Apprentissage Automatique
Master Sciences Cognitives

Alexis Nasr

27 mars 2020

Chapitre 1

Réseaux récurrents

Un réseau convolutionnel permet de prendre en entrée une séquence d'observations tout en l'analysant par petit bouts à l'aide d'une *fenêtre glissante*.

Mais un tel réseau ne permet pas de prendre en entrée une séquence de longueur quelconque.

Nous nous sommes servi de ce type de réseaux pour des séquences de caractères de longueur fixe (100, 1000, ...) dans le cas de l'identification de la langue d'un texte ou d'images de 28×28 pixels pour la reconnaissance de chiffres manuscrits.

On pourrait imaginer un réseau dont le but est de prédire, par exemple, quel événement va avoir lieu à un moment donné dans un film.

Il n'est pas évident de concevoir un tel réseau car on ne sait pas à quels moments dans le passé ont lieu des événements qui aideront à prédire un événement au temps présent.

Afin d'autoriser la prise en compte d'événements arbitrairement distants, on introduit la notion d'**état** du réseau au temps t [1], noté $\mathbf{s}_t$ (s pour state en anglais).

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 $t-1$: $\mathbf{s}_{t-1}$.

$\mathbf{s}_i$ véhicule de l'information utile pour des prédictions au temps $t > i$, il se présente sous la forme d'un vecteur.

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.

1. De manière plus général, 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.

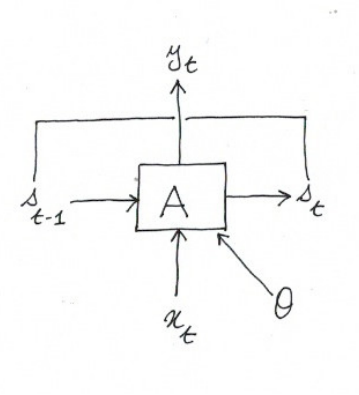

FIGURE 1.1 – un réseau récurrent

C'est cette caractéristique qui lui confère le nom de **réseau récurrent**, noté RNN dans la suite de ce document.

Un RNN est généralement représenté à l'aide d'une boucle, comme dans la figure 1.1 qui illustre bien son caractère récurrent.

Si on observe le fonctionnement d'un réseau dans le temps, on obtient une vision "dépliée" de ce réseau, comme dans la figure 1.2.

Dans la version dépliée, chaque occurrence du réseau A correspond à un instant, on voit ainsi le déroulement du réseau dans le temps : à chaque instant t (dans la figure t prend respectivement les valeurs 1, 2, 3 et 4) l'entrée $\mathbf{x}_t$ est lue, ainsi que le vecteur d'état $\mathbf{s}_{t-1}$, un sortie $\hat{\mathbf{y}}_t$ est calculée, ainsi que le vecteur d'état $\mathbf{s}_t$.

Il est important de bien remarquer que, comme dans un réseau convolutionnel, le réseau A est "dupliqué" plusieurs fois, il s'agit du même réseau, et donc des mêmes paramètres.

Nous avons représenté de manière explicite les paramètres θ dans la figure 1.2 pour insister sur le fait qu'il s'agit d'un même jeu de paramètres, pour toutes les étapes du traitement.

Notations :

— L'entrée du réseau est constitué d'une séquence $\mathbf{x}_{1\ldots n}$ de vecteurs de dimension d_{in}.

— La sortie du réseau au temps t est un vecteur $\hat{\mathbf{y}}_t$ de dimension d_{out} :

$$\hat{\mathbf{y}}_t = rnn(\mathbf{x}_{1\ldots n})$$

— Pour chaque sous-séquence $\mathbf{x}_{1\ldots t}$, le réseau produit une sortie $\mathbf{y_t}$.

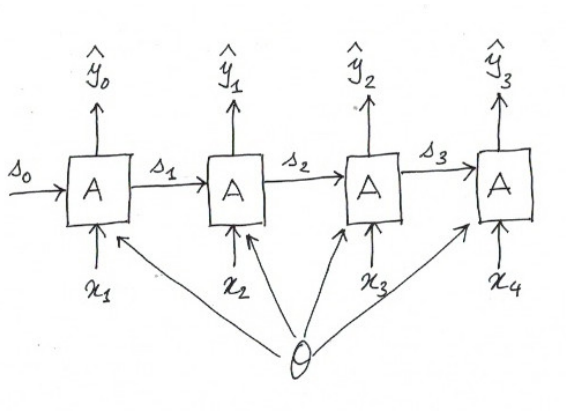

FIGURE 1.2 – un réseau récurrent déplié

On note rnn^* la fonction qui produit la séquence $\mathbf{y}_{1...t}$

$$\hat{\mathbf{y}}_{1...t} = rnn^*(\mathbf{x}_{1...t})$$

Le vecteur de sortie $\hat{\mathbf{y}}_t$ est généralement utilisé pour effectuer une prédiction, par exemple à l'aide d'un MLP.

De manière plus précise, le RNN peut être défini à l'aide d'une fonction R qui prend en entrée un état $\mathbf{s}_{t-1}$ et une entrée x_t pour produire un nouvel état $\mathbf{s}_t$:

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

On définit de plus une fonction O qui produit une sortie $\mathbf{y_i}$ à partir de $\mathbf{s}_i$:

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

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, ce que l'on fera plus tard.

La définition de R et de O confère au réseau un caractère récursif, qui permet de prendre en compte, lors de la production de y_i, toutes les entrées $x_{j \leq i}$.

Illustration pour $i = 4$

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

Comme on peut l'observer ci-dessus, la valeur de $\mathbf{s}_4$ est calculée à partir des entrées $\mathbf{x}_{1...4}$.

$\mathbf{s}_n$ et $\hat{\mathbf{y}}_n$ peuvent être vus comme des manières de représenter (on dit aussi d'**encoder**) toute la séquence d'entrée $\mathbf{x}_{1...n}$.

C'est lors de l'étape d'apprentissage que les paramètres de R et de O sont déterminés, de manière que les informations pertinentes pour une prédiction au temps t soient véhiculées par le vecteur d'état.

L'apprentissage des paramètres est réalisé sur le réseau sous forme déplié.

Pour cela, il faut définir une fonction de perte et utliser l'algorithme de descente du gradient, par exemple, pour calculer les paramètres du réseau qui permettent de minimiser la fonction de perte sur des données d'apprentissage.

C'est par le biais de la minimisation de la fonction de perte 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.

1.1 Accepteurs et Transducteurs

1.1.1 Accepteurs

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 :

1. 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.

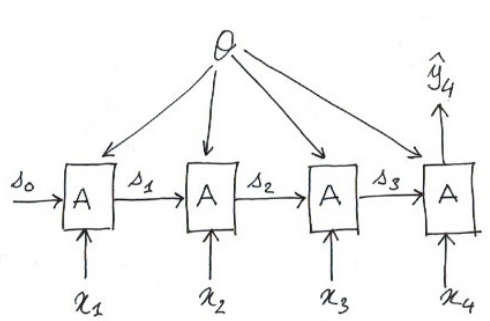

FIGURE 1.3 – un accepteur

2. 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, comme dans l'exemple ci-dessous :
   - *J'ai vraiment détesté ce cours* → NEGATIF
   - *rarement un cours m'a autant intéressé* → POSITIF
   - *Le cours a lieu toutes les semaines en salle 202* → NEUTRE

Le vecteur d'état $\mathbf{s}_n$ produit à l'issue du traitement d'une séquence $\mathbf{x}_{1...n}$ est souvent utilisée comme entrée d'une autre tâche de prédiction.

On peut imaginer par exemple que $\mathbf{s}_n$ constitue l'entrée d'un autre réseau R qui peut être, par exemple, MLP ou un CNN, comme l'illustre la figure 1.4.

$\mathbf{s}_n$ correspond à une **représentation** de la séquence $\mathbf{x}_{1...n}$, on dit aussi son **encodage**.

Le RNN est appelé dans ce cas là un **encodeur**.

La prédiction est réalisée par R et c'est donc à la sortie de R que la fonction de perte est calculée et que la rétro-propagation est réalisée, pour mettre à jour les paramètres du RNN.

1.1.2 Transducteurs

De manière générale, un transducteur est un programme qui lit une séquence et en produit une autre.

On peut imaginer par exemple un programme qui effectue une traduction d'une langue source vers une langue cible. Il lit une séquence dans la langue

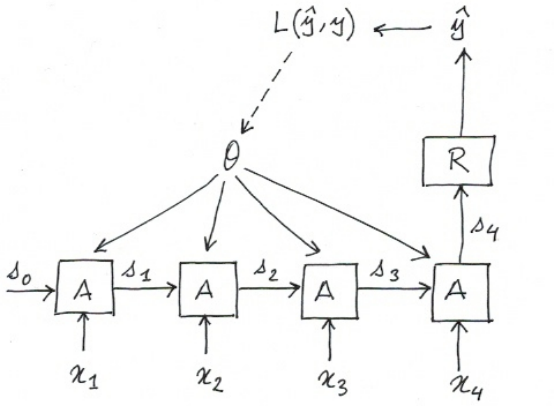

FIGURE 1.4 – un encodeur

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 $\mathbf{x}_t$.

Ceci permet de calculer 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 :

$$L(\hat{\mathbf{y}}_{1...n}, \mathbf{y}_{1...n}) = \sum_{t=1}^{n} L(\hat{\mathbf{y}}_t, \mathbf{y}_t)$$

Exemple :

On veut attribuer une partie de discours (catégorie grammaticale), tel que : *Verbe*, *Nom*, *Adj* aux différents mots d'un texte, comme dans l'exemple suivant :

$\mathbf{x}$	Marie	donne	le	la
$\hat{\mathbf{y}}$	Nom	Verbe	Det	Nom

La prédiction de la partie de discours d'un mot dépend bien sûr 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.

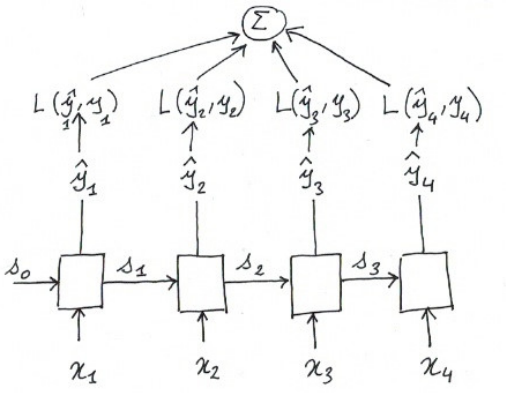

FIGURE 1.5 – un transducteur

1.2 Quelques types de réseaux récurrents

Le modèle de RNN présenté jusque là est très général, il est défini par les deux équations suivantes :

$$\begin{aligned}\mathbf{s}_t &= R(\mathbf{x}_t, \mathbf{s}_{t-1}) \\ \hat{\mathbf{y}}_t &= O(\mathbf{s}_t)\end{aligned}$$

Selon la forme que prennent R et O, différents types de RNN peuvent être définis.

On en verra deux :

— Les RNN simples, notés SRNN, ou réseaux de Elman

— Les *Gated Recurrent Units*, notés GRU

1.2.1 RNN simples

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 $\mathbf{s}_t$.

Le modèle SRNN est confronté au problème de la disparition du gradient (*vanishing gradient* en anglais) : lors de la rétro-propagation de l'erreur, cette dernière diminue rapidement (car elle est multipliée successivement par des poids dont la valeur est souvent 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 **porte** ou de **filtre** (*gate* en anglais).

Un filtre $\mathbf{f}$ se présente comme un vecteur composé de 0 et de 1.

Il est combiné à un autre vecteur $\mathbf{x}$ 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 $\mathbf{f}$ est multipliée par la composante i de $\mathbf{x}$), comme dans l'exemple suivant :

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

Dans cet exemple, le filtre a permis “d'oublier” les composantes 2 et 3 du vecteur $\mathbf{x}$.

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 :

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

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

Exemple :

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

L'idée, bien entendu, est d'apprendre les filtres lors de l'apprentissage : le réseau apprend l'information qu'il doit retenir de l'état précédent.

Cela pose un 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**.

Pour cela, on autorise les composantes du filtre à prendre pour valeur n'importe quel réel, ce qui constitue un filtre $\mathbf{f}'$ a 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, comme on peut le voir dans la figure 1.6.

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.

1.2.2 Les Gated Recurrent Units

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 $\mathbf{s}_t$.

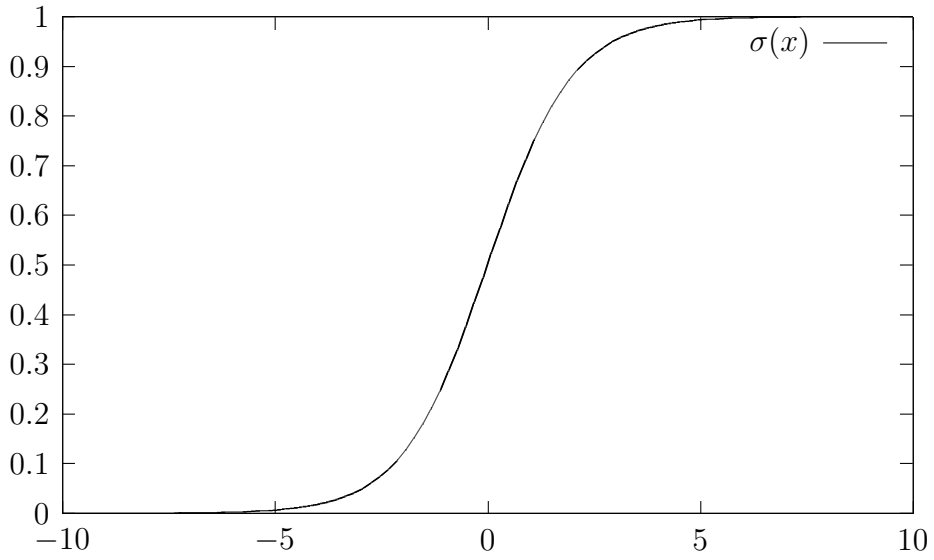

FIGURE 1.6 – La fonction sigmoïde

— 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 de $\mathbf{z_t}$

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

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

Ce calcul est représenté schématiquement dans la figure 1.7.

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 de $\mathbf{r_t}$

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

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

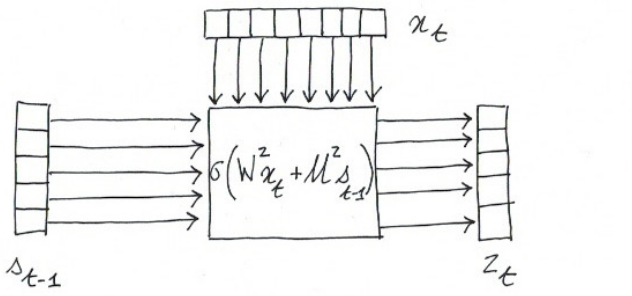

FIGURE 1.7 – Représentation schématique du calcul du filtre $\mathbf{z_t}$

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 :

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

$\mathbf{s'_t}$ 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 (voir chapitre sur les fonctions).

Ce calcul est représenté schématiquement dans la figure 1.8 qui donne une idée générale du comportement du GRU.

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 $\mathbf{s}_{t-1}$.

On observe, dans la partie basse du schéma, que les composantes 4 et 5, correspondant à d et e, du vecteur d'état $\mathbf{s}_{t-1}$ sont recopiés 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 $1 - \mathbf{z_t}$.

Le vecteur d'état $\mathbf{s}_{t-1}$ est aussi combiné, dans la partie haute du schéma, 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.

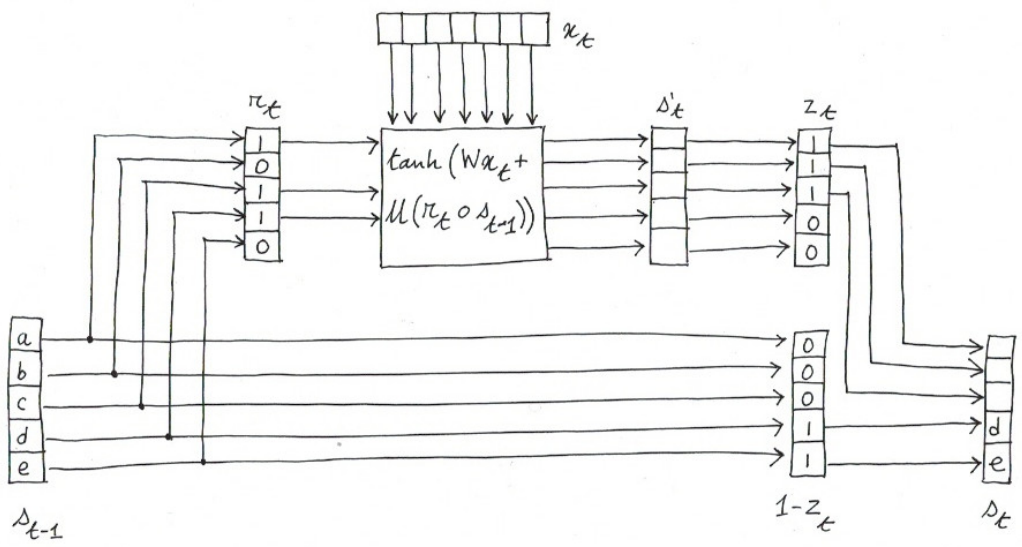

FIGURE 1.8 – Représentation schématique du calcul du vecteur d'état $\mathbf{s}_t$

Les composantes 1, 3 et 4 sont combinées avec l'entrée $\mathbf{x}_t$, à l'aide des matrices W et U pour former l'état interne $\mathbf{s'_t}$. Ce dernier est alors combiné avec le filtre $\mathbf{z_t}$, qui n'en garde que les trois premières composantes.

Le vecteur d'état $\mathbf{s}_t$ est alors construit, en gardant les trois premières composantes de $\mathbf{s'_t}$ et les deux dernières composantes de $\mathbf{s}_{t-1}$.