- Réseaux de neurones artificiels et apprentissage profond

Introduction

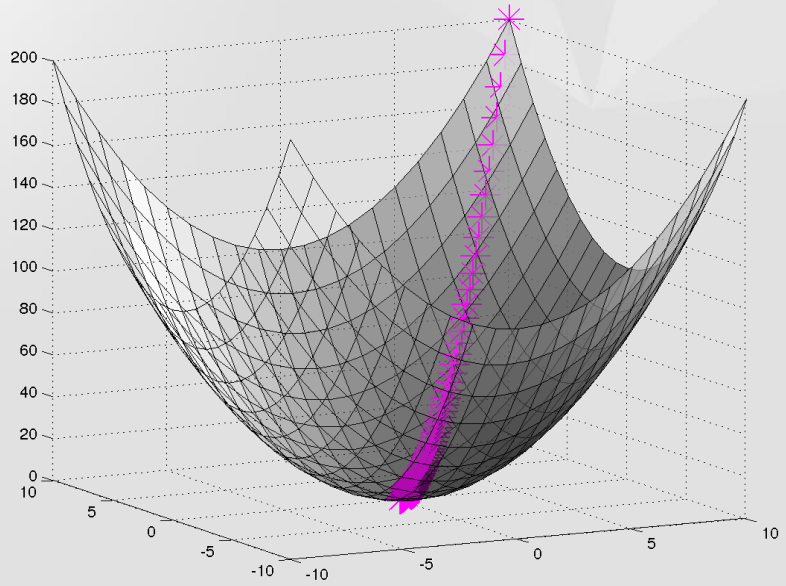
Part 1

Perceptron multi-couches

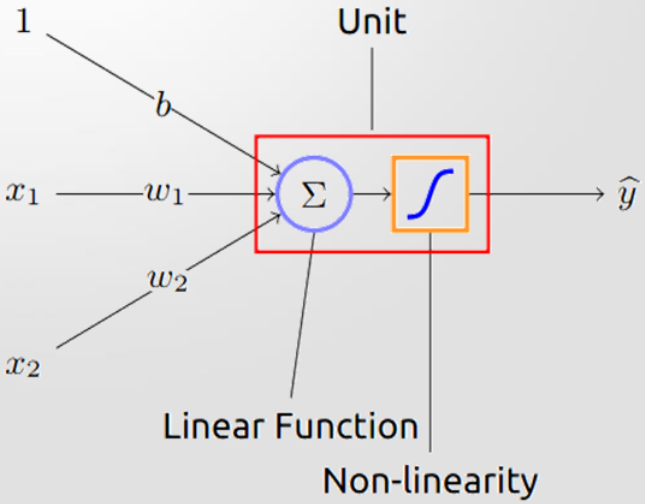
Régression logistique

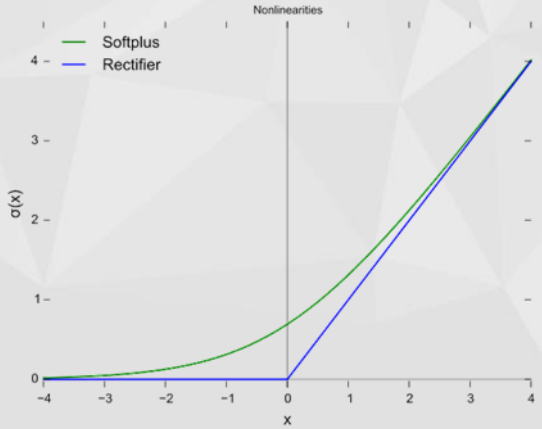
- Combinaison linéaire des variables
- Injection dans une sigmoïde
- Ajustement du modèle:
  - Fonction objectif dépendant des coefficients de la combinaison linéaire (entropie croisée, typiquement)
  - Minimisation de la fonction objectif (par descente de gradient, typiquement)

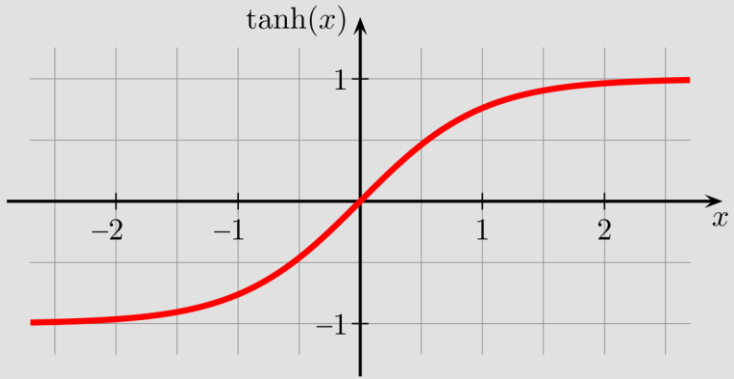
Optimisation par descente de gradient



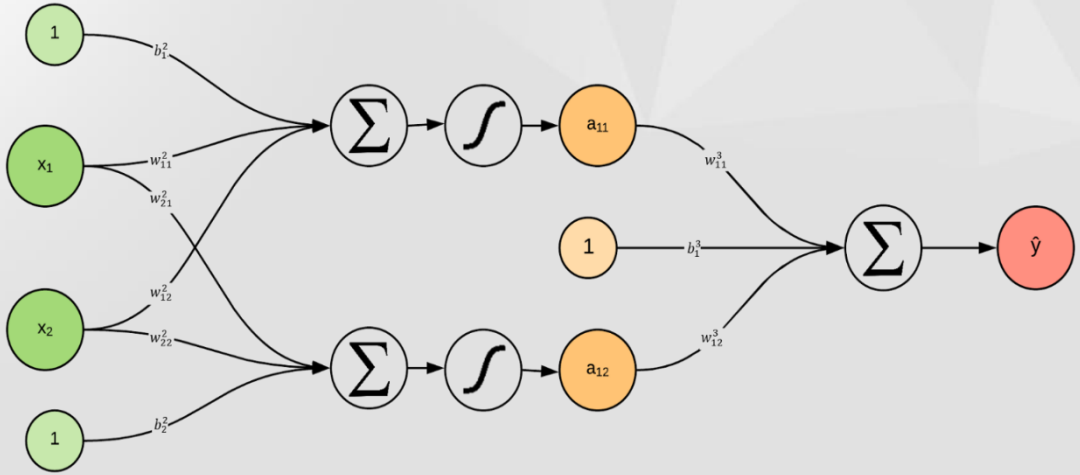
Neurone artificiel







Perceptron multi-couches



$$\hat{y} = w_{11}^3 \cdot f_1(w_{11}^2 \cdot x_1 + w_{12}^2 \cdot x_2 + b_1^2) + w_{12}^3 \cdot f_2(w_{21}^2 \cdot x_1 + w_{22}^2 \cdot x_2 + b_2^2) + b_1^3$$

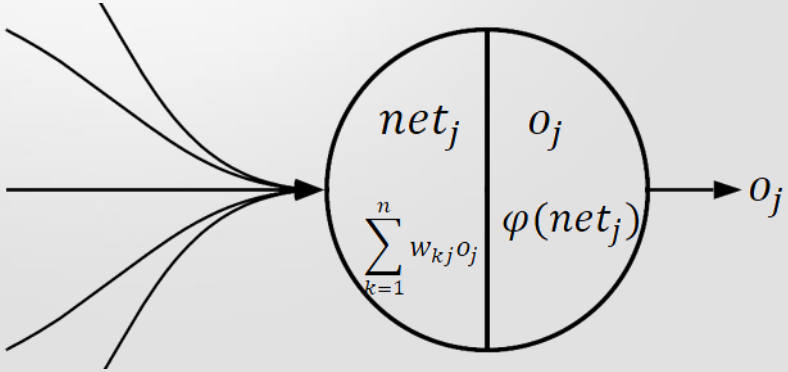
Perceptron multi-couches

- Fonction arbitrairement complexe des variables (mais « différentiable »)
- Ajustement du modèle:
  - Fonction objectif dépendant des coefficients de la combinaison linéaire (maximum de vraisemblance, typiquement)
  - Minimisation de la fonction objectif (par descente de gradient, typiquement)
  - Problème du calcul de gradient: méthode de retro-propagation

Descente de gradient stochastique

- Tirage au sort (sans remise) d'une observation du jeu de données à itération de la descente
- Estimation de la fonction objectif et de son gradient a partir de cette seule observation
- Converge en moyenne vers le même résultat
- Variante: Tirer plusieurs observations à la fois (typiquement entre 32 et 500)

Méthode de rétropropagation



$Couche\ h_{t-1}$ $Couche\ h_t$ $Couche\ h_{t+1}$

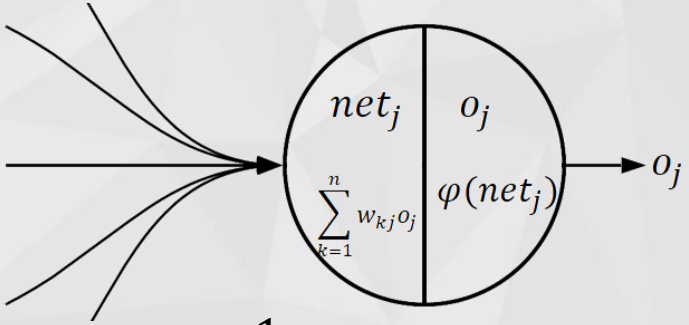
$$E = \frac{1}{2}(t - y)^2$$

$$\varphi(z) = \frac{1}{1 + e^{-z}}$$

$$\varphi'(z) = \frac{\varphi(z)}{1 - \varphi(z)}$$

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j} \cdot \frac{\partial o_j}{\partial net_j} \cdot \frac{\partial net_j}{\partial w_{ij}}$$

Méthode de rétropropagation



$$\frac{\partial net_j}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}}\left(\sum_{k=1}^{n} w_{kj} o_j\right)$$
$$= \frac{\partial}{\partial w_{ij}}(w_{ij} o_j)$$
$$= o_j$$

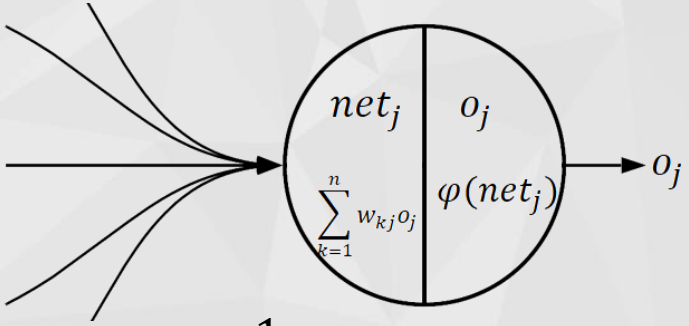
$$E = \frac{1}{2}(t - y)^2$$

$$\varphi(z) = \frac{1}{1 + e^{-z}}$$

$$\varphi'(z) = \frac{\varphi(z)}{1 - \varphi(z)}$$

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j} \cdot \frac{\partial o_j}{\partial net_j} \cdot \frac{\partial net_j}{\partial w_{ij}}$$

Méthode de rétropropagation



$$\frac{\partial o_j}{\partial net_j} = \frac{\partial}{\partial net_j}\varphi(net_j)$$

$$= \frac{\varphi(net_j)}{1-\varphi(net_j)}$$

$$E = \frac{1}{2}(t-y)^2$$

$$\varphi(z) = \frac{1}{1+e^{-z}}$$

$$\varphi'(z) = \frac{\varphi(z)}{1-\varphi(z)}$$

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j} \cdot \frac{\partial o_j}{\partial net_j} \cdot \frac{\partial net_j}{\partial w_{ij}}$$

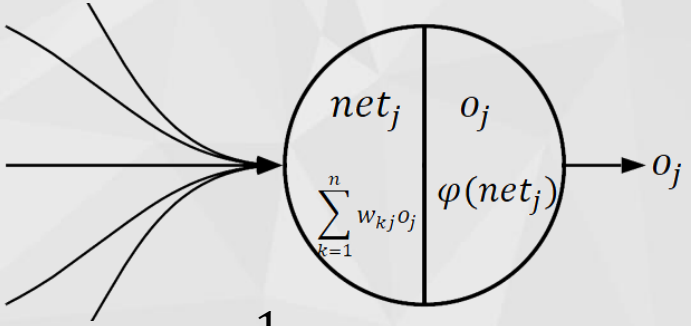
Méthode de rétropropagation

- Couche de sortie

$$\frac{\partial E}{\partial o_j} = \frac{\partial E}{\partial y}$$

$$= \frac{\partial}{\partial y}\frac{1}{2}(t-y)^2$$

$$= y - t$$



$$E = \frac{1}{2}(t-y)^2$$

$$\varphi(z) = \frac{1}{1+e^{-z}}$$

$$\varphi'(z) = \frac{\varphi(z)}{1-\varphi(z)}$$

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j} \cdot \frac{\partial o_j}{\partial net_j} \cdot \frac{\partial net_j}{\partial w_{ij}}$$

Méthode de rétropropagation

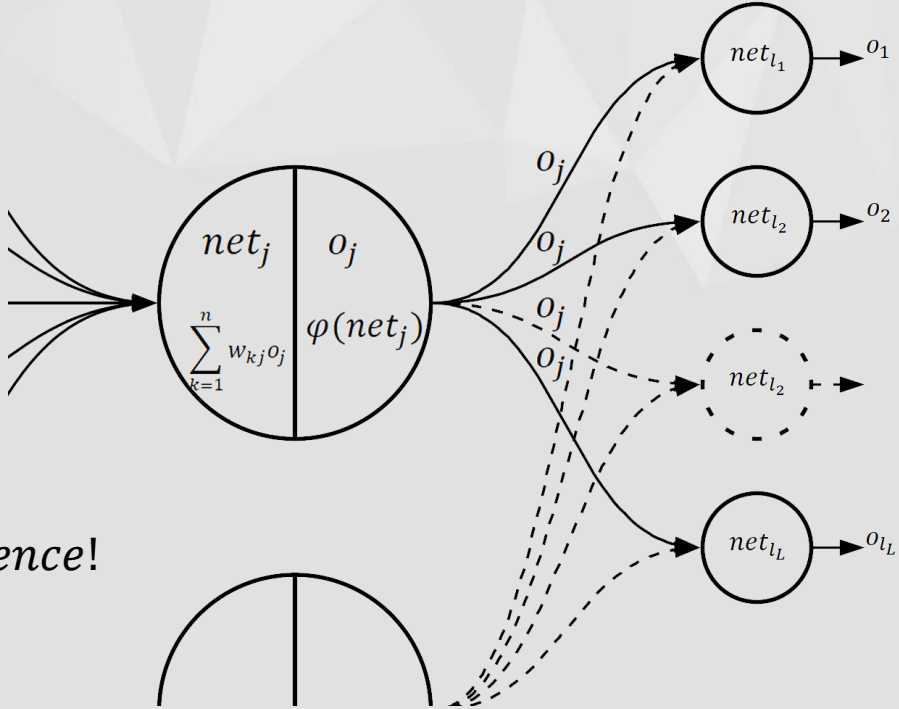
– Couche interne

$$\frac{\partial E(o_j)}{\partial o_j} = \frac{\partial E(net_{l_1}, net_{l_2}, \dots, net_{l_L})}{\partial o_j}$$

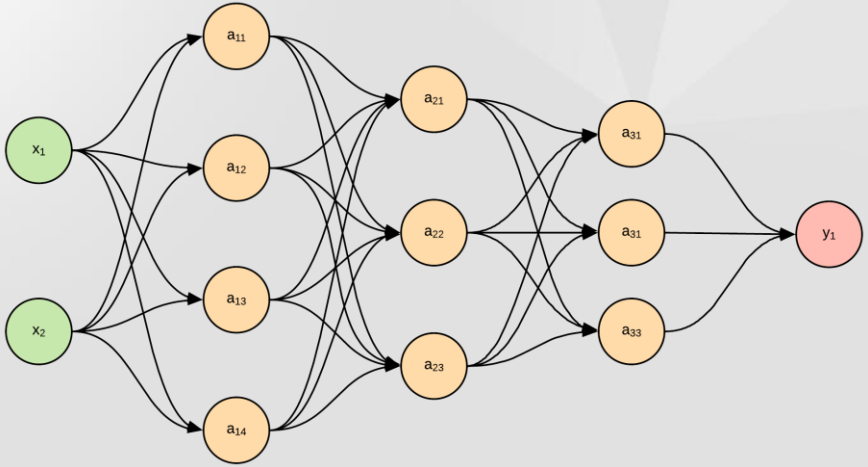
$$= \sum_{i=1}^{L} \frac{\partial E}{\partial net_{l_i}} \frac{\partial net_{l_i}}{\partial o_j}$$

$$= \sum_{i=1}^{L} \frac{\partial E}{\partial o_{l_i}} \frac{\partial o_{l_i}}{\partial net_{l_i}} w_{jl_i}$$

recurrence!



Perceptron multi-couches



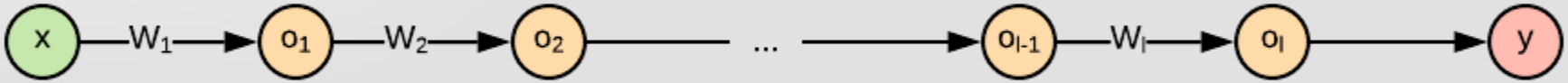
Commande Tensorflow: tf.keras.layers.Dense

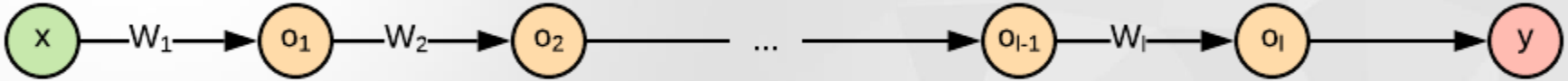
Perceptron multi-couches: avantages et inconvénients

- Théorème d'approximation universel
- Fort risque de surajustement
- Fonction objectif non convexe et difficulté du problème d'optimisation (phénomènes d'explosion et d'évaporation de gradient)

Evaporation de gradient

- Perceptron de n couches, avec un neurone par couches (sans biais)



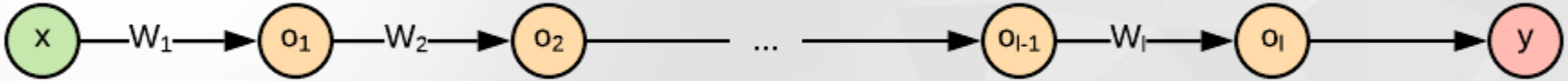


$$\frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial o_1}\frac{\partial o_1}{\partial net_1}\frac{\partial net_1}{\partial w_1}$$

$$net_1 = w_1 x \quad \Rightarrow \quad \frac{\partial net_1}{\partial w_1} = x$$

$$o_1 = \varphi(net_1) \quad \Rightarrow \quad \frac{\partial o_1}{\partial net_1}$$
$$= \varphi(net_1)\left(1 - \varphi(net_1)\right)$$

$$\frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial o_1}\varphi(net_1)\left(1 - \varphi(net_1)\right)x$$



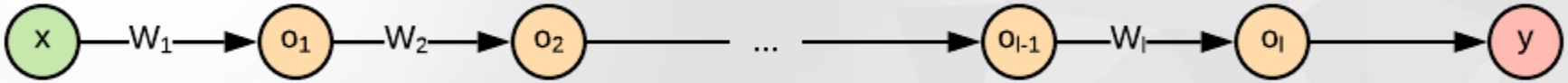
$$\frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial o_1}\varphi(net_1)\left(1 - \varphi(net_1)\right)x$$

$$\frac{\partial E}{\partial o_1} = \frac{\partial E}{\partial o_2}\frac{\partial o_2}{\partial net_2}\frac{\partial net_2}{\partial o_1}$$

$$net_2 = w_2 o_1 \quad \Rightarrow \quad \frac{\partial net_2}{\partial o_1} = w_2$$

$$\frac{\partial E}{\partial o_1} = \frac{\partial E}{\partial o_2}\varphi(net_2)\left(1 - \varphi(net_2)\right)w_2$$

$$\frac{\partial E}{\partial o_1} = (y - t)\prod_{i=2}^{l}\varphi(net_i)\left(1 - \varphi(net_i)\right)w_i$$



$$\frac{\partial E}{\partial w_1} = x(y-t)\,\varphi(net_1)\left(1-\varphi(net_1)\right)\prod_{i=2}^{l}\varphi(net_i)\left(1-\varphi(net_i)\right)w_i$$

$$\left|\frac{\partial E}{\partial w_1}\right| < \left|x(y-t)\,0.25^l\prod_{i=2}^{l}w_i\right|$$

$$w_i \sim N(0,1)$$

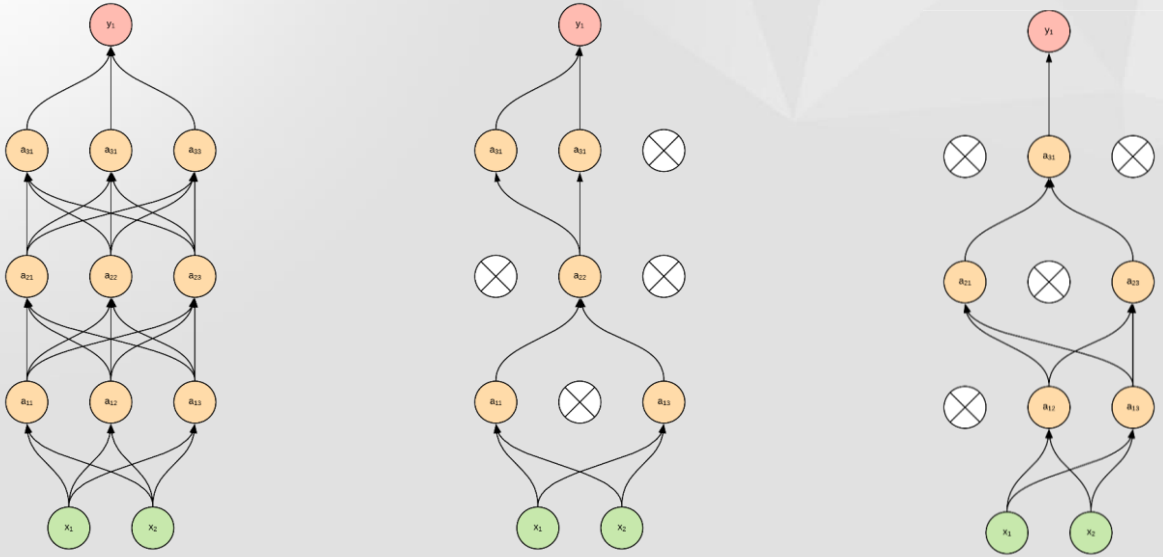
Risque de surajustement

- Méthodes de régularisation (drop-out, ridge, lasso)

Difficultés d'ajustement

- Amélioration de l'algorithme de descente (Adam)
- Amélioration de la méthode d'initialisation (Xavier)
- Modification de l'architecture neuronale (non linéarité, normalisation, connections résiduelles)

Régularisation: Drop-out



Commande Tensorflow: tf.keras.layers.Dropout

Régularisation: Drop-out

- Couper aléatoirement avec probabilité p des connections entre neurones a chaque itération de la descente de gradient
- Plusieurs variantes:
  - Couper différentes connections pour chaque élément de la batch
  - Injection additive de bruit
- Comportement en phase de prédiction?

Régularisation: Drop-out

- Couper aléatoirement avec probabilité p des connections entre neurones a chaque itération de la descente de gradient
- Plusieurs variantes:
  - Couper différentes connections pour chaque élément de la batch
  - Injection additive de bruit
- Comportement en phase de prédiction?
  - Multiplier les connections du modèle par p

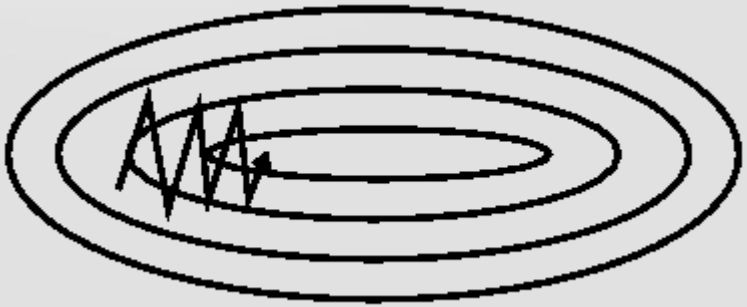
Descente stochastique

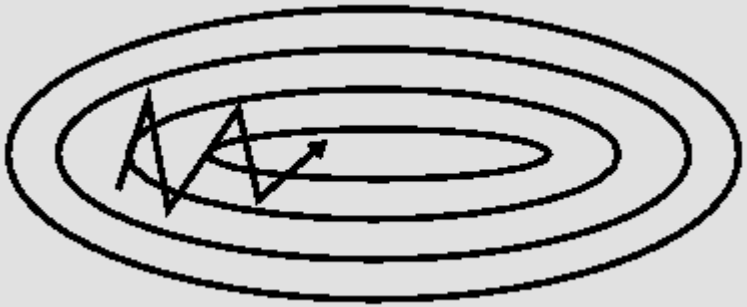
- Permet d'estimer le gradient malgré les demandes en ressources machine
- Bruité

Descente stochastique avec moment

- Lisser le gradient en utilisant une moyenne glissante exponentielle

$$Update_t = \beta . Update_{t-1} + (1 - \beta) . Grad_t \; avec \; \beta \in]0, 1[$$





Descente stochastique avec moment

- Lisser le gradient en utilisant une moyenne glissante exponentielle

$$Update_t = \beta . Update_{t-1} + (1 - \beta) . Grad_t \; avec \; \beta \in]0, 1[$$

- $\beta = 0.9$ typiquement

Descente stochastique RMSprop

- Adapter le taux d'apprentissage pour chaque paramètre
- Alimenter une moyenne glissante du carré du gradient
- Diviser le taux d'apprentissage par sa racine carrée

$$E[g^2]_t = \beta E[g^2]_{t-1} + (1-\beta)(\frac{\partial L}{\partial W})^2$$

$$w_t = w_{t-1} - \frac{\eta}{\sqrt{E[g^2]_t}} \frac{\partial L}{\partial W}$$

Descente stochastique Adam

- Combine les idées de moment et RMSprop

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\nabla f(a_{t-1})$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)\left(\nabla f(a_{t-1}) \odot \nabla f(a_{t-1})\right)$$

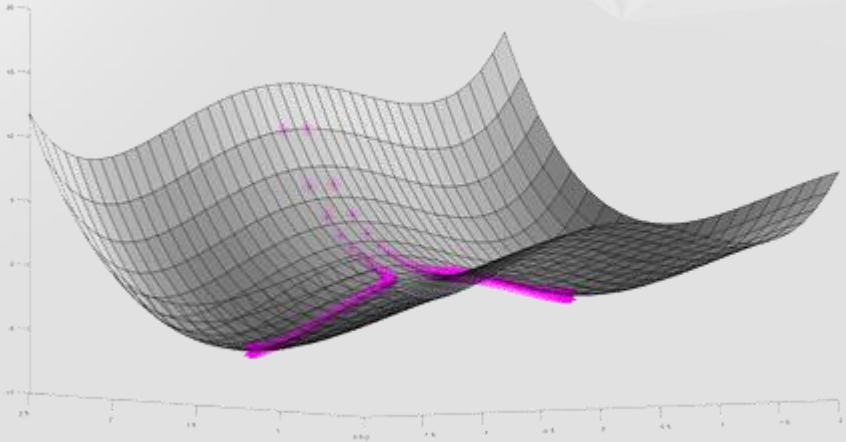
$$\hat{m}_t = \frac{m_t}{(1-\beta_1)^t}$$

$$\hat{v}_t = \frac{v_t}{(1-\beta_2)^t}$$

$$\forall i \in [\![1, n]\!], \qquad (a_t)_i = (a_{t-1})_i - \frac{\eta}{\sqrt{(\hat{v}_t)_i} + \varepsilon}(\hat{m}_t)_i$$

Méthodes d'initialisation de descente

- Crucial lorsque la fonction à optimiser n'est pas convexe



Méthodes d'initialisation Xavier

- Initialiser les paramètres d'une couche du réseau en les tirant aléatoirement d'une loi normale de moyenne nulle et de variance

$$\sigma^2 = \sqrt{\frac{2}{n^{i-1}}}$$

- Avec n le nombre de neurones présents dans la couche précédente

Commande Tensorflow: tf.keras.initializers.VarianceScaling

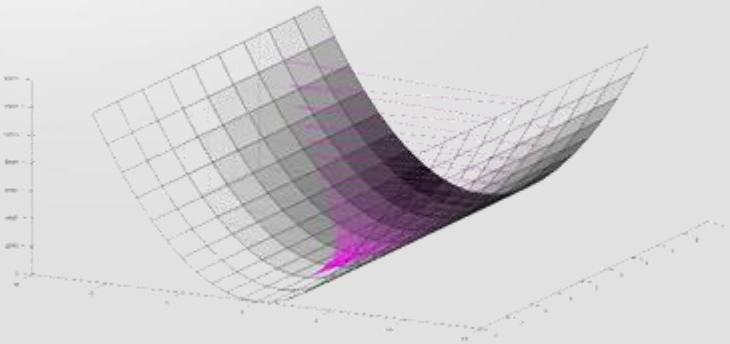
Normalisation des variables d'entrées

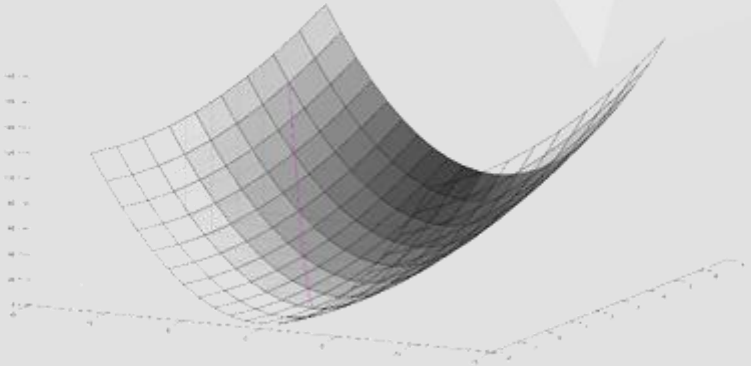
- A faire systématiquement (y compris en ML non profond)

- Rescaling des variables d'entrées de manières a ce qu'elles aient une moyenne nulle et variance unitaire

$$f(x_i) = \frac{x_i - \mu(x_i)}{\sigma(x_i)}$$

Normalisation des variables d'entrées

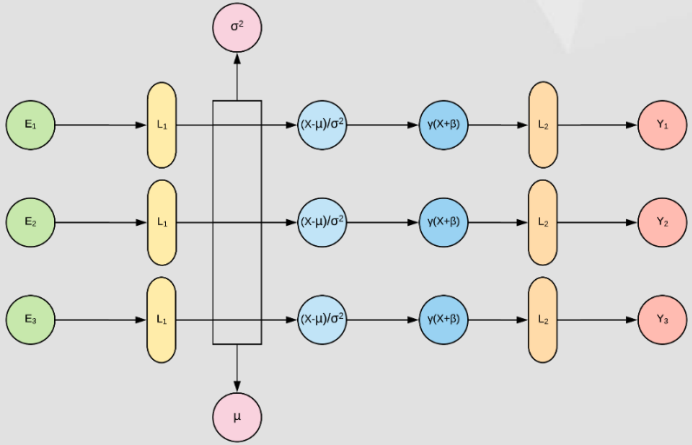




Normalisation par batch

- Les sorties d'une couche sont considerees comme des variables d'entrees pour la couche suivante

- Naturel de normaliser ces sorties



Normalisation par batch

- Comportement lors de la prédiction?

Normalisation par batch

- Comportement lors de la prédiction?
- Nécessite de tenir compte d'une moyenne coulissante des moments statistiques des sorties de chaque couche
- Couteux

Normalisation par couche

- Garder l'idée de figer les moments statistiques des sorties de chaque couche, tout en se débarrassant des moyennes coulissantes

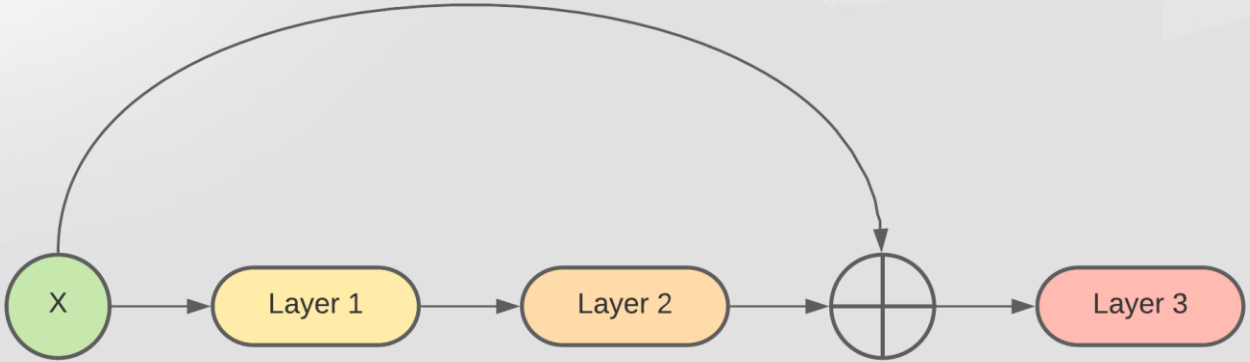
- Normaliser les sorties d'une couche pour qu'elles soient collectivement de moyenne nulle et variance unitaire

Commande Tensorflow: tf.keras.layers.LayerNormalization

Connections résiduelles

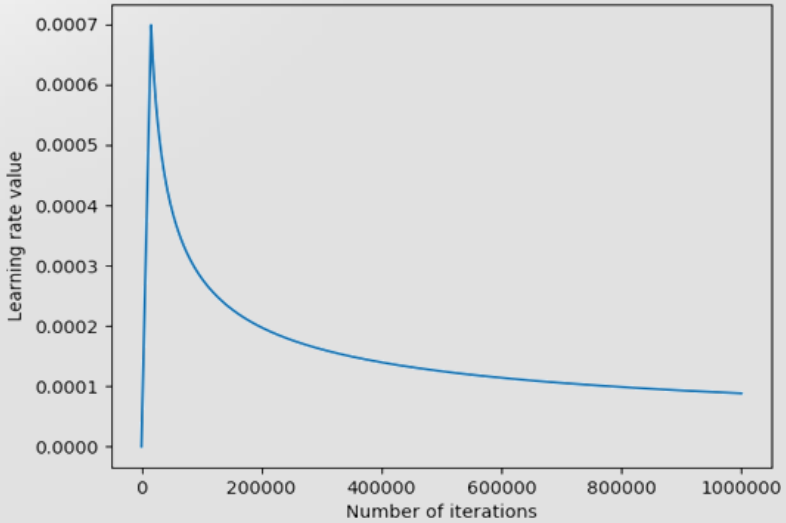
- Les phénomènes d'évaporation et explosion de gradient dépendent directement de la profondeur du réseau
- Le pouvoir explicatif aussi
- Peut on essayer de construire des réseaux très profonds qui arrivent tout de même à converger?

Connections résiduelles



Connections résiduelles

- Nécessite une gestion particulière du pas de descente
- « échauffement linéaire » en début d'ajustement



Conclusion

- Puissants modèles prédictifs
- Nécessitent un arsenal de méthodes pour converger
- Couteux en ressources machines et en données
- Intérêt peu évident tel quel

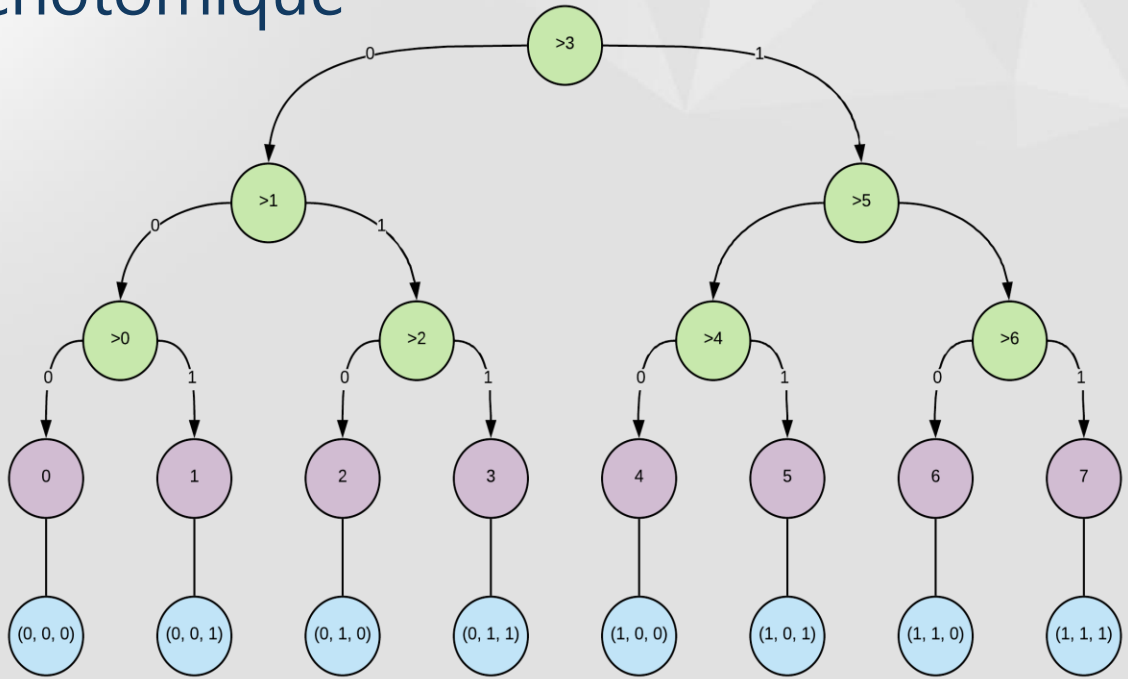
Partie 2

Apprentissage profond et analyse de séquences

Exemple préliminaire, analyse de régression ordinale

- Exercice hybride entre la classification de variables quantitatives ou qualitatives
- Méthodes a seuil ou adaptation de l'objectif de moindre carre
- Hypothèse forte (« proportional odds assumption », variable latente quantitative

Représentation d'une variable ordinale sur un arbre dichotomique



Représentation d'une variable ordinale sur un arbre dichotomique

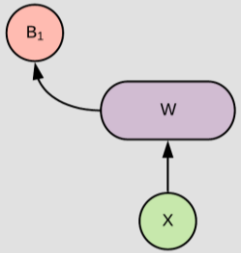
$$P(y \mid X, \theta) = P\left(\left\{B_{i,n}, i \in [\![1, n]\!]\right\} \mid X, \theta\right)$$

$$avec\ B_{i,n} = \left\lfloor \frac{y}{2^{n-i}} \right\rfloor\ mod\ 2\ \forall (i,n) \in \mathbb{N}^2$$

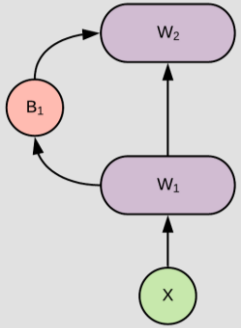
$$= P\left(B_{1,n} \mid X, \theta\right) \prod_{i=1}^{n} P\left(B_{i,n} \mid \left\{B_{j,n}, j \in [\![1, i-1]\!]\right\}, X, \theta\right)$$

Variables à prédire dépendantes entre elles

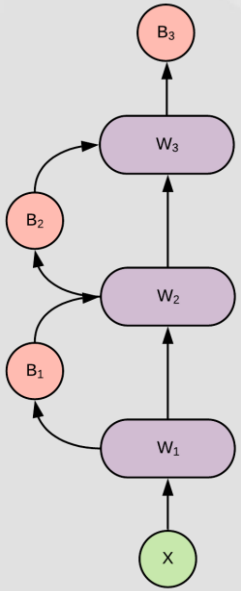
Modélisation séquentielle des variables à prédire



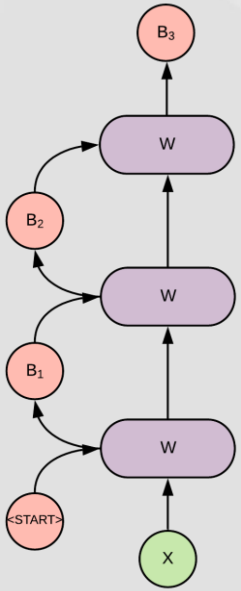
Modélisation séquentielle des variables à prédire



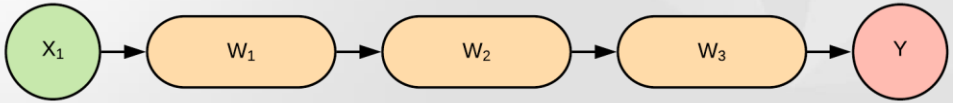
Modélisation séquentielle des variables à prédire



Modélisation séquentielle des variables à prédire

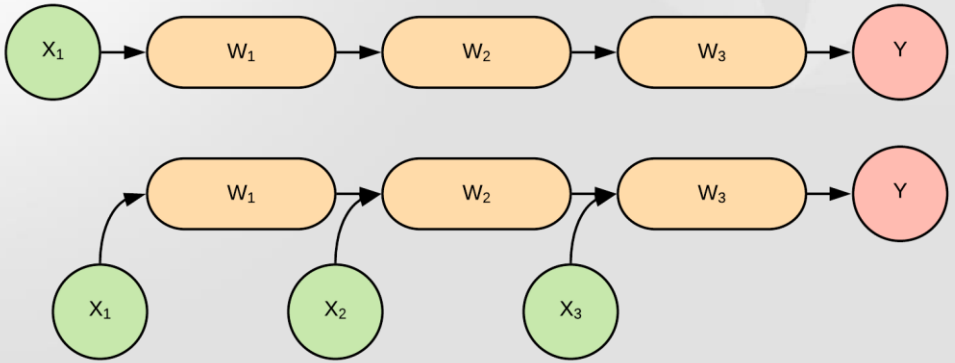


Réseau de neurones récurrent « vanille »

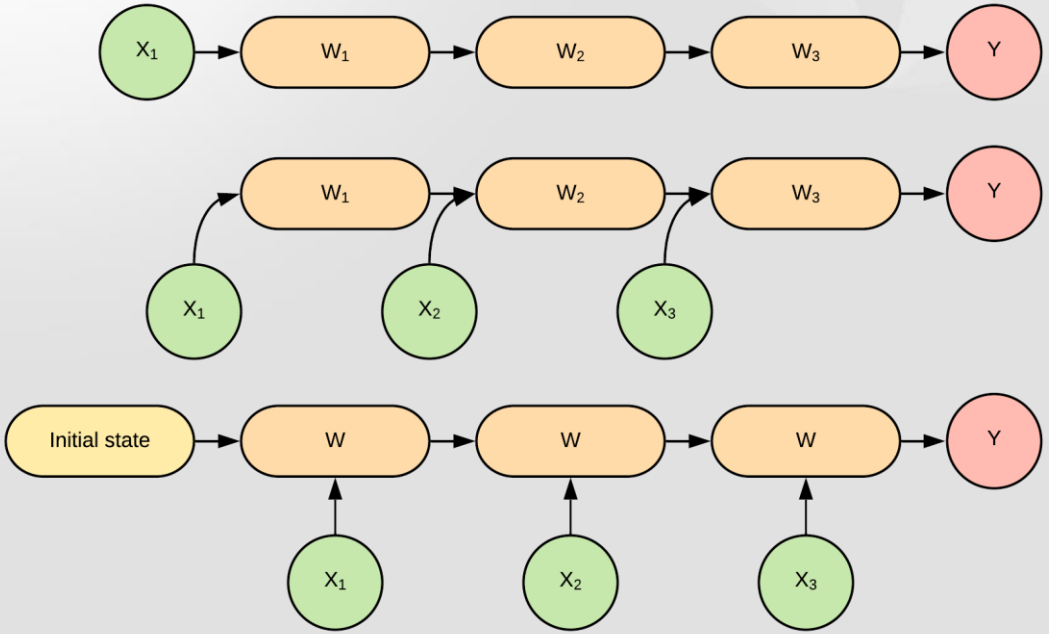


Standard multilayer perceptron

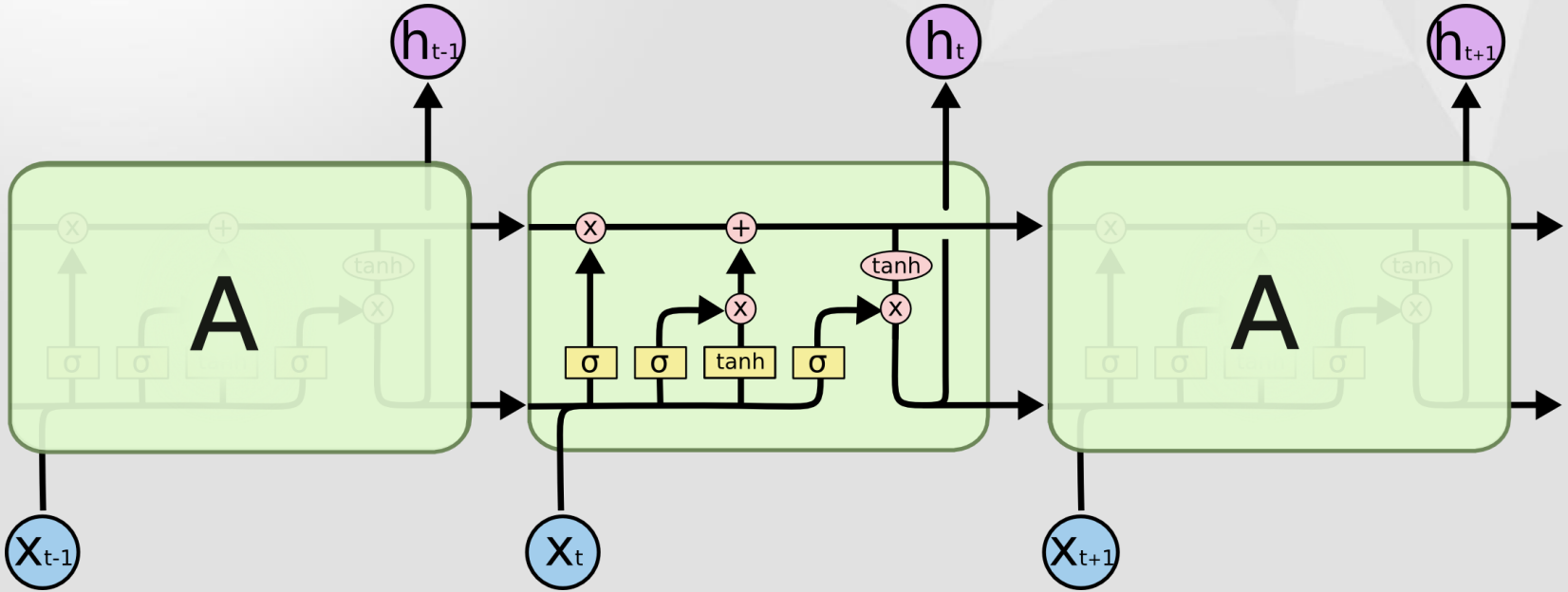
Réseau de neurones récurrent « vanille »



Réseau de neurones récurrent « vanille »



Long Short Term Memory network (LSTM)



Long Short Term Memory network (LSTM)

$$f_t = \sigma_g(W_f \cdot x_t + U_f \cdot h_{t-1} + b_f)$$

$$i_t = \sigma_g(W_i \cdot x_t + U_i \cdot h_{t-1} + b_i)$$

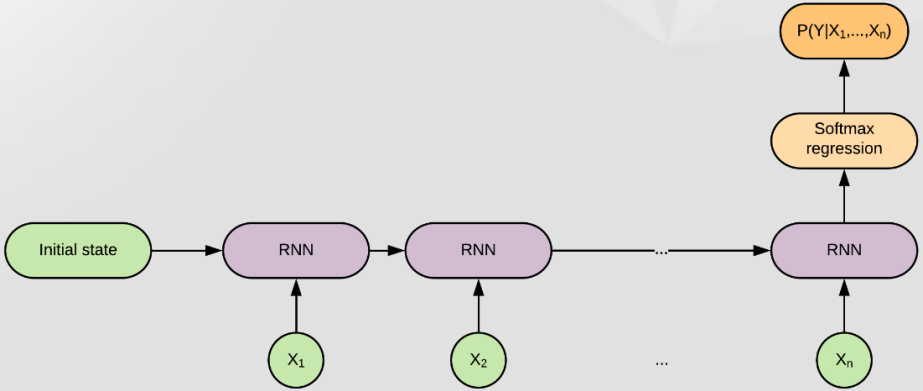
$$o_t = \sigma_g(W_o \cdot x_t + U_o \cdot h_{t-1} + b_o)$$

$$c_t = f_t \circ c_{t-1} + i_t \circ \sigma_g(W_f \cdot x_t + U_f \cdot h_{t-1} + b_f)$$

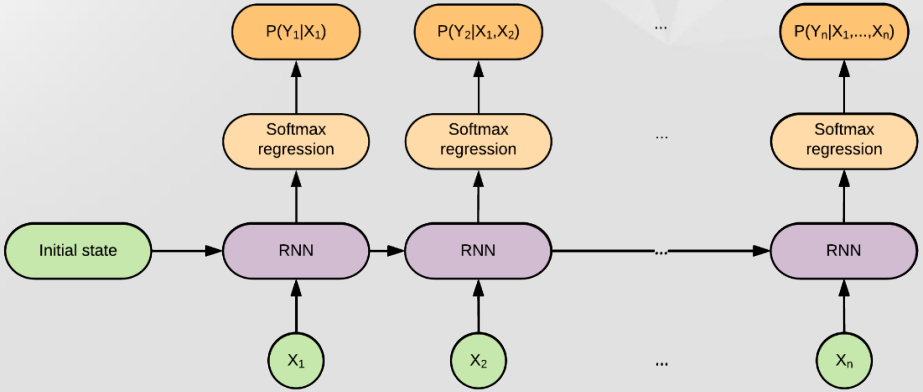
$$h_t = o_t \circ \sigma_h(c_t)$$

Commande Tensorflow: tf.keras.layers.LSTM

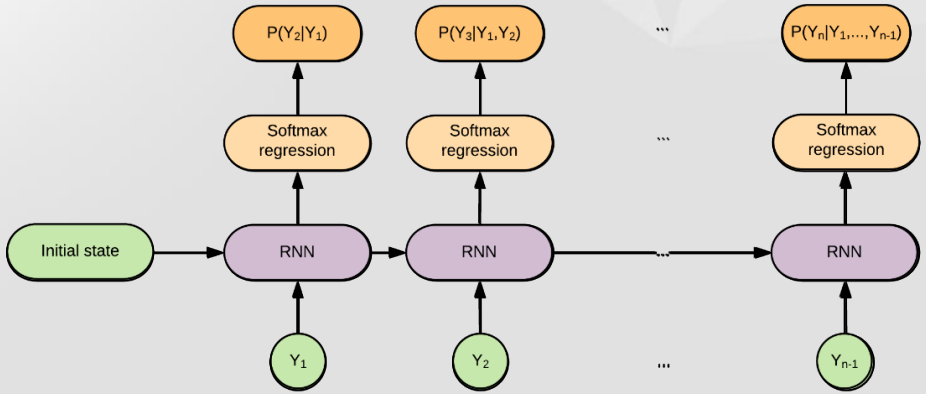
Long Short Term Memory network (LSTM)



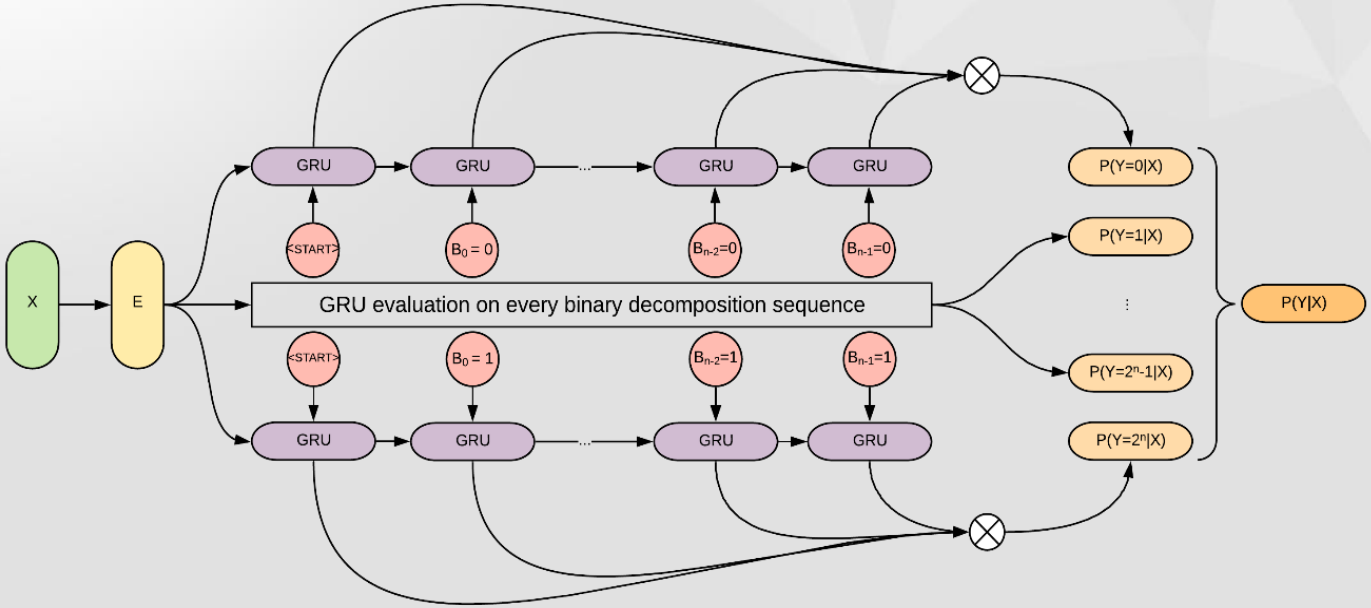
Long Short Term Memory network (LSTM)



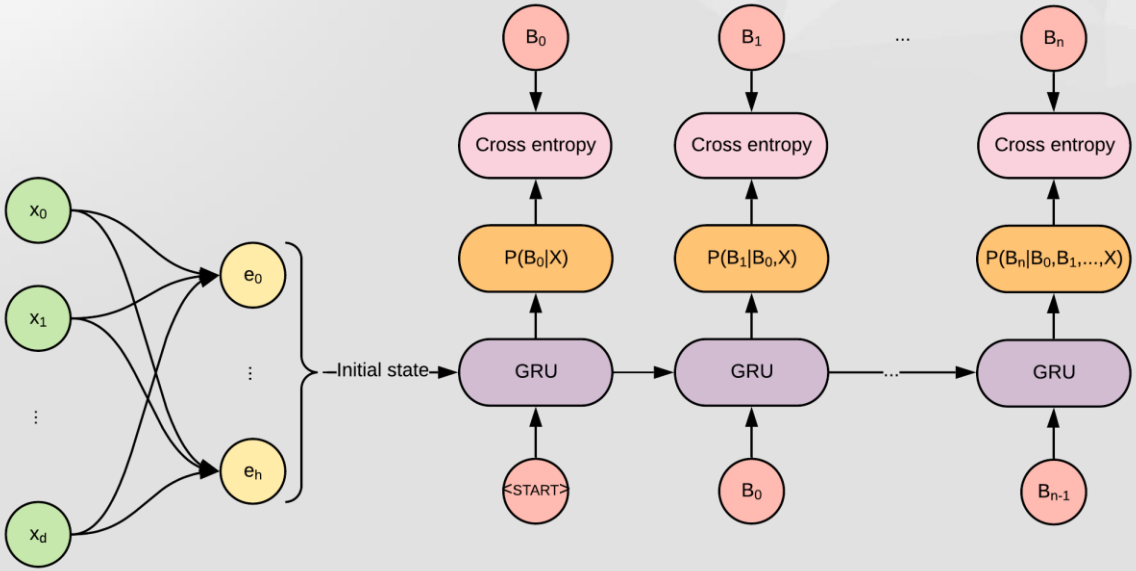
Long Short Term Memory network (LSTM)



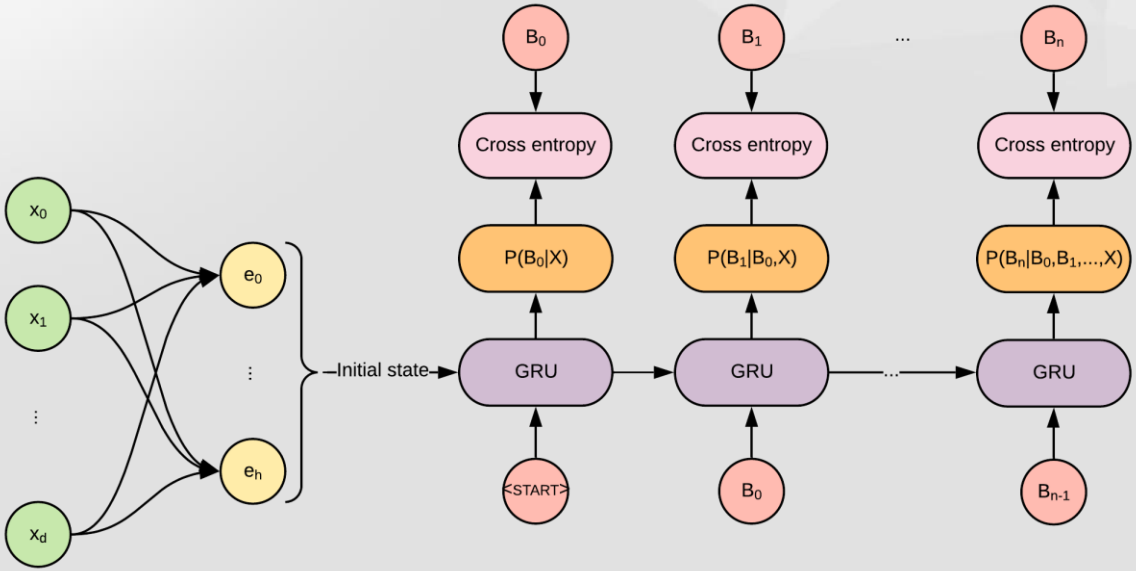
Modèle de régression ordinale



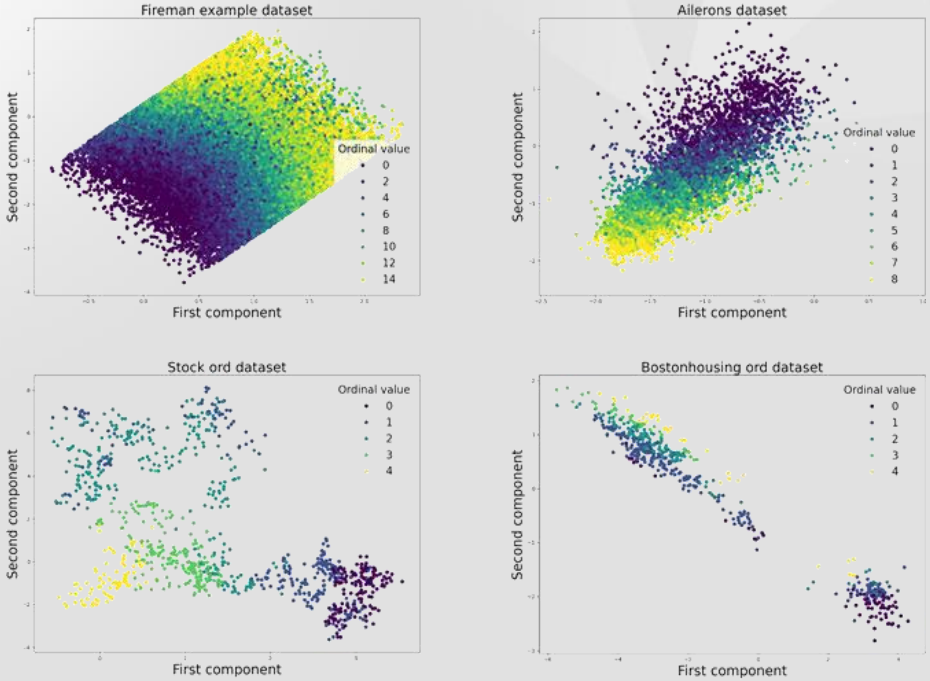
Apprentissage forcé



Apprentissage forcé



Réduction de dimensionnalité et visualisation



Seq2seq et traduction neuronale

- Objectif: construire un modèle qui à partir d'une phrase dans un langage source prédit une phrase dans un langage cible

- Corpus de phrase appariées dans les deux langues

- Idée saugrenue: modèle de classification de type régression multinomiale
  - Considérer les phrases du langage source comme une variable catégorielle avec autant d'états que de phrases possibles
  - Pareil pour les phrases du langage cible

Seq2seq et traduction neuronale

- Définition du modèle prédictif étudié

$$P(Seq_{cible}|Seq_{source},\theta) = f_\theta(Seq_{source},G)$$

- Aspect séquentiel du langage naturel

$$Seq_{source} = (w_1, w_2, \dots, w_n)$$

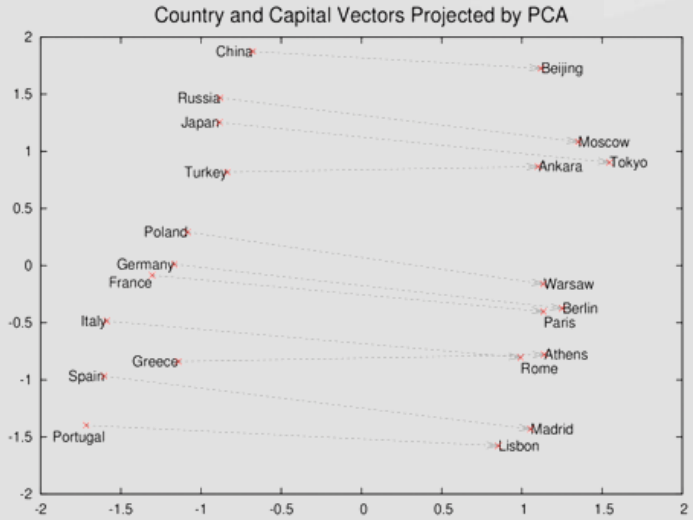
$$P(Seq_{cible}|Seq_{source},\theta) = P(w_1|Seq_{source},\theta)\prod_{i=2}^{n} P(w_i|Seq_{source},\theta,w_1,..,w_{i-1})$$

Différences avec l'exemple de régression ordinale

- Variable explicative séquentielle
  - Projection linéaire des mots
  - Construire l'état initial du décodeur avec un réseau de neurones récurrent
- Densité de probabilité jointe intraitable
  - Teacher forcing
  - Exploration des zones intéressantes de la distribution
  - Modèle autorégressif glouton ou par recherche a faisceau
- Séquences cibles a longueur variables
  - Création d'un critère d'arrêt dans l'exploration des séquences candidates (token <STOP>)

Projection linéaire de mots

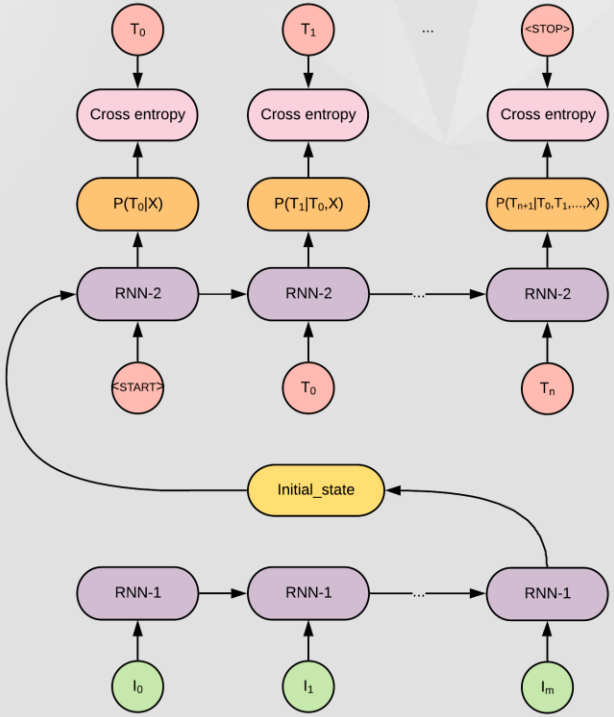
- Projeter les mots exprimes sous forme de one-hot encoding dans un espace de faible dimension (entre 32 et 512 typiquement)



Projection linéaire de mots

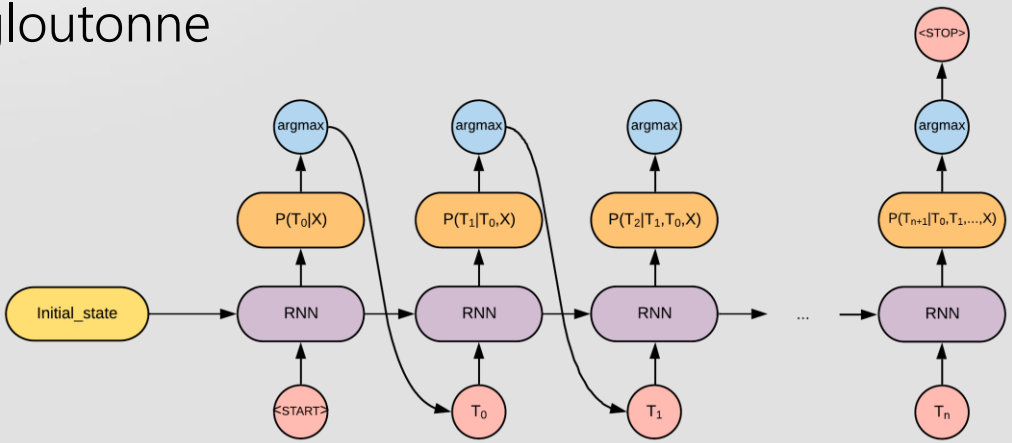
- Utiliser des embeddings pré-entraînés de manière non supervisée (word2vec, glove)
- Considérer ces embeddings comme paramètres du modèle et ajuster l'ensemble
- Hybridation des deux méthodes

Définition du modèle: Ajustement



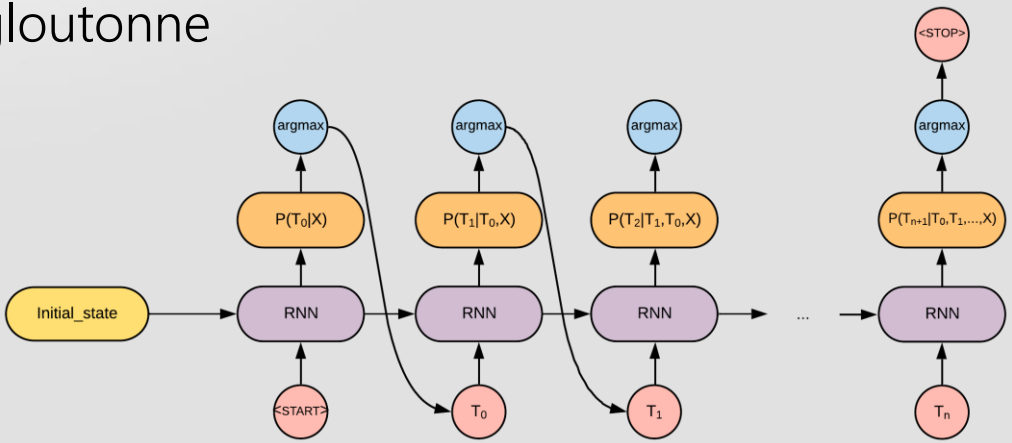
Définition du modèle: Prédiction

- Apres ajustement du modèle, possibilité d'estimer la probabilité d'une phrase cible candidate conditionnée sur une phrase source donnée
- Exploration de l'espace des phrases candidates par recherche gloutonne



Définition du modèle: Prédiction

- Apres ajustement du modèle, possibilité d'estimer la probabilité d'une phrase cible candidate conditionnée sur une phrase source donnée
- Exploration de l'espace des phrases candidates par recherche gloutonne



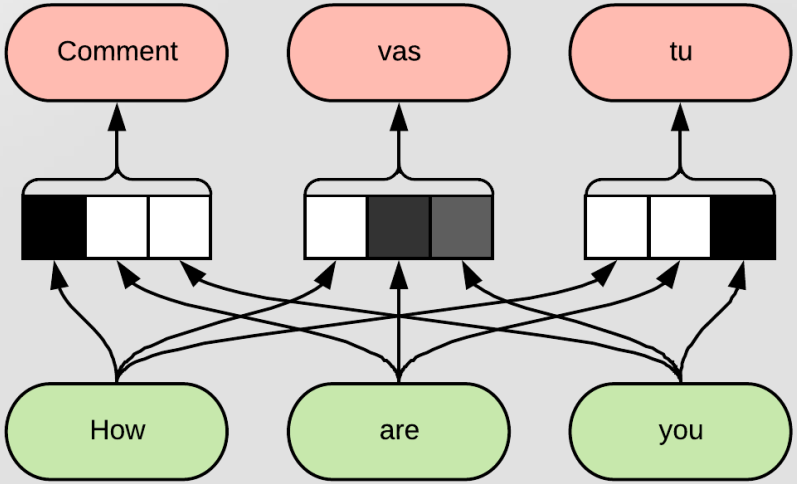
Modèles à attention

- L'architecture encodeur-décodeur converti la phrase source (de longueur variable) en un vecteur de dimensionnalité fixe

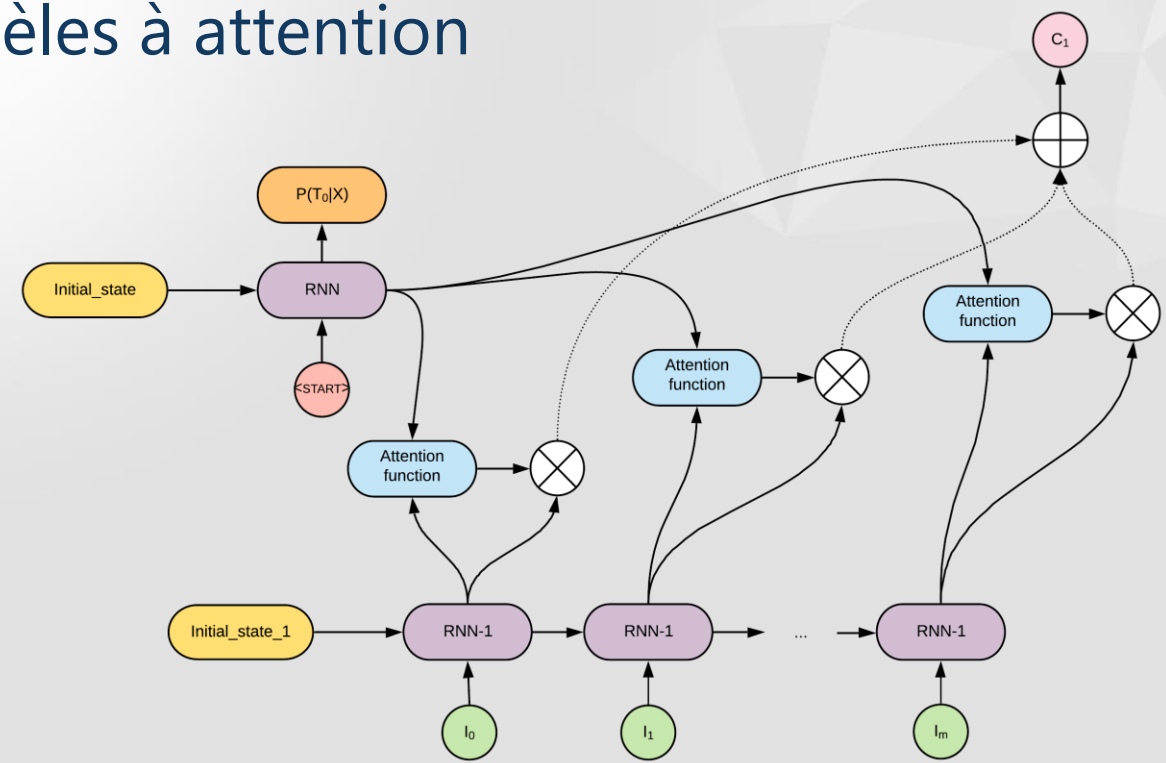
- Approche maladroite d'un point de vu humain (on ne traduit pas une phrase d'un coup mais plusieurs bouts à la fois)

Modèles à attention

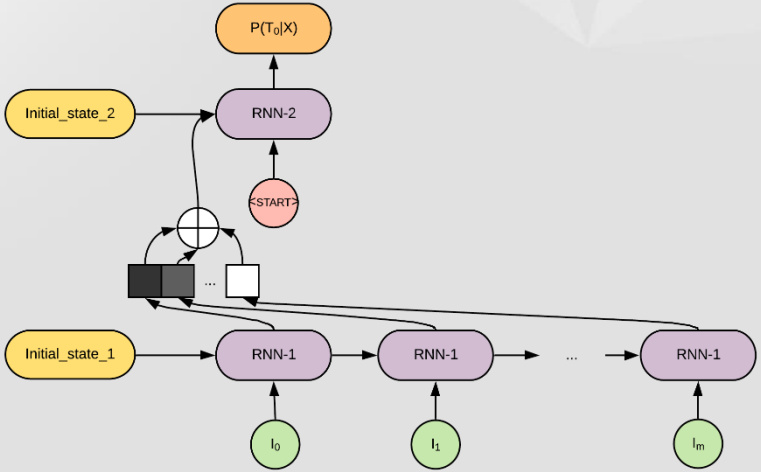
- Construire des représentations contextuelles pour chaque mot généré



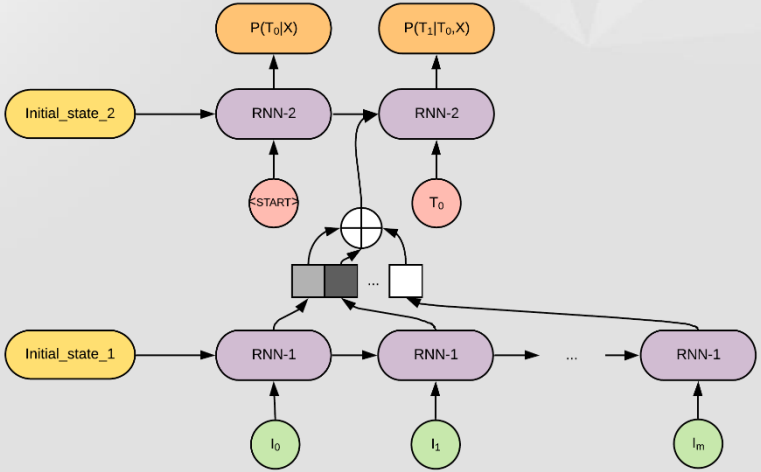
Modèles à attention



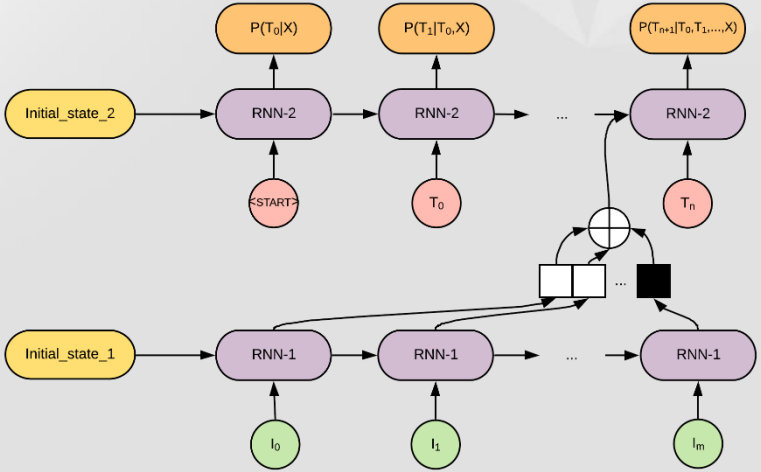
Modèles à attention



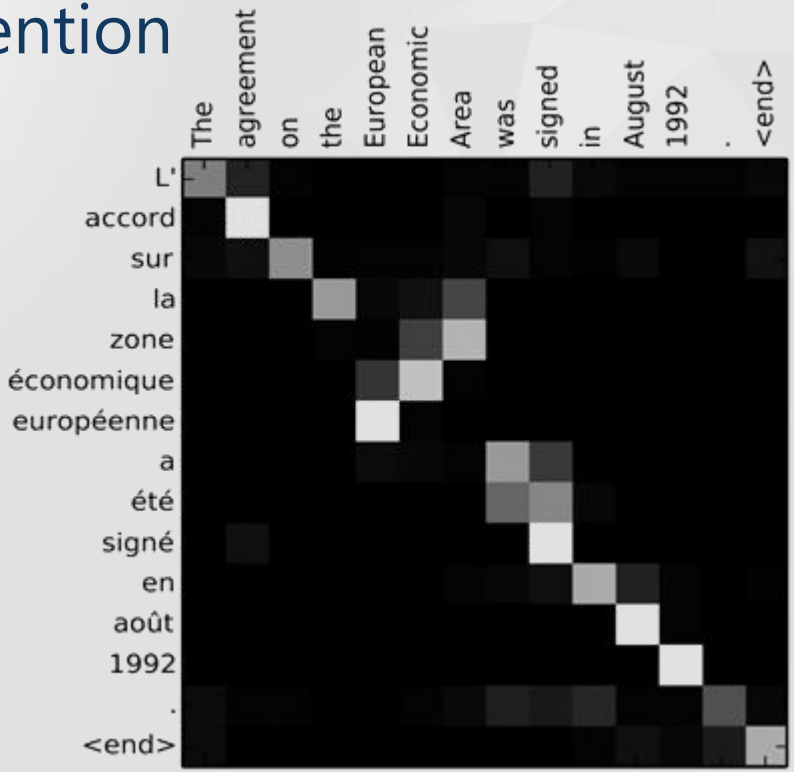
Modèles à attention



Modèles à attention



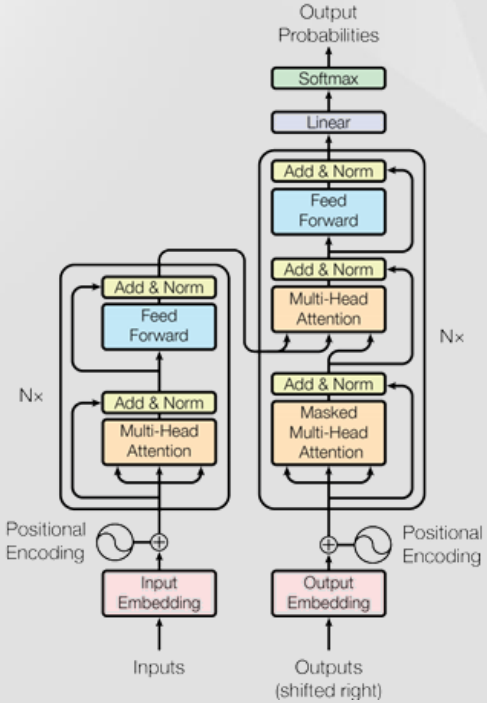
Modèles à attention



Méthodes modernes: Transformer

- RNN difficilement parallélisable
- Seulement plus performants que les méthodes à dire d'expert avec l'ajout de modules d'attention
- Peut on faire un modèle seq2seq purement attentionnel?
- Idée d'auto-attention

Méthodes modernes: Transformer



Applications pratiques

- Tokenization pour utiliser le caractère étymologique du langage et gérer les mots rares
- Plusieurs librairies et approches disponibles

- Choix du modèle dépend du nombre d'observations, particulièrement exigeant en apprentissage profond
- Ajuster un Transformer correctement demande le plus souvent plusieurs millions d'observations

BERT

- Transformer ajusté de manière non supervisée (similaire a word2vec)
- Version pré-entrainées disponibles en ligne
  - BERT classique
  - CamenBERT et FlauBERT pour des versions françaises
  - BioBERT ou ClinicalBERT pour des modèles adaptes au langage biomédical

BERT

- Deux approches possibles:

  - Considérer le modèle BERT comme une initialisation des paramètres du Transformer et réajuster de manière supervisée

  - Ajuster un petit modèle (CRF, LSTM) directement sur les sorties du BERT