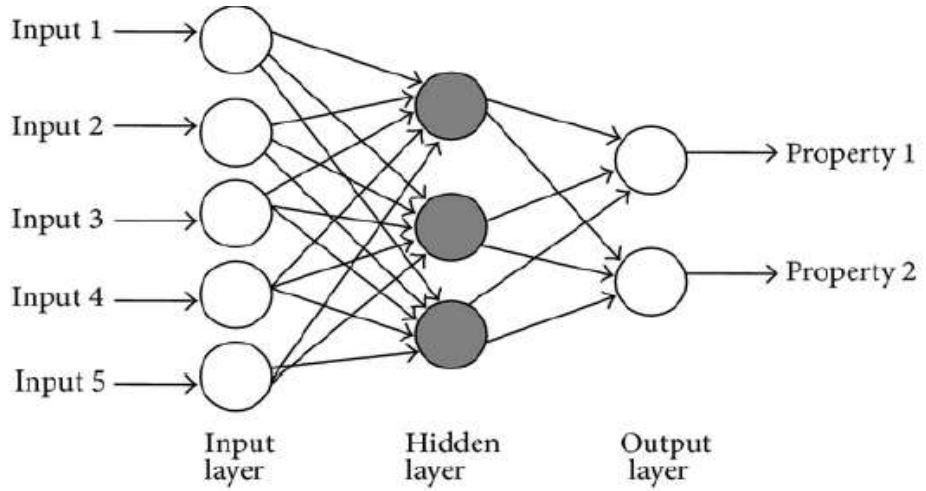
Apprentissage Automatique

Le Perceptron multi-couches



Définition

- Au moins une couche avec fonction d'activation non-linéaire
- Les neurones de la couche i servent d'entrées aux neurones de la couche $i + 1$
- Neurones d'entrées, neurones de sortie, neurones cachés
- Généralement : tous les neurones d'une couche sont connectés à tous les neurones des couches précédentes et suivantes
- Toutes les connections $i \rightarrow j$ sont pondérées par le poids w_{ij}

Le Perceptron multi-couches: Apprendre avec un risque le plus faible

Cas d'un PMC à une couche cachée n entrées, p sorties

- Soit un exemple $(x, y) \in S, x = (x_1, \cdots x_n), y = (y_1, \cdots, y_p)$, la fonction de décision à la sortie k est :

$$h_k(x) = A_o(W_o A(\langle w, x \rangle))$$

- avec A non linéaire, non polynomiale, continue, dérivable, approximant la fonction de Heaviside :
  - $A(x) = x tanh(x)$
  - $A(x) = \frac{1}{1+e^{-x}}$ fonction logistique (sigmoïde)
- et A_o continue dérivable, softmax en classification multi-classes

Quel que soit le nombre de couches

Erreur et risque : dépend de tous les w

- Apprentissage avec risque minimal = minimiser $\ell(h_k(x), y_k)$
- Fixons la fonction de risque $\ell(h_k(x), y) = (h_k(x) - y_k)^2$, continue dérivable si $h_k(x), y \in \mathbb{R}$ (régression).

Le Perceptron multi-couches: Apprendre avec un risque le plus faible

Expression de la fonction à minimiser

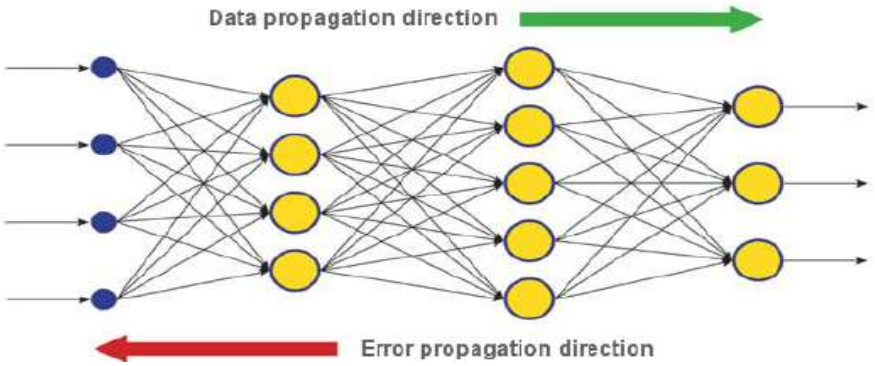
L'erreur (si fonction de perte = mean-square, h_k sortie du neurone k) :

$$E(w) = \frac{1}{2} \sum_{(x,y)\in S} \sum_{k=1}^{p} (h_k - y_k)^2$$

avec $h_k = A_0(w_{k0} + \sum_{j\in \mathrm{Pred}(k)}(w_{kj} a_j))$

- Contribution de tous les neurones et de toutes les synapses
- Prise en compte de tous les exemples de l'échantillon

Activation dans un sens, propagation de l'erreur dans l'autre



Apprentissage PMC: Problème d'optimisation

Optimisation non-linéaire

Fonctions continues, dérivables

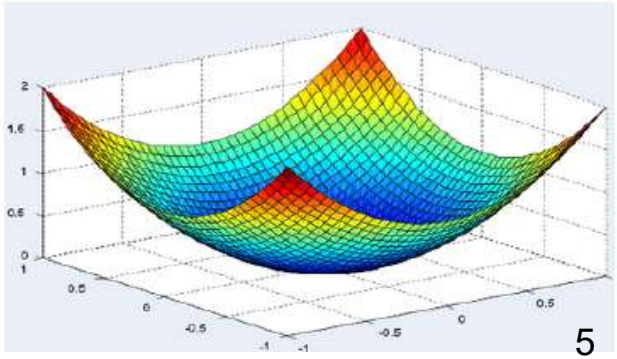
Similarité avec le perceptron

- Minimisation de l'erreur sur chaque présentation individuelle des exemples (règle de Widrow-Hoff)
- On doit minimiser :

$$E = E_{(x,y)}(w) = \frac{1}{2}\sum_{k=1}^{p}(h_k - y_k)^2$$

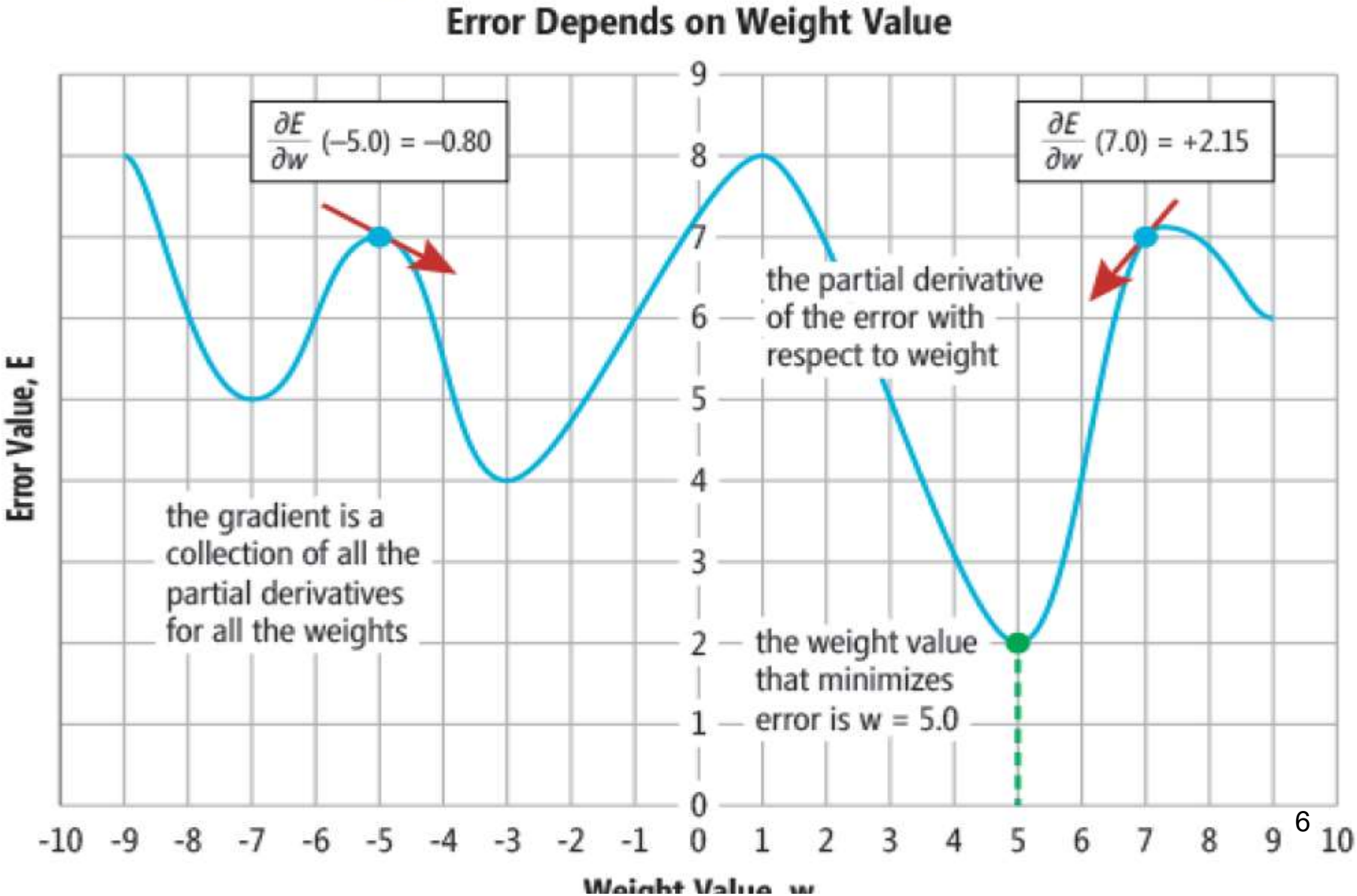
- Plus généralement $\arg\min_\theta L(f_\theta, S) = \arg\min_\theta \frac{1}{n}\sum_{i=1}^{n} \ell(f_\theta(x^i), y^i)$,

Minimiser une fonction : recherche descendante du point de dérivée nulle de la fonction

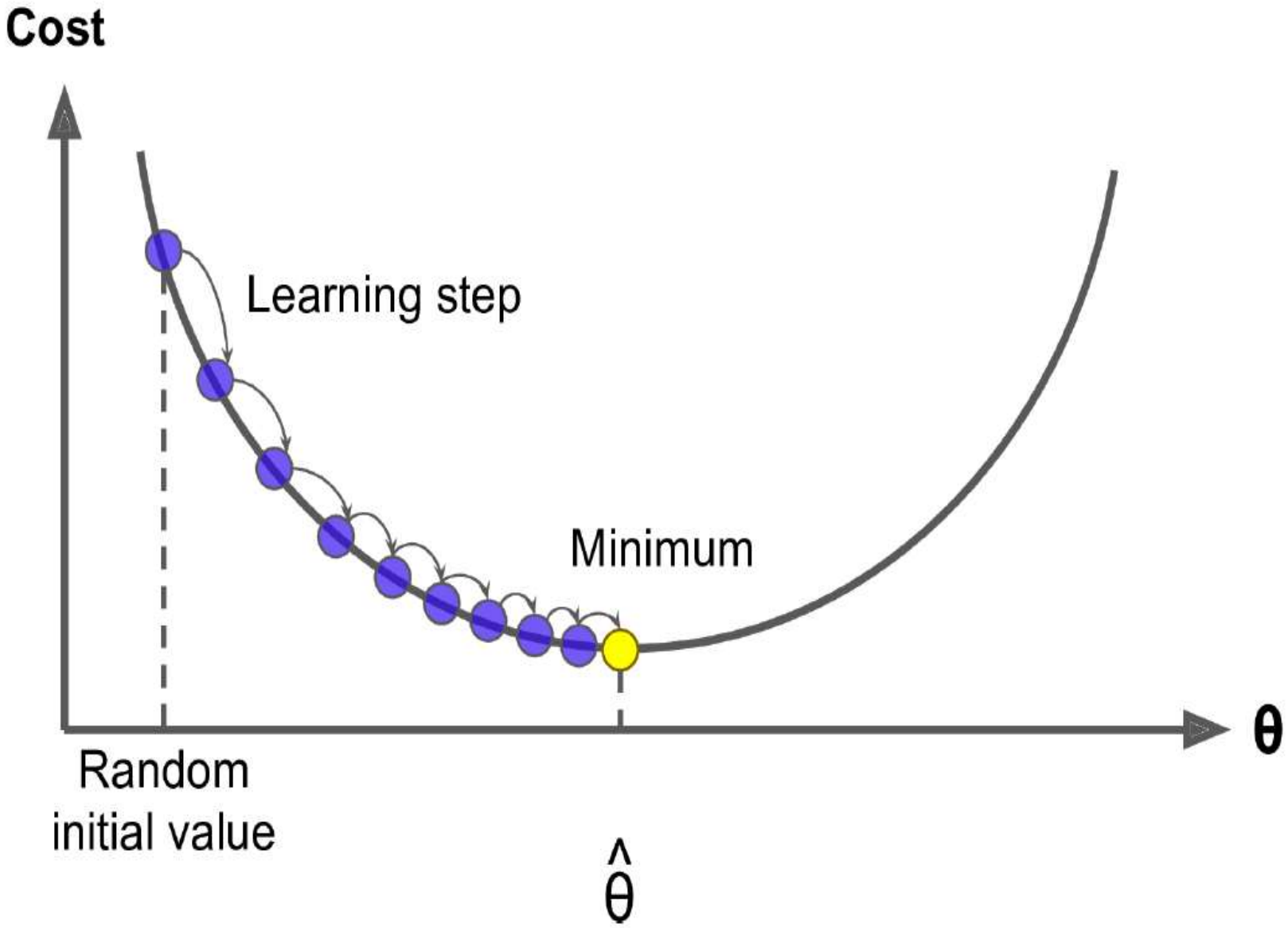


Apprentissage PMC: Problème d'optimisation

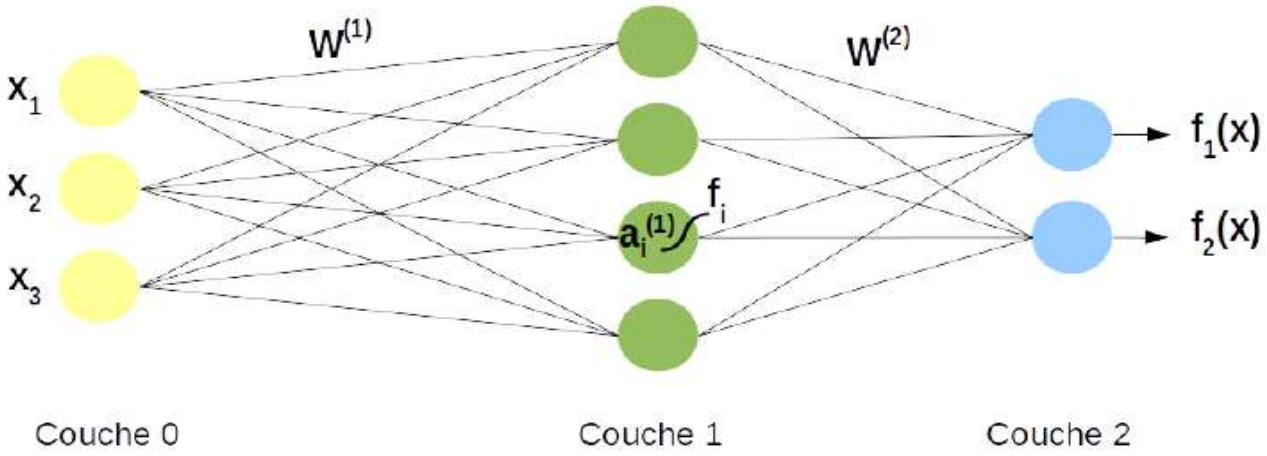
Trouver les *w* qui mènent à la plus petite erreur, si possible



Apprentissage PMC: Descente de gradient



Apprentissage PMC: Algorithme dans le cas d'une seule couche cachée



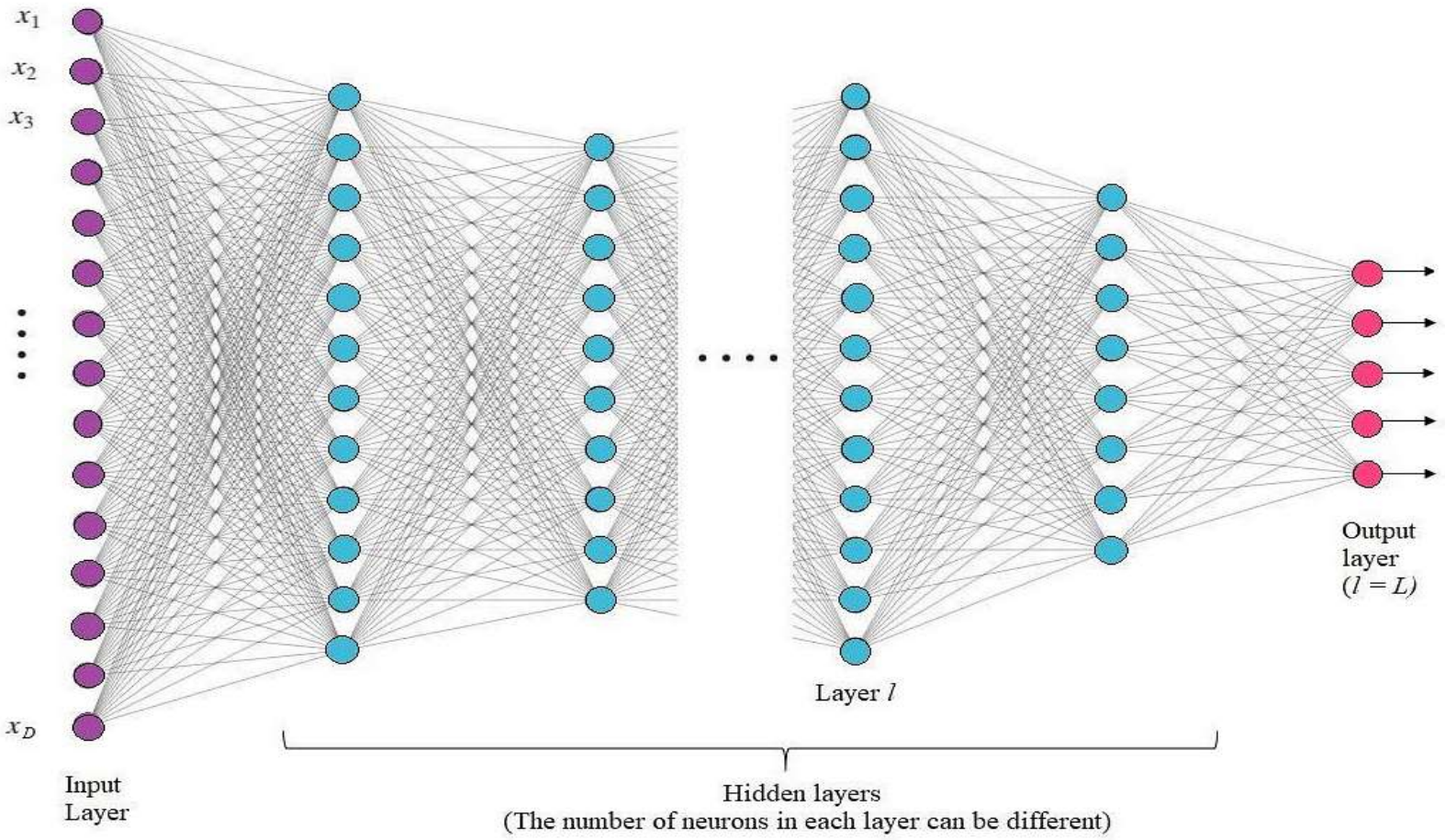
1. Initialisation de tous les poids du réseau (petites valeurs)
2. Pour chaque exemple (x, y) de S faire :
   - 2.1 prediction = $f_k(x)$ = activation-du-MLP(x)
   - 2.2 calculer l'erreur $e_k = \ell(f_k - y_k)$ pour chaque sortie, et $E = \sum_k e_k$
   - 2.3 calculer Δw_{jk} pour tous les poids entre la couche cachée et la couche de sortie
   - 2.4 calculer Δw_{ij} pour tous les poids entre la couche d'entrée et la couche cachée
   - 2.5 Mettre à jour tous les poids

   Jusqu'à satisfaction d'un critère d'arrêt
3. Retourner le réseau

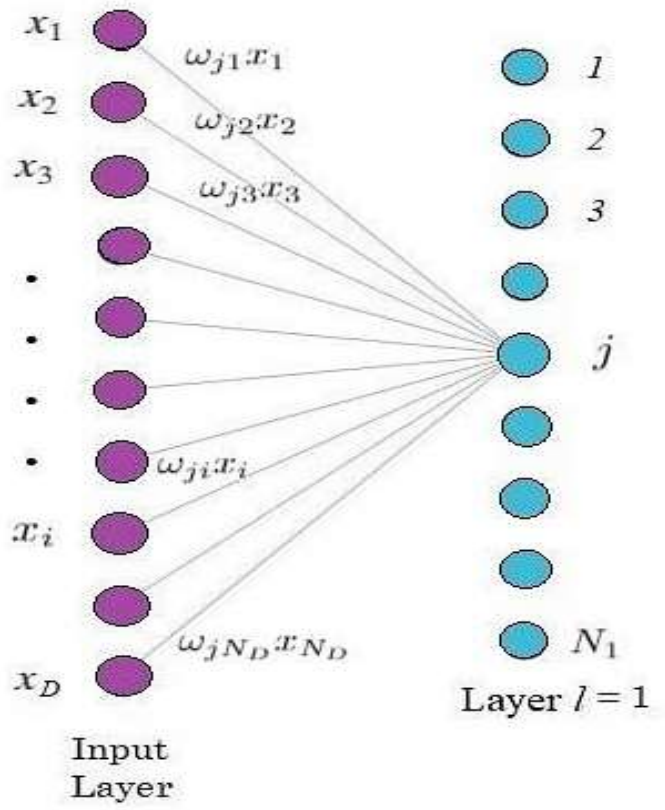
Apprentissage PMC: Algorithme générique

1. initialiser le vecteur de paramètres w
2. initialiser les paramètres - méthode de minimisation
3. **Répéter**
4. **Pour tout** $(x, y) \in D$ **faire**
5. appliquer x au réseau et calculer la sortie correspondante
6. calculer $\frac{\partial \ell(f(x), y)}{\partial W_{ij}}$ pour toutes les pondérations
7. **Fin pour**
8. calculer $\frac{\partial L(f, D)}{\partial W_{ij}}$ en sommant sur toutes les données d'entrée
9. appliquer une mise à jour du vecteur de pondération - méthode de minimization
10. **Tant que** l'erreur n'a pas convergé

Réseaux de neurones avec plusieurs couches cachées



Réseaux de neurones Feed-Forward



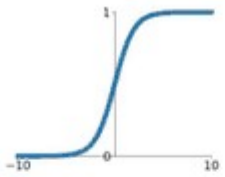
$$z_j^{(1)} = h(a_j^{(1)})$$
$$= h(\sum_{i=1}^{D} \omega_{ji}^{(1)} x_i + b_j^{(1)})$$

- Les neurones de la première couche (***l* = *1***) reçoivent une somme pondérée des éléments du vecteur d'entrée ***x*** ainsi qu'un biais ***b***
- Chaque neurone transforme cette somme pondérée reçue à l'entrée, ***a***, à l'aide d'une fonction d'activation différentiable et non linéaire ***h***(.) pour obtenir la sortie ***z***

Fonctions d'activation

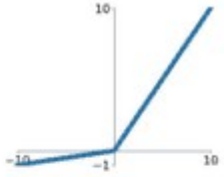
Sigmoid

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



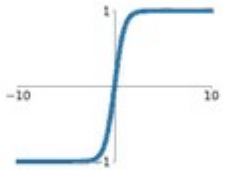
Leaky ReLU

$\max(0.1x, x)$



tanh

$\tanh(x)$

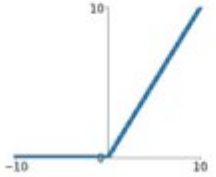


Maxout

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

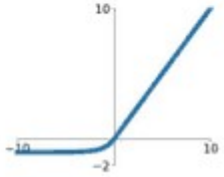
ReLU

$\max(0, x)$



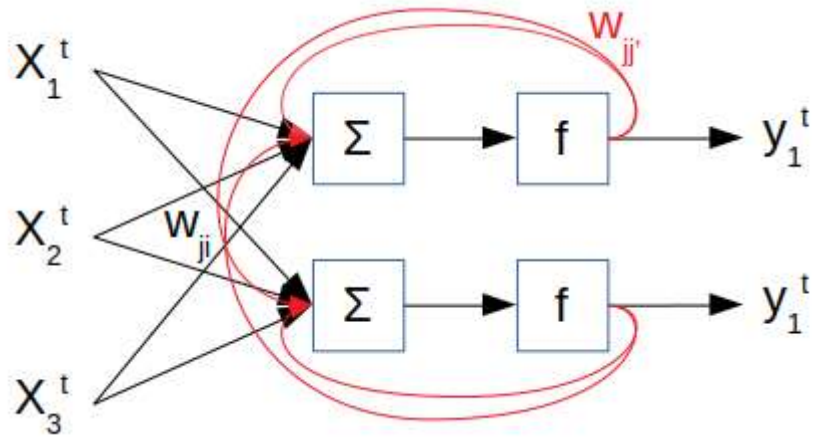
ELU

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



- **Sigmoïd:** produit une courbe en forme de S. Bien que de nature non linéaire, elle ne tient toutefois pas compte des légères variations des entrées, ce qui entraîne des résultats similaires.
- **Tanh** (Fonctions de tangente hyperbolique): Il s'agit d'une fonction similaire à Sigmoid. Cependant, elle est plus lente à converger.
- **ReLu** (Unité linéaire rectifiée): Cette fonction converge plus rapidement, optimise et produit la valeur souhaitée plus rapidement. C'est de loin la fonction d'activation la plus utilisée dans les couches cachées.
- **Softmax**: Cette fonction est utilisée dans la couche de sortie car elle réduit les dimensions et peut représenter une distribution catégorique.

Réseaux de neurones récurrents (RNNs)



- Les x^t_i et les y^t_j désignent respectivement les entrées et les sorties de la couche à l'instant t.
- Les poids w_{ji} relient les entrées à la sortie, et les poids $r_{jj'}$ relient la sortie et l'entrée de la couche (connexions récurrentes).

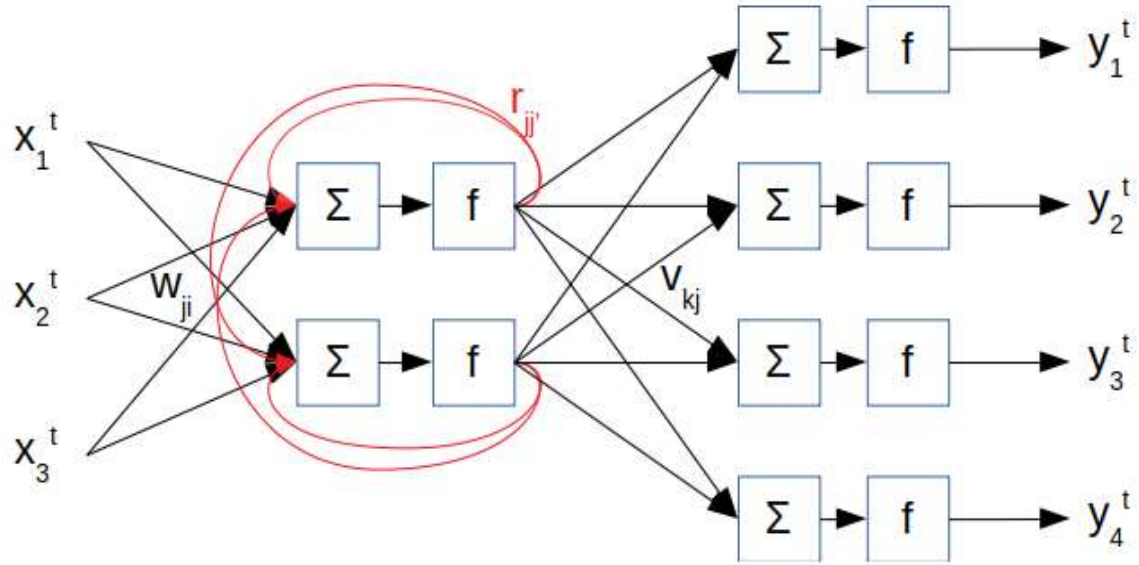
Le calcul de la sortie d'une couche de neurones peut donc se faire par l'équation :

$$y_j^t = f(\sum_i W_{ji} x_i^t + \sum_{j'} r_{ii'} y_i^{t-1})$$

⇨ Les connexions récurrentes réinjectent bien les sorties précédentes y^{t-1}_j en entrée de la couche à l'instant t.

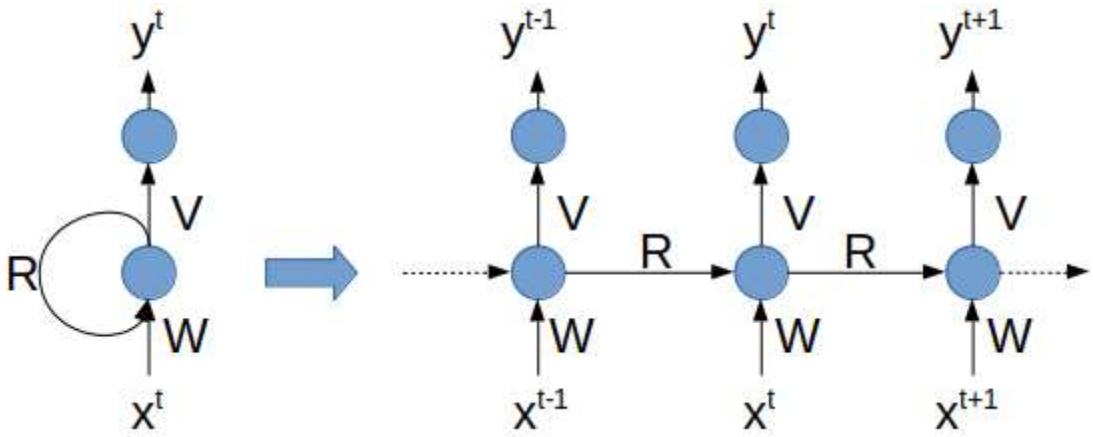
⇨ Le deuxième terme modélise la récurrence du réseau

Réseaux de neurones récurrents suivis d'une couche dense



- Deux matrices de poids pour la couche récurrente : les w_{ji} sont les poids reliant la *i* ème entrée au *j* ème neurone de la couche récurrente, et les $r_{jj'}$ sont les poids de la récurrence, reliant la sortie du *j'* ème neurone récurrent à l'entrée du *j* ème neurone récurrent.
- Une matrice de poids v_{kj} pour la couche dense, reliant le *j* ème neurone de la couche récurrente au *k* ème neurone de la couche de sortie.

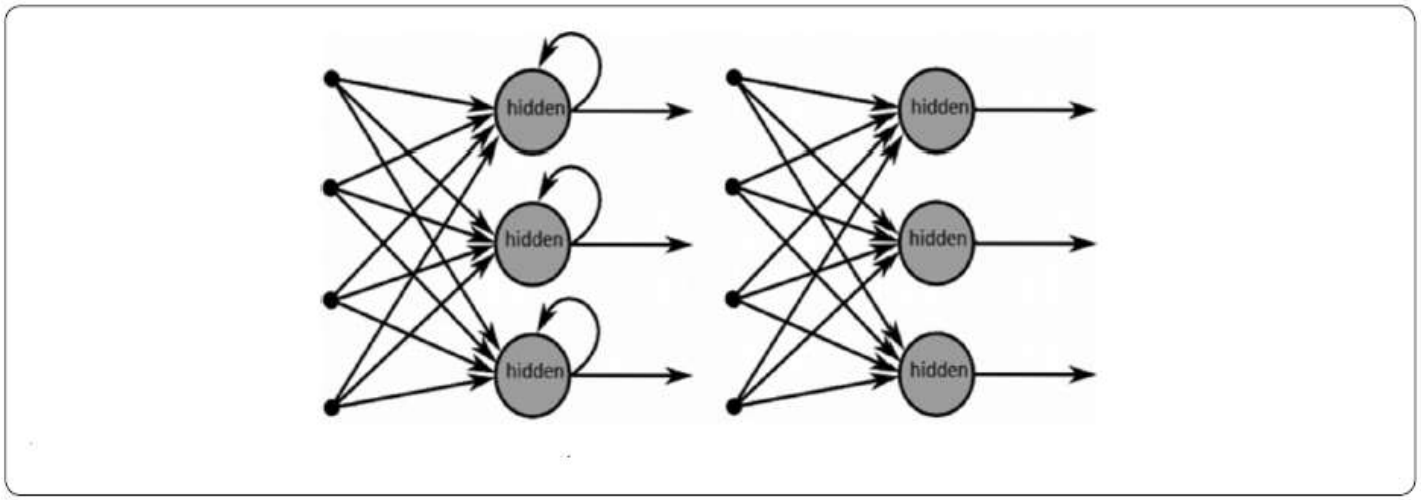
Représentation d'un réseau de neurones récurrents



- Les matrices W, R et V sont dupliquées et apparaissent autant de fois que le nombre de dépliements du réseau dans le temps.

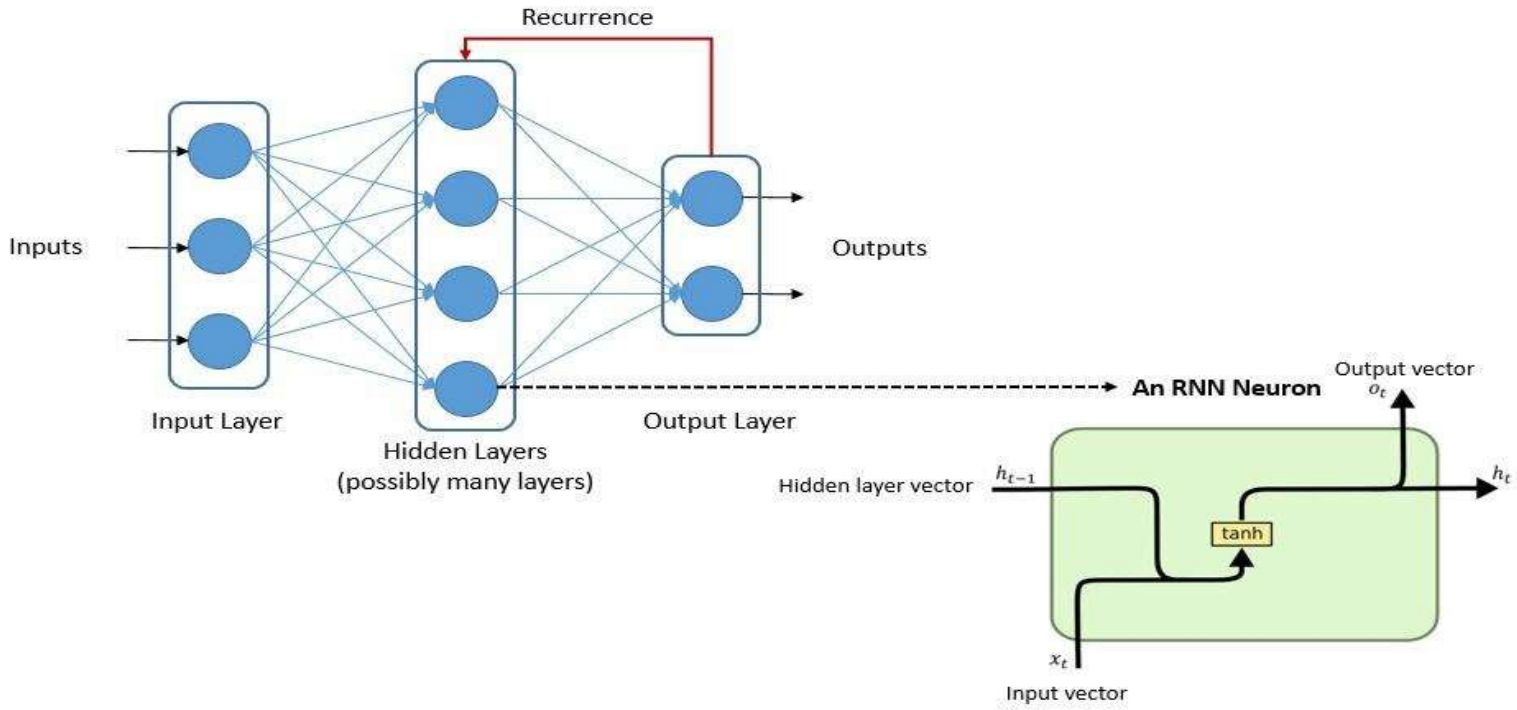
- Les connexions récurrentes utilisent l'information du temps précédent.

Réseaux de neurones récurrents / Réseaux de neurones Feed-Forward



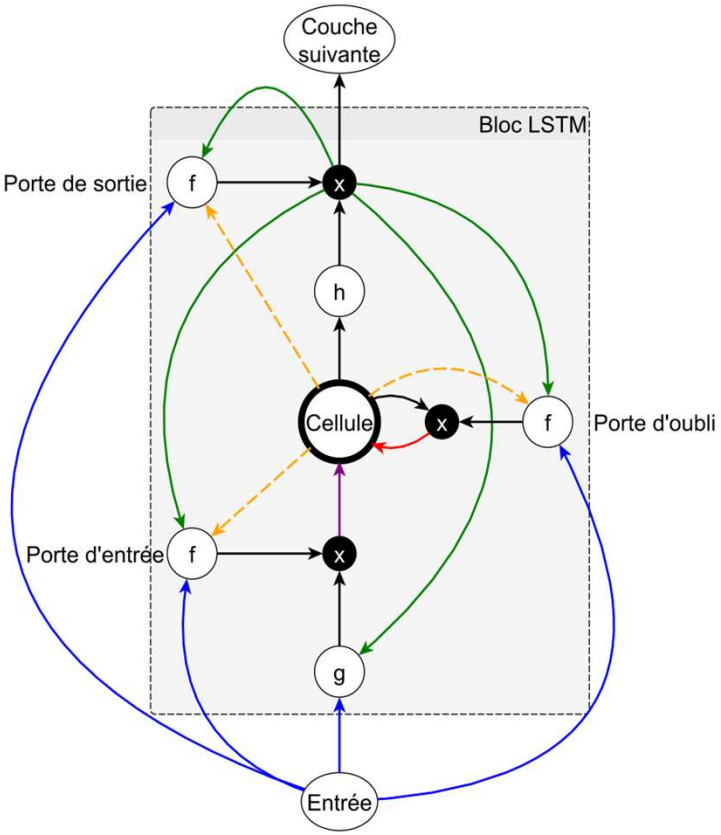
- Les réseaux de neurones de type Feed-Forward sont incapables de maintenir leur état.
- Les RNNs ont des connexions cycliques qui facilitent les mises à jour de leur état en fonction des états précédents et des entrées actuelles.
- Les RNNs sont efficaces pour modéliser les tâches de séquence à séquence comme celles du Traitement Automatique des Langues.

Réseaux de neurones récurrents – Fonctionnement de la récurrence



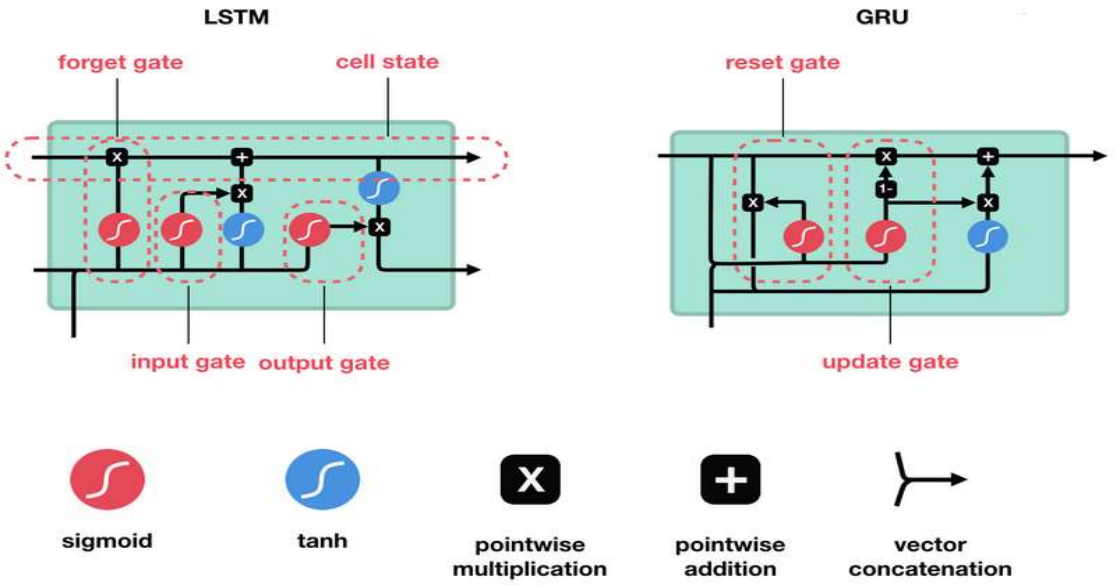
- Les RNNs utilisent leur état interne pour traiter des séquences d'entrées de longueur variable.
- Il y a deux variantes de RNNs qui proposent une mémoire à long terme et une mémoire à court terme : Long-short Term Memory (LSTM) et Gated Recurrent Units (GRU).

RNNs – Fonctionnement de d'une cellule LSTM



- Les RNNs de type LSTM possèdent une mémoire interne appelée cellule. La cellule permet de maintenir un état aussi longtemps que nécessaire. Cette cellule consiste en une valeur numérique que le réseau peut piloter en fonction des situations.
- La cellule mémoire peut être pilotée par trois portes (vannes) de contrôle:
  1. La porte d'entrée décide si l'entrée doit modifier le contenu de la cellule.
  2. La porte d'oubli décide s'il faut remettre à 0 le contenu de la cellule.
  3. La porte de sortie décide si le contenu de la cellule doit influer sur la sortie du neurone.
- L'ouverture/la fermeture de la vanne est modélisée par une fonction f (généralement une sigmoïde) appliquée à la somme pondérée des entrées (en bleu), des sorties (en vert) et de la cellule (en orange).

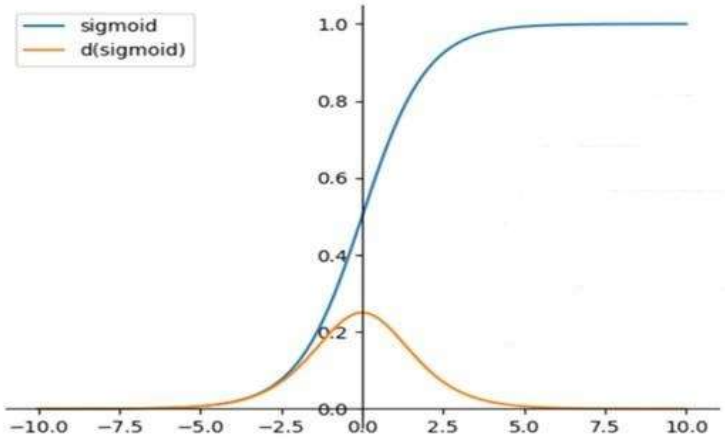
Réseaux de neurones récurrents LSTM-GRU



- Compte tenu de toutes les connexions nécessaires au pilotage de la cellule mémoire, les couches de neurones de type LSTM sont deux fois plus "lourdes" que les couches récurrentes simples, qui elles-mêmes sont deux fois plus lourdes que les couches denses classiques.
- Afin de réduire le nombre de paramètres des modèles, la variante GRU (Gated Recurrent Unit) des LSTM supprime la porte d'oubli. Cette variante permet d'obtenir des performances similaires avec moins de paramètres.
- Les GRU s'entraînent plus rapidement, mais leurs performances sont moins bonnes sur des datasets de taille importante.
- Les couches LSTM sont généralement combinées avec des couches denses (FCL: Full Connected Layer) ou des couches convolutionnelles (CNN).

Limites des Réseaux de neurones récurrents LSTM-GRU

- Les réseaux de neurones de type LSTM sont très longs à entrainer
- L'introduction des données se fait de manière séquentielle
  - ➔ Pas de possibilité de paralléliser l'apprentissage,
- Les modèles LSTM et GRU souffrent du problème de disparition du gradient (vanishing gradient)
  - Ce problème est rencontré dans les modèles utilisant une méthode d'apprentissage par rétro-propagation du gradient.
  - Lorsque les entrées de la fonction sigmoid deviennent grandes ou petites, la dérivée devient proche de zéro,
  - Lorsque les *n* couches cachées utilisent une fonction d'activation comme la sigmoid, les *n* petites valeurs des dérivées sont multipliées entre elles,
    - ➔ Le gradient décroit exponentiellement en propageant vers les couches initiales, ce qui provoque le problème de disparition du gradient.
    - ➔ Cela empêche effectivement les poids de modifier leur valeur. Ce qui peut empêcher le réseau neuronal de poursuivre son apprentissage. Ce problème rend l'entrainement des RNNs pour les tâches TAL pratiquement inefficace.



Mécanisme d'attention dans les RNNs de type LSTM

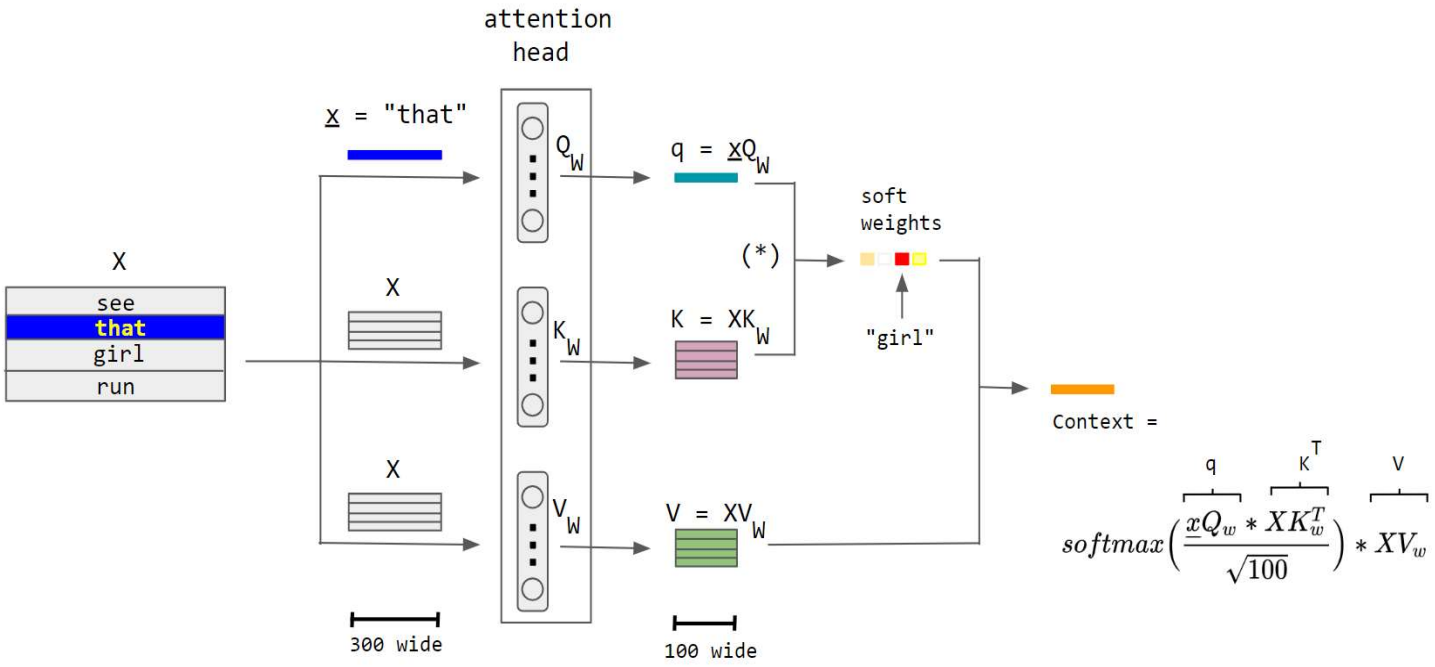
- Dans les RNNs de type LSTM, la quantité d'informations qui peut être propagée est limitée et la fenêtre d'informations conservées est plus courte

- Le mécanisme d'attention permet à cette fenêtre d'information d'être considérablement élargie

- Le mécanisme d'attention est une technique qui permet d'améliorer certaines parties des données d'entrée tout en diminuant d'autres parties

➔ Motivation: le réseau doit se concentrer davantage sur les parties importantes des données

Mécanisme d'attention dans les RNNs de type LSTM



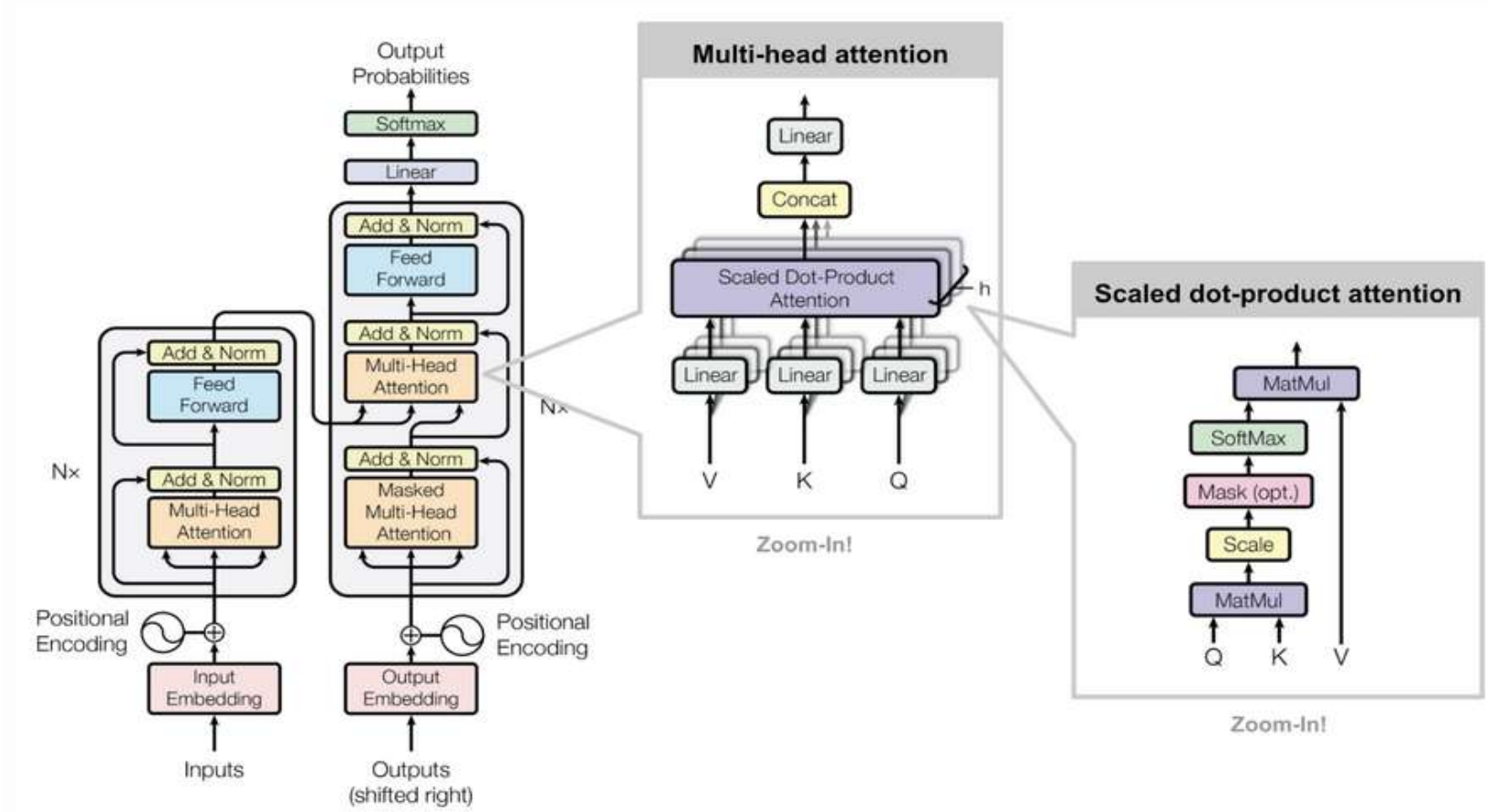
$$softmax\left(\frac{\underset{q}{xQ_w} * \underset{K^T}{XK_w^T}}{\sqrt{100}}\right) * \underset{V}{XV_w}$$

Identification des corrélations entre les entrées une fois qu'un réseau a été entrainé:

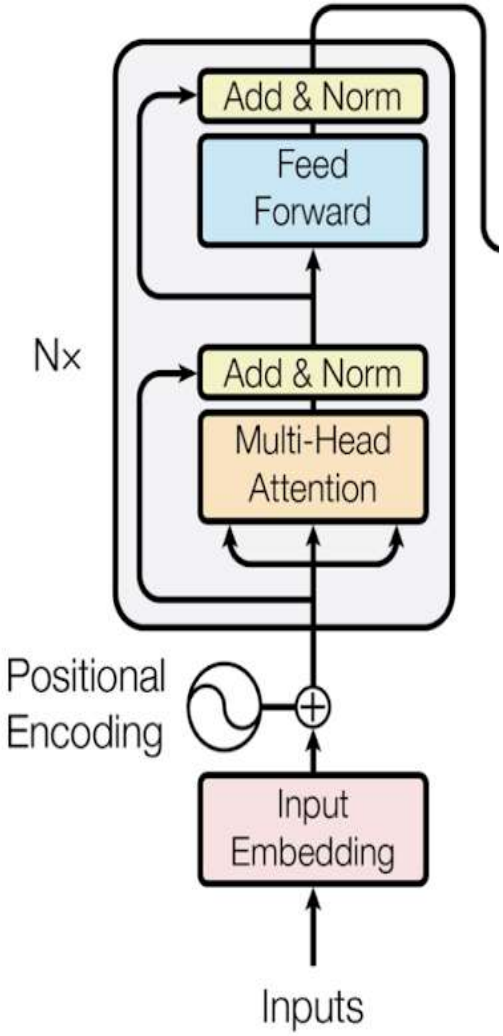
Exemple: *see that girl run*

- ➔ Lorsque le modèle examine le mot "that" dans la phrase "see that girl run", le réseau devrait être en mesure d'identifier "girl" comme un mot fortement corrélé.
- ➔ Dans cet exemple, on se concentre sur le mot "that", mais tous les mots reçoivent ce traitement en parallèle et les poids et les vecteurs de contexte qui en résultent sont empilés dans des matrices.
- ➔ Le vecteur de la requête (q) est comparé à chaque mot des clés (k). Cela aide le modèle à découvrir le mot le plus pertinent pour le mot de la requête. Dans ce exemple, le mot "girl" est le plus pertinent pour "that".

Architecture des Transformers



Architecture des Transformers – Structure de l'Encodeur



- La couche de l'encodeur contient deux sous-couches:
  1. un Multi-Head Attention.
  2. un réseau Feed Forward.

- Il existe également des connexions résiduelles autour de chacune des deux sous-couches suivies d'une normalisation de couche (pour accélérer l'apprentissage).

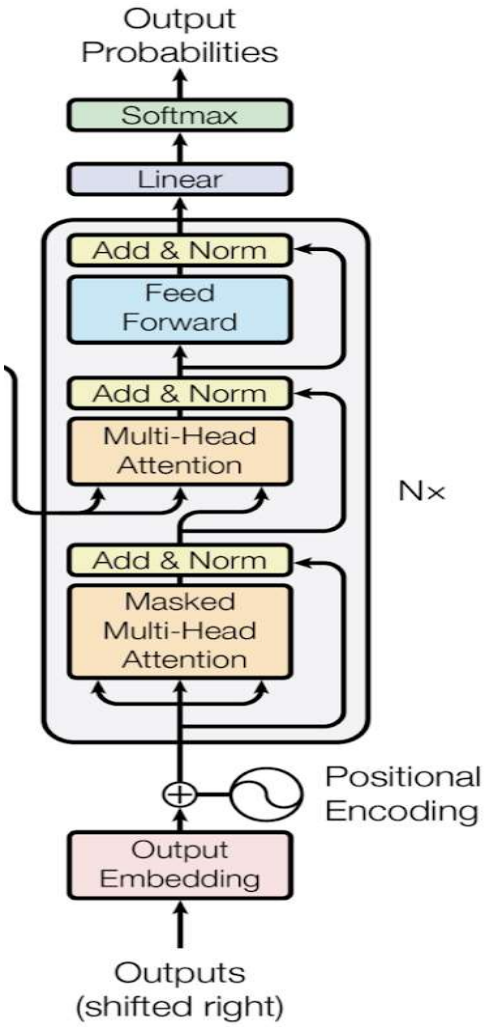
- **L'encodeur** peut être empilé *n* fois (la sortie d'un encoder sera l'entrée de l'encoder suivant) pour coder davantage les informations, où chaque couche a la possibilité d'apprendre différentes représentations de l'attention, augmentant ainsi la puissance prédictive du réseau de transformers.

- **L'Input Embeddings** consiste à envoyer l'entrée dans une couche *embedding layer*: Chaque mot est représenté par un vecteur avec des valeurs continues.

- Le **Positional Encoding** consiste à injecter des informations de position dans les embeddings.

- Le **Multi-headed Attention** applique un mécanisme de Self-Attention.

Architecture des Transformers – Structure du Décodeur



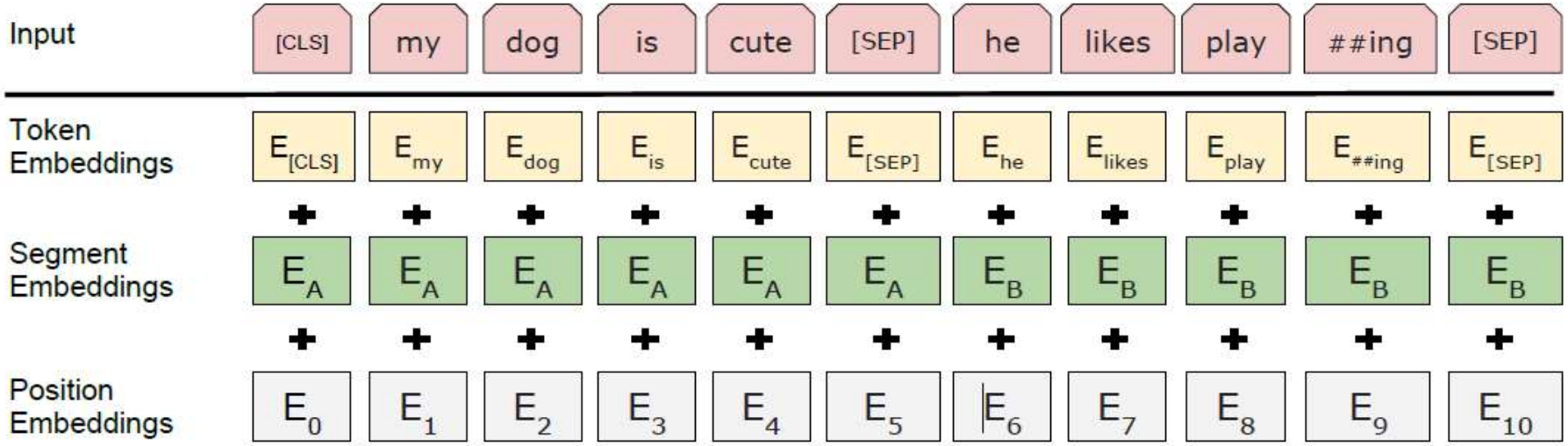
- La couche du décodeur contient trois sous-couches:
  1. un Multi-Head Attention masqué.
  2. un Multi-Head Encoder-Decoder Attention.
  3. Un Réseau Feed Forward

- Chacune de ces sous-couches est suivie d'une connexion résiduelle et une normalisation. Le décodeur est bouclé d'une couche linéaire qui agit comme un classificateur et d'un softmax pour obtenir les probabilités d'un mot.

- La sortie de la couche Feed Forward passe par une couche linéaire qui agit comme un classificateur.

- Le décodeur prend ensuite la sortie, l'ajoute à la liste des entrées du décodeur et continue à décoder à nouveau jusqu'à ce qu'un token soit prédit.

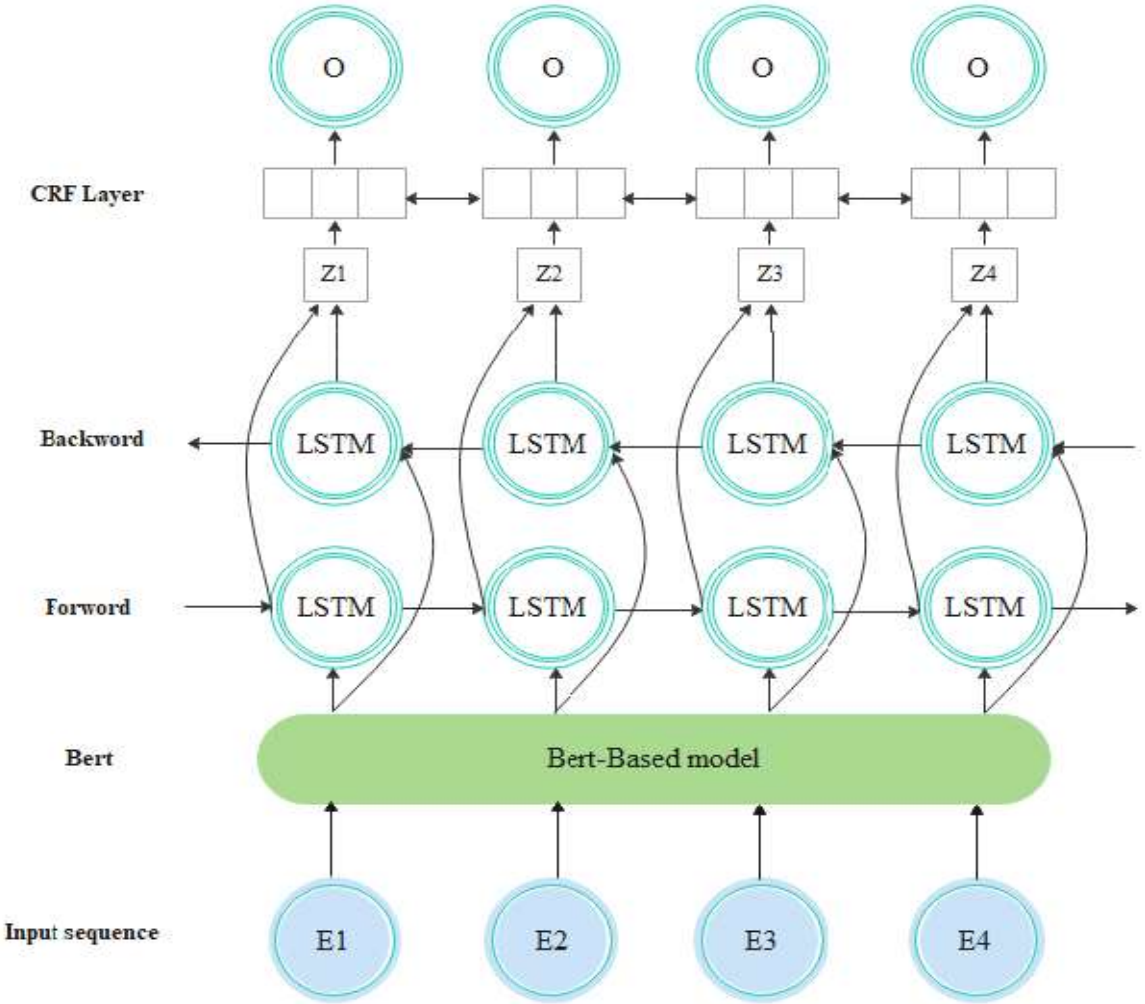
Architecture du modèle de langue BERT (Bidirectional Encoder Representations from Transformers)



- Représentation de l'entrée de BERT: Les embeddings d'entrée sont la somme des token embeddings, des segmentation embeddings et des position embeddings.
- Le modèle de langue BERT a deux objectifs en pré-entrainement (tâches non supervisées):
  1. Masked LM (MLM): L'objectif est de masquer un nombre au hasard de tokens d'entrée, puis prédire les tokens masqués.
  2. Next Sentence Prediction (NSP): L'objectif ici est de prédire la phrase qui suit une phrase donnée (courante).

Exemple d'utilisation de BERT pour la Reconnaissance d'Entitées Nommées

Architecture BERT-BiLSTM-CRF



Quelques liens sur l'Apprentissage Automatique

- **Les réseaux de neurones récurrents**

https://blog.octo.com/les-reseaux-de-neurones-recurrents-des-rnn-simples-aux-lstm

- **Mécanisme d'attention**

https://lilianweng.github.io/posts/2018-06-24-attention/

- **Fonctions d'Activation et Fonctions Loss**

https://indraneeldb1993ds.medium.com/activation-functions-and-loss-functions-for-neural-networks-how-to-pick-the-right-one-542e1dd523e0

https://towardsdatascience.com/deep-learning-which-loss-and-activation-functions-should-i-use-ac02f1c56aa8

- **Les réseaux de neurones Transformers, GPT, BERT, etc.**

https://france.devoteam.com/paroles-dexperts/lstm-transformers-gpt-bert-guide-des-principales-techniques-en-nlp/

https://www.assemblyai.com/blog/how-chatgpt-actually-works/

- **Modèles de langue larges (Large Language Models)**

https://www.baeldung.com/cs/large-language-models