Introduction au Deep LEarning

Réseaux récurrents et texte

Olivier Schwander <olivier.schwander@sorbonne-universite.fr>

Master MIND
Sorbonne Université

2025-2026

Réseau récurrent

Entrée séquentielle

- Suite de symboles
- Évolution temporelle

Pourquoi pas des convolutions ?

- Taille d'entrée variable
- Pas de changement d'échelle possible
- Dépendances à long terme

Pourquoi pas des MLP ?

- Trop de paramètres
- Pas assez de structure dans le réseau
- Ignore la structure de l'entrée

Définition

Équation récurrente

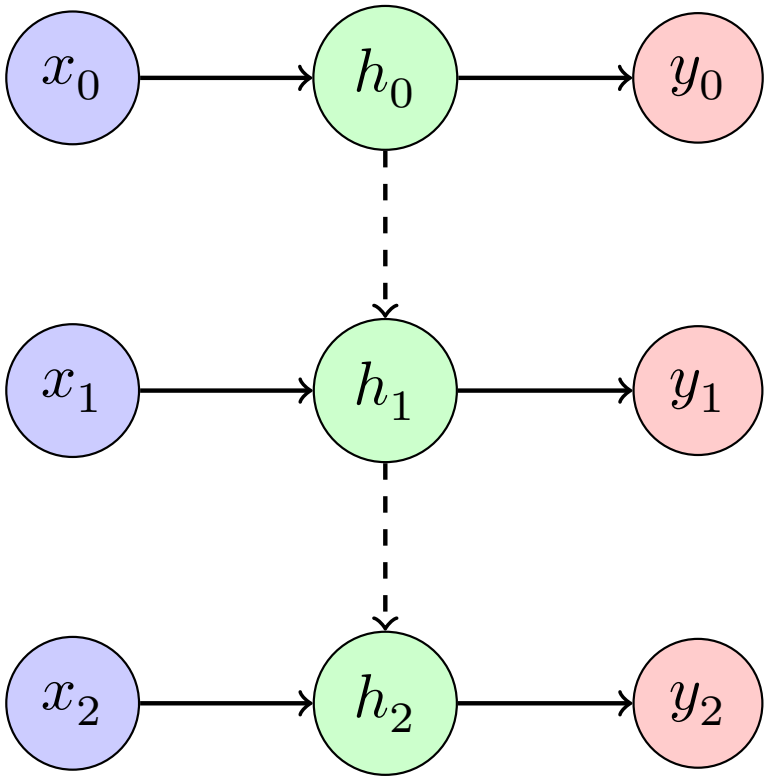
État caché h_t qui évolue avec le temps :

$$h_t = \sigma(W_{ih}x_t + W_{hh}h_{t-1} + b_h)$$

$$y_t = W_{hy}h_t + b_y$$

- $x_t \in \mathbb{R}^{d_{in}}$: entrée au temps t
- $h_t \in \mathbb{R}^{d_h}$: état caché au temps t
- $y_t \in \mathbb{R}^{d_{out}}$: sortie au temps t
- W_{ih}, W_{hh}, W_{hy} : matrices de poids
- σ : fonction d'activation

RNN déroulé



Backpropagation Through Time (BPTT)

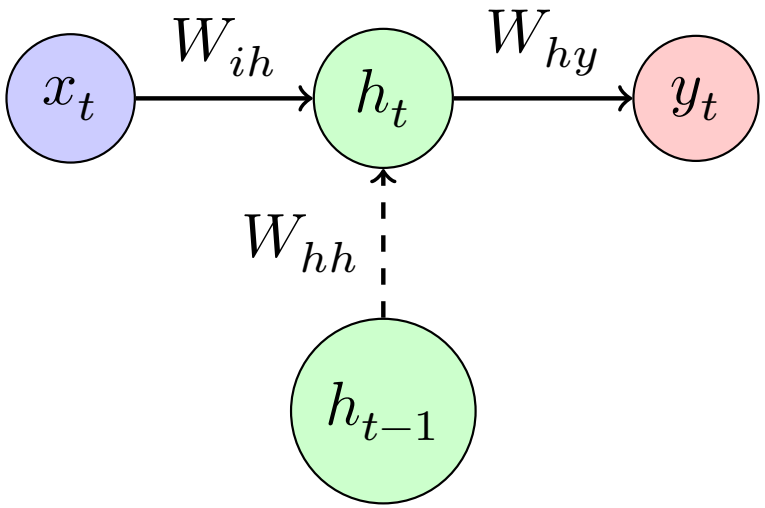
Backprop classique

- Sur le RNN déroulé
- Propagation à travers les pas de temps

Dépendance temporelle

$$\frac{\partial L}{\partial h_t} = \frac{\partial L}{\partial h_T} \prod_{k=t}^{T-1} \frac{\partial h_{k+1}}{\partial h_k}$$

Réseau structuré



Mêmes paramètres à chaque pas de temps

Nombre de paramètres

Partage

- $W_{ih} \in \mathbb{R}^{d_h \times d_{in}}$
- $W_{hh} \in \mathbb{R}^{d_h \times d_h}$
- $W_{hy} \in \mathbb{R}^{d_{out} \times d_h}$
- Total : $d_h(d_{in} + d_h + d_{out})$ (+ les biais)

Avantage

- Indépendant de la longueur de séquence d'entrée
- Apprend des motifs réutilisables à différentes positions

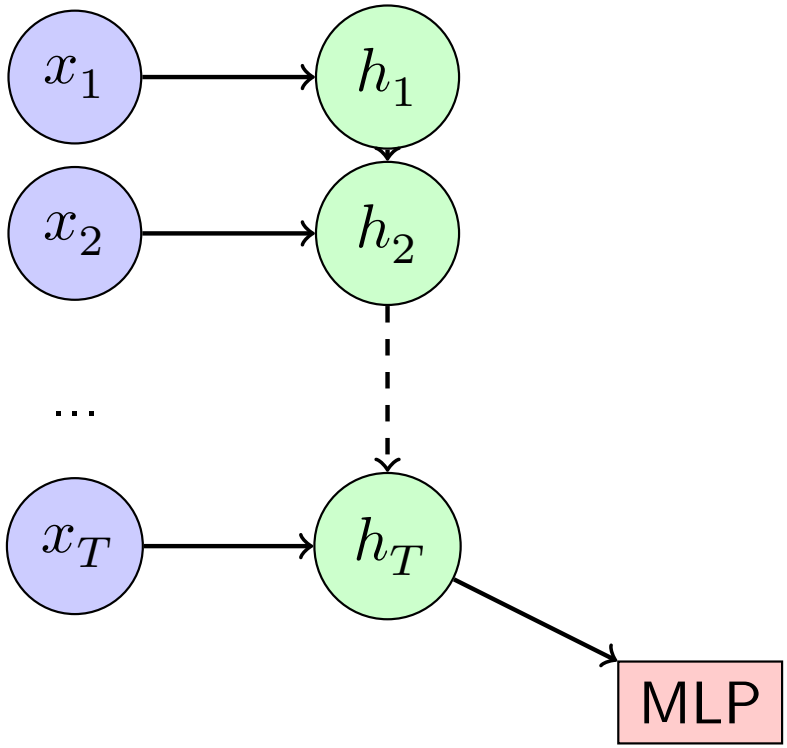
RNN profond

Empilement de couches

- Plusieurs couches RNN empilées
- Sortie branchée sur l'entrée
- Capture des dépendances hiérarchiques

$$h_t^{(l)} = \sigma(W_{ih}^{(l)} h_t^{(l-1)} + W_{hh}^{(l)} h_{t-1}^{(l)} + b_h^{(l)})$$

Classification



En pratique

- Utilisation de l'état caché final h_T
- Suivi d'un MLP puis loss classique

Génération

One-to-many

[au tableau]

Many-to-many

[au tableau]

Résumé

Many-to-One

- Séquence en entrée
- Une seule sortie
- **Exemple** : analyse de sentiments

One-to-Many

- Graine en entrée (plusieurs en général)
- Séquence en sortie
- **Exemple** : génération de texte à partir d'un prompt

Many-to-Many

- Séquence en entrée
- Séquence en sortie
- **Exemples** : traduction, résumé, question-réponse

Vanishing gradient

Rétro-propagation

- Multiplication successive de gradients
- Sur une longue séquence
- Si $|W_{hh}| < 1$: le gradient décroît exponentiellement

Conséquence

- Apprentissage impossible
- Dépendance uniquement sur une fenêtre très courte
- Solution, LSTM et cie

Autre problèmes

Exploding gradient

- Si $|W_{hh}| > 1$, explosion
- Solution, gradient clipping

Écrasement de l'information

- Mise à jour à chaque pas de temps
- Compression dans l'état caché de taille fixe
- Solution, LSTM et cie

Absence de parallèlisme

- Sur la dimension temporelle
- Solution, *transformers*

LSTM - Long Short Term Memory

Motivation

- Mémoriser l'information sur le long terme
- Contrôler explicitement ce qu'on garde et ce qu'on oublie
- Éviter le vanishing gradient

Avantages

- Meilleur apprentissage : chemin additif pour la backprop
- Constant error carousel : pas de vanishing gradient
- Gestion des dépendance à long terme

Architecture

Portes

- **Forget gate** $f_t = \sigma(W_f x_t + U_f c_{t-1} + b_f)$
- **Input gate** $i_t = \sigma(W_i x_t + U_i c_{t-1} + b_i)$
- **Output gate** $o_t = \sigma(W_o x_t + U_o c_{t-1} + b_o)$

États

- **État de cellule**

$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W_c x_t + U_c h_t + b_c)$$

- **État caché** $h_t = o_t \odot \tanh(c_t)$

Interprétation

Rétro-propagation

- Pas seulement le chemin multiplicatif
- Également une chemin additif
- Penser au ResNet !

Dictionnaire dérivable

f_t	i_t	o_t	Comportement
~ 0	~ 1		Reset
~ 1	~ 0		Pas de modification
~ 1	~ 1		Mise à jour
		~ 0	Pas de lecture
		~ 1	Lecture

Bi-RNN / Bi-LSTM

Motivation

- Exploiter l'information du "futur"
- Utile quand on connait globalement la séquence
- Pas quand les symboles arrivent l'un après l'autre

Architecture

- Un réseau agit sur $1 \dots T$
- Un réseau agit sur $T \dots 1$

Application

- Tâches où le contexte complet est important
- Compréhension
- Mais pas génération à partir d'un prompt

One-Hot encoding

Vecteurs binaires

- Dimension : taille du vocabulaire $|V|$

$$w_i = [0, 0, ..., 1, ..., 0]^T$$

Vocabulaire : {“chat”, “chien”, “souris”, “maison”}

```
"chat"    → [1, 0, 0, 0]
"chien"   → [0, 1, 0, 0]
"souris"  → [0, 0, 1, 0]
"maison"  → [0, 0, 0, 1]
```

Problème

- Dimension gigantesque
- Pas de sémantique

Matrice d'embeddings

Représentation dense

- Représenter chaque mot par un vecteur dans un espace vectoriel
- Dimension réduite : entre 50 et 500
- Représentation apprise pendant l'entraînement

Embedding layer

- Chaque ligne de la matrice correspond à la représentation du mot

$$E \in \mathbb{R}^{|V| \times d_{emb}}$$

Objectifs

- Mots sémantiquement proches $\rightarrow$ vecteurs proches
- Capture de similarité syntaxique et sémantique

Apprentissage des embeddings

Architecture

- Couche intiale : embeddings
- Couches récurentes
- Couches denses

Rétro-propagation

- À travers une seule ligne de la matrice
- Apprentissage guidé par la tâche

Embeddings pré-entraînés

From scratch

- En pratique : pas assez de données pour les tâches complexes

Modèles de mot

- Pré-entraînement sur de gros corpus
- Auto-supervisé : pas de label externe (prédiction de mot masqué par exemple)

Nouvelle architecture

- Couche intiale : word2vec
- Couches récurentes
- Couches denses

Modèles de fondation

Apprentissage de représentation

- Plus besoin de feature engineering
- Représentation apprise from scratch
- Et guidée par la tâche

Modèles réutilisables

- CNN + Finetuning
- Modèles de mots utilisables directement : Word2vec
- Modèles de langue : Large Language Models

Vers le zero-shot et le few-shots

Zero-shot learning

- Tâche sans exemples d'entraînement
- Utiliser les connaissances générales du modèle
- Prompting : formuler la tâche en langage naturel

Few-shot learning

- Très faible nombre d'exemples seulement (quelques dizaines max)
- Learning in-context : exemples fournis dans le prompt
- Adaptation sans ré-entraînement
- Généralisation à partir des quelques exemples

Conclusion

Réseaux récurrents

Avantages

- Traitement naturel de séquences de longueur variable
- Capture de dépendances séquentielles
- Paramètres partagés (efficacité)

Limitations

- Vanishing/exploding gradient
- Difficile à paralléliser (séquentiel)
- Dépendances à long terme limitées