



Traduction automatique

TA neuronale

Enzo Doyen

enzo.doyen@unistra.fr

2025-09-19

Plan

- I.** Rappels sur les réseaux de neurones
- II.** Représentations des mots et plongements lexicaux
- III.** Réseaux de neurones récurrents
- IV.** Mise en pratique



I. Rappels sur les réseaux de neurones

Réseaux de neurones

- ♦ Fonction complexe, composée d'unités nommées « neurones ».
- ♦ Chaque neurone reçoit :
 - en entrée (x) : des vecteurs ou matrices ;
 - en sortie (y) : des vecteurs, matrices ou scalaires.

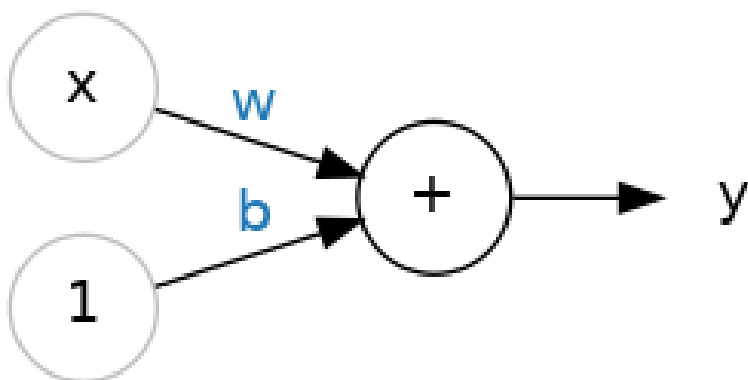
Chaque unité a un poids correspondant sous forme de vecteur (w) et un biais (b), qui sont des paramètres à apprendre.

On applique une fonction d'activation non linéaire (sigmoid, tanh, ReLU) g pour déterminer la sortie :

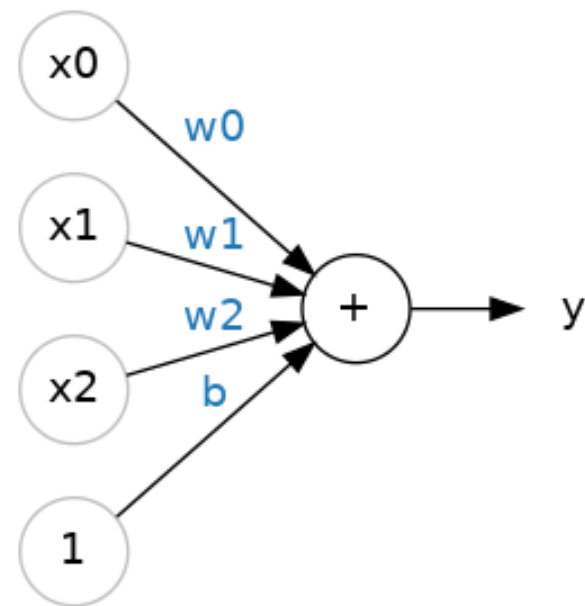
$$y = g(wx + b)$$

Structure d'un neurone

Source : <https://www.kaggle.com/code/ryanholbrook/a-single-neuron>





Neurone simple avec une entrée



Neurone simple avec 3 entrées

Réseaux de neurones : fonctions d'activation non linéaires

Pourquoi utiliser des fonctions non linéaires ?

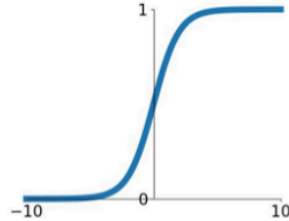
- ♦ Sans elles, le réseau serait équivalent à une simple combinaison linéaire des entrées.
- ♦ Les fonctions non linéaires permettent de capturer les relations de non-linéarité.
- ♦ Comparer¹ : sans activation  et avec activation .
- ♦ Pour approfondir : <https://stackoverflow.com/questions/9782071/why-must-a-nonlinear-activation-function-be-used-in-a-backpropagation-neural-net>

¹Source : <https://www.dailydoseofds.com/why-is-relu-a-non-linear-activation-function/>
Essayez vous-même : <https://playground.tensorflow.org/>

Activation Functions

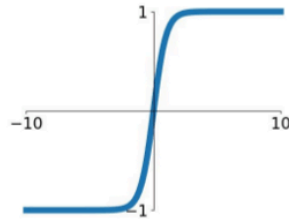
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



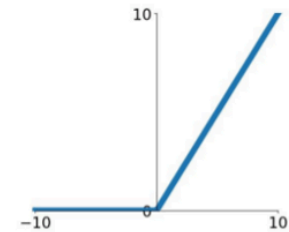
tanh

$$\tanh(x)$$



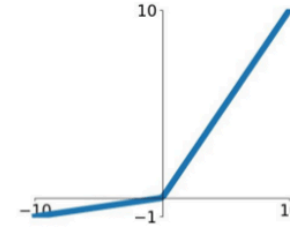
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

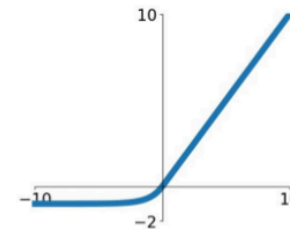


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Source : <https://medium.com/machine-learning-world/how-to-debug-neural-networks-manual>

Réseau de neurones à action directe (*feedforward neural network*)

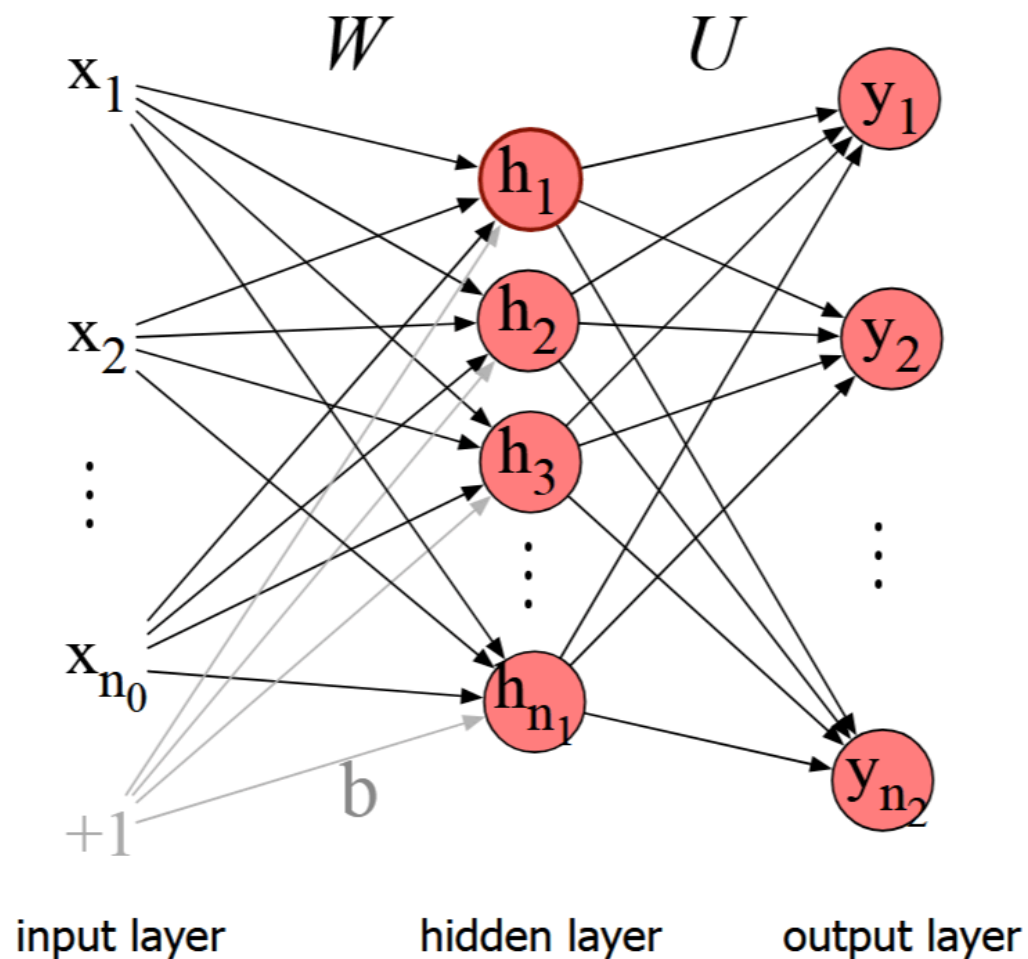
- ♦ Réseau multicouches avec des couches d'entrées, cachées et de sortie.
- ♦ Chaque couche est composée de neurones.
- ♦ Les neurones de chaque couche sont entièrement connectés aux neurones de la couche suivante : chaque neurone de chaque couche prend comme entrée les sorties de tous les neurones de la couche précédente.

Réseau de neurones à action directe (*feedforward neural network*) : couches cachées

Ce sont les couches cachées (une couche est représentée sous h) qui permettent de capturer les relations non linéaires.

$$h = g(Wx + b)$$

où W représente la matrice de poids (contient tous les poids w de la couche).



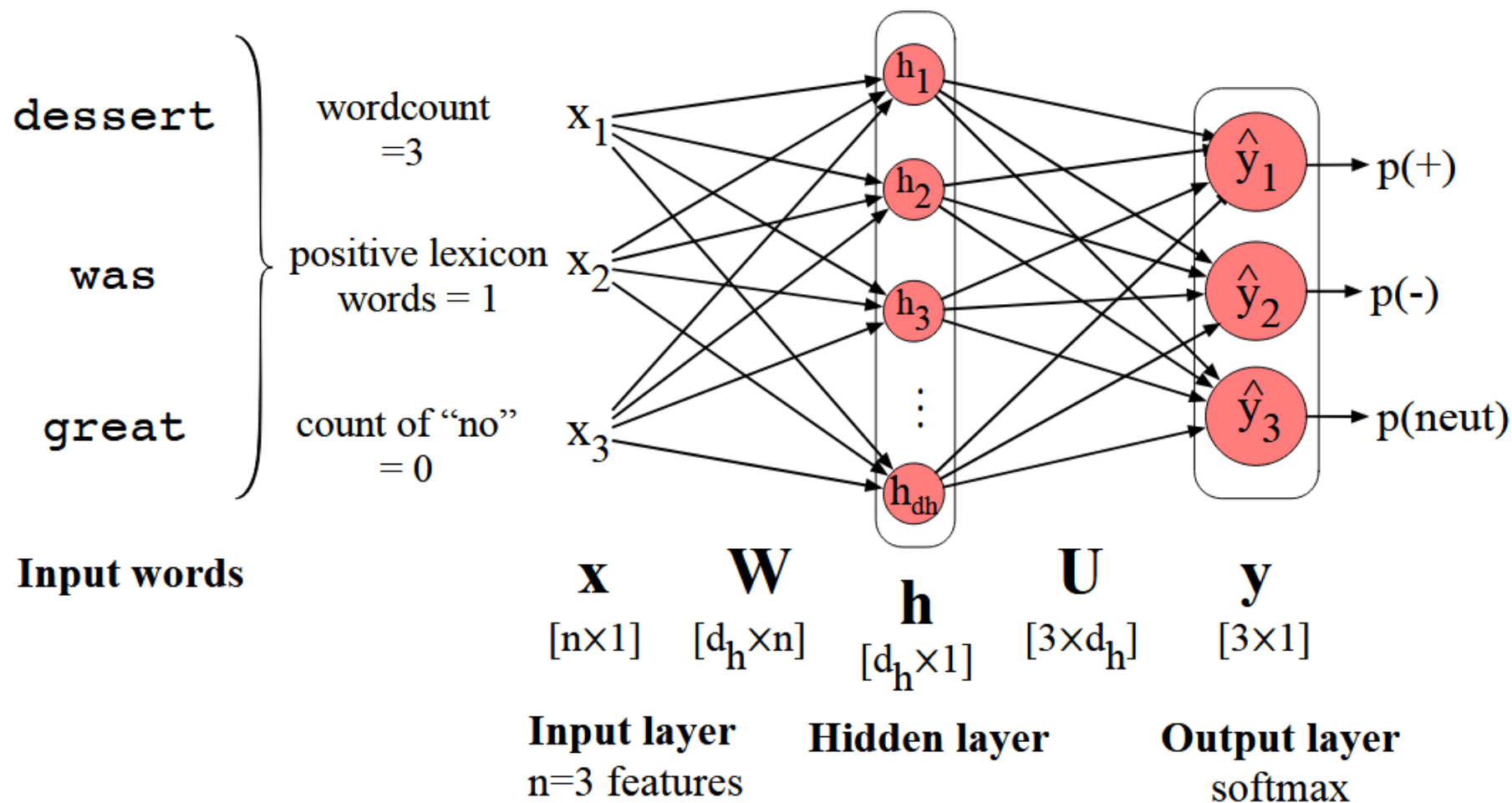
Source : **Jurafsky et Martin (2025)**

Réseau de neurones à action directe (*feedforward neural network*) : softmax

- ♦ Application d'une fonction softmax à la sortie pour obtenir un vecteur de probabilités (qui s'additionne à 1).

$$z = [0.6, 1.1, -1.5, 1.2, 3.2, -1.1]$$

$$\text{softmax}(z) = [0.055, 0.090, 0.0067, 0.10, 0.74, 0.010]$$



Exemple pour classification de sentiments | Source : **Jurafsky et Martin (2025)**

Entraînement d'un réseau

- ♦ Initialisation des poids au hasard.
- ♦ **Étape forward** : on calcule la sortie du réseau en appliquant les fonctions d'activation à l'entrée.
- ♦ **Étape backward** : on ajuste les gradients des poids et biais par rapport à la fonction de perte en utilisant la rétropropagation.
 - Différentes méthodes pour calculer les paramètres qui minimisent l'erreur : SGD (Stochastic Gradient Descent), Adam, RMSprop, etc.
- ♦ On répète ces étapes jusqu'à un certain point d'arrêt (par exemple, un nombre d'itérations ou une convergence de la fonction de perte).

Entraînement d'un réseau : étape *forward*

Exemple avec un réseau à 2 couches cachées h_0 et h_1 , où x est le vecteur d'entrée, b le biais, W_i le vecteur de poids de la couche i et g_i la fonction d'activation de la couche i . y est la sortie du réseau.

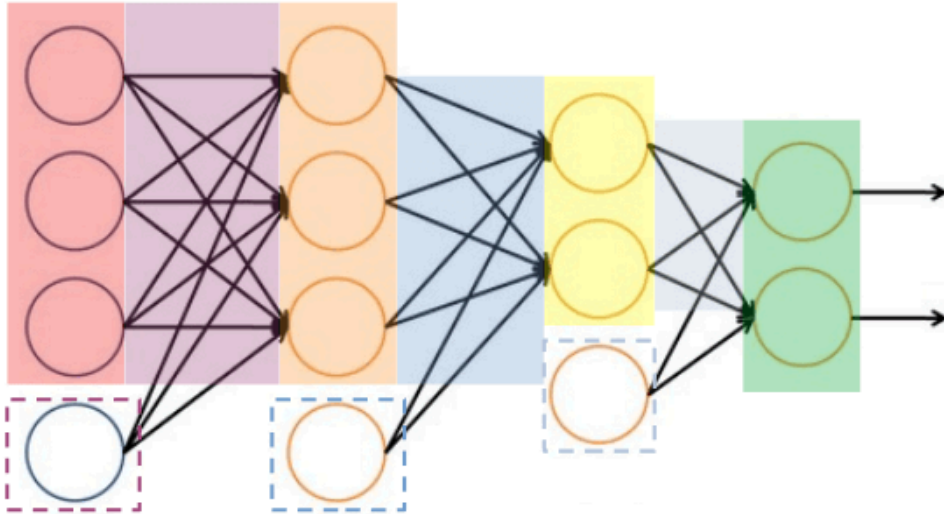
$$h_0 = g_0(W_0x + b_0)$$

$$h_1 = g_1(W_1h_0 + b_1)$$

$$y = g_2(W_2h_1 + b_2)$$

$$\text{Généralisation : } h_i = g_i(W_i h_{i-1} + b_i)$$

Entraînement d'un réseau : étape *forward*

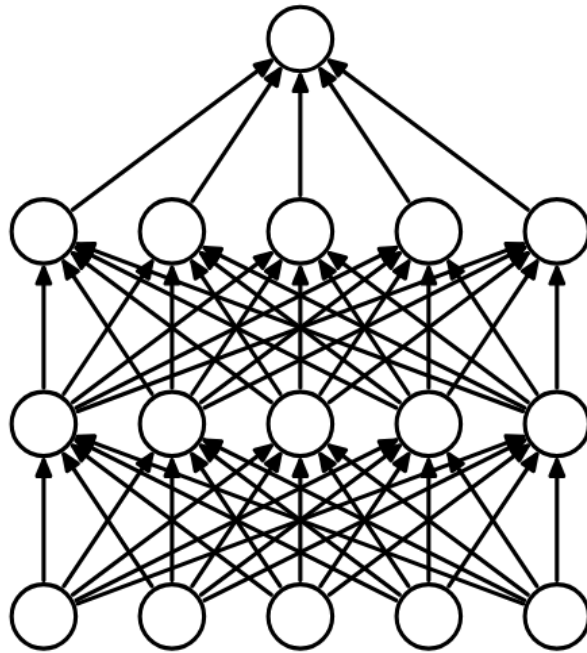


$$y = f(W_2 h_1 + b_2)$$

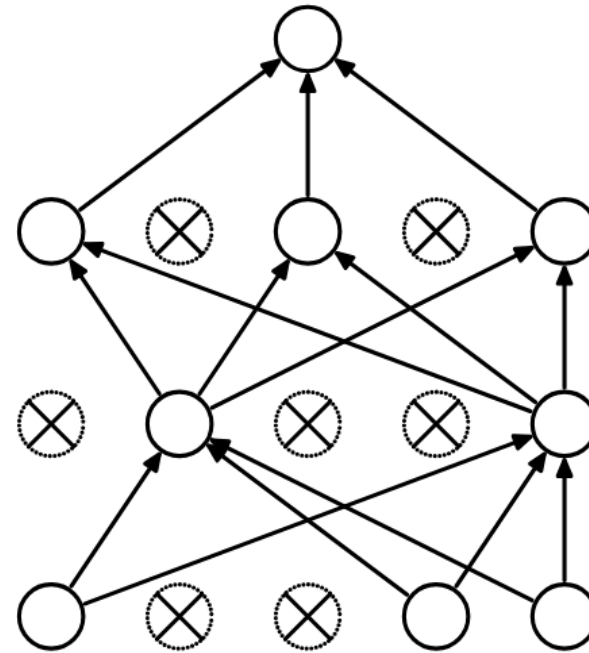
$$h_1 = f(W_1 h_0 + b_1)$$

$$h_0 = f(W_0 x + b_0)$$

Entraînement d'un réseau : dropout



(a) Standard Neural Net



(b) After applying dropout.

Source : **Srivastava et al. (2014)**

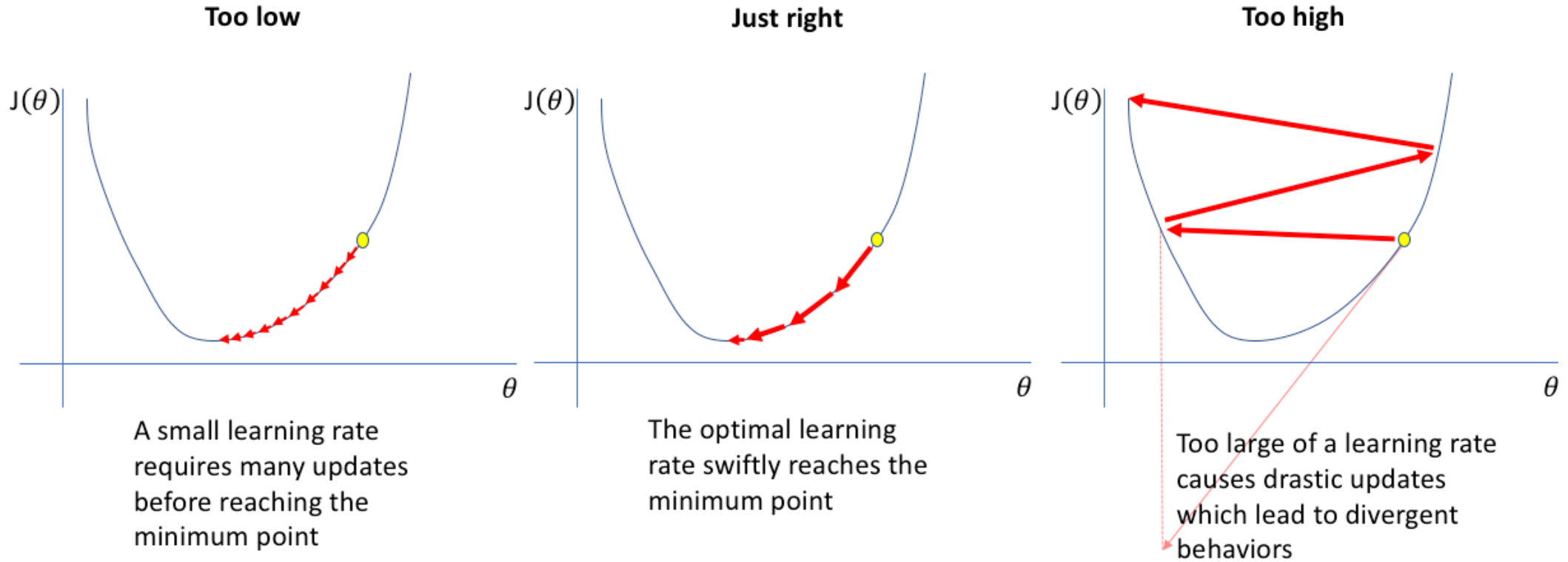
Entraînement d'un réseau : *backpropagation* (rétropropagation du gradient)

- ♦ Le **gradient** représente le taux de changement de la fonction de perte par rapport aux poids du réseau.
- ♦ Il sert à calculer la direction dans laquelle la fonction de perte diminue le plus rapidement, dans le but de minimiser l'erreur.

Entraînement d'un réseau : *backpropagation* (rétropropagation du gradient) et *learning rate*

- ♦ **Learning rate** : coefficient avec lequel on multiplie l'erreur lors de la rétropropagation.
- ♦ Impacte l'ampleur avec laquelle les paramètres sont modifiés à chaque itération.
- ♦ Si trop grand, risque de ne pas converger ; si trop petit, risque de converger trop lentement.
- ♦ Demande un équilibre entre vitesse d'entraînement et qualité des résultats

Entraînement d'un réseau : *backpropagation* (rétropropagation du gradient) et *learning rate*



Source : <https://www.jeremyjordan.me/nn-learning-rate/>

Entraînement d'un réseau : exploitation du corpus

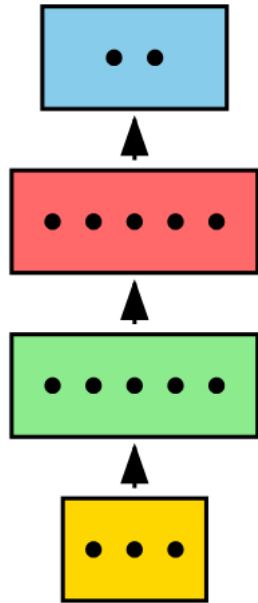
- ♦ Pour l'entraînement, le corpus est mélangé en **mini-batches** (petits lots d'exemples).
 - ♦ L'erreur selon la fonction de perte est calculée sur chaque mini-batch, et les poids sont mis à jour en conséquence.
-
- ♦ **Epoch** : une itération complète sur l'ensemble du corpus.
 - ♦ Après chaque epoch, le corpus est à nouveau mélangé et divisé en mini-batches.

Entraînement d'un réseau : *backpropagation* (rétropropagation du gradient)

- ♦ Actualisation des poids W selon le gradient ∇ de la fonction de perte J avec un *learning rate* η .
- ♦ Entraînement avec les entrées x , de 0 à K (taille du minibatch = K).

$$W = W - \eta \cdot \nabla J(x_{[0:K]}; W)$$

Entraînement d'un réseau : résumé



- Initialisation avec paramètres aléatoires : W
- À chaque epoch
 - Mélange des données d'entraînement
 - À chaque mini-batch (ensemble de K exemples)
 - Calcul de la perte (forward)
 - Calcul du gradient de la perte (backward)
 - Mise à jour de W / b :
(learning rate η) $W = W - \eta \nabla J_{x[0:K]}(W)$
(fonc. de cout J)
 - Mesure de l'exactitude train/dev
- Continuer jusqu'à convergence / fin du temps fixé / arrêt augmentation exactitude dev

Entraînement d'un réseau : résumé des hyperparamètres

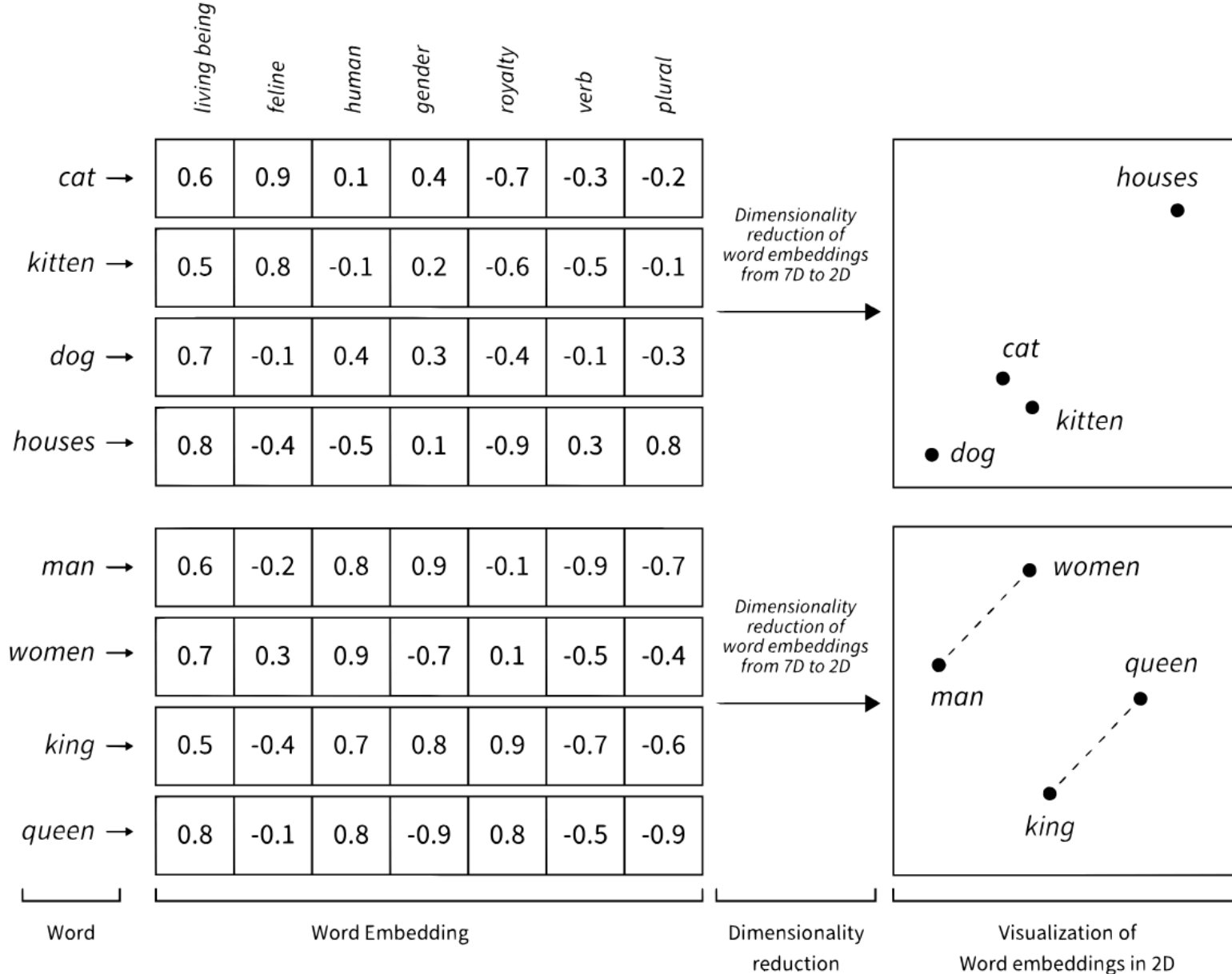
- ♦ Topologie du réseau : nombre de couches, nombre de neurones par couche, type de fonction d'activation.
- ♦ **Algorithme d'optimisation** : méthode pour ajuster les poids et biais du réseau (SGD, Adam, RMSprop, etc.).
- ♦ **Dropout** : technique qui désactive aléatoirement certains neurones pendant l'entraînement pour éviter le surapprentissage.
- ♦ **Learning rate** : contrôle la vitesse d'apprentissage du modèle (c'est-à-dire à quel point le modèle ajuste ses paramètres à chaque itération de l'algorithme d'optimisation).
- ♦ **Batch size** : nombre d'exemples traités avant la mise à jour des poids.
- ♦ ...



II. Représentations des mots et plongements lexicaux

Plongements lexicaux

En TAL neuronal, l'unité de base du traitement n'est plus le token, mais l'**embedding** (ou **plongement lexical**) : un vecteur dense qui représente le sens d'un mot dans un espace vectoriel selon ses contextes d'apparition.



Représentation des mots

- ♦ Nécessaire de représenter les mots par des vecteurs pour les utiliser dans les réseaux de neurones.
- ♦ Comment faire ?

Représentation des mots : *one-hot encoding*

- ♦ Représentation binaire de chaque mot.
- ♦ Chaque mot est représenté par un vecteur de taille égale au vocabulaire, avec une seule valeur à 1 et le reste à 0.

Soit un corpus :

$$V = \{\text{le, chat, chien, dort}\}$$

| Mot | Vecteur |
|-------|--------------|
| le | [1, 0, 0, 0] |
| chat | [0, 1, 0, 0] |
| chien | [0, 0, 1, 0] |
| dort | [0, 0, 0, 1] |

Représentation des mots : *one-hot encoding*

- ♦ Représentations très éparses (plupart des valeurs égales à 0), peu informatives.
- ♦ Ne capture pas les relations sémantiques entre les mots.

Représentation des mots : matrice de cooccurrence

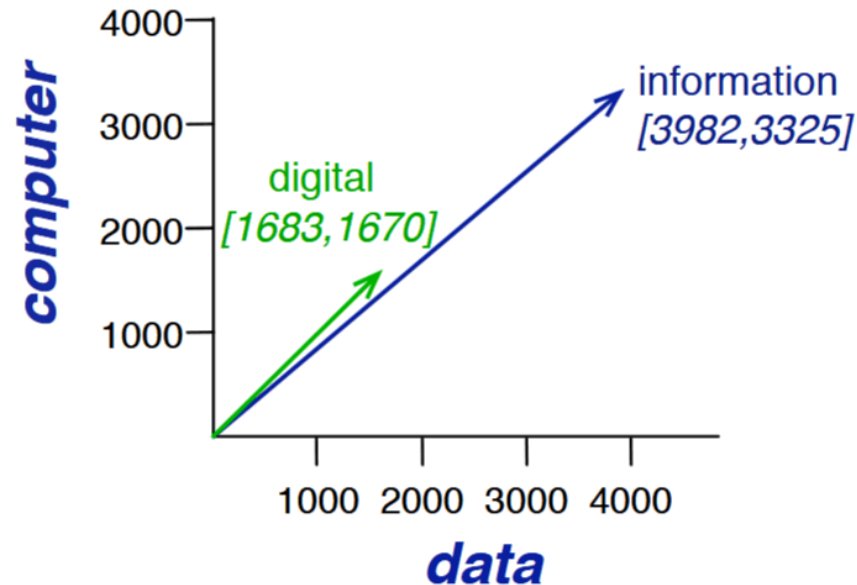
- Représentation basée sur les cooccurrences des mots dans un contexte donné.
- On construit une matrice de cooccurrence où chaque ligne représente un mot et chaque colonne un contexte (par exemple, les mots voisins dans une fenêtre de taille fixe).

| | aardvark | ... | computer | data | result | pie | sugar | ... |
|-------------|----------|-----|----------|------|--------|-----|-------|-----|
| cherry | 0 | ... | 2 | 8 | 9 | 442 | 25 | ... |
| strawberry | 0 | ... | 0 | 0 | 1 | 60 | 19 | ... |
| digital | 0 | ... | 1670 | 1683 | 85 | 5 | 4 | ... |
| information | 0 | ... | 3325 | 3982 | 378 | 5 | 13 | ... |

Source : **Jurafsky et Martin (2025)**

Représentation des mots : matrice de cooccurrence

- ♦ Possibilité de comparer la similarité entre les mots en utilisant des mesures de similarité.

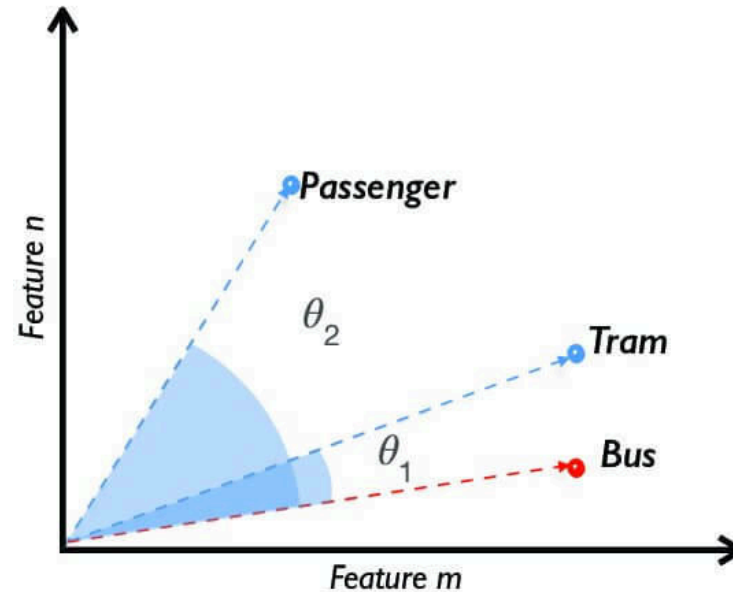


Source : Jurafsky et Martin (2025)

Représentation des mots : matrice de cooccurrence

- ♦ Possibilité de comparer la similarité entre les mots.

$$\text{sim}(a, b) = \cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|}$$



Source : **Kalwar et al. (2023)**

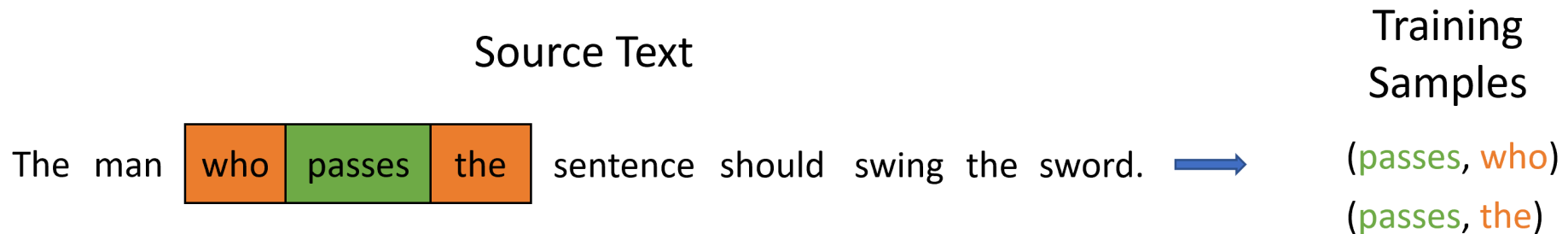
Réduction de la dimensionnalité

- ♦ Les matrices de cooccurrence ne résolvent pas le problème de la représentation éparses.
- ♦ Pour résoudre ce problème, on peut utiliser des techniques de **réduction de la dimensionnalité** (SVD, PCA, t-SNE, UMAP, etc.) pour obtenir des plongements lexicaux denses, qui contiennent davantage d'informations.

Représentation des mots basée sur le contexte :

modèle *skip-gram*

- ♦ Modèle neuronal entraîné à prédire les mots autour d'un mot cible.
- ♦ La taille du contexte est définie par une fenêtre glissante autour du mot cible (hyperparamètre) ; ici, 2.

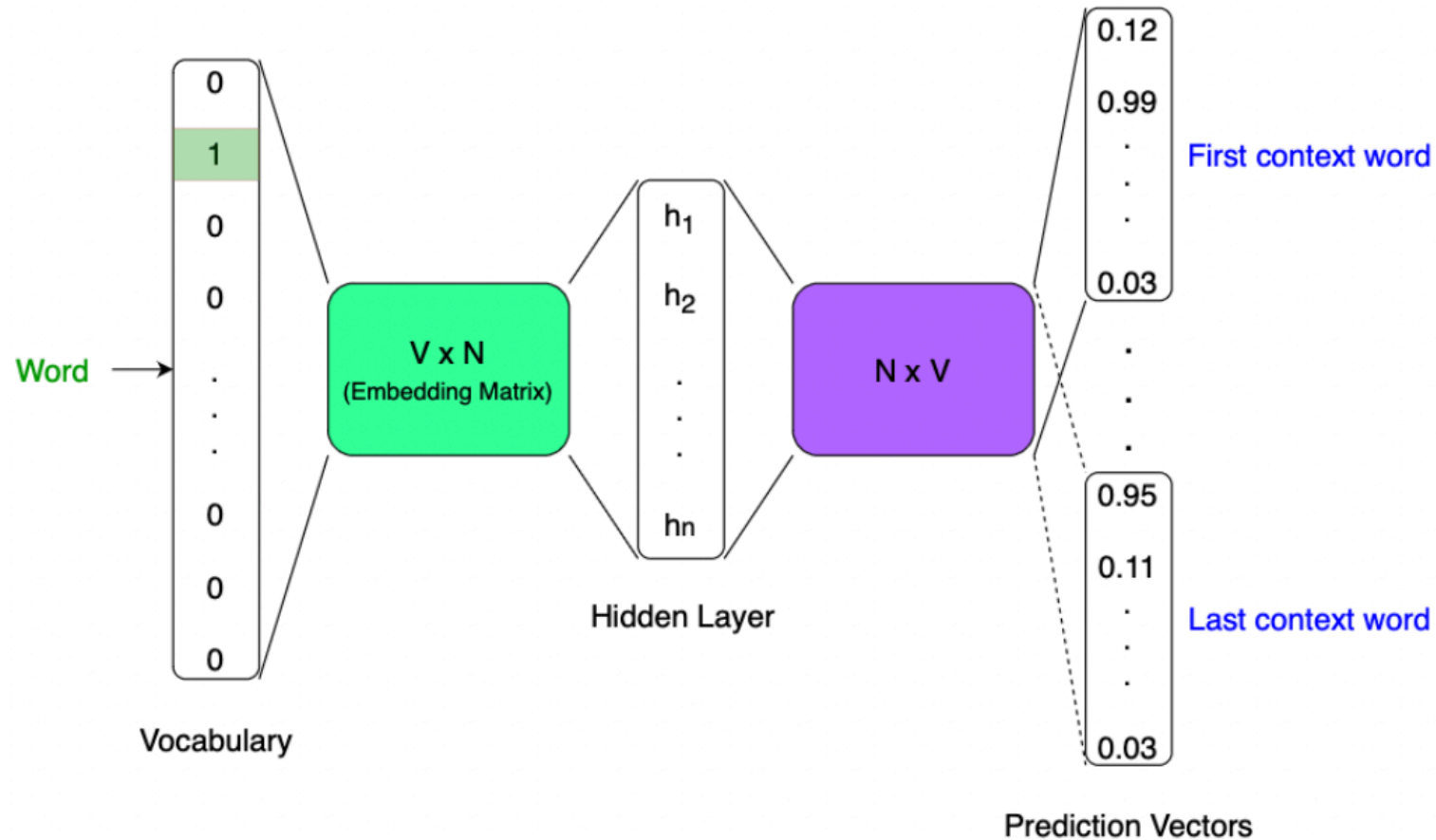


Source : https://aegis4048.github.io/demystifying_neural_network_in_skip_gram_language_modeling

Représentation des mots basée sur le contexte :

modèle *skip-gram*

- ♦ Les plongements lexicaux sont d'abord initialisés au hasard.
- ♦ Pour chaque mot cible, des paires (cible, contexte) sont générées.
- ♦ Le modèle est entraîné à maximiser la probabilité de prédire les mots du contexte à partir du mot cible.
- ♦ *Negative sampling* : on essaie de maximiser le produit scalaire entre les mots qui partagent des contextes, et le minimiser avec des mots « négatifs » (hors du contexte).



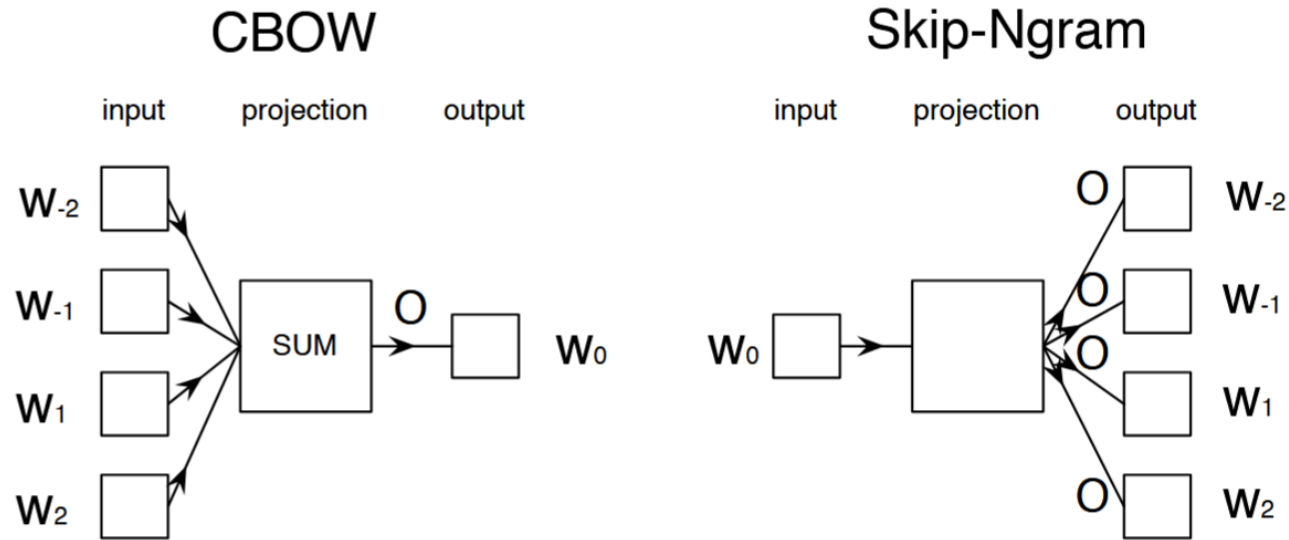
Source : <https://towardsdatascience.com/implementing-word2vec-in-pytorch-from-the-ground-up>

Représentation des mots basée sur le contexte :

modèle *CBOW*

- ♦ Modèle très similaire au modèle *skip-gram*, mais au lieu de prédire les mots du contexte à partir du mot cible, il prédit le mot cible à partir des mots du contexte (tâche *fill-in-the-blank*).

Représentation des mots basée sur le contexte : modèle *CBOW*

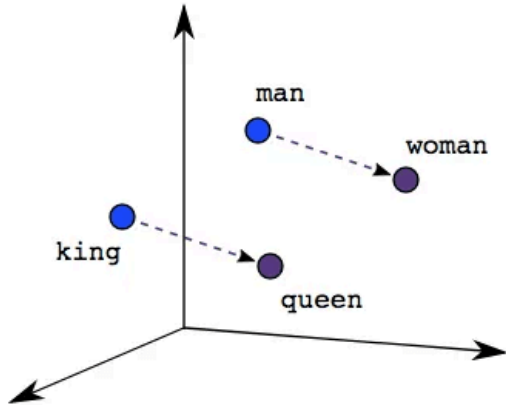


Source : **Ling et al. (2015)**

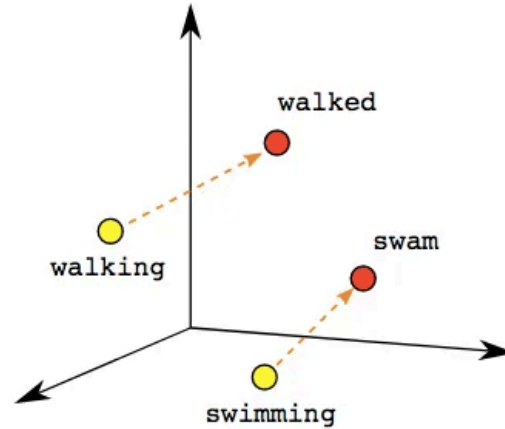
Représentation des mots basée sur le contexte

- ♦ Architectures initialement intégrées dans Word2vec (**Mikolov et al., 2013**).
- ♦ Autres implémentations : GloVe (**Pennington et al., 2014**), fastText (**Bojanowski et al., 2016**).

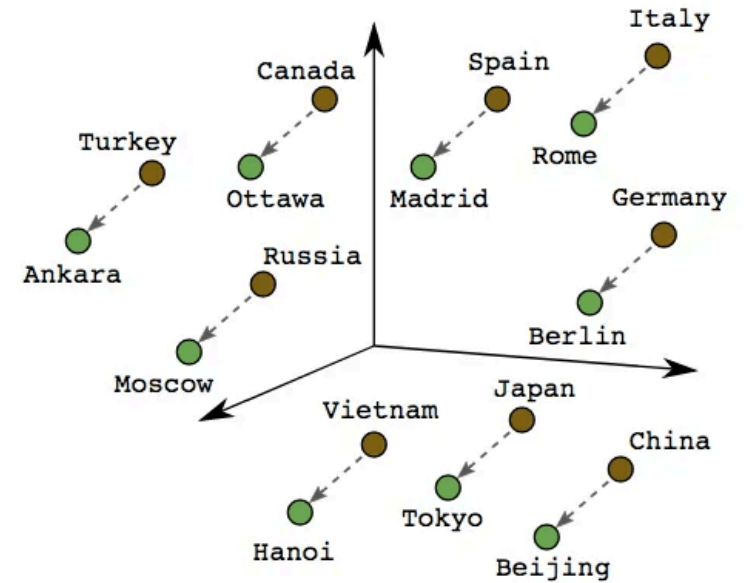
Plongements lexicaux



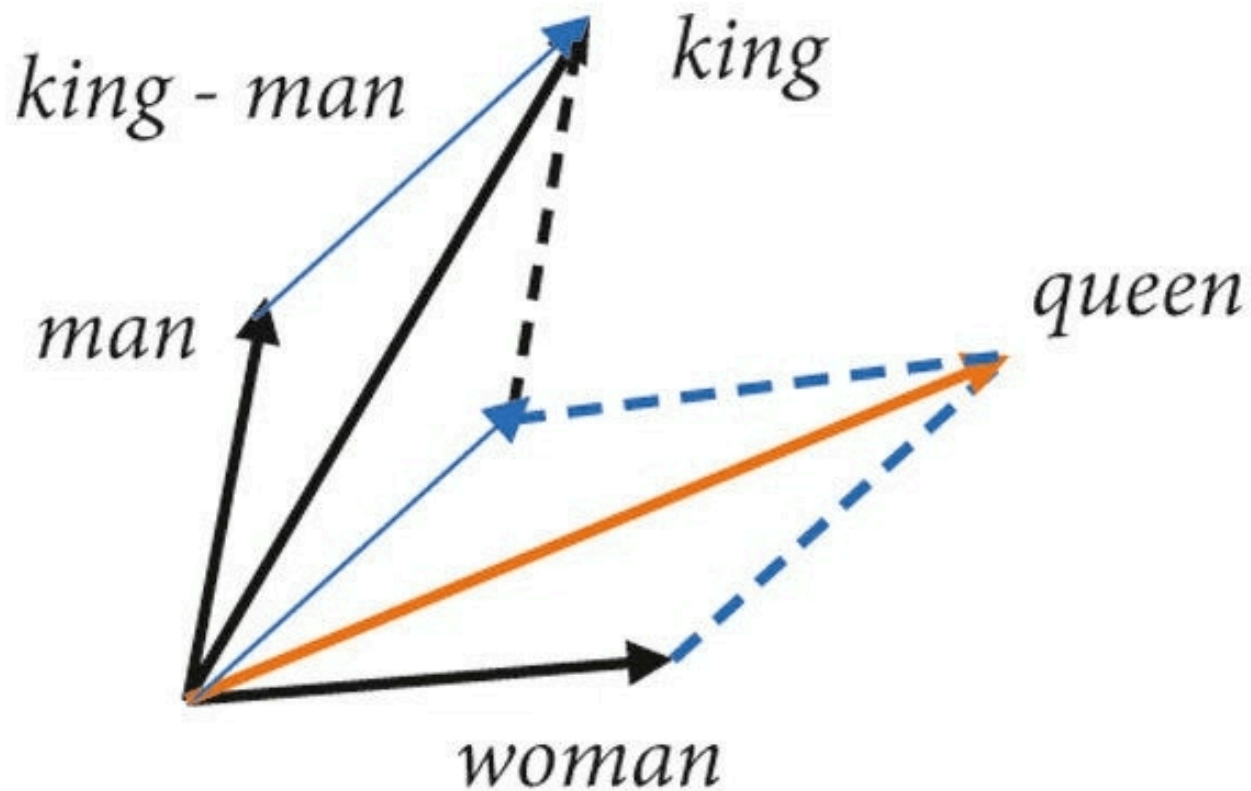
Male-Female



Verb Tense



Country-Capital



$$\text{king} - \text{man} + \text{woman} \approx \text{queen}$$

Analogies avec plongements lexicaux | Source : [https://dev.to/mshojaei77/](https://dev.to/mshojaei77/words-to-vectors-a-gentle-introduction-to-word-embeddings-4mia)



III. Réseaux de neurones récurrents

Réseaux de neurones récurrents (RNN)

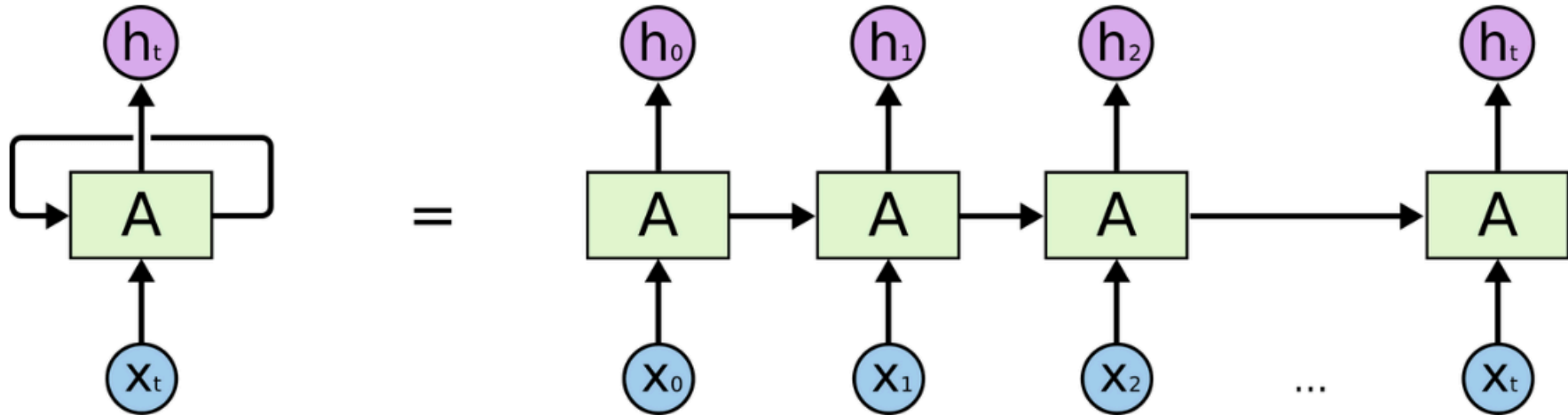
Nous avons vu jusqu'à présent que les réseaux de neurones étaient capables de créer des représentations des mots et de les prédire selon un contexte...

Mais qu'en est-il des phrases ? Et comment passer d'une langue à une autre ?

Réseaux de neurones récurrents (RNN)

- ♦ **RNN** : type de modèle déjà utilisé pour la reconnaissance vocale ou la reconnaissance d'écriture manuscrite.
- ♦ Architecture qui permet d'utiliser les sorties précédentes comme entrées pour les neurones suivants (d'où l'adjectif « récurrent »).
- ♦ Dispose d'« états cachés » (*hidden states*, h) qui permettent de conserver une mémoire des entrées précédentes.
- ♦ Ainsi, permet de modéliser des séquences de données (comme des phrases).

Réseaux de neurones récurrents



Source : <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

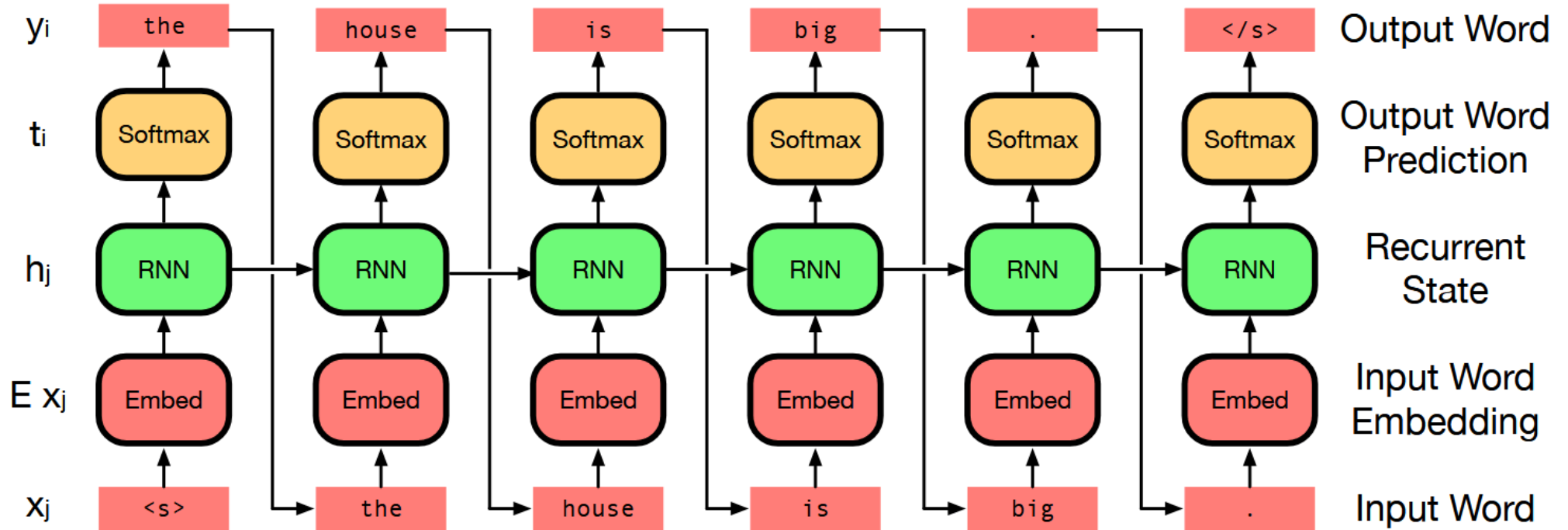
Réseaux de neurones récurrents

Les RNN sont considérés comme des **modèles de langue** : à partir d'une séquence de mots, on veut savoir quel serait le possible mot suivant.

$$P(w_1 \dots w_n) = P(w_1)P(w_2 \mid w_1) \dots P(w_n \mid w_1, \dots, w_{n-1})$$

Cela se fait sur la base d'un entraînement sur de grands corpus textuels.

Réseaux de neurones récurrents

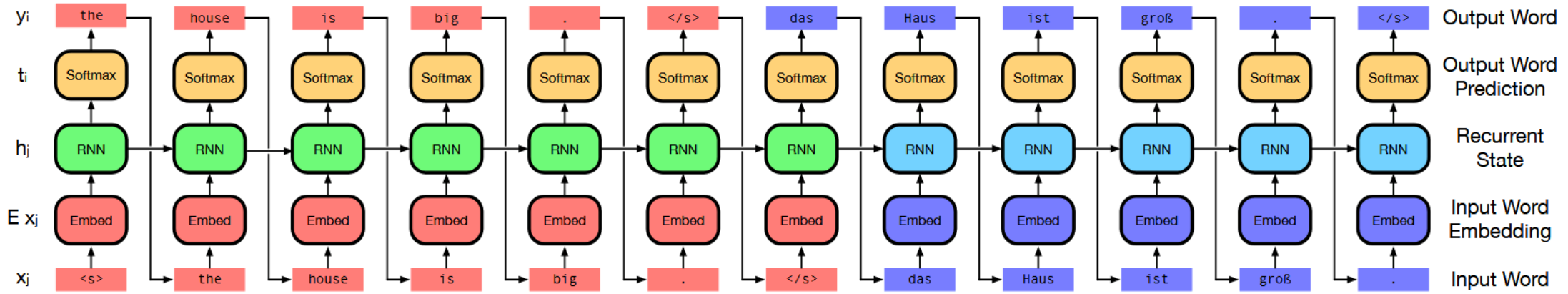


Source : Philipp Koehn

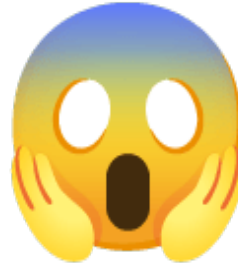
Réseaux de neurones récurrents

- ♦ Premières utilisations pour la traduction automatique par **Sutskever et al. (2014)** et **Cho et al. (2014)**
- ♦ Un RNN n'est pas suffisant ! Il faut en ajouter un deuxième pour modéliser la deuxième langue...

Réseaux de neurones récurrents



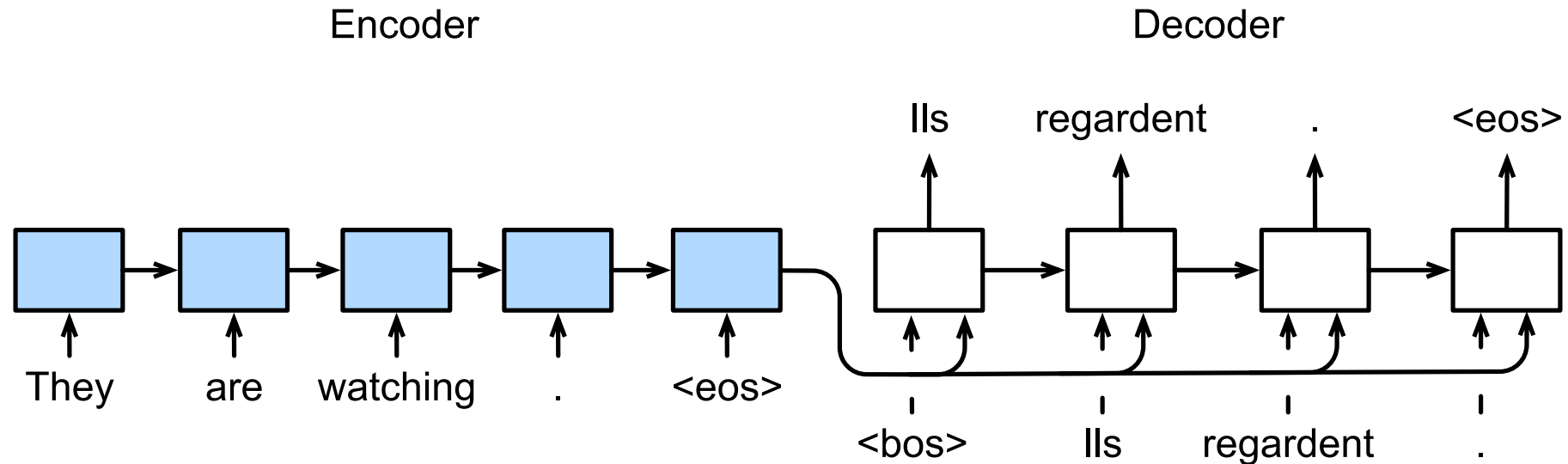
Source : Philipp Koehn



Réseaux de neurones récurrents


- ♦ Premières utilisations pour la traduction automatique par **Sutskever et al. (2014)** et **Cho et al. (2014)**
- ♦ Un RNN n'est pas suffisant ! Il faut en ajouter un deuxième pour modéliser la deuxième langue...
- ♦ Architecture **encodeur-décodeur** (ou **sequence-to-sequence** [Seq2Seq]) :
 - Premier RNN « **encodeur** » : prend la phrase source et la transforme en un vecteur de taille fixe. Pas de sortie.
 - Deuxième RNN « **décodeur** » : prend le dernier état caché de l'encodeur (en langue source), et génère la phrase cible (traduction), mot par mot.
 - Entraînement sur corpus parallèles.

Architecture encodeur-décodeur



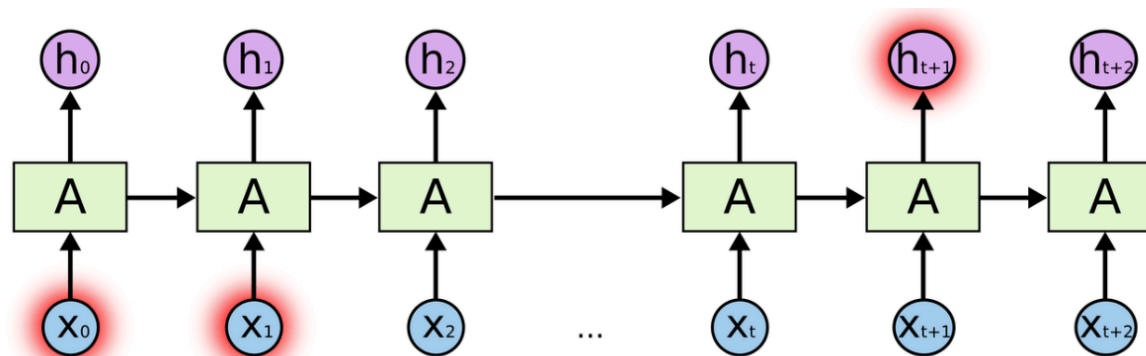
Architecture Seq2Seq | Source : https://d2l.ai/chapter_recurrent-modern/seq2seq.html 

Architecture encodeur-décodeur

- ♦ Premier RNN « **encodeur** » : prend la phrase source et la transforme en un vecteur de taille fixe. Pas de sortie.
- ♦ Deuxième RNN « **décodeur** » : prend le dernier état caché de l'encodeur (en langue source), et génère la phrase cible (traduction), mot par mot.
- ♦ Entraînement sur corpus parallèles.
- ♦ **Teacher forcing**  : pendant l'entraînement, on utilise la sortie attendue (d'après le corpus) comme entrée pour le décodeur, plutôt que la sortie du décodeur lui-même. Cela permet d'accélérer l'entraînement et de stabiliser les gradients en ayant des entrées correctes (qui correspondent au corpus d'entraînement).

Architecture encodeur-décodeur : problème des RNN traditionnels

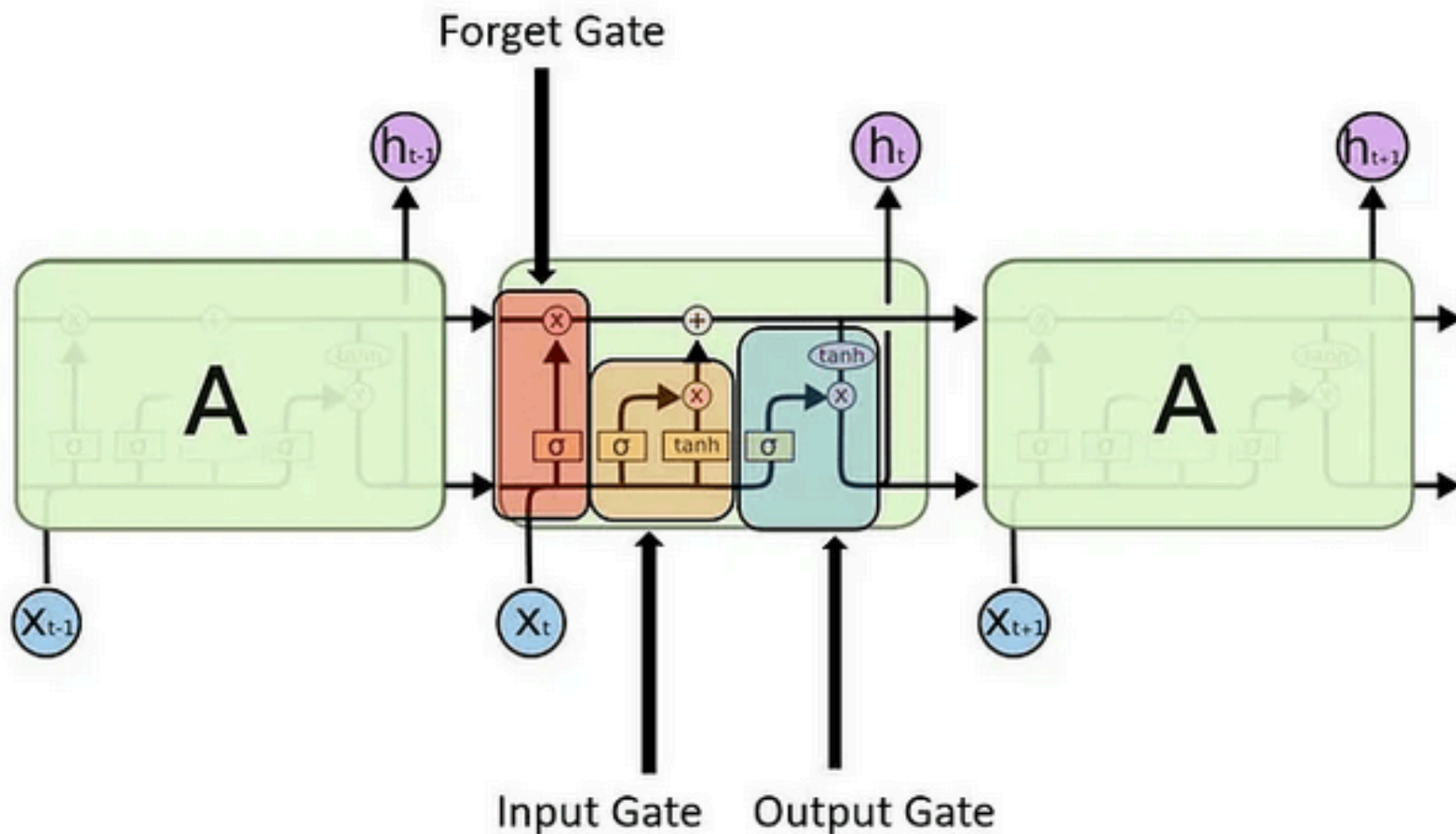
- ♦ Risque d'oubli de l'information lorsqu'elle est trop loin.
 - Au bout d'un certain temps, le modèle ne parvient plus à capturer les informations pertinentes vues en début de séquence (**problème du *vanishing gradient***).



Source : <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

Architecture encodeur-décodeur : LSTM

- ♦ Nouveau type de RNN : **Long Short-Term Memory** (LSTM).
- ♦ Ajoute un mécanisme de « mémoire à court terme » pour conserver les informations pertinentes sur de longues séquences.
- ♦ Contient trois différentes « gates » :
 - **Forget gate** : décide quelles informations de l'état caché précédent doivent être oubliées.
 - **Input gate** : décide quelles nouvelles informations doivent être ajoutées à l'état caché.
 - **Output gate** : décide quelles informations de l'état caché doivent être utilisées pour la sortie.



Source : <https://aditi-mittal.medium.com/understanding-rnn-and-lstm>

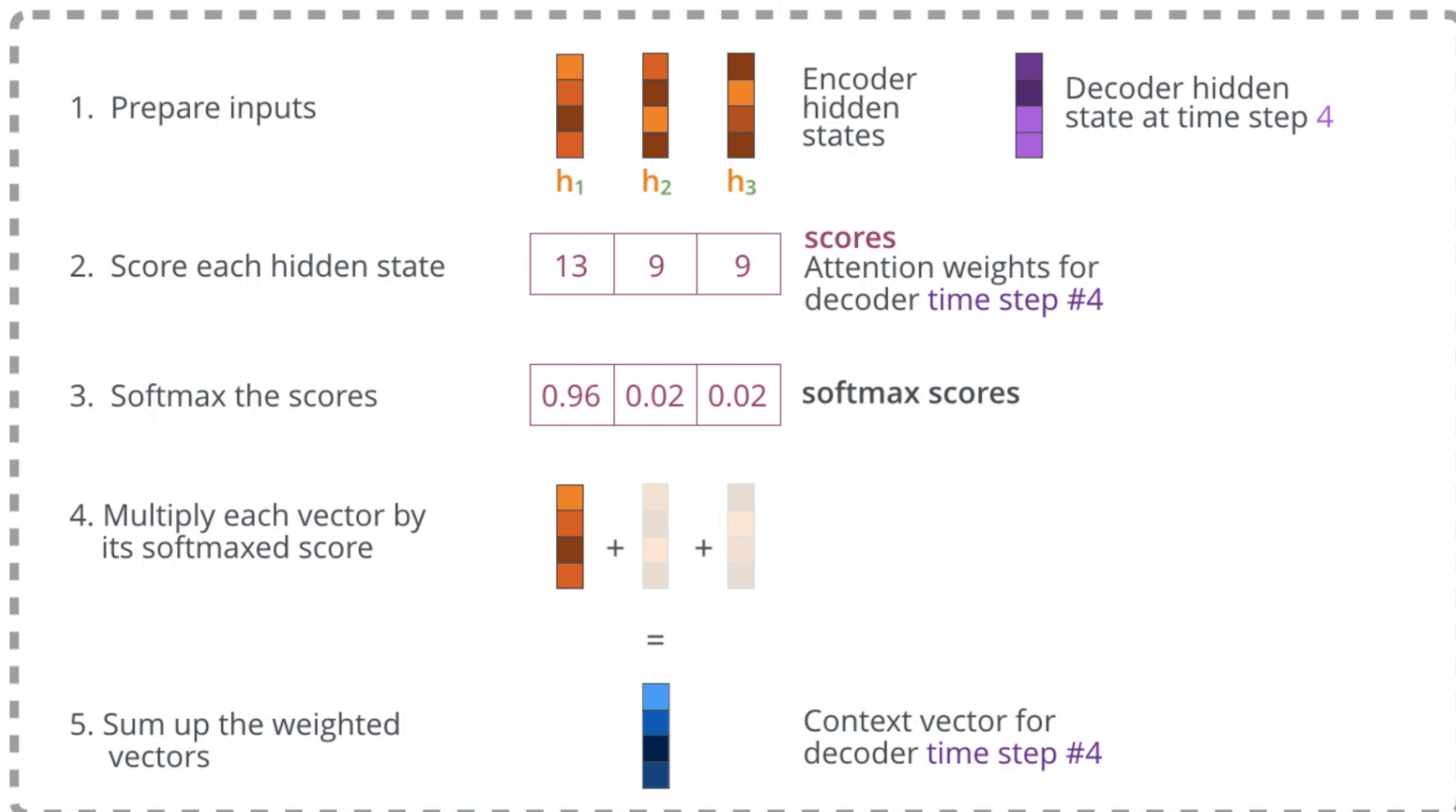
Architecture encodeur-décodeur : GRU

- ♦ Même principe que les LSTM, mais avec une architecture simplifiée.
- ♦ Contient deux gates :
 - **Update gate** : décide quelles informations de l'état caché précédent doivent être conservées.
 - **Reset gate** : décide quelles nouvelles informations doivent être ajoutées à l'état caché.

Architecture encodeur-décodeur : attention

- ♦ Mécanisme d'**attention** (Bahdanau et al., 2016) : assigne un score (« poids d'attention ») à chaque état caché, ce qui permet au décodeur de se concentrer sur différentes parties de la phrase source à chaque étape de génération.
- ♦ Le décodeur peut ainsi donner plus ou moins d'importance à certains mots de la phrase source en fonction du mot qu'il est en train de générer.
- ♦ Permet de mieux capturer les relations entre les mots de la phrase source et ceux de la phrase cible.

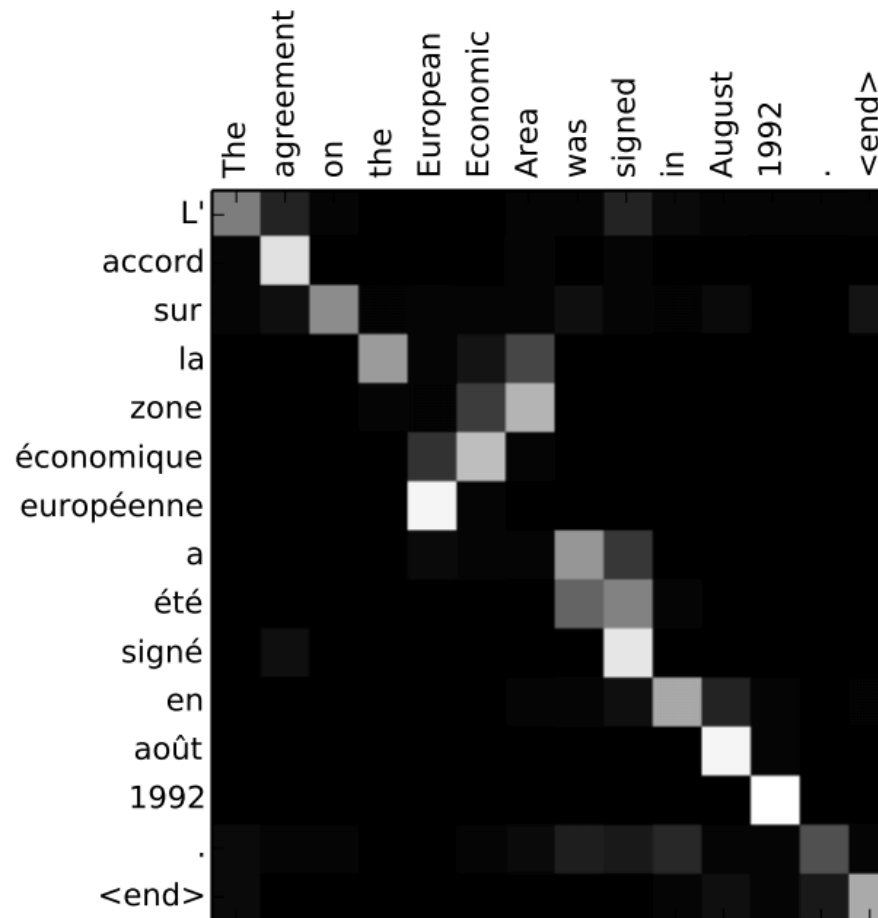
Attention at time step 4



Source : <https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/>

Architecture encodeur-décodeur : attention

- ♦ Mécanisme d'**attention** (Bahdanau et al., 2016) : assigne un score (« poids d'attention ») à chaque état caché, ce qui permet au décodeur de se concentrer sur différentes parties de la phrase source à chaque étape de génération.
- ♦ Le décodeur peut ainsi donner plus ou moins d'importance à certains mots de la phrase source en fonction du mot qu'il est en train de générer.
- ♦ Permet de mieux capturer les relations entre les mots de la phrase source et ceux de la phrase cible.
- ♦ Pour les tâches de traduction, le mécanisme d'attention agit comme un mécanisme d'alignement entre les mots de la phrase source et ceux de la phrase cible.



Source : <https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/>



IV. Mise en pratique

Mise en pratique

- Notebook disponible sur Moodle pour l'entraînement d'un modèle Seq2Seq (RNN) pour de la traduction automatique de l'anglais vers le français.
- Autre notebook facultatif créé par Pablo Ruiz Fabo pour la création d'un réseau à action directe (*feedforward neural network*) pour de la classification (binaire et à classes multiples) : https://colab.research.google.com/drive/1K7aIIRXgAuEPrZTMEfWQdixIslqS_n_L?usp=sharing#scrollTo=JE6QVMa3akD2

Ressources complémentaires

- ♦ **Neural Network Simulator**: <https://www.mladdict.com/neural-network-simulator>

Bibliographie

- Bahdanau, D., Cho, K., et Bengio, Y. (mai 2016). *Neural Machine Translation by Jointly Learning to Align and Translate* (Numéro arXiv:1409.0473). arXiv. [**10.48550/arXiv.1409.0473**](#)
- Bojanowski, P., Grave, E., Joulin, A., et Mikolov, T. (2016). Enriching Word Vectors with Subword Information. *arXiv preprint arXiv:1607.04606*.
- Cho, K., Merrienboer, B. van, Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., et Bengio, Y. (septembre 2014). *Learning Phrase Representations Using RNN Encoder-Decoder for Statistical Machine Translation* (Numéro arXiv:1406.1078). arXiv. [**10.48550/arXiv.1406.1078**](#)
- Jurafsky, D., et Martin, J. H. (2025). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models* (3rd éd.).
- Kalwar, S., Rossi, M., et Sadeghi, M. (2023). Automated Creation of Mappings Between Data Specifications Through Linguistic and Structural Techniques. *IEEE Access*, 11, 30324–30339. [**10.1109/ACCESS.2023.3259904**](#)
- Koehn, P. (2020). *Neural Machine Translation*. Cambridge University Press.
- Ling, W., Dyer, C., Black, A. W., et Trancoso, I. (mai 2015). Two/Too Simple Adaptations of Word2Vec for Syntax Problems. In R. Mihalcea, J. Chai, et A. Sarkar (éds.), *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. [**10.3115/v1/N15-1142**](#)
- Mikolov, T., Chen, K., Corrado, G., et Dean, J. (septembre 2013). *Efficient Estimation of Word Representations in Vector Space* (Numéro arXiv:1301.3781). arXiv. [**10.48550/arXiv.1301.3781**](#)
- Pennington, J., Socher, R., et Manning, C. (octobre 2014). GloVe: Global Vectors for Word Representation. In A. Moschitti, B. Pang, et W. Daelemans (éds.), *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP): Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. [**10.3115/v1/D14-1162**](#)

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., et Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15(56), 1929–1958.

Sutskever, I., Vinyals, O., et Le, Q. V. (décembre 2014). *Sequence to Sequence Learning with Neural Networks* (Numéro arXiv:1409.3215). arXiv. [10.48550/arXiv.1409.3215](#)

Remerciements

- ♦ Pablo Ruiz Fabo pour le contenu de certaines diapositives.