

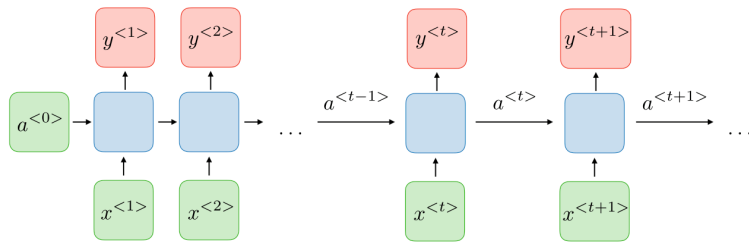
Pense-bête VIP : Réseaux de neurones récurrents

Afshine AMIDI et Shervine AMIDI

6 janvier 2019

Vue d'ensemble

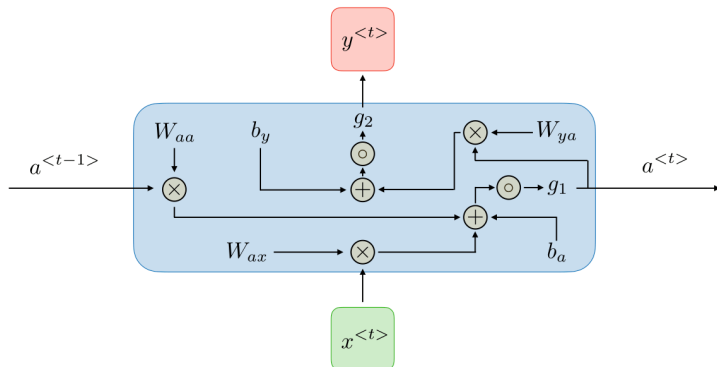
□ **Architecture d'un RNN traditionnel** – Les réseaux de neurones récurrents (en anglais *recurrent neural networks*), aussi appelés RNNs, sont une classe de réseaux de neurones qui permettent aux prédictions antérieures d'être utilisées comme entrées, par le biais d'états cachés (en anglais *hidden states*). Ils sont de la forme suivante :



À l'instant t , l'activation $a^{<t>}$ et la sortie $y^{<t>}$ sont de la forme suivante :

$$a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a) \quad \text{et} \quad y^{<t>} = g_2(W_{ya}a^{<t>} + b_y)$$

où $W_{ax}, W_{aa}, W_{ya}, b_a, b_y$ sont des coefficients indépendants du temps et où g_1, g_2 sont des fonctions d'activation.



Les avantages et inconvénients des architectures de RNN traditionnelles sont résumés dans le tableau ci-dessous :

Avantages	Inconvénients
<ul style="list-style-type: none"> - Prend en compte des entrées de toute taille - La taille du modèle n'augmente pas avec la taille de l'entrée - Les calculs prennent en compte les infos antérieures - Les coefficients sont indépendants du temps 	<ul style="list-style-type: none"> - Le temps de calcul est long - Difficulté d'accéder à des informations d'un passé lointain - Impossibilité de prendre en compte des informations futures un état donné

□ **Applications des RNNs** – Les modèles RNN sont surtout utilisés dans les domaines du traitement automatique du langage naturel et de la reconnaissance vocale. Le tableau suivant détaille les applications principales à retenir :

Type de RNN	Illustration	Exemple
Un à un $T_x = T_y = 1$		Réseau de neurones traditionnel
Un à plusieurs $T_x = 1, T_y > 1$		Génération de musique
Plusieurs à un $T_x > 1, T_y = 1$		Classification de sentiment
Plusieurs à plusieurs $T_x = T_y$		Reconnaissance d'entité
Plusieurs à plusieurs $T_x \neq T_y$		Traduction machine

□ **Fonction de loss** – Dans le contexte des réseaux de neurones récurrents, la fonction de loss \mathcal{L} prend en compte le loss à chaque temps T de la manière suivante :

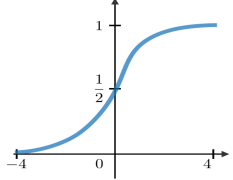
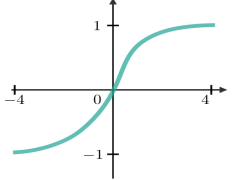
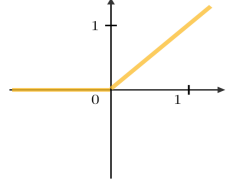
$$\mathcal{L}(\hat{y}, y) = \sum_{t=1}^{T_y} \mathcal{L}(\hat{y}^{<t>, y^{<t>})$$

□ **Backpropagation temporelle** – L'étape de backpropagation est appliquée dans la dimension temporelle. À l'instant T , la dérivée du loss \mathcal{L} par rapport à la matrice de coefficients W est donnée par :

$$\frac{\partial \mathcal{L}^{(T)}}{\partial W} = \sum_{t=1}^T \frac{\partial \mathcal{L}^{(T)}}{\partial W} \Big|_{(t)}$$

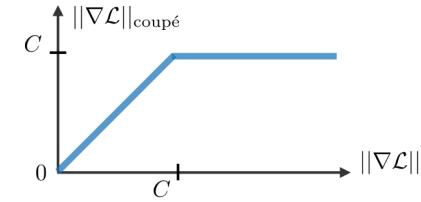
Dépendances à long terme

□ **Fonctions d'activation communément utilisées** – Les fonctions d'activation les plus utilisées dans les RNNs sont décrits ci-dessous :

Sigmoïde	Tanh	RELU
$g(z) = \frac{1}{1 + e^{-z}}$	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0, z)$
		

□ **Gradient qui disparaît/explose** – Les phénomènes de gradient qui disparaît et qui explose (en anglais *vanishing gradient* et *exploding gradient*) sont souvent rencontrés dans le contexte des RNNs. Ceci est dû au fait qu'il est difficile de capturer des dépendances à long terme à cause du gradient multiplicatif qui peut décroître/croître de manière exponentielle en fonction du nombre de couches.

□ **Coupage de gradient** – Cette technique est utilisée pour atténuer le phénomène de gradient qui explose qui peut être rencontré lors de l'étape de backpropagation. En plafonnant la valeur qui peut être prise par le gradient, ce phénomène est maîtrisé en pratique.



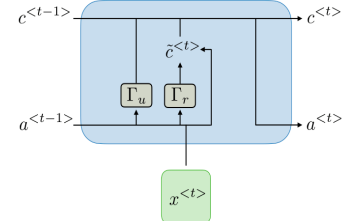
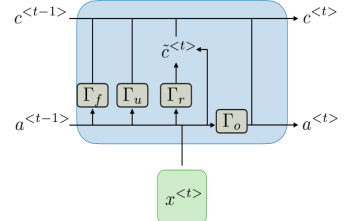
□ **Types de porte** – Pour remédier au problème du gradient qui disparaît, certains types de porte sont spécifiquement utilisés dans des variantes de RNNs et ont un but bien défini. Les portes sont souvent notées Γ et sont telles que :

$$\Gamma = \sigma(Wx^{<t>} + Ua^{<t-1>} + b)$$

où W, U, b sont des coefficients spécifiques à la porte et σ est une sigmoïde. Les portes à retenir sont récapitulées dans le tableau ci-dessous :

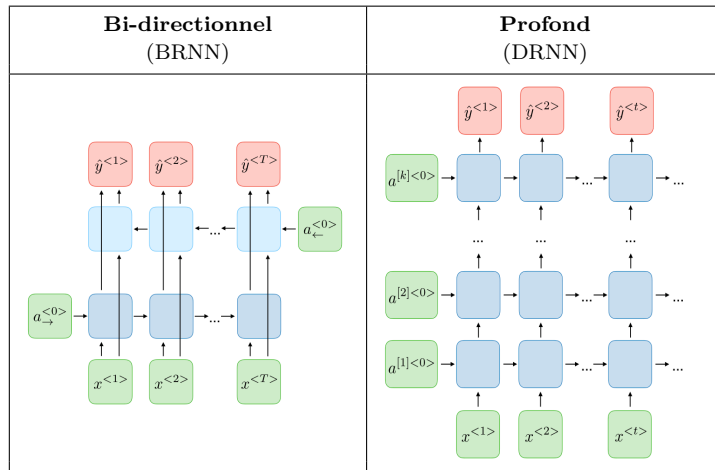
Type de porte	Rôle	Utilisée dans
Porte d'actualisation Γ_u	Dans quelle mesure le passé devrait être important ?	GRU, LSTM
Porte de pertinence Γ_r	Enlever les informations précédentes ?	GRU, LSTM
Porte d'oubli Γ_f	Enlever une cellule ?	LSTM
Porte de sortie Γ_o	Combien devrait-on révéler d'une cellule ?	LSTM

□ **GRU/LSTM** – Les unités de porte récurrente (en anglais *Gated Recurrent Unit*) (GRU) et les unités de mémoire à long/court terme (en anglais *Long Short-Term Memory units*) (LSTM) appaisent le problème du gradient qui disparaît rencontré par les RNNs traditionnels, où le LSTM peut être vu comme étant une généralisation du GRU. Le tableau ci-dessous résume les équations caractéristiques de chacune de ces architectures :

	Gated Recurrent Unit (GRU)	Long Short-Term Memory (LSTM)
$\tilde{c}^{<t>}$	$\tanh(W_c[\Gamma_r \star a^{<t-1>}, x^{<t>}] + b_c)$	$\tanh(W_c[\Gamma_r \star a^{<t-1>}, x^{<t>}] + b_c)$
$c^{<t>}$	$\Gamma_u \star \tilde{c}^{<t>} + (1 - \Gamma_u) \star c^{<t-1>}$	$\Gamma_u \star \tilde{c}^{<t>} + \Gamma_f \star c^{<t-1>}$
$a^{<t>}$	$c^{<t>}$	$\Gamma_o \star c^{<t>}$
Dépendances		

Remarque : le signe \star dénote le produit de Hadamard entre deux vecteurs.

□ **Variantes des RNNs** – Le tableau ci-dessous récapitule les autres architectures RNN communément utilisées :



Apprentissage de la représentation de mots

Dans cette section, on note V le vocabulaire et $|V|$ sa taille.

□ **Techniques de représentation** – Les deux manières principales de représenter des mots sont décrits dans le tableau suivant :

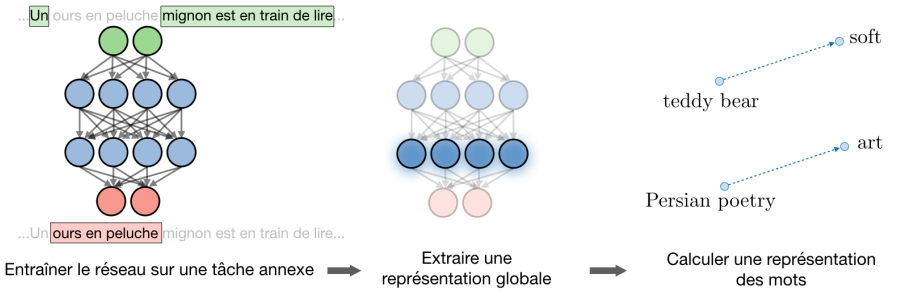
Représentation binaire	Représentation du mot
<ul style="list-style-type: none"> - Noté o_w - Approche naïve, pas d'information de similarité 	<ul style="list-style-type: none"> - Noté e_w - Prend en compte la similarité des mots

□ **Matrice de représentation** – Pour un mot donné w , la matrice de représentation (en anglais *embedding matrix*) E est une matrice qui relie une représentation binaire o_w à sa représentation correspondante e_w de la manière suivante :

$$e_w = E o_w$$

Remarque : l'apprentissage d'une matrice de représentation peut être effectuée en utilisant des modèles probabilistiques de cible/contexte.

□ **Word2vec** – Word2vec est un ensemble de techniques visant à apprendre comment représenter les mots en estimant la probabilité qu'un mot donné a d'être entouré par d'autres mots. Le skip-gram, l'échantillonnage négatif et le CBOW font parti des modèles les plus populaires.



□ **Skip-gram** – Le skip-gram est un modèle de type supervisé qui apprend comment représenter les mots en évaluant la probabilité de chaque mot cible t donné dans un mot contexte c . En notant θ_t le paramètre associé à t , la probabilité $P(t|c)$ est donnée par :

$$P(t|c) = \frac{\exp(\theta_t^T e_c)}{\sum_{j=1}^{|V|} \exp(\theta_j^T e_c)}$$

Remarque : le fait d'additionner tout le vocabulaire dans le dénominateur du softmax rend le modèle coûteux en temps de calcul. CBOW est un autre modèle utilisant les mots avoisinants pour prédire un mot donné.

□ **Échantillonnage négatif** – Cette méthode utilise un ensemble de classifieurs binaires utilisant des régressions logistiques qui visent à évaluer dans quelle mesure des mots contexte et cible sont susceptible d'apparaître simultanément, avec des modèles étant entraînés sur des ensembles de k exemples négatifs et 1 exemple positif. Étant donnés un mot contexte c et un mot cible t , la prédiction est donnée par :

$$P(y = 1|c, t) = \sigma(\theta_t^T e_c)$$

Remarque : cette méthode est moins coûteuse en calcul par rapport au modèle skip-gram.

□ **GloVe** – Le modèle GloVe (en anglais *global vectors for word representation*) est une technique de représentation des mots qui utilise une matrice de co-occurrence X où chaque $X_{i,j}$ correspond au nombre de fois qu'une cible i se produit avec un contexte j . Sa fonction de coût J est telle que :

$$J(\theta) = \frac{1}{2} \sum_{i,j=1}^{|V|} f(X_{i,j})(\theta_i^T e_j + b_i + b'_j - \log(X_{i,j}))^2$$

où f est une fonction à coefficients telle que $X_{i,j} = 0 \implies f(X_{i,j}) = 0$.

Étant donné la symétrie que e et θ ont dans un modèle, la représentation du mot final $e_w^{(\text{final})}$ est donnée par :

$$e_w^{(\text{final})} = \frac{e_w + \theta_w}{2}$$

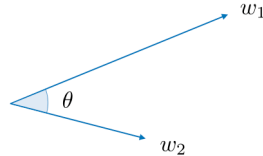
Remarque : les composantes individuelles de la représentation d'un mot n'est pas nécessairement facilement interprétable.

Comparaison de mots

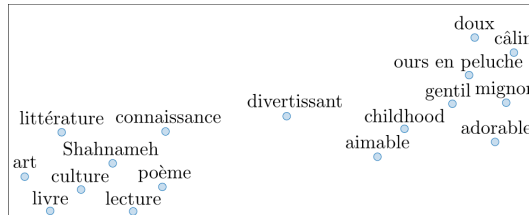
□ **Similarité cosinus** – La similarité cosinus (en anglais *cosine similarity*) entre les mots w_1 et w_2 est donnée par :

$$\text{similarity} = \frac{w_1 \cdot w_2}{\|w_1\| \|w_2\|} = \cos(\theta)$$

Remarque : θ est l'angle entre les mots w_1 et w_2 .



□ **t-SNE** – La méthode *t*-SNE (en anglais *t-distributed Stochastic Neighbor Embedding*) est une technique visant à réduire une représentation dans un espace de haute dimension en un espace de plus faible dimension. En pratique, on visualise les vecteur-mots dans un espace 2D.



Modèle de langage

□ **Vue d'ensemble** – Un modèle de langage vise à estimer la probabilité d'une phrase $P(y)$.

□ **Modèle *n*-gram** – Ce modèle consiste en une approche naïve qui vise à quantifier la probabilité qu'une expression apparaisse dans un corpus en comptabilisant le nombre de son apparition dans le training data.

□ **Perplexité** – Les modèles de langage sont communément évalués en utilisant la perplexité, aussi noté PP, qui peut être interprété comme étant la probabilité inverse des données normalisée par le nombre de mots T . La perplexité est telle que plus elle est faible, mieux c'est. Elle est définie de la manière suivante :

$$PP = \prod_{t=1}^T \left(\frac{1}{\sum_{j=1}^{|V|} y_j^{(t)} \cdot \hat{y}_j^{(t)}} \right)^{\frac{1}{T}}$$

Remarque : PP est souvent utilisée dans le cadre du *t*-SNE.

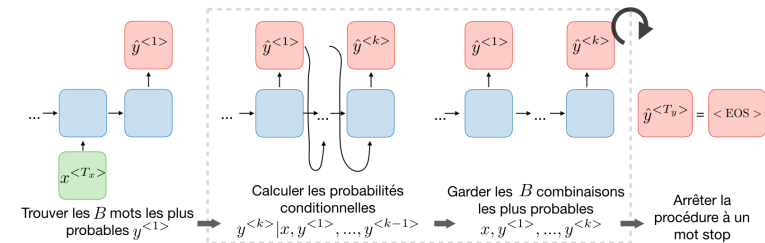
Traduction machine

□ **Vue d'ensemble** – Un modèle de traduction machine est similaire à un modèle de langage ayant un auto-encodeur placé en amont. Pour cette raison, ce modèle est souvent surnommé modèle conditionnel de langage. Le but est de trouver une phrase y telle que :

$$y = \arg \max_{y^{<1>}, \dots, y^{<T_y>}} P(y^{<1>}, \dots, y^{<T_y>} | x)$$

□ **Recherche en faisceau** – Cette technique (en anglais *beam search*) est un algorithme de recherche heuristique, utilisé dans le cadre de la traduction machine et de la reconnaissance vocale, qui vise à trouver la phrase la plus probable y sachant l'entrée x .

- Étape 1 : Trouver les B mots les plus probables $y^{<1>}$
- Étape 2 : Calculer les probabilités conditionnelles $y^{<k>} | x, y^{<1>}, \dots, y^{<k-1>}$
- Étape 3 : Garder les B combinaisons les plus probables $x, y^{<1>}, \dots, y^{<k>}$



Remarque : si la largeur du faisceau est prise égale à 1, alors ceci est équivalent à un algorithme glouton.

□ **Largeur du faisceau** – La largeur du faisceau (en anglais *beam width*) B est un paramètre de la recherche en faisceau. De grandes valeurs de B conduisent à avoir de meilleurs résultats mais avec un coût de mémoire plus lourd et à un temps de calcul plus long. De faibles valeurs de B conduisent à de moins bons résultats mais avec un coût de calcul plus faible. Une valeur de B égale à 10 est standard et est souvent utilisée.

□ **Normalisation de longueur** – Pour que la stabilité numérique puisse être améliorée, la recherche en faisceau utilise un objectif normalisé, souvent appelé l'objectif de log-probabilité normalisé, défini par :

$$\text{Objectif} = \frac{1}{T_y^\alpha} \sum_{t=1}^{T_y} \log \left[p(y^{<t>} | x, y^{<1>}, \dots, y^{<t-1>}) \right]$$

Remarque : le paramètre α est souvent comprise entre 0.5 et 1.

□ **Analyse d'erreur** – Lorsque l'on obtient une mauvaise traduction prédite \hat{y} , on peut se demander la raison pour laquelle l'algorithme n'a pas obtenu une bonne traduction y^* en faisant une analyse d'erreur de la manière suivante :

Cas	$P(y^* x) > P(\hat{y} x)$	$P(y^* x) \leq P(\hat{y} x)$
Cause	Recherche en faisceau défectueuse	RNN défectueux
Remèdes	Augmenter la largeur du faisceau	- Essayer une différente architecture - Régulariser - Obtenir plus de données

□ **Score bleu** – Le score bleu (en anglais *bilingual evaluation understudy*) a pour but de quantifier à quel point une traduction est bonne en calculant un score de similarité basé sur une précision *n*-gram. Il est défini de la manière suivante :

$$\text{score bleu} = \exp \left(\frac{1}{n} \sum_{k=1}^n p_k \right)$$

où p_n est le score bleu unique basé sur les n -gram, défini par :

$$p_n = \frac{\sum_{\text{n-gram} \in \hat{y}} \text{count}_{\text{clip}}(\text{n-gram})}{\sum_{\text{n-gram} \in \hat{y}} \text{count}(\text{n-gram})}$$

Remarque : une pénalité de brièveté peut être appliquée aux traductions prédites courtes pour empêcher que le score bleu soit artificiellement haut.

Attention

□ **Modèle d'attention** – Le modèle d'attention (en anglais *attention model*) permet au RNN de mettre en valeur des parties spécifiques de l'entrée qui peuvent être considérées comme étant importantes, ce qui améliore la performance du modèle final en pratique. En notant $\alpha^{<t,t'>}$ la quantité d'attention que la sortie $y^{<t>}$ devrait porter à l'activation $a^{<t'>}$ et au contexte $c^{<t>}$ à l'instant t , on a :

$$c^{<t>} = \sum_{t'} \alpha^{<t,t'>} a^{<t'>} \quad \text{avec} \quad \sum_{t'} \alpha^{<t,t'>} = 1$$

Remarque : les scores d'attention sont communément utilisés dans la génération de légende d'image ainsi que dans la traduction machine.



Un ours en peluche mignon est en train de lire de la littérature persane



Un ours en peluche mignon est en train de lire de la littérature persane

□ **Coefficient d'attention** – La quantité d'attention que la sortie $y^{<t>}$ devrait porter à l'activation $a^{<t'>}$ est donné $\alpha^{<t,t'>}$, qui est calculé de la manière suivante :

$$\alpha^{<t,t'>} = \frac{\exp(e^{<t,t'>})}{\sum_{t''=1}^{T_x} \exp(e^{<t,t''>})}$$

Remarque : la complexité de calcul est quadratique par rapport à T_x .

★ ★ ★