Réseaux de neurones récursifs

Introduction à l'apprentissage automatique
Master Sciences Cognitives
Aix Marseille Université

Alexis Nasr

Motivations

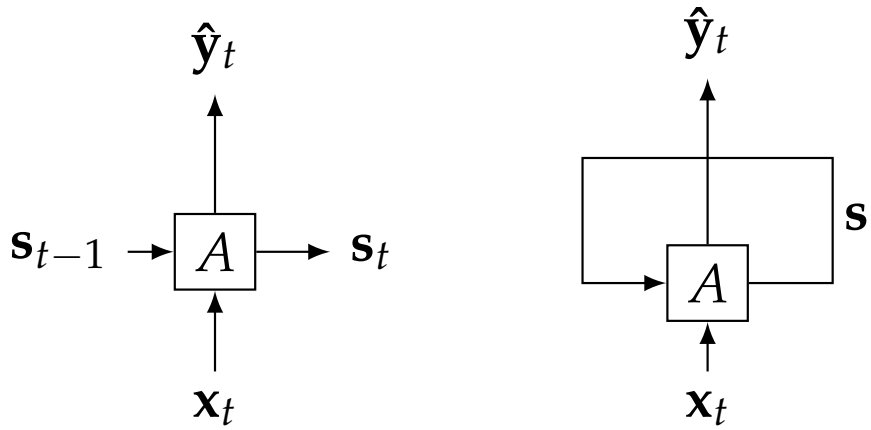
- Les réseaux convolutionnels peuvent déceler des motifs **locaux** dans les données.
- Il ne sont pas adaptés pour déceler des relations pouvant relier des événements arbitrairement distants.
- Exemple : on aimerait concevoir un réseau dont le but est de prédire quel événement va avoir lieu à un moment donné dans un film.
- Concevoir un réseau n'est pas évident car on ne sait pas à quels moments dans le passé ont lieu des événements qui aideront à prédire un événement au temps présent.

Plan

Idée générale

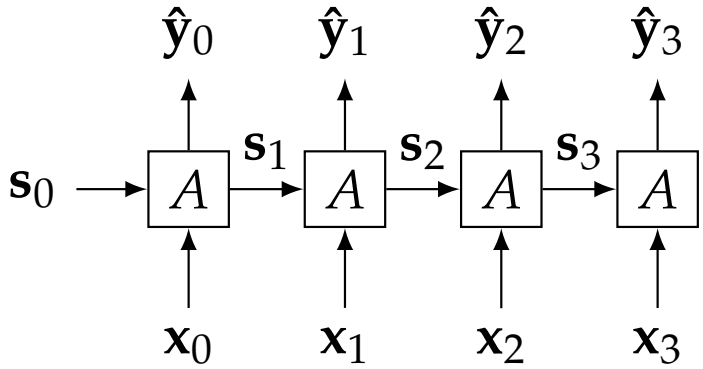
- Les réseaux récurrents sont conçus pour traiter des **séquences** d'entrée de la forme : $x_1 \ldots x_n$
- Afin d'autoriser la prise en compte d'événements arbitrairement distants, on introduit la notion d'**état** du réseau au temps t, noté $\mathbf{s}_t$ (s pour state en anglais).
- Ce qu'on appelle *temps* ici est en fait une position dans une séquence, tel que la position d'une lettre dans une séquence de lettres.
- $\mathbf{s}_i$ véhicule de l'information utile pour des prédictions au temps $t > i$, il se présente sous la forme d'un vecteur, appelé **vecteur d'état**.

Idée générale



- Au temps t, le réseau a accès à l'entrée au temps t : $\mathbf{x}_t$, ainsi qu'à l'état du réseau au temps $t-1$: $\mathbf{s}_{t-1}$.
- L'état du réseau constitue en même temps une sortie et une entrée du réseau : il est produit par ce dernier (sortie) et pris en compte (entrée) pour des prédictions futures.
- C'est cette caractéristique qui lui confère le nom de **réseau récurrent**, noté RNN.
- Le caractère récurrent est souvent représenté par une boucle.

Réseau “déplié”



- Si on observe le fonctionnement d’un réseau dans le temps, on obtient une vision “dépliée” de ce réseau.
- Dans la version dépliée, chaque occurrence du réseau A correspond à un instant, on voit ainsi le déroulement du réseau dans le temps.
- A chaque instant t l’entrée $\mathbf{x}_t$ est lue, ainsi que le vecteur d’état $\mathbf{s}_{t-1}$, la sortie $\hat{\mathbf{y}}_t$ est calculée, ainsi que le vecteur d’état $\mathbf{s}_t$.
- Comme dans un réseau convolutionnel, le réseau A est **dupliqué** plusieurs fois, il s’agit du même réseau, et donc des mêmes paramètres.

Notations

- L'entrée du réseau est constituée d'une séquence $\mathbf{x}_1 \dots \mathbf{x}_n$ de vecteurs de dimension d_{in}.
- La sortie du réseau au temps t est un vecteur $\hat{\mathbf{y}}_t$ de dimension d_{out} :

$$\hat{\mathbf{y}}_t = rnn(\mathbf{x}_1 \dots \mathbf{x}_n)$$

- Pour chaque sous-séquence $\mathbf{x}_1 \dots \mathbf{x}_t$, le réseau produit une sortie $\hat{\mathbf{y}}_t$.
- On note rnn^* la fonction qui produit la séquence $\hat{\mathbf{y}}_1 \dots \hat{\mathbf{y}}_t$

$$\hat{\mathbf{y}}_1 \dots \hat{\mathbf{y}}_t = rnn^*(\mathbf{x}_1 \dots \mathbf{x}_t)$$

Modèle abstrait

- De manière abstraite, un RNN peut être défini à l'aide de deux fonctions R et O.
- R prend en entrée un état $\mathbf{s}_{t-1}$ et une entrée $\mathbf{x}_t$ pour produire un nouvel état $\mathbf{s}_t$:

$$\mathbf{s}_t = R(\mathbf{s}_{i-1}, \mathbf{x_t})$$

- O produit une sortie $\hat{\mathbf{y}}_t$ à partir de $\mathbf{s}_t$:

$$\hat{\mathbf{y}}_t = O(\mathbf{s}_t)$$

- Pour amorcer la pompe, on a besoin d'un état $\mathbf{s}_0$ que l'on considèrera pour l'instant comme un vecteur nul (toutes ses composantes valent 0).
- Le modèle présenté ci-dessus, à l'aide des deux fonctions R et O est un modèle abstrait.
- Pour définir un réseau concret, il faut spécifier la forme exacte de R et de O.

Modèle abstrait

- La définition de R confère au réseau son caractère récursif : $\mathbf{s}_t$ est produit à partir de $\mathbf{s}_{t-1}$.
- Illustration pour $t = 4$

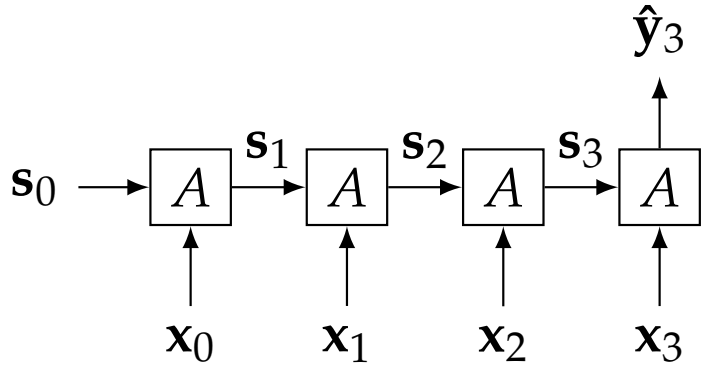
$$\begin{aligned}\mathbf{s}_4 &= R(\mathbf{s}_3, \mathbf{x}_4) \\ &= R(R(\mathbf{s}_2, \mathbf{x}_3), \mathbf{x}_4) \\ &= R(R(R(\mathbf{s}_1, \mathbf{x}_2), \mathbf{x}_3), \mathbf{x}_4) \\ &= R(R(R(R(\mathbf{s}_0, \mathbf{x}_1), \mathbf{x}_2), \mathbf{x}_3), \mathbf{x}_4)\end{aligned}$$

- La valeur de $\mathbf{s}_4$ est calculée à partir des entrées $\mathbf{x}_1 \ldots \mathbf{x}_4$.

Apprentissage des paramètres des fonctions R et 0

- C'est lors de l'étape d'apprentissage que les paramètres des fonctions R et O sont déterminés.
- L'apprentissage des paramètres est réalisé sur le réseau sous forme déplié.
- On procède comme d'habitude : on détermine les valeurs des paramètres de R et O qui permettent de minimiser une fonction d'erreur sur des données d'apprentissage.
- C'est par le biais de la minimisation de la fonction d'erreur que le réseau apprend ce dont il faut se souvenir du passé, en d'autres termes, ce qui doit être représenté dans le vecteur d'état.

Encodeurs

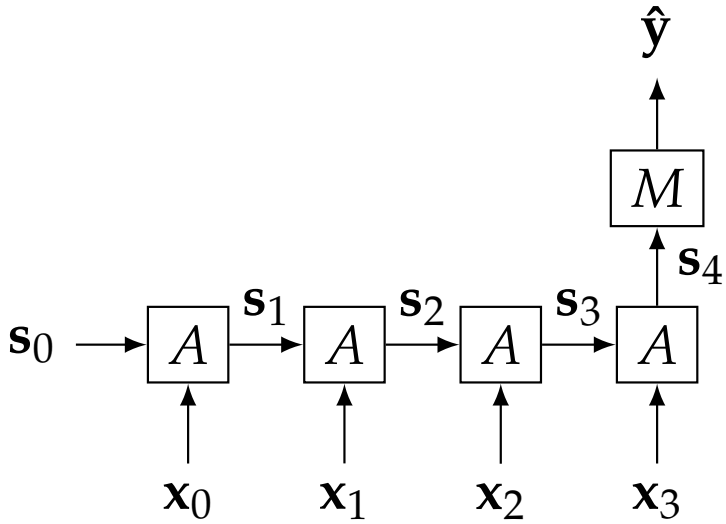


- Il est possible de faire en sorte que le réseau ne fasse qu'une prédiction, à l'issue de la lecture de toute la séquence.
- Lors de l'apprentissage, ce n'est qu'à l'issue de la prédiction finale que l'on calcule la fonction de perte et que la mise à jour des paramètres est réalisée.

Exemples

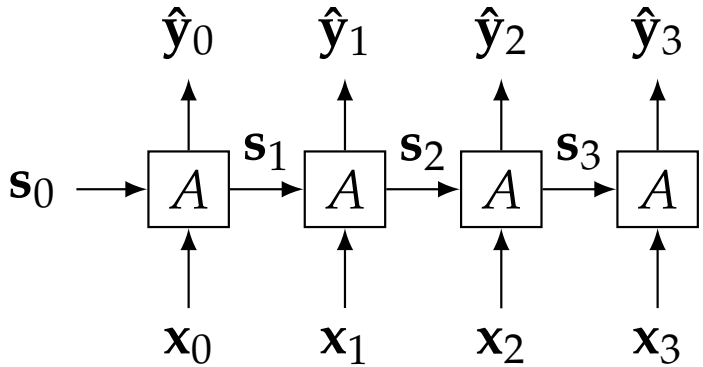
- **Prédiction de la langue**
  Contrairement au CNN, on a maintenant des séquences de longueurs quelconques et c'est à l'issue de la lecture du dernier caractère qu'une prédiction est réalisée.
- **Prédiction du jugement véhiculé par une phrase**
  On veut concevoir un réseau qui lit une phrase et prédit, à la fin de la lecture si la phrase véhicule un jugement positif, négatif ou neutre :
  - *J'ai vraiment détesté ce cours* $\rightarrow$ NEGATIF
  - *rarement un cours m'a autant intéressé* $\rightarrow$ POSITIF
  - *Le cours a lieu toutes les semaines en salle 202* $\rightarrow$ NEUTRE

Encodeurs



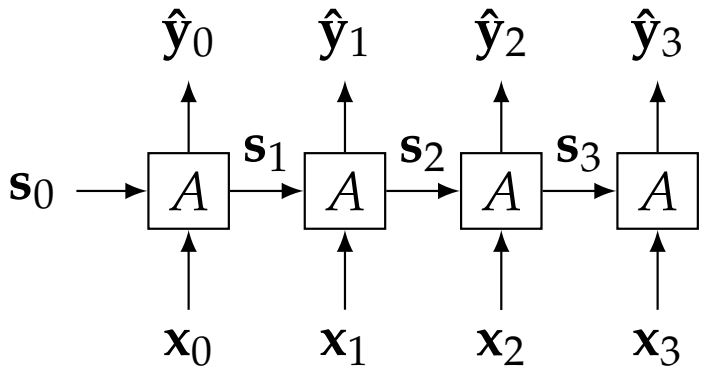
- Le vecteur d'état $\mathbf{s}_n$ produit à l'issue du traitement d'une séquence $\mathbf{x}_1 \ldots \mathbf{x}_n$ est souvent utilisé comme entrée d'une autre tâche de prédiction.
- On peut imaginer par exemple que $\mathbf{s}_n$ constitue l'entrée d'un autre réseau M qui peut être, par exemple, MLP ou un CNN
- $\mathbf{s}_n$ correspond à une **représentation** de la séquence $\mathbf{x}_1 \ldots \mathbf{x}_n$, on dit aussi son **encodage**.
- Le RNN est appelé dans ce cas là un **encodeur**.
- La prédiction est réalisée par M et c'est donc à la sortie de M que la fonction de perte est calculée et que la rétro-propagation est réalisée, pour mettre à jour les paramètres du RNN.

Transducteurs



- Un transducteur est un programme qui lit une séquence et en produit une autre.
- Exemple : traduction, le réseau lit une séquence dans la langue source et produit une séquence dans la langue cible.
- Une transduction peut être réalisée par un RNN qui produit une sortie $\hat{\mathbf{y}}_t$ pour toute entrée $\mathbf{x}_t$.

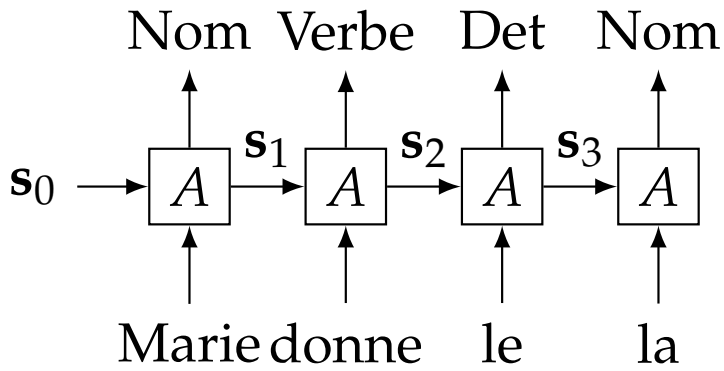
Transducteurs



- On calcule une perte pour chaque sortie $\hat{\mathbf{y}}_t$ en la comparant avec la sortie correcte $\mathbf{y}_t$: $L(\hat{\mathbf{y}}_t, \mathbf{y}_t)$
- La perte pour toute la séquence peut alors être calculée à partir des pertes locales, en faisant, par exemple, la somme :

$$L(\hat{\mathbf{y}}_1 \dots \hat{\mathbf{y}}_n, \mathbf{y}_1 \dots \mathbf{y}_n) = \sum_{t=1}^{n} L(\hat{\mathbf{y}}_t, \mathbf{y}_t)$$

Exemple



- On veut attribuer une partie de discours (catégorie grammaticale), tel que : *Verbe*, *Nom*, *Adj* aux différents mots d'un texte.
- La prédiction de la partie de discours d'un mot dépend du mot (*manger* est un verbe),
- mais aussi du contexte du mot (le mot *la* dans notre exemple est un nom car il est précédé d'un déterminant).
- C'est le contexte du mot qui est représenté par le vecteur d'état.

RNN simple ou machine de Elman

- Il s'agit d'un des modèles récurrents les plus simples.
- Il est défini de la façon suivante :

$$\begin{aligned}
\mathbf{s}_t &= R_{SRNN}(\mathbf{x}_t, \mathbf{s}_{t-1}) \\
&= g(W^s \mathbf{s}_{t-1} + W^x \mathbf{x}_t + \mathbf{b}) \\
\hat{\mathbf{y}}_t &= O_{SRNN}(\mathbf{s}_t) \\
&= \mathbf{s}_t
\end{aligned}$$

- où g est une fonction d'activation non linéaire.
- Comme on peut le voir :
  - l'état courant $\mathbf{s}_t$ est calculé à partir de l'état précédent $\mathbf{s}_{t-1}$ et de l'entrée courante $\mathbf{x}_t$ en multipliant $\mathbf{s}_{t-1}$ par la matrice W^s et $\mathbf{x}_t$ par la matrice W^x puis en faisant la somme des deux vecteurs produits et d'un vecteur de biais $\mathbf{b}$;
  - La sortie au temps t n'est autre que l'état $\mathbf{s}_t$.

Disparition du gradient

- Le modèle SRNN est confronté au problème de la disparition du gradient (*vanishing gradient*) :
- lors de la rétro-propagation de l'erreur, cette dernière **diminue** rapidement (elle est multipliée successivement par des paramètres dont la valeur est généralement comprise entre 0 et 1).
- Cette caractéristique empêche la bonne prise en compte de dépendances entre observations éloignées dans le temps (le réseau *oublie* trop vite).
- Pour pallier ce problème, on introduit un système de **filtre** (*gate* en anglais).

Filtres

- Un filtre **f** se présente comme un vecteur composé de 0 et de 1.
- Il est combiné à un autre vecteur **x** à l'aide d'une opération appelée **produit de Hadamard** : $\mathbf{f} \circ \mathbf{x}$.
- Cette opération consiste simplement à effectuer le produit terme à terme de deux vecteurs (la composante i de **f** est multipliée par la composante i de **x**) :

$$\begin{bmatrix} 4 \\ 5 \\ 7 \\ 9 \end{bmatrix} \circ \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 4 \\ 0 \\ 0 \\ 9 \end{bmatrix}$$

- Dans cet exemple, le filtre a permis "d'oublier" les composantes 2 et 3 du vecteur **x**.

Filtres

- Un filtre permet donc de décider ce que l'on garde et ce que l'on oublie dans un état $\mathbf{s}_i$.
- Les filtres offrent un moyen simple de combiner au sein d'un vecteur d'état $\mathbf{s}_i$ le contenu de l'entrée $\mathbf{x}_i$ et du vecteur d'état $\mathbf{s}_{i-1}$ de la manière suivante :

$$\mathbf{s}_i = \mathbf{f} \circ \mathbf{x}_i + (1 - \mathbf{f}) \circ \mathbf{s}_{i-1}$$

- On garde les composantes de l'entrée courante $\mathbf{x}_i$ qui correspondent à des 1 dans le filtre et on oublie les autres.
- On garde les composantes de l'état précédent $\mathbf{s}_{i-1}$ qui correspondent à des 0 dans le filtre.

$$\begin{bmatrix}0\\1\\0\\0\\0\\1\end{bmatrix} \circ \begin{bmatrix}10\\11\\12\\13\\14\\15\end{bmatrix} + \begin{bmatrix}1\\0\\1\\1\\1\\0\end{bmatrix} \circ \begin{bmatrix}8\\9\\3\\7\\5\\8\end{bmatrix} = \begin{bmatrix}8\\11\\3\\7\\5\\15\end{bmatrix}$$

Filtre dérivable

- Les filtres sont appris lors de l'apprentissage : le réseau **apprend** l'information qu'il doit retenir de l'état précédent.
- Problème technique : l'algorithme de rétro-propagation impose que toutes les fonctions qui participent à la fonction de perte soient dérivables, ce qui n'est pas le cas des filtres dont les composantes ne peuvent prendre qu'une des deux valeurs 0 ou 1.
- Une solution consiste à approximer le comportement des filtres binaires à l'aide d'une fonction dérivable pour donner naissance à des **filtres dérivables**.

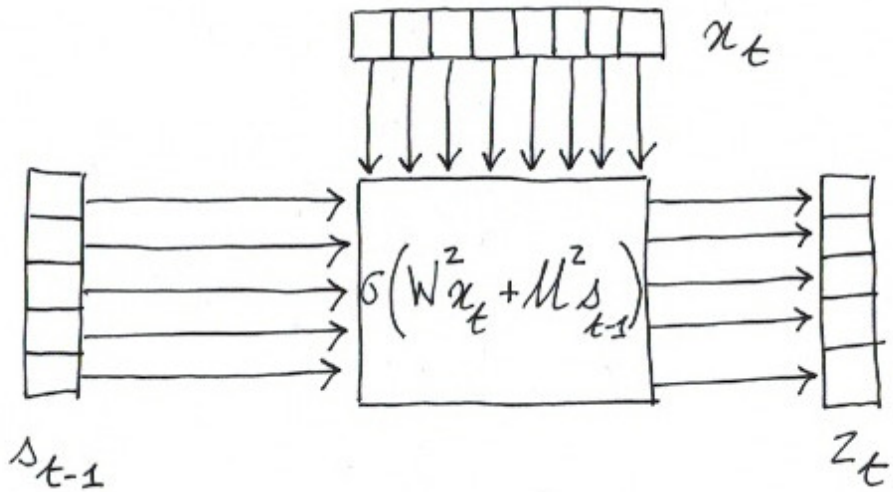
Filtre dérivable

- On autorise les composantes du filtre à prendre pour valeur n'importe quel réel, ce qui constitue un filtre $\mathbf{f}'$ à valeurs réelles.
- Ce filtre est ensuite transformé à l'aide de la fonction sigmoïde (σ), ce qui va avoir pour effet de ramener toutes les valeurs à l'intervalle $[0, 1]$ et "repousser" la majorité des valeurs qui composent le vecteur vers 0 ou 1
- Le filtre $\sigma(\mathbf{f}')$ est par conséquent composé de valeurs proches de 0 et de 1.
- Lorsqu'on réalise le produit $\sigma(\mathbf{f}') \circ \mathbf{x}$, les composantes de $\mathbf{x}$ correspondant à des valeurs proches de 1 dans $\sigma(\mathbf{f}')$ sont conservées alors que les autres sont oubliées.
- Ce mécanisme de filtre dérivable est utilisé dans deux types de RNN, les GRU et les LSTM.
- A chaque étape, de tels réseaux décident, grâce à des filtres dérivables, quelles informations de l'entrée vont être prises en compte dans l'état courant et quelles informations de l'état précédent vont être oubliées.

Les Gated Recurrent Units (GRU)

- Les GRU mettent en œuvre deux filtres :
  - Le filtre *update gate*, noté $\mathbf{z_t}$, qui contrôle quelles informations contenues dans $\mathbf{s}_{t-1}$ vont être transmises à $\mathbf{s}_t$.
  - Le filtre *reset gate*, noté $\mathbf{r_t}$, qui contrôle quelles informations contenues dans $\mathbf{s}_{t-1}$ vont être combinées à $\mathbf{x}_t$.

Calcul du filtre $\mathbf{z_t}$



- Le filtre $\mathbf{z_t}$ est calculé à partir de :
  - l'entrée courante : $\mathbf{x}_t$
  - l'état précédent : $\mathbf{s}_{t-1}$
  - deux matrices de paramètres X^z et U^z

$$\mathbf{z_t} = \sigma(W^z \mathbf{x}_t + U^z \mathbf{s}_{t-1})$$

- Lors de la prédiction, les matrices W^z et U^z sont constantes (elles ont été calculées lors de la phase d'apprentissage) mais pour tout couple $(\mathbf{x}_t, \mathbf{s}_{t-1})$, une nouvelle valeur du filtre est calculée.

Calcul du filtre $\mathbf{r_t}$

- Le calcul de $\mathbf{r_t}$ suit le même principe que celui de $\mathbf{z_t}$, avec les matrices X^r et U^r :

$$\mathbf{r_t} = \sigma(W^r \mathbf{x}_t + U^r \mathbf{s}_{t-1})$$

Calcul de $\mathbf{s}_t$

- L'équation générale de calcul de $\mathbf{s}_t$ est :

$$\mathbf{s}_t = R(\mathbf{s}_{t-1}, \mathbf{x}_t)$$

- Dans le cas des GRU, cette équation prend la forme suivante :

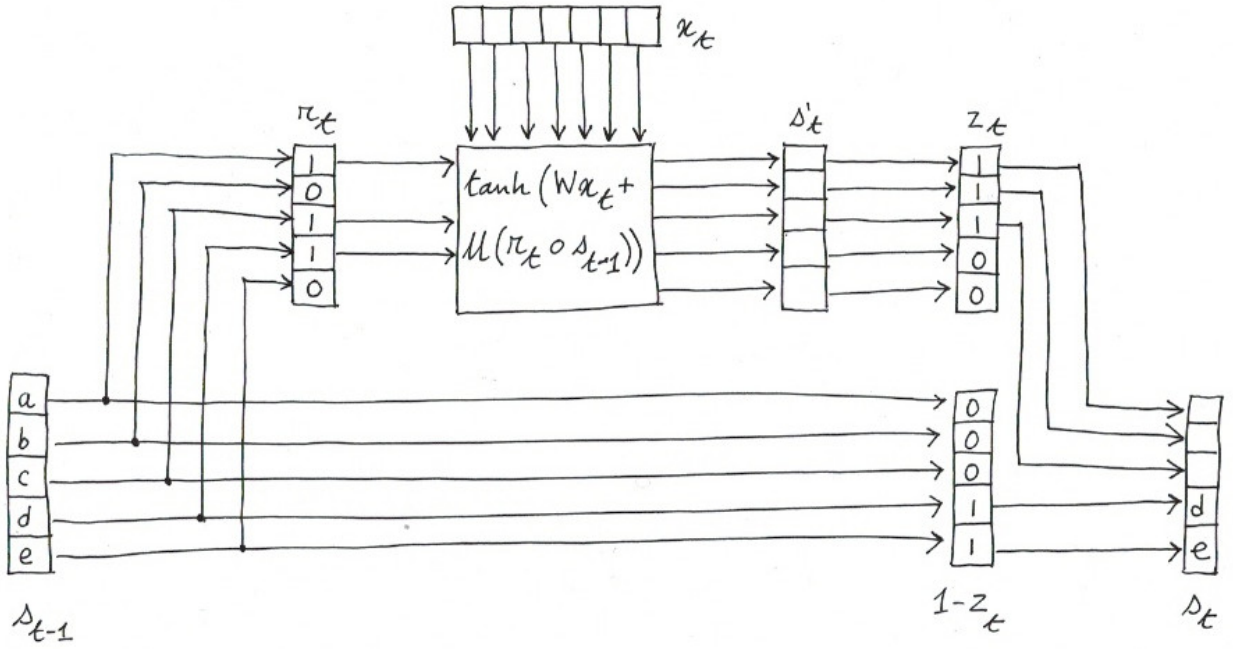
$$\mathbf{s}_t = (1 - \mathbf{z_t}) \circ \mathbf{s}_{t-1} + \mathbf{z_t} \circ \mathbf{s'_t}$$

- $\mathbf{s'_t}$ est applelé l'état interne au temps t, il est calculé de la manière suivante :

$$\mathbf{s'_t} = \tanh(W\mathbf{x}_t + U(\mathbf{r_t} \circ \mathbf{s}_{t-1}))$$

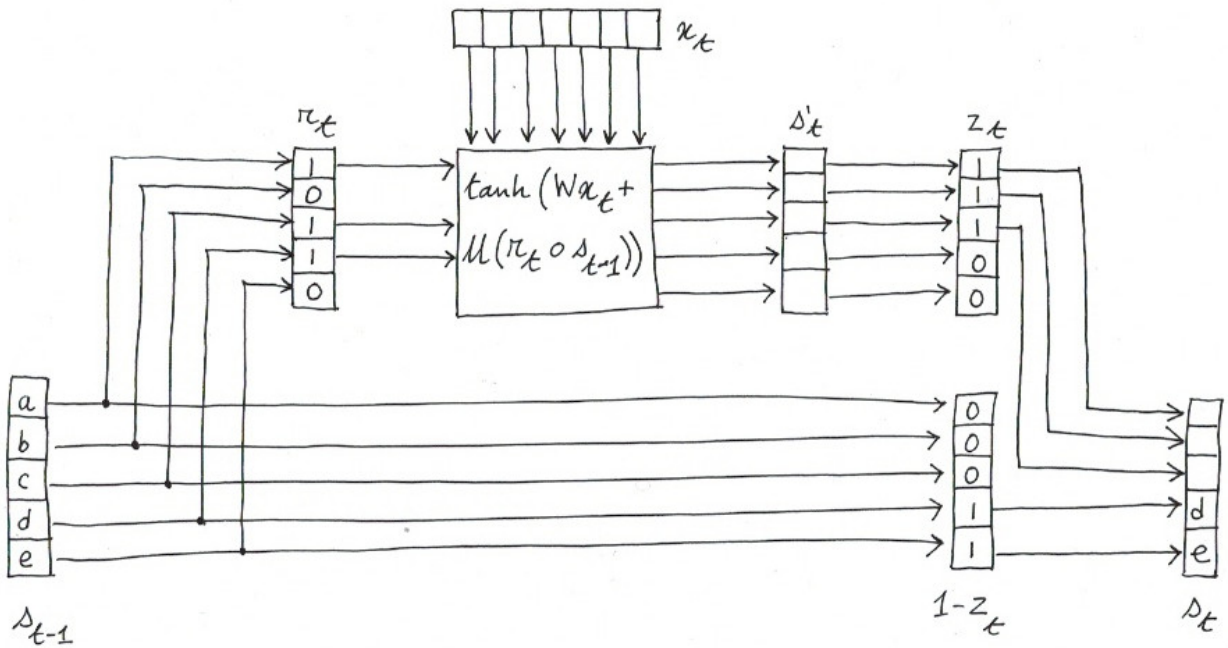
- où tanh est la fonction tangente hyperbolique.

Calcul de $\mathbf{s}_t$



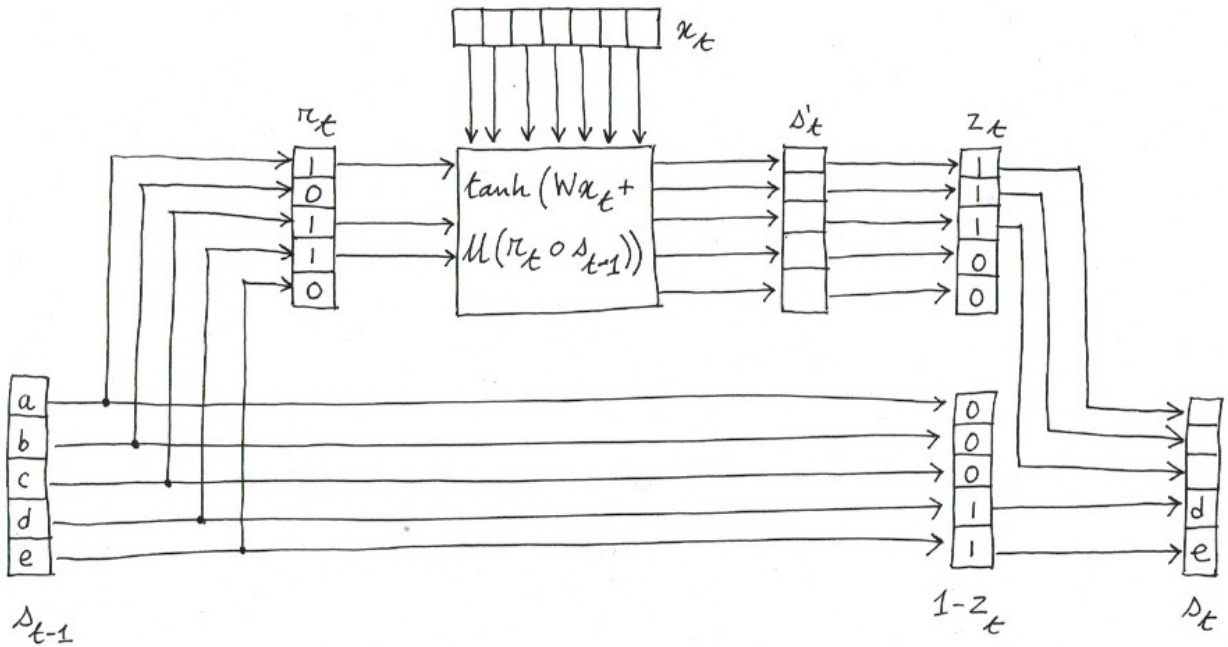
- Afin de ne pas surcharger le schéma, nous avons omis le calcul des filtres $\mathbf{z_t}$ et $\mathbf{r_t}$ à partir de l'entrée $\mathbf{x}_t$ et du vecteur d'état $\mathbf{s}_{t-1}$.
- Dans la partie basse du schéma, les composantes 4 et 5, correspondant à d et e, du vecteur d'état $\mathbf{s}_{t-1}$ sont recopiées dans le vecteur d'état $\mathbf{s}_t$.
- Cette recopie est réalisée par la combinaison du vecteur d'état $\mathbf{s}_{t-1}$ et du filtre $1 - \mathbf{z_t}$:$(1 - \mathbf{z_t}) \circ \mathbf{s}_{t-1}$

Calcul de $\mathbf{s}_t$



- Dans la partie haute du schéma, le vecteur d'état $\mathbf{s}_{t-1}$ est combiné avec le filtre $\mathbf{r_t}$, qui ne garde du premier que les composantes 1, 3 et 4 (les valeurs a, c et d) et oublie par conséquent les valeurs b et e.
- Les composantes 1, 3 et 4 sont combinées avec l'entrée $\mathbf{x}_t$, à l'aide des matrices W et U pour former le vecteur $\mathbf{s'_t}$, appelé *état interne*.

Calcul de $\mathbf{s}_t$



- L'état interne est combiné avec le filtre $\mathbf{z_t}$, qui n'en garde que les trois premières composantes.
- Le vecteur d'état $\mathbf{s}_t$ est construit en gardant les trois premières composantes de $\mathbf{s'_t}$ et les deux dernières composantes de $\mathbf{s}_{t-1}$.

Résumé

Entrée au temps t	$\mathbf{x}_t$
Etat au temps $t-1$	$\mathbf{s}_t$
Update gate	$\mathbf{z_t} = \sigma(W^z \mathbf{x}_t + U^z \mathbf{s}_{t-1})$
Reset gate	$\mathbf{r_t} = \sigma(W^r \mathbf{x}_t + U^r \mathbf{s}_{t-1})$
Etat interne	$\mathbf{s'_t} = \tanh(W \mathbf{x}_t + U(\mathbf{r_t} \circ \mathbf{s}_{t-1}))$
Etat au temps t	$\mathbf{s}_t = (1 - \mathbf{z_t}) \circ \mathbf{s}_{t-1} + \mathbf{z_t} \circ \mathbf{s'_t}$

Implémentation avec keras

```
tf.keras.layers.GRU(
    units,                          # dimension du vecteur d'état
    activation="tanh",              # fonction d'activation de l'état interne
    recurrent_activation="sigmoid", # fonction d'activation des filtres
    use_bias=True,                  # ajout d'un biais
    return_sequences=False,         # renvoie la dernière sortie ou toutes les sorties
    return_state=False,             # renvoie le dernier vecteur d'état ou pas
    ...
    )
```

Lemmatisation

- Lemme : forme canonique d'un mot variable
  - mangeront $\rightarrow$ manger
  - arbres $\rightarrow$ arbre
  - bleues $\rightarrow$ bleu
- Objectif : concevoir un GRU qui réalise la lemmatisation.
- Deux cas :
  - Un modèle par catégorie
  - Un modèle pour toutes les catégories

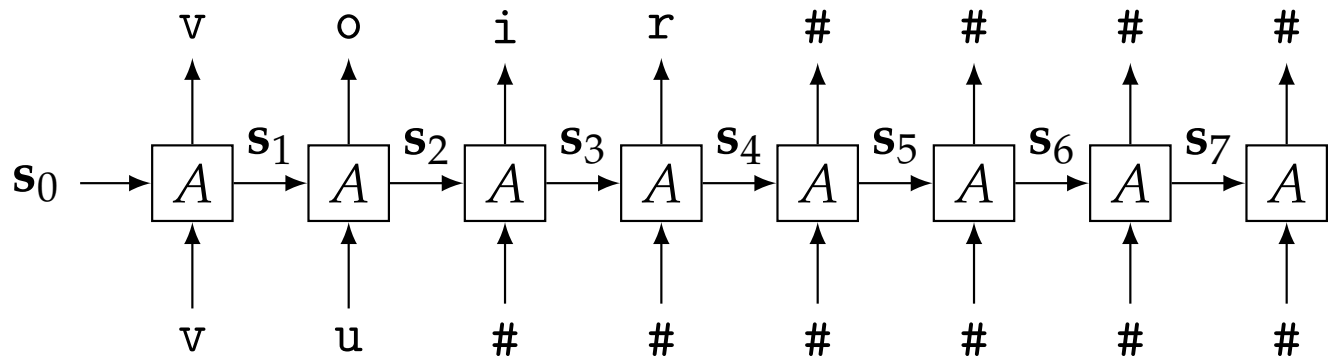
Lemmatisation

- Type de réseau : transducteur
- Encodage de l'entrée : chaque caractère est représenté par un vecteur one-hot
- Différence de longueur de l'entrée et de la sortie
  - Entrée plus longue : *pourrait* $\rightarrow$ *pouvoir*
  - Sortie plus longue : *vu* $\rightarrow$ *voir*
- En général, le lemme est plus court que la forme
- Padding : on ajoute des # en fin de lemme et de forme pour que les deux aient la même longueur :
  - p o u r r a i t # # # # # #
p o u v o i r # # # # # # #
  - v u # # # # # #
v o i r # # # #
- On ajoute systématiquement 6 # à la fin de la forme, afin de pouvoir absorber jusqu'à 6 lettres de plus dans le lemme.

Fichiers de données

```
Certes######    certes######    ADV
rien######      rien######      PRO
ne######        ne######        ADV
dit######       dire#####       V
une######       un#######       D
seconde######   second#######   A
motion######    motion######    N
de######        de######        P
censure######   censure######   N
sur######       sur######       P
son######       son######       D
projet######    projet######    N
de######        de######        P
loi######       loi######       N
reprenant###### reprendre###### V
accord######    accord######    N
```

Implémentation à l'aide de transducteurs



Chaque lettre est encodée à l'aide d'un vecteur *one-hot*

Rôle de la catégorie

- Il est des fois difficile de lemmatiser sans connaître la catégorie de la forme
  - (couvent, V) $\rightarrow$ couver
  - (couvent, N) $\rightarrow$ couvent
- Deux cas :
  - Lemmatiseur catégoriel : un lemmatiseur par catégorie
  - Lemmatiseur général : un lemmatiseur pour toutes les catégories confondues
- On ajoute en entrée la catégorie de la forme.

Sources

- Yoav Goldberg, *Neural Network Methods for Natural Language Processing*, Morgan & Claypool, 2017.
- Ian Goodfellow, Yoshua Bengio, Aaron Courville, *Deep Learning*, MIT Press, 2016.