et $y = \{1, 2, 1, 1\}$
2. $x = \{2, 1, 1\}$ et $y = \{2, 1, 2, 1, 1\}$
3. $x = \{juste\}$ et $y = \{sujet\}$

Question 2 – Un ami vous mentionne qu'il a entraîné un modèle basé sur des réseaux récurrents pour prédire le prix de l'action `FNW` du lendemain. Après avoir entraîné son modèle sur une séquence composée de la valeur des actions `FNW`, il prétend obtenir une erreur quadratique moyenne presque nulle sur son ensemble de test, et vous encourage à investir rapidement. Vous êtes sceptique et inspectez le schéma de l'architecture du modèle (voir Figure 5). Quelles conclusions en tirez-vous?

Question 3 – Pour entrainer un modèle de traduction de mots de l'anglais vers le français, vous utilisez le petit ensemble suivant. L'extraction des symboles se fait au niveau des lettres. On vous demande d'utiliser un modèle de traduction de base.

$$\begin{cases} \{hello\} & \mapsto \{allo\} \\ \{hat\} & \mapsto \{chapeau\} \\ \{gold\} & \mapsto \{or\} \end{cases} \tag{1}$$

1. En utilisant un dictionnaire de jetons où $\{a \mapsto 0, b \mapsto 1, ..., z \mapsto 25\}$, réécrivez l'ensemble de données sous la forme de jetons.

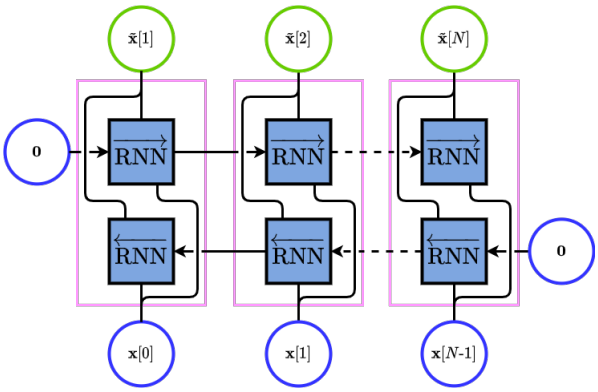


Figure 5: Architecture du réseau de prédiction du prix de l'action `FNW`

2. Comment devrait-on modifier les données pour pouvoir les traiter par lots? À quoi ressembleraient les données?
3. Dessinez l'architecture typique d'un réseau de neurones récurrents pour un modèle de traduction.
4. Quelle fonction de coût serait appropriée pour la tâche?
5. Si la taille du lot est de 3, quels seraient la taille et le type du tenseur en sortie du modèle? Si une couche *nn.embedding* est présente à l'entrée du modèle, quels devraient être la taille et le type du tenseur d'entrée?

Question 4 – Les sous-questions suivantes portent sur le concept d'attention/alignement.

1. À quoi sert un module d'attention? Dans quelle(s) situation(s) est-il utile?
2. De quoi est composé un module d'attention en général? Comment l'utilise-t-on dans une tâche de traduction comme au numéro précédent?
3. Reprenez l'exemple de la figure 17 en utilisant la mesure de similarité cosinus à la place du produit scalaire. Quels sont les poids et le vecteur d'attention? La mesure de similarité cosinus est donnée par: $cos(\theta) = \frac{x \cdot y}{\|x\|\|y\|}$
4. Proposez et dessinez une architecture de réseau récurrent pour traduction avec attention et couche bidirectionnelle.
5. Pouvez-vous imaginer des exemples de tâches où l'attention pourrait être exploitée?

14 Formation à la pratique en laboratoire 2

14.1 Buts de l'activité

- Se familiariser avec la métrique de distance d'édition.
- Se familiariser avec le concept de remplissage.
- Se familiariser avec l'architecture Seq2Seq.
- Se familiariser avec l'architecture Seq2Seq avec module d'attention.

14.2 Guide de lecture

Aide de la bibliothèque logicielle PyTorch

- Réseaux récurrents: `https://pytorch.org/docs/stable/nn.html#recurrent-layers`
- Couche GRU : `https://pytorch.org/docs/stable/generated/torch.nn.GRU.html#torch.nn.GRU`
- Couche de vectorisation : `https://pytorch.org/docs/stable/generated/torch.nn.Embedding.html`
- Multiplication matricielle de lots : `https://pytorch.org/docs/stable/generated/torch.bmm.html`
- Copie de tenseur : `https://pytorch.org/docs/stable/tensors.html?highlight=repeat#torch.Tensor.repeat`
- Fonction Softmax : `https://pytorch.org/docs/stable/nn.functional.html?highlight=softmax#torch.nn.functional.softmax`
- Fonction Sommation : `https://pytorch.org/docs/stable/generated/torch.sum.html`

14.3 Contenu

Dans ce laboratoire, vous travaillerez avec des séquences de mots dans le contexte de la traduction de phrases.

Question 1 – Distance d'édition

Compléter la fonction *edit_distance* du fichier `metrics.py`. Dans le fichier `metrics.py`, un exemple d'utilisation de la fonction vous est fourni dans le *main*. Lors du calcul de la distance d'édition entre le mot *'allo'* et *'apollo2'*, vous devriez obtenir un résultat de 3.

Exécutez directement le fichier `metrics.py` pour lancer le test.

Question 2 – Remplissage

Compléter la classe *Fr_En* du fichier `dataset.py`. Afin de compléter la classe, vous devez faire l'ajout du remplissage aux données provenant de la base de données. Cela vous permettra de faire l'entraînement de lots. Dans le fichier `dataset.py`, un exemple d'utilisation de la classe vous est fourni dans le *main*. Cet exemple affiche une séquence en français avec la traduction anglaise.

Voici ce que vous devriez obtenir :

Sans padding —

Francais: *je n ' y parviens pas .*

Anglais: *i can ' t make it .*

Avec padding —

Francais: *je n ' y parviens pas . <eos ><pad ><pad ><pad >*

Anglais: *i can ' t make it . <eos ><pad >*

Question 3 – Seq2Seq

Complétez la classe *Seq2Seq* du fichier `models.py`. Vous devez effectuer les étapes suivantes:

- Compléter la fonction membre *encodeur*.
- Compléter la fonction membre *decodeur*.

Afin de tester le modèle, vous devez lancer l'exécution du fichier `main.py`. Ce fichier ne doit pas être modifié pour ce numéro. En lançant l'exécution, l'entraînement du réseau débutera et le programme lancera le test.

Vous devriez obtenir un résultat similaire à ceci :

Input: tom a besoin d ' aide . <eos >

Target: tom needs help . <eos >

Output: tom ruined it . <eos >

Input: je me suis senti triste . <eos >

Target: i felt sad . <eos >

Output: i felt happy . <eos >

Input: tom l ' a pris . <eos >

Target: tom took it . <eos >

Output: tom needs it . <eos >

Quel est le problème avec la prédiction? Pourquoi cela se produit-il?

Question 4 – Seq2Seq avec attention

Complétez la classe *Seq2Seq_attn* du fichier `models.py`. Vous devez effectuer les étapes suivantes:

- Compléter la fonction membre *encodeur*.
- Compléter la fonction membre *attentionModule*.
- Compléter la fonction membre *decoderWithAttn*.

Afin de tester le modèle, vous devez lancer l'exécution du fichier `main.py`. Vous devez modifier l'instanciation du modèle dans le fichier `main.py` pour utiliser le *Seq2Seq_attn*. En lancant l'exécution, le programme débutera l'entraînement du réseau et ensuite lancera le test de celui-ci.

Vous devriez obtenir un résultat similaire à ceci :

Input: je dispose d ' une clé . <eos >

Target: i have a key . <eos >

Output: i have a key . <eos >

Input: nous n ' avons rien vu . <eos >

Target: we saw nothing . <eos >

Output: we saw nothing . <eos >

Input: ça ne fera pas mal . <eos >

Target: it won ' t hurt . <eos >

Output: it won ' t hurt . <eos >

Pourquoi les résultats sont-ils meilleurs? Que remarquez-vous au niveau des poids de l'attention?

15 Notes de cours

Cette section introduit certains concepts non-couverts ou couverts rapidement dans l'ouvrage de référence proposé pour ce module.

15.1 Traitement de séquences

Une séquence est un ensemble d'éléments (discrets ou continus) pour lesquels l'ordre est important. Par exemple, les lettres d'un mot, les mots d'une phrase, les phonèmes d'un segment de paroles et les coordonnés GPS d'une trajectoire constituent des éléments d'une séquence.

Soit la séquence de vecteurs $[\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{N-1}]$. Plusieurs tâches peuvent être effectuées:

1. Génération: $[\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{N\text{-}1}] \mapsto [\tilde{\mathbf{x}}_N, \tilde{\mathbf{x}}_{N+1}, \ldots, \tilde{\mathbf{x}}_{N+M\text{-}1}]$, où $N, M \in \mathbb{N}$
2. Prédiction: $[\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{N\text{-}1}] \mapsto \mathbf{y}$, où $N \in \mathbb{N}$
3. Annotation: $[\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{N\text{-}1}] \mapsto [\mathbf{y}_0, \mathbf{y}_1, \ldots, \mathbf{y}_{N\text{-}1}]$, où $N \in \mathbb{N}$
4. Traduction: $[\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{N\text{-}1}] \mapsto [\mathbf{y}_0, \mathbf{y}_1, \ldots, \mathbf{y}_{M\text{-}1}]$, où $N, M \in \mathbb{N}$

Dans le cas de la **génération**, on cherche à déterminer la correspondance entre une séquence d'entrée et une séquence de sortie cible qui correspond à la suite de l'entrée (e.g. générer une séquence de notes de musique à partir des premières notes). Pour la **prédiction**, on veut déterminer la correspondance entre une séquence complète et une seule cible (e.g. classifier un son à partir d'une trame audio). Pour l'**annotation**, chaque élément de la séquence est associé à une cible (e.g. déterminer la classe de chaque mot d'une phrase). Finalement, pour une **traduction**, on cherche à déterminer la correspondance entre une séquence d'entrée complète et une séquence cible de taille différente (e.g. traduire une phrase du français vers anglais).

Une première stratégie consiste à utiliser un réseau entièrement connecté pour traiter la séquence complète en une passe. Deux problèmes ressortent cependant de cette approche: 1) la quantité importante de paramètres et 2) la redondance des paramètres. En effet, la longueur de la séquence vient directement affecter la taille du réseau et peut devenir problématique pour des séquences d'entrées de plusieurs milliers d'éléments. De plus, puisque la position des éléments peut varier en fonction de la taille de la séquence, il y aura de la redondance inutile dans les poids.

Pour pallier à ces problèmes, il est pratique d'utiliser un réseau récurrent pour chaque élément de la séquence. Ce réseau a deux types d'entrées, l'élément de la séquence ainsi qu'un vecteur de contexte provenant de l'itération précédente, tel qu'illustré à la figure 6.

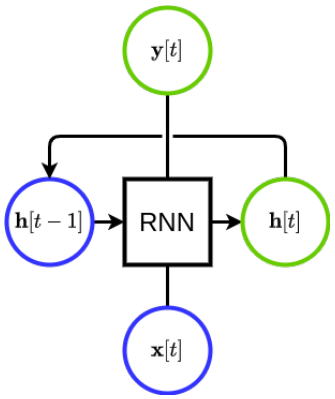
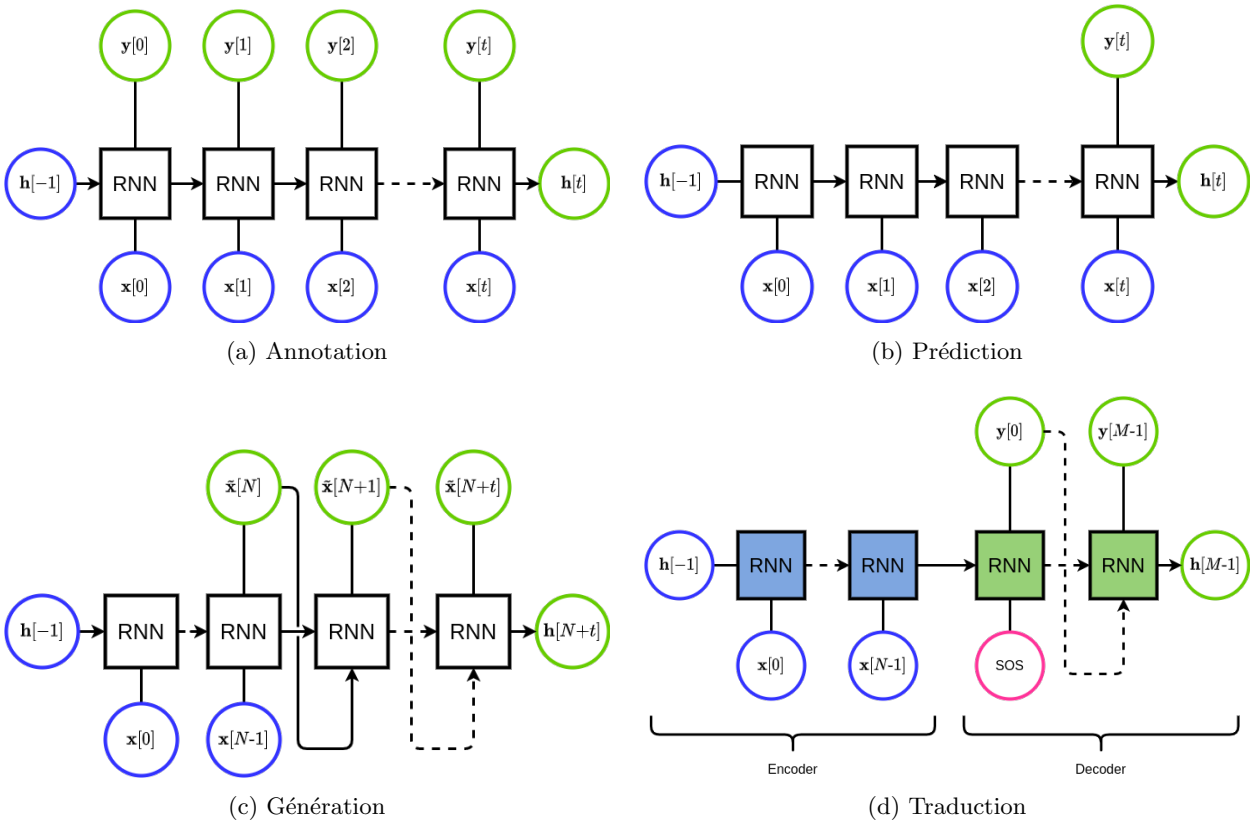


Figure 6: Réseau de neurones récurrents générique

La rétroaction de $\mathbf{h}_i$ vers $\mathbf{h}_{i-1}$ rend l'analyse de ce type de réseaux difficile tel que représenté à la figure 6. Afin, de mieux visualiser les différentes itérations, il faut "dérouler" le réseau dans le temps, tel qu'illustré à la figure 7. Dépendamment de la tâche, la sortie $\mathbf{y}_i$ pourra être utilisée ou non. Les figures 7a, 7b, 7c et 7d présentent le déroulement de réseaux pour différentes tâches.



(a) Annotation (b) Prédiction

(c) Génération (d) Traduction

Figure 7: Réseaux de neurones récurrents déroulés

Finalement, les réseaux récurrents peuvent être empilés afin d'obtenir des représentations de plus en plus abstraites. La figure 8 présente un exemple de réseaux empilés pour une tâche d'annotation.

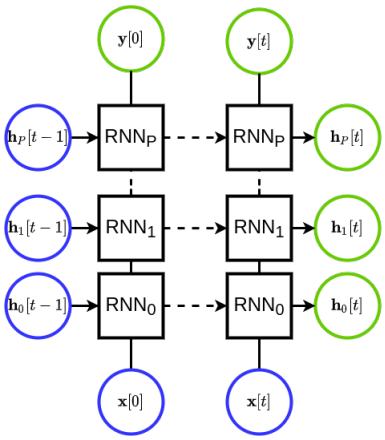


Figure 8: Réseaux de neurones récurrents empilés

15.2 Types de réseaux récurrents

15.2.1 Réseaux récurrents simples (Elman)

Un réseau récurrent prend toujours deux vecteurs en entrées, soit le vecteur à traiter et un vecteur de contexte. Au plus simple, un réseau de neurones récurrent peut être construit à l'aide de couches denses. Pour traiter le $i^{\text{ème}}$ vecteur de la séquence, le réseau prend deux vecteurs en entrée, soit le vecteur d'entrée et le vecteur de contexte. La figure 9 présente l'architecture d'un réseau simple de Elman.

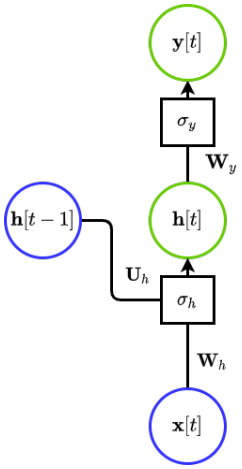


Figure 9: Réseau de neurones récurrent simple (Elman)

Mathématiquement, on peut représenter ce réseau en (2), où $\mathbf{h}[t-1]$ et $\mathbf{x}[t]$ sont le vecteur de contexte de l'itération précédente et le vecteur d'entrée au temps t. Les paramètres du réseau (poids et biais) sont définis par les matrices et vecteurs $\mathbf{W}$, $\mathbf{U}$, et $\mathbf{b}$. Finalement, la sortie du réseau est le contexte de l'itération présente sont définies par les vecteurs $\mathbf{y}[t]$ et $\mathbf{h}[t]$.

$$\begin{aligned} \mathbf{h}[t] &= \sigma_h(\mathbf{W}_h\mathbf{x}[t] + \mathbf{U}_h\mathbf{h}[t-1] + \mathbf{b}_h) \\ \mathbf{y}[t] &= \sigma_y(\mathbf{W}_y\mathbf{h}[t] + \mathbf{b}_y) \end{aligned} \tag{2}$$

Lors de la mise à jour des paramètres, on utilise la méthode de rétropropagation des gradients de la fonction de coût L. Le calcul des gradients doit se faire du temps futur vers le temps passé dans la séquence en utilisant le réseau récurrent déroulé (voir figure 10). Chaque élément de la séquence est similaire à une couche qui aurait été ajoutée. Le calcul du gradient d'un paramètre revient à faire la somme des gradients obtenus par tous les chemins du schéma. L'équation 3 présente un exemple de calcul du gradient à partir des connexions de la figure 10.

$$\frac{\partial L}{\partial \mathbf{W}_y} = \sum \frac{\partial L}{\partial \mathbf{W}_y} = \frac{\partial L}{\partial \mathbf{y}[t]}\frac{\partial \mathbf{y}[t]}{\partial \mathbf{W}_y} + \frac{\partial L}{\partial \mathbf{y}[t+1]}\frac{\partial \mathbf{y}[t+1]}{\partial \mathbf{W}_y} + \frac{\partial L}{\partial \mathbf{y}[t+2]}\frac{\partial \mathbf{y}[t+2]}{\partial \mathbf{W}_y} \tag{3}$$

15.2.2 Disparition et explosion du gradient

Une des limitations des réseaux récurrents simples est la disparition du gradient (*gradient vanishing*). La figure 11 illustre les variables d'un RNN simple pour une séquence de N éléments. La variable d'entrée $\mathbf{x}[t]$ est reliée à la variable de sortie $\mathbf{y}[N-1]$ par le théorème de dérivation des fonctions composées (*chain rule*) (voir (4)). Lorsque la valeur absolue des dérivées partielles est inférieure à un, le produit tend de plus en plus vers zéro. Ceci fait en sorte que l'amplitude du gradient pour des sorties éloignées dans le temps devient insignifiante par rapport à celles des sorties plus rapprochées dans le temps. Dans le contexte d'un RNN, ceci implique une difficulté à apprendre des relations entrées/sorties qui sont distantes, ce qui donne au réseau une mémoire dite à court terme. À l'inverse, lorsque la valeur absolue des dérivées partielles est supérieure à un, le gradient peut augmenter très rapidement

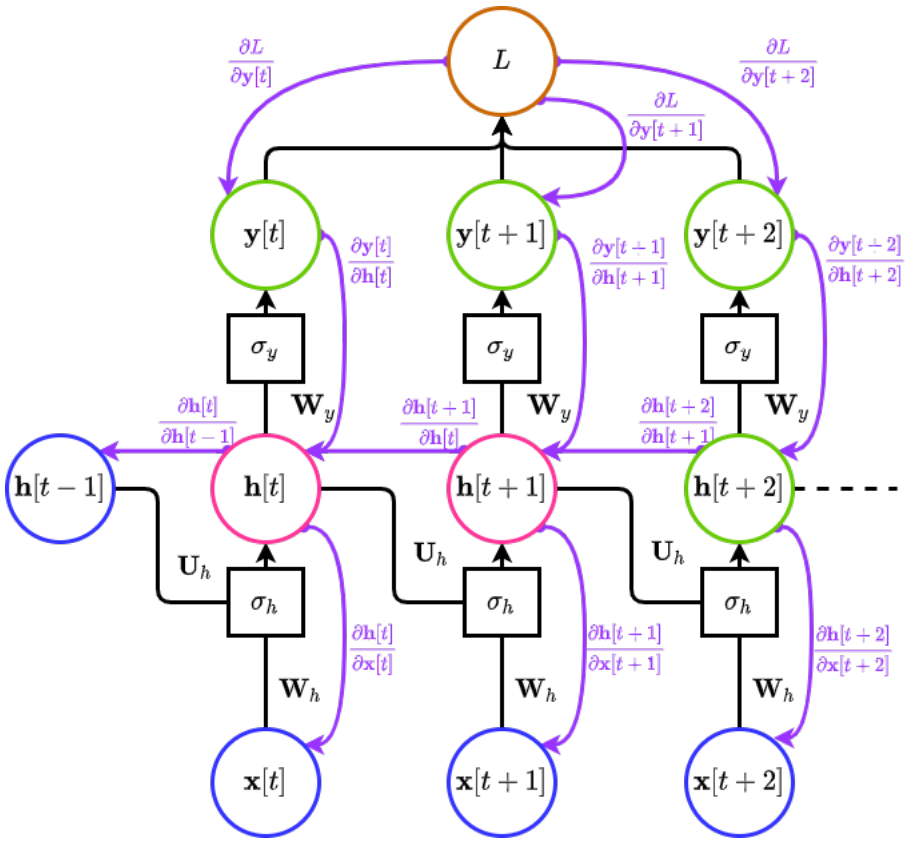


Figure 10: Réseau de neurones récurrents simple pour une séquence de deux éléments

(tendre vers $\pm\infty$) pour de longues séquences (explosion du gradient ou *exploding gradient*). Une solution simple consiste à limiter la valeur positive et négative des gradients (*gradient clipping*).

$$\frac{\partial \mathbf{y}[N\text{-}1]}{\partial \mathbf{x}[t]} = \frac{\partial \mathbf{h}[t]}{\partial \mathbf{x}[t]} \prod_{j=t}^{N-2} \left[\frac{\partial \mathbf{h}[j+1]}{\partial \mathbf{h}[j]} \right] \frac{\partial \mathbf{y}[N\text{-}1]}{\partial \mathbf{h}[N\text{-}1]} \tag{4}$$

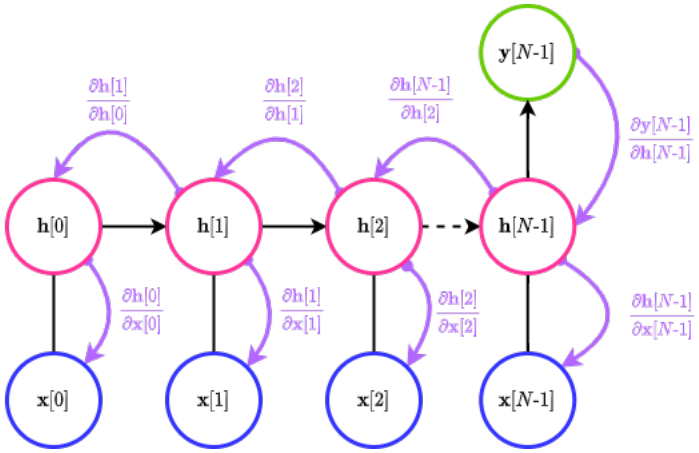


Figure 11: Illustration de la disparition du gradient

15.2.3 Réseaux récurrents à vannes

Afin de répondre au problème de disparition du gradient, les réseaux avec vannes (*gated*) ont été introduits à la fin des années 1990. Le réseau *Long Short Time Memory* (LSTM) est le premier de ce genre, et est illustré à la figure 12. Le réseau possède trois vecteurs en entrée, soit le vecteur de la séquence ($\mathbf{x}[t]$), le vecteur de contexte de l'itération précédente ($\mathbf{h}[t-1]$) et un vecteur correspondant à la cellule de mémoire de l'itération précédente ($\mathbf{c}[t-1]$). Deux vecteurs sont générés par le réseau, soit celui du contexte ($\mathbf{h}[t]$) et de la cellule de mémoire ($\mathbf{c}[t]$). L'idée générale du LSTM est d'utiliser des vannes (*gates*) afin de gérer ce qui est lu, écrit et effacé de la cellule de mémoire. Les sorties des sigmoïdes agissent comme des actionneurs qui contrôlent les vannes (avec le produit élément par élément $\odot$) et régulent le flux d'information. La vanne d'oublie (*forget gate*) permet de réguler ce qui est conservé dans la cellule de l'itération précédente. La vanne d'entrée (*input gate*) permet de réguler ce qui est ajouté à la cellule de mémoire. Finalement, la vanne de sortie (*output gate*) permet de réguler ce qui est lu de la cellule et passé au contexte de l'itération courante. Ces relations sont exprimées plus formellement en (5).

$$
\begin{aligned}
\mathbf{f}[t] &= \text{sig}(\mathbf{U}_f\mathbf{h}[t-1] + \mathbf{W}_f\mathbf{x}[t] + \mathbf{b}_f) \\
\mathbf{i}[t] &= \text{sig}(\mathbf{U}_i\mathbf{h}[t-1] + \mathbf{W}_i\mathbf{x}[t] + \mathbf{b}_i) \\
\mathbf{o}[t] &= \text{sig}(\mathbf{U}_o\mathbf{h}[t-1] + \mathbf{W}_o\mathbf{x}[t] + \mathbf{b}_o) \\
\tilde{\mathbf{c}}[t] &= \tanh(\mathbf{U}_c\mathbf{h}[t-1] + \mathbf{W}_c\mathbf{x}[t] + \mathbf{b}_c) \\
\mathbf{c}[t] &= \mathbf{f}[t] \odot \mathbf{c}[t-1] + \mathbf{i}[t] \odot \tilde{\mathbf{c}}[t] \\
\mathbf{h}[t] &= \mathbf{o}[t] \odot \tanh(\mathbf{c}[t])
\end{aligned}
\tag{5}
$$

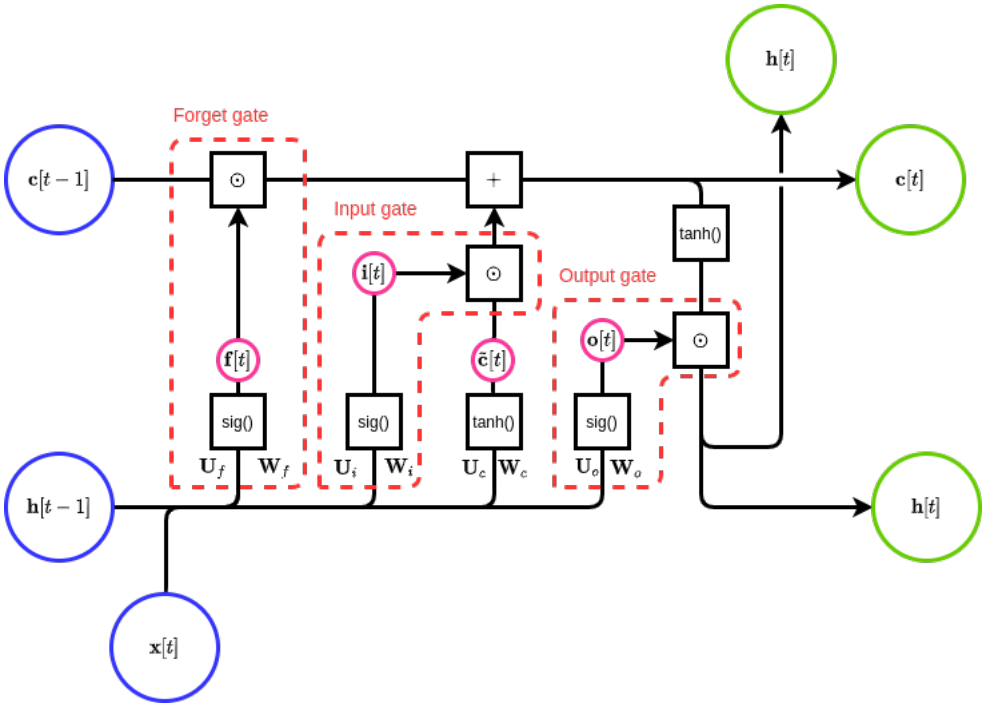


Figure 12: Réseau LSTM simplifié

Un autre type de réseau récurrent à vannes populaire est le *Gated Recurrent Unit* (GRU). Ce réseau a été introduit comme alternative au LSTM avec moins de paramètres. Ce réseau a seulement deux vecteurs en entrée, soit le contexte de l'itération précédente $\mathbf{h}[t-1]$ et le vecteur de la séquence $\mathbf{x}[t]$. Il possède deux vannes (*gates*) pour contrôler ce qui est ajouté et retiré au vecteur de contexte $\mathbf{h}[t]$. La vanne de réinitialisation (*reset gate*) permet d'extraire l'information pertinente du vecteur de contexte précédent afin de calculer un vecteur de contexte candidat $\tilde{\mathbf{h}}[t]$. La vanne de mise à jour (*update gate*) permet d'effacer de l'information du contexte précédent et de la remplacer

par celle du contexte candidat. Ces relations sont exprimées plus formellement en (6).

$$\begin{aligned}
\mathbf{r}[t] &= \text{sig}(\mathbf{U}_r\mathbf{h}[t-1] + \mathbf{W}_r\mathbf{x}[t] + \mathbf{b}_r) \\
\mathbf{z}[t] &= \text{sig}(\mathbf{U}_z\mathbf{h}[t-1] + \mathbf{W}_z\mathbf{x}[t] + \mathbf{b}_z) \\
\tilde{\mathbf{h}}[t] &= \tanh(\mathbf{U}_h(\mathbf{r}[t] \odot \mathbf{h}[t-1]) + \mathbf{W}_h\mathbf{x}[t] + \mathbf{b}_h) \\
\mathbf{h}[t] &= (\mathbf{1} - \mathbf{z}[t]) \odot \mathbf{h}[t-1] + \mathbf{z}[t] \odot \tilde{\mathbf{h}}[t]
\end{aligned} \tag{6}$$

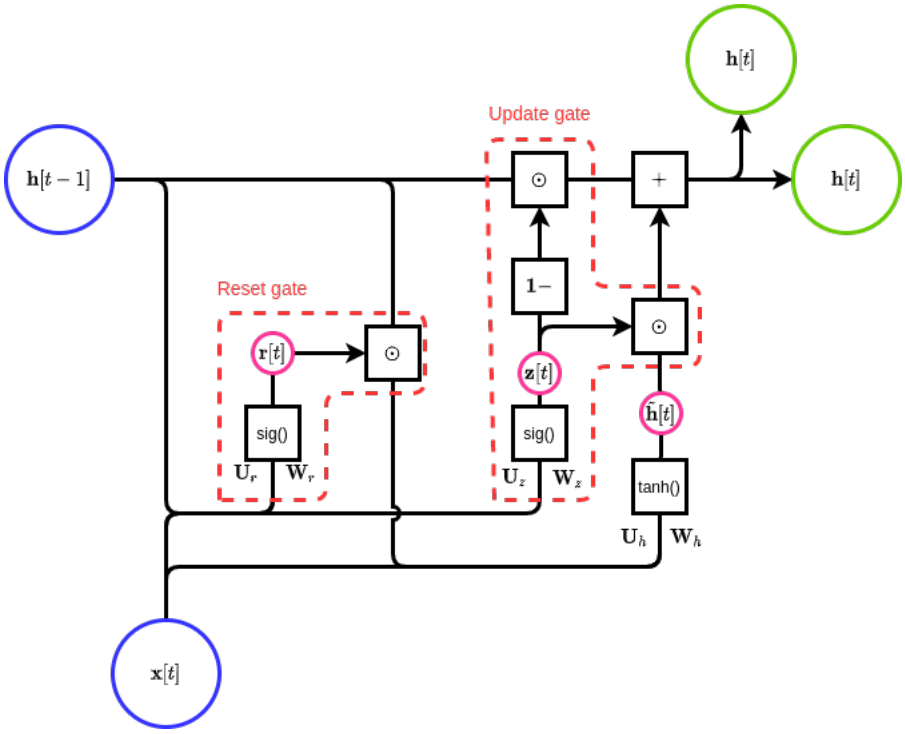


Figure 13: Réseau GRU simplifié

15.3 Réseaux récurrents bidirectionnels

Dans certaines applications, il est possible d'avoir accès à la séquence d'entrée complète lors du déploiement. En d'autres mots, dans ce genre de scénario, le réseau récurrent n'a pas à adopter un comportement causal. Voici quelques exemples:

- Traduction de séquences
  - Traduction de langues
  - Voix vers texte
  - Écriture cursive vers texte
  - Résumé de texte
- Annotation de séquences
  - Analyse de sentiments
  - Analyse de types de mots

Dans ce type d'applications, les observations du passé et celles du futur peuvent être utilisées afin d'avoir une meilleure idée du contexte. Pour obtenir l'information des éléments futurs en plus des éléments du passé, il suffit d'utiliser une autre couche RNN en parallèle qui itère sur la séquence de la fin vers le début, tel qu'illustré à la figure 14. La sortie d'une couche bidirectionnelle est la concaténation de la sortie des deux couches de RNN (double d'unités cachées).

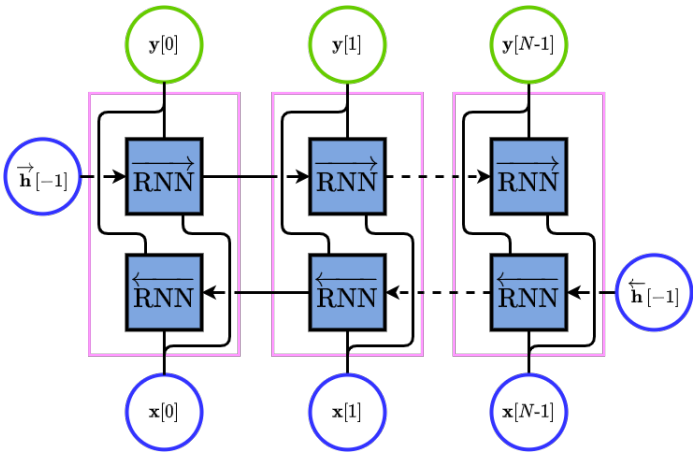


Figure 14: Réseau récurrent bidirectionnel

15.4 Remplissage

L'un des avantages des réseaux récurrents réside dans leur capacité à traiter des séquences de tailles variables. Toutefois, pour tirer avantage des bibliothèques logicielles optimisées pour les calculs vectorisés en lots, les échantillons doivent être de mêmes dimensions. Pour y parvenir, lorsqu'on utilise des séquences de symboles, certains symboles spéciaux sont utilisés pour baliser les séquences:

- Le symbole de remplissage (*Padding*) PAD est utilisé pour combler les espaces vides dans une séquence. L'objectif est de permettre au modèle d'apprendre à ignorer ce symbole.
- Le symbole de fin (*End Of Sequence*) EOS est utilisé pour indiquer la fin d'une séquence.
- Le symbole de début (*Start Of Sequence*) SOS est utilisé pour indiquer le début d'une séquence.

15.5 Module d'attention

Dans une tâche de traduction, on désire générer une séquence de sortie à partir d'une séquence d'entrée. Ces séquences peuvent être de tailles différentes. L'architecture typique utilisée est un encodeur/décodeur (voir figure 15). La portion encodeur permet de représenter le contexte de la séquence entière à l'aide d'un seul vecteur qui est passé à la portion décodeur. Les couches de l'encodeur et du décodeur sont généralement différentes (i.e. elles ne partagent pas les mêmes poids), mais elles ont le même nombre de neurones cachés. Le décodeur est un modèle génératif qui prend un symbole de début (SOS) afin de commencer le décodage. Lors de l'entraînement, le décodage se poursuit jusqu'à la longueur maximum de séquence de sortie (M). Lors du déploiement, on désire parfois arrêter le décodage lorsque le modèle renvoie un symbole de fin (EOS).

Le vecteur de contexte entre l'encodeur et le décodeur peut être vu comme un goulot d'étranglement (*bottleneck*), qui contient l'information de la séquence d'entrée encodée. Pour de longues séquences en entrée cela peut être problématique, car le nombre d'unités cachées doit être augmenté et ceci augmente le temps d'entraînement. Pour palier à ce problème, il est possible d'ajouter un module d'attention/alignement au modèle (voir Figure 16). L'idée générale derrière le module d'attention est d'utiliser un mécanisme pour déterminer quelles sorties (vecteurs de context) de l'encodeur $\mathbf{v}[0:N\text{-}1]$ sont les plus pertinentes à l'élément décodé $\mathbf{q}[t]$ pour prédire la sortie $\mathbf{y}[t]$.

Il existe plusieurs façons de faire un module d'attention, mais celles-ci suivent généralement les mêmes étapes. Le vecteur $\mathbf{q}[t] \in \mathbb{R}^L$ provenant de l'élément décodé (contexte du décodeur) est appelé "requête" (*Query*). Les vecteurs de contexte de l'encodeur $\mathbf{v}[0:N\text{-}1] \in \mathbb{R}^{L \times N}$ sont les "valeurs" (*Values*). La première étape est d'utiliser une métrique pour donner un pointage (*scores*) $\tilde{\mathbf{w}} \in \mathbb{R}^N$ de similarité entre la requête et chacun des vecteurs de valeur. Plusieurs types de fonctions existent pour inférer la similarité (e.g. produit scalaire, réseau dense, similarité cosinus). Ensuite il faut normaliser le vecteur de pointage afin d'obtenir une distribution représentant les poids d'attention $\mathbf{w} = \text{softmax}(\tilde{\mathbf{w}}) \in \mathbb{R}^N_{>0}, \sum w_i = 1$. Ces poids représentent donc le niveau de similitude normalisé entre le vecteur

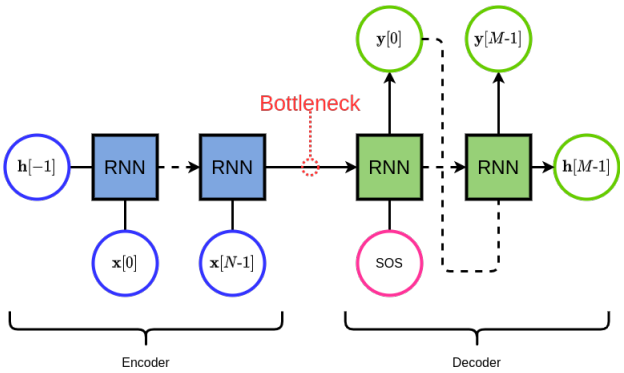


Figure 15: Goulot d'étranglement d'une architecture de traduction

de requête et chacun des vecteurs de valeurs, ou bien le "niveau d'attention" porté à chaque étape d'encodage pour une valeur du décodeur. Finalement, il faut pondérer les vecteurs de valeurs aux poids d'attention pour obtenir le vecteur d'attention $\mathbf{a} = \mathbf{v}[0 : N\text{-}1]\mathbf{w} \in \mathbb{R}^L$. Dans une architecture d'attention de Luong, ce vecteur d'attention est concaténé au vecteur de requête puis passé dans un réseau dense (*Feed forward*) vers le vecteur de sortie $\mathbf{y}[t]$. La figure 17 présente un exemple numérique d'un module d'attention avec trois vecteurs de valeur ayant $L = 2$ unités cachées. La fonction de similitude utilisée est le produit scalaire. Ce genre de module permet d'enlever les limitations liées aux longues séquences, car le décodeur peut utiliser l'information du contexte de n'importe quel endroit de la séquence d'entrée. De plus, ce module offre un nouveau chemin pour la propagation du gradient de façon analogue à des connexions résiduelles, pouvant aider durant l'entraînement.

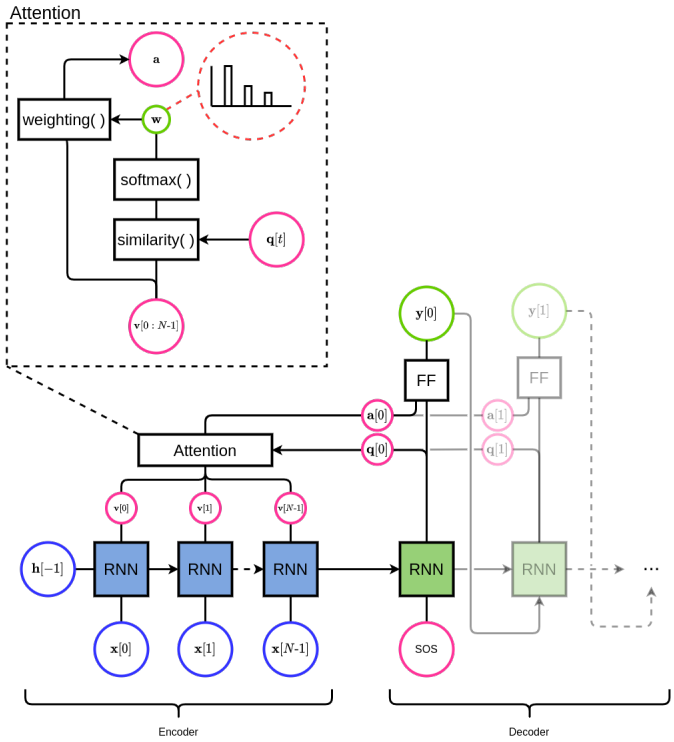


Figure 16: Attention de Luong

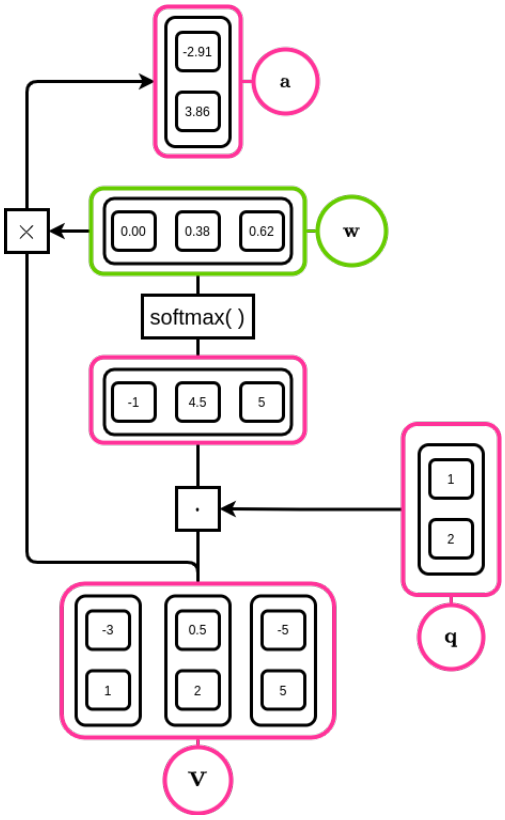


Figure 17: Exemple numérique d'un module d'attention

15.6 Métrique pour séquences

La distance d'édition (ou distance de Lavenshtein) est une mesure permettant de déterminer la **plus petite** distance (ou dissimilitude) entre deux chaînes de symboles. Elle permet de déterminer le nombre d'ajouts et de substitutions de symboles entre deux séquences de tailles arbitraires. Par exemple, avec les séquences $\{1, 2, 3, 4\}$ et $\{1, 2, 5, 3, 6\}$, il y a ajout du symbole 5 et la substitution du symbole 4 vers 6, donnant une distance de 2. De la même façon, les mots *Allo* et *Apollo2* aurait une distance de Lavenshtein de 3, en raison de l'ajout des caractères *po2*. Afin de calculer cette métrique, on peut utiliser une fonction récursive en (7). L'opérateur $|\dots|$ correspond à la cardinalité et renvoie le nombre d'éléments de la séquence. Sans utiliser de récursion, on peut utiliser l'algorithme équivalent présenté à la figure 18.

$$\text{dist}(x, y) = \begin{cases} |y| & \text{Si } |x| = 0 \\ |x| & \text{Si } |y| = 0 \\ \text{dist}(x[1:], y[1:]) & \text{Si } x[0] = y[0] \\ 1 + \min \begin{cases} \text{dist}(x[1:], y) \\ \text{dist}(x, y[1:]) \\ \text{dist}(x[1:], y[1:]) \end{cases} & \text{Sinon} \end{cases} \tag{7}$$

15.7 Extraction de symboles vers jetons (*tokenization*)

Un modèle ne peut pas directement travailler avec des symboles (e.g. mots, notes de musique etc.). Pour travailler avec ces symboles, on doit utiliser une représentation sous la forme de jetons (*tokens*). Le but est d'assigner un jeton unique à chaque symbole (ou groupe de symboles) pour la tâche. Généralement, le jeton correspond à un indice

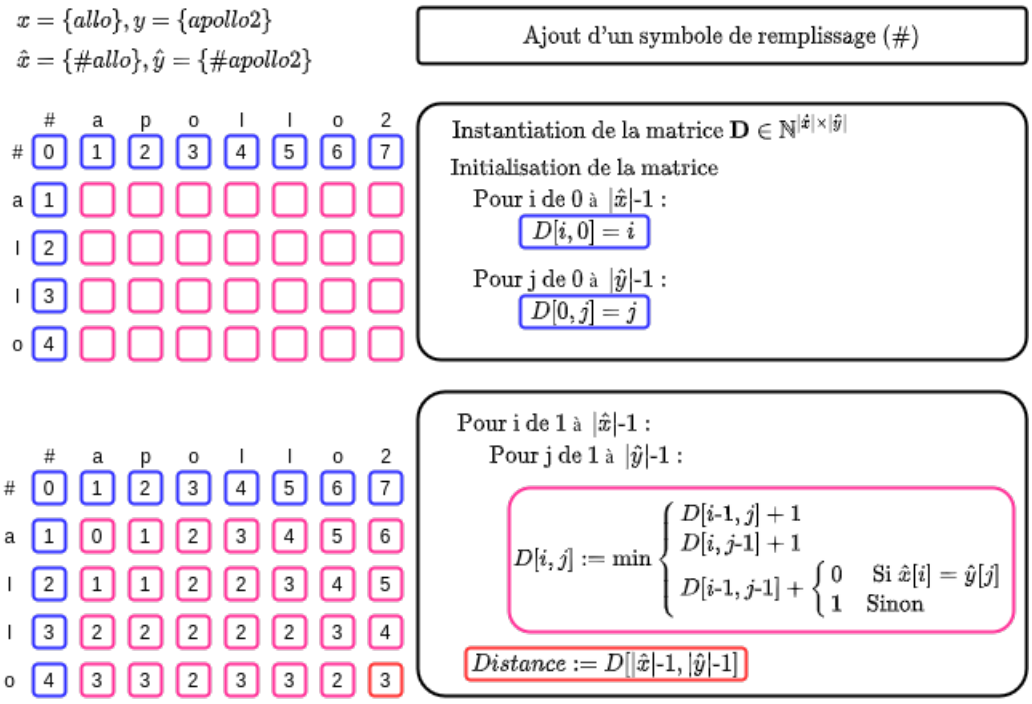


Figure 18: Algorithm de distance d'édition avec matrice

de dictionnaire, allant de 0 à la taille du dictionnaire moins un. Par exemple, si l'ensemble de données contient uniquement l'échantillon {a,p,o,l,l,o}, on pourrait construire le dictionnaire de jetons suivant:

$$jetons = \begin{cases} a \mapsto & 0 \\ p \mapsto & 1 \\ o \mapsto & 2 \\ l \mapsto & 3 \end{cases}$$

La séquence {a,p,p,o,l,o} sous la forme de jetons serait donc {0, 1, 2, 3, 3, 2}. Pour passer cette séquence à un modèle, il suffit de transformer les jetons en représentation 1 parmi N (*one-hot*), en ajoutant un 1 à l'indice du jeton dans un vecteur de taille du dictionnaire. Le processus de *tokenization* pour l'exemple se résume en (8).

$$\{a, p, o, l, l, o\} \xrightarrow{jetons} \{0, 1, 2, 3, 3, 2\} \xrightarrow{one-hot} \begin{Bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{Bmatrix} \tag{8}$$

15.8 Couche de vectorisation (*embedding layer*)

On représente généralement un symbole sous la forme d'un vecteur 1 parmi N (*one-hot*). Cette représentation parcimonieuse peut être d'une dimension assez élevée selon la taille du dictionnaire. Une couche de vectorisation est généralement nécessaire afin d'encoder le vecteur 1 parmi N en un vecteur dense (généralement de plus petite dimension) riche en information. Dans des implémentations comme *nn.embedding* de Pytorch, la couche prend directement un jeton (*token*) et fait la représentation 1 parmi N à l'interne avant de passer le tout dans une couche linéaire. La figure 19 illustre un exemple de couche de vectorisation.

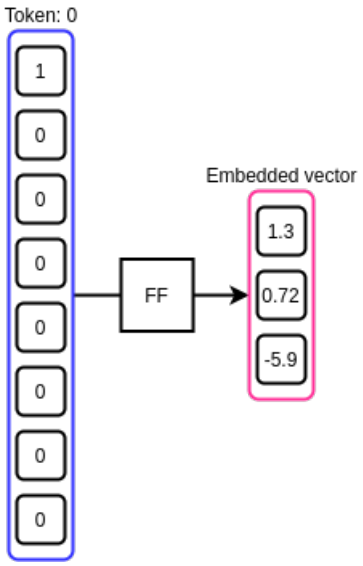


Figure 19: Exemple de couche de vectorisation

15.9 Stratégies

Lorsqu'on utilise des modèles génératifs (où la sortie du réseau est directement copiée à l'entrée de l'itération suivante), il est possible d'utiliser la stratégie d'entraînement *Teacher forcing* afin d'aider le modèle à converger. Lors de l'entraînement, on passe au modèle la séquence cible à titre de référence. Après chaque itération, on échantillonne une variable aléatoire uniforme $\sim U[0,1]$, et si le résultat est sous un certain seul prédéfini (T_{teach}), on copie la réponse attendue de la cible pour la prochaine itération. Dans le cas contraire, on copie le symbole prédit par le modèle à la prochaine itération comme à l'habitude.