

# Super Pense-bête VIP : Apprentissage profond

Afshine AMIDI et Shervine AMIDI

6 janvier 2019

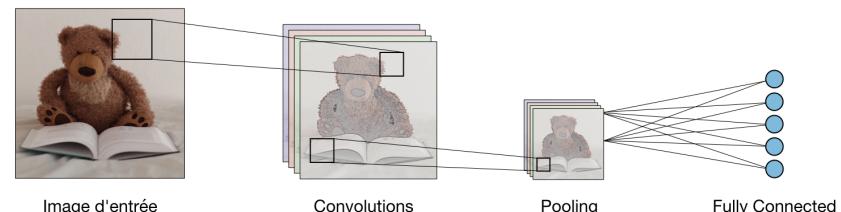
## Table des matières

|   |           |
|---|-----------|
| <b>1 Réseaux de neurones convolutionnels</b>                  | <b>1</b>  |
| 1.1 Vue d'ensemble . . . . .                                  | 1         |
| 1.2 Types de couche . . . . .                                 | 1         |
| 1.3 Paramètres du filtre . . . . .                            | 2         |
| 1.4 Réglage des paramètres . . . . .                          | 2         |
| 1.5 Fonctions d'activation communément utilisées . . . . .    | 3         |
| 1.6 Détection d'objet . . . . .                               | 3         |
| 1.6.1 Vérification et reconnaissance de visage . . . . .      | 5         |
| 1.6.2 Transfert de style neuronal . . . . .                   | 5         |
| 1.6.3 Architectures utilisant des astuces de calcul . . . . . | 6         |
| <b>2 Réseaux de neurones récurrents</b>                       | <b>7</b>  |
| 2.1 Vue d'ensemble . . . . .                                  | 7         |
| 2.2 Dépendances à long terme . . . . .                        | 8         |
| 2.3 Apprentissage de la représentation de mots . . . . .      | 9         |
| 2.3.1 Motivation et notations . . . . .                       | 9         |
| 2.3.2 Représentation de mots . . . . .                        | 9         |
| 2.4 Comparaison de mots . . . . .                             | 10        |
| 2.5 Modèle de langage . . . . .                               | 10        |
| 2.6 Traduction machine . . . . .                              | 10        |
| 2.7 Attention . . . . .                                       | 11        |
| <b>3 Petites astuces</b>                                      | <b>12</b> |
| 3.1 Traitement des données . . . . .                          | 12        |
| 3.2 Entraîner un réseau de neurones . . . . .                 | 12        |
| 3.2.1 Définitions . . . . .                                   | 12        |
| 3.2.2 Recherche de coefficients optimaux . . . . .            | 12        |
| 3.3 Réglage des paramètres . . . . .                          | 13        |
| 3.3.1 Initialisation des coefficients . . . . .               | 13        |
| 3.3.2 Optimisation de la convergence . . . . .                | 13        |
| 3.4 Régularisation . . . . .                                  | 13        |
| 3.5 Bonnes pratiques . . . . .                                | 14        |

## 1 Réseaux de neurones convolutionnels

### 1.1 Vue d'ensemble

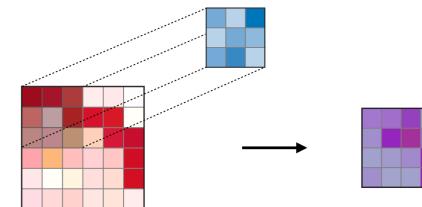
□ **Architecture d'un CNN traditionnel** – Les réseaux de neurones convolutionnels (en anglais *Convolutional neural networks*), aussi connus sous le nom de CNNs, sont un type spécifique de réseaux de neurones qui sont généralement composés des couches suivantes :



La couche convolutionnelle et la couche de pooling peuvent être ajustées en utilisant des paramètres qui sont décrites dans les sections suivantes.

### 1.2 Types de couche

□ **Couche convolutionnelle (CONV)** – La couche convolutionnelle (en anglais *convolution layer*) (CONV) utilise des filtres qui scannent l'entrée  $I$  suivant ses dimensions en effectuant des opérations de convolution. Elle peut être réglée en ajustant la taille du filtre  $F$  et le stride  $S$ . La sortie  $O$  de cette opération est appelée *feature map* ou aussi *activation map*.

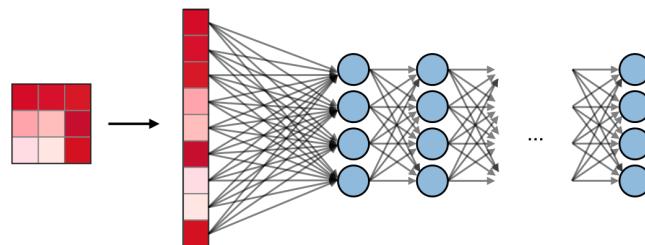


Remarque : l'étape de convolution peut aussi être généralisée dans les cas 1D et 3D.

□ **Pooling (POOL)** – La couche de pooling (en anglais *pooling layer*) (POOL) est une opération de sous-échantillonage typiquement appliquée après une couche convolutionnelle. En particulier, les types de pooling les plus populaires sont le max et l'average pooling, où les valeurs maximales et moyennes sont prises, respectivement.

|              | Max pooling  | Average pooling  |
|--------------|--|--|
| But          | Chaque opération de pooling sélectionne la valeur maximale de la surface   | Chaque opération de pooling sélectionne la valeur moyenne de la surface  |
| Illustration |  |  |
| Commentaires | <ul style="list-style-type: none"> <li>- Garde les caractéristiques détectées</li> <li>- Plus communément utilisé</li> </ul> | <ul style="list-style-type: none"> <li>- Sous-échantillonne la <i>feature map</i></li> <li>- Utilisé dans LeNet</li> </ul> |

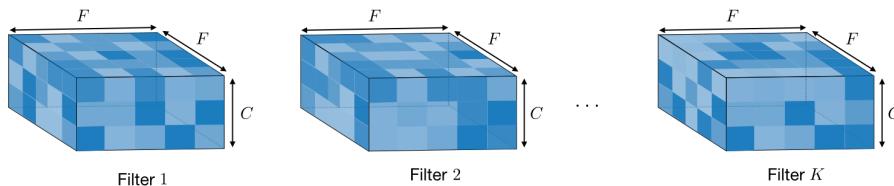
□ **Fully Connected (FC)** – La couche de fully connected (en anglais *fully connected layer*) (FC) s'applique sur une entrée préalablement aplatie où chaque entrée est connectée à tous les neurones. Les couches de fully connected sont typiquement présentes à la fin des architectures de CNN et peuvent être utilisées pour optimiser des objectifs tels que les scores de classe.



### 1.3 Paramètres du filtre

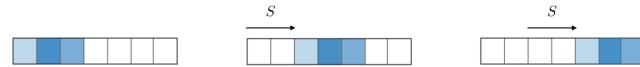
La couche convolutionnelle contient des filtres pour lesquels il est important de savoir comment ajuster ses paramètres.

□ **Dimensions d'un filtre** – Un filtre de taille  $F \times F$  appliqué à une entrée contenant  $C$  canaux est un volume de taille  $F \times F \times C$  qui effectue des convolutions sur une entrée de taille  $I \times I \times C$  et qui produit un *feature map* de sortie (aussi appelé *activation map*) de taille  $O \times O \times 1$ .



Remarque : appliquer  $K$  filtres de taille  $F \times F$  engendre un *feature map* de sortie de taille  $O \times O \times K$ .

□ **Stride** – Dans le contexte d'une opération de convolution ou de pooling, la stride  $S$  est un paramètre qui dénote le nombre de pixels par lesquels la fenêtre se déplace après chaque opération.



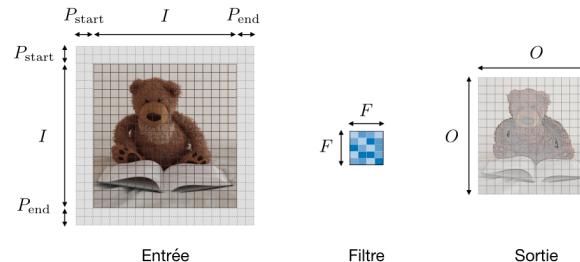
□ **Zero-padding** – Le zero-padding est une technique consistant à ajouter  $P$  zéros à chaque côté des frontières de l'entrée. Cette valeur peut être spécifiée soit manuellement, soit automatiquement par le biais d'une des configurations détaillées ci-dessous :

|              | Valide   | Pareil   | Total   |
|--------------|--|--|---|
| Valeur       | $P = 0$  | $P_{\text{start}} = \left\lceil \frac{S \lceil \frac{I}{S} \rceil - I + F - S}{2} \right\rceil$<br>$P_{\text{end}} = \left\lceil \frac{S \lceil \frac{I}{S} \rceil - I + F - S}{2} \right\rceil$   | $P_{\text{start}} \in [0, F - 1]$<br>$P_{\text{end}} = F - 1$   |
| Illustration |  |  |   |
| But          | <ul style="list-style-type: none"> <li>- Pas de padding</li> <li>- Enlève la dernière opération de convolution si les dimensions ne collent pas</li> </ul> | <ul style="list-style-type: none"> <li>- Le padding tel que la feature map est de taille <math>\left\lceil \frac{I}{S} \right\rceil</math></li> <li>- La taille de sortie est mathématiquement satisfaisante</li> <li>- Aussi appelé 'demi' padding</li> </ul> | <ul style="list-style-type: none"> <li>- Padding maximum tel que les dernières convolutions sont appliquées sur les bords de l'entrée</li> <li>- Le filtre 'voit' l'entrée du début à la fin</li> </ul> |

### 1.4 Réglage des paramètres

□ **Compatibilité des paramètres dans la couche convolutionnelle** – En notant  $I$  le côté du volume d'entrée,  $F$  la taille du filtre,  $P$  la quantité de zero-padding,  $S$  la stride, la taille  $O$  de la feature map de sortie suivant cette dimension est telle que :

$$O = \frac{I - F + P_{\text{start}} + P_{\text{end}}}{S} + 1$$



Remarque : on a souvent  $P_{\text{start}} = P_{\text{end}} \triangleq P$ , auquel cas on remplace  $P_{\text{start}} + P_{\text{end}}$  par  $2P$  dans la formule au-dessus.

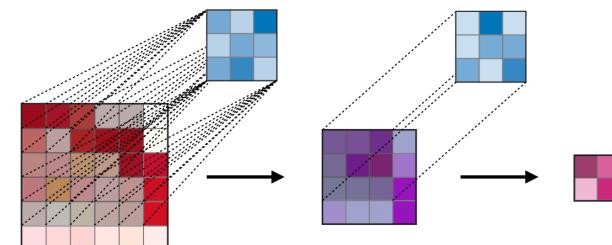
**Comprendre la complexité du modèle** – Pour évaluer la complexité d'un modèle, il est souvent utile de déterminer le nombre de paramètres que l'architecture va avoir. Dans une couche donnée d'un réseau de neurones convolutionnels, on a :

|                      | CONV  | POOL  | FC  |
|----------------------|---|---|---|
| Illustration         |   |   |   |
| Taille d'entrée      | $I \times I \times C$   | $I \times I \times C$   | $N_{\text{in}}$   |
| Taille de sortie     | $O \times O \times K$   | $O \times O \times C$   | $N_{\text{out}}$  |
| Nombre de paramètres | $(F \times F \times C + 1) \cdot K$   | 0   | $(N_{\text{in}} + 1) \times N_{\text{out}}$   |
| Remarques            | <ul style="list-style-type: none"> <li>- Un paramètre de biais par filtre</li> <li>- Dans la plupart des cas, <math>S &lt; F</math></li> <li>- <math>2C</math> est un choix commun pour <math>K</math></li> </ul> | <ul style="list-style-type: none"> <li>- L'opération de pooling est effectuée pour chaque canal</li> <li>- Dans la plupart des cas, <math>S = F</math></li> </ul> | <ul style="list-style-type: none"> <li>- L'entrée est aplatie</li> <li>- Un paramètre de biais par neurone</li> <li>- Le choix du nombre de neurones de FC est libre</li> </ul> |

**Champ récepteur** – Le champ récepteur à la couche  $k$  est la surface notée  $R_k \times R_k$  de l'entrée que chaque pixel de la  $k$ -ième *activation map* peut 'voir'. En notant  $F_j$  la taille du filtre de la couche  $j$  et  $S_i$  la valeur de stride de la couche  $i$  et avec la convention  $S_0 = 1$ , le champ récepteur à la couche  $k$  peut être calculé de la manière suivante :

$$R_k = 1 + \sum_{j=1}^k (F_j - 1) \prod_{i=0}^{j-1} S_i$$

Dans l'exemple ci-dessous, on a  $F_1 = F_2 = 3$  et  $S_1 = S_2 = 1$ , ce qui donne  $R_2 = 1+2 \cdot 1+2 \cdot 1 = 5$ .



## 1.5 Fonctions d'activation communément utilisées

**Unité linéaire rectifiée** – La couche d'unité linéaire rectifiée (en anglais *rectified linear unit layer*) (ReLU) est une fonction d'activation  $g$  qui est utilisée sur tous les éléments du volume. Elle a pour but d'introduire des complexités non-linéaires au réseau. Ses variantes sont récapitulées dans le tableau suivant :

| ReLU  | Leaky ReLU   | ELU   |
|---|--|---|
| $g(z) = \max(0, z)$   | $g(z) = \max(\epsilon z, z)$ with $\epsilon \ll 1$ | $g(z) = \max(\alpha(e^z - 1), z)$ with $\alpha \ll 1$ |
|   |  |   |
| Complexités non-linéaires interprétables d'un point de vue biologique | Répond au problème de <i>dying ReLU</i>            | Dérivable partout                                     |

**Softmax** – L'étape softmax peut être vue comme une généralisation de la fonction logistique qui prend comme argument un vecteur de scores  $x \in \mathbb{R}^n$  et qui renvoie un vecteur de probabilités  $p \in \mathbb{R}^n$  à travers une fonction softmax à la fin de l'architecture. Elle est définie de la manière suivante :

$$p = \begin{pmatrix} p_1 \\ \vdots \\ p_n \end{pmatrix} \quad \text{where} \quad p_i = \frac{e^{x_i}}{\sum_{j=1}^n e^{x_j}}$$

## 1.6 Détection d'objet

**Types de modèles** – Il y a 3 principaux types d'algorithme de reconnaissance d'objet, pour lesquels la nature de ce qui est prédit est différent. Ils sont décrits dans la table ci-dessous :

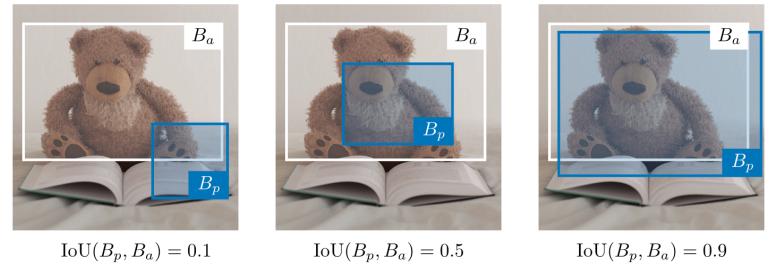
| Classification d'image                                      | Classification avec localisation  | Détection   |
|---|---|---|
| Ours en peluche   | Ours en peluche   | Ours en peluche<br>Livre  |
| - Classifie une image<br>- Prédit la probabilité d'un objet | - Déetecte un objet dans une image<br>- Prédit la probabilité de présence d'un objet et où il est situé | - Peut détecter plusieurs objets dans une image<br>- Prédit les probabilités de présence des objets et où ils sont situés |
| CNN traditionnel  | YOLO simplifié, R-CNN   | YOLO, R-CNN   |

□ **Détection** – Dans le contexte de la détection d'objet, des méthodes différentes sont utilisées selon si l'on veut juste localiser l'objet ou alors détecter une forme plus complexe dans l'image. Les deux méthodes principales sont résumées dans le tableau ci-dessous :

| Détection de zone délimitante                                | Détection de forme complexe   |
|--|---|
| Déetecte la partie de l'image où l'objet est situé           | - Déetecte la forme ou les caractéristiques d'un objet (e.g. yeux)<br>- Plus granulaire |
|  |   |
| Zone de centre $(b_x, b_y)$ , hauteur $b_h$ et largeur $b_w$ | Points de référence $(l_{1x}, l_{1y}), \dots, (l_{nx}, l_{ny})$                         |

□ **Intersection sur Union** – Intersection sur Union (en anglais *Intersection over Union*), aussi appelé IoU, est une fonction qui quantifie à quel point la zone délimitante prédite  $B_p$  est correctement positionnée par rapport à la zone délimitante vraie  $B_a$ . Elle est définie de la manière suivante :

$$\text{IoU}(B_p, B_a) = \frac{B_p \cap B_a}{B_p \cup B_a}$$

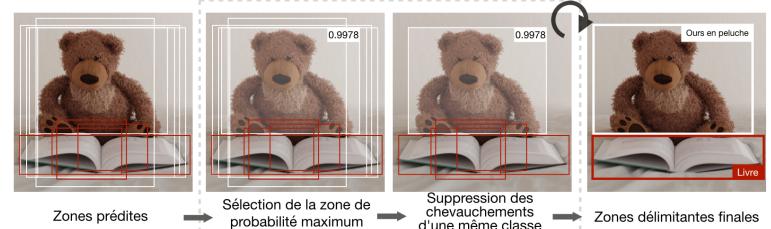


Remarque : on a toujours  $\text{IoU} \in [0,1]$ . Par convention, la prédiction  $B_p$  d'une zone délimitante est considérée comme étant satisfaisante si l'on a  $\text{IoU}(B_p, B_a) \geq 0.5$ .

□ **Zone d'accroche** – La technique des zones d'accroche (en anglais *anchor boxing*) sert à prédire des zones délimitantes qui se chevauchent. En pratique, on permet au réseau de prédire plus d'une zone délimitante simultanément, où chaque zone prédict doit respecter une forme géométrique particulière. Par exemple, la première prédiction peut potentiellement être une zone rectangulaire d'une forme donnée, tandis qu'une seconde prédiction doit être une zone rectangulaire d'une autre forme.

□ **Suppression non-max** – La technique de suppression non-max (en anglais *non-max suppression*) a pour but d'enlever des zones délimitantes qui se chevauchent et qui prédisent un seul et même objet, en sélectionnant les zones les plus représentatives. Après avoir enlevé toutes les zones ayant une probabilité prédite de moins de 0.6, les étapes suivantes sont répétées pour éliminer les zones redondantes : Pour une classe donnée,

- Étape 1 : Choisir la zone ayant la plus grande probabilité de prédiction.
- Étape 2 : Enlever toute zone ayant  $\text{IoU} \geq 0.5$  avec la zone choisie précédemment.



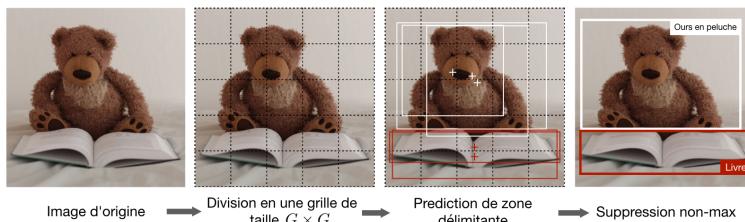
□ **YOLO** – L'algorithme You Only Look Once (YOLO) est un algorithme de détection d'objet qui fonctionne de la manière suivante :

- Étape 1 : Diviser l'image d'entrée en une grille de taille  $G \times G$ .
- Étape 2 : Pour chaque cellule, faire tourner un CNN qui prédit  $y$  de la forme suivante :

$$y = \underbrace{[p_c, b_x, b_y, b_h, b_w, c_1, c_2, \dots, c_p, \dots]}_k^T \in \mathbb{R}^{G \times G \times k \times (5+p)}$$

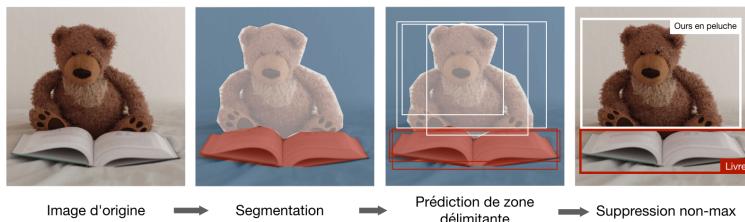
où  $p_c$  est la probabilité de détecter un objet,  $b_x, b_y, b_h, b_w$  sont les propriétés de la zone délimitante détectée,  $c_1, \dots, c_p$  est une représentation binaire (en anglais *one-hot representation*) de l'une des  $p$  classes détectée, et  $k$  est le nombre de zones d'accroche.

- Étape 3 : Faire tourner l'algorithme de suppression non-max pour enlever des doublons potentiels qui chevauchent des zones délimitantes.



Remarque : lorsque  $p_c = 0$ , le réseau ne détecte plus d'objet. Dans ce cas, les prédictions correspondantes  $b_x, \dots, c_p$  doivent être ignorées.

**R-CNN** – L'algorithme de région avec des réseaux de neurones convolutionnels (en anglais *Region with Convolutional Neural Networks*) (R-CNN) est un algorithme de détection d'objet qui segmente l'image d'entrée pour trouver des zones délimitantes pertinentes, puis fait tourner un algorithme de détection pour trouver les objets les plus probables d'apparaître dans ces zones délimitantes.



Remarque : bien que l'algorithme original soit lent et coûteux en temps de calcul, de nouvelles architectures ont permis de faire tourner l'algorithme plus rapidement, tels que le Fast R-CNN et le Faster R-CNN.

### 1.6.1 Vérification et reconnaissance de visage

**Types de modèles** – Deux principaux types de modèle sont récapitulés dans le tableau ci-dessous :

| Vérification de visage                    | Reconnaissance de visage   |
|---|--|
| - Est-ce la bonne personne ?<br>- Un à un | - Est-ce une des $K$ personnes dans la base de données ?<br>- Un à plusieurs |
| Requête<br><br>Référence<br>              | Requête<br><br>Base de données<br>   |

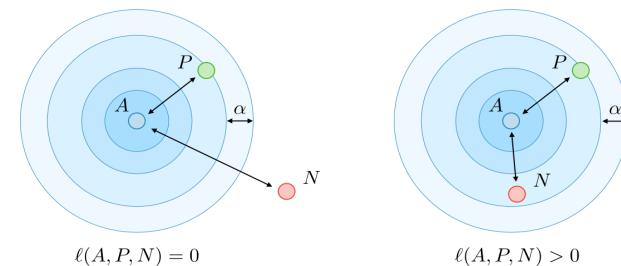
**Apprentissage par coup** – L'apprentissage par coup (en anglais *One Shot Learning*) est un algorithme de vérification de visage qui utilise un training set de petite taille pour apprendre

une fonction de similarité qui quantifie à quel point deux images données sont différentes. La fonction de similarité appliquée à deux images est souvent notée  $d(\text{image 1}, \text{image 2})$ .

**Réseaux siamois** – Les réseaux siamois (en anglais *Siamese Networks*) ont pour but d'apprendre comment encoder des images pour quantifier le degré de différence de deux images données. Pour une image d'entrée donnée  $x^{(i)}$ , l'encodage de sortie est souvent notée  $f(x^{(i)})$ .

**Loss triple** – Le loss triple (en anglais *triplet loss*)  $\ell$  est une fonction de loss calculée sur une représentation encodée d'un triplet d'images  $A$  (accroche),  $P$  (positif), et  $N$  (négatif). L'exemple d'accroche et l'exemple positif appartiennent à la même classe, tandis que l'exemple négatif appartient à une autre. En notant  $\alpha \in \mathbb{R}^+$  le paramètre de marge, le loss est défini de la manière suivante :

$$\ell(A, P, N) = \max(d(A, P) - d(A, N) + \alpha, 0)$$



### 1.6.2 Transfert de style neuronal

**Motivation** – Le but du transfert de style neuronal (en anglais *neural style transfer*) est de générer une image  $G$  à partir d'un contenu  $C$  et d'un style  $S$ .



**Activation** – Dans une couche  $l$  donnée, l'activation est notée  $a^{[l]}$  et est de dimensions  $n_H \times n_w \times n_c$ .

**Fonction de coût de contenu** – La fonction de coût de contenu (en anglais *content cost function*), notée  $J_{\text{content}}(C, G)$ , est utilisée pour quantifier à quel point l'image générée  $G$  diffère de l'image de contenu original  $C$ . Elle est définie de la manière suivante :

$$J_{\text{content}}(C, G) = \frac{1}{2} \|a^{[l](C)} - a^{[l](G)}\|^2$$

**Matrice de style** – La matrice de style (en anglais *style matrix*)  $G^{[l]}$  d'une couche  $l$  donnée est une matrice de Gram dans laquelle chacun des éléments  $G_{kk'}^{[l]}$  quantifie le degré de corrélation des canaux  $k$  and  $k'$ . Elle est définie en fonction des activations  $a^{[l]}$  de la manière suivante :

$$G_{kk'}^{[l]} = \sum_{i=1}^{n_H^{[l]}} \sum_{j=1}^{n_w^{[l]}} a_{ijk}^{[l]} a_{ijk'}^{[l]}$$

Remarque : les matrices de style de l'image de style et de l'image générée sont notées  $G^{[l]}(S)$  and  $G^{[l]}(G)$  respectivement.

**Fonction de coût de style** – La fonction de coût de style (en anglais *style cost function*), notée  $J_{\text{style}}(S, G)$ , est utilisée pour quantifier à quel point l'image générée  $G$  diffère de l'image de style  $S$ . Elle est définie de la manière suivante :

$$J_{\text{style}}^{[l]}(S, G) = \frac{1}{(2n_H n_w n_c)^2} \|G^{[l]}(S) - G^{[l]}(G)\|_F^2 = \frac{1}{(2n_H n_w n_c)^2} \sum_{k,k'=1}^{n_c} \left( G_{kk'}^{[l]}(S) - G_{kk'}^{[l]}(G) \right)^2$$

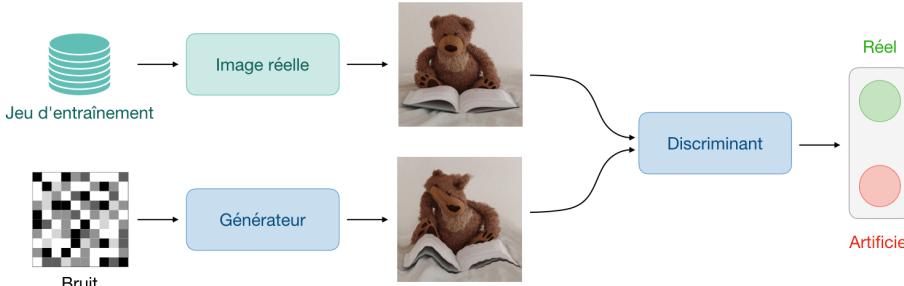
**Fonction de coût total** – La fonction de coût total (en anglais *overall cost function*) est définie comme étant une combinaison linéaire des fonctions de coût de contenu et de style, pondérées par les paramètres  $\alpha, \beta$ , de la manière suivante :

$$J(G) = \alpha J_{\text{content}}(C, G) + \beta J_{\text{style}}(S, G)$$

Remarque : plus  $\alpha$  est grand, plus le modèle privilégiera le contenu et plus  $\beta$  est grand, plus le modèle sera fidèle au style.

### 1.6.3 Architectures utilisant des astuces de calcul

**Réseau antagoniste génératif** – Les réseaux antagonistes génératifs (en anglais *generative adversarial networks*), aussi connus sous le nom de GANs, sont composés d'un modèle génératif et d'un modèle discriminatif, où le modèle génératif a pour but de générer des prédictions aussi réalistes que possibles, qui seront ensuite envoyées dans un modèle discriminatif qui aura pour but de différencier une image générée d'une image réelle.



Remarque : les GANs sont utilisées dans des applications pouvant aller de la génération de musique au traitement de texte vers image.

**ResNet** – L'architecture du réseau résiduel (en anglais *Residual Network*), aussi appelé ResNet, utilise des blocs résiduels avec un nombre élevé de couches et a pour but de réduire l'erreur de training. Le bloc résiduel est caractérisé par l'équation suivante :

$$a^{[l+2]} = g(a^{[l]} + z^{[l+2]})$$

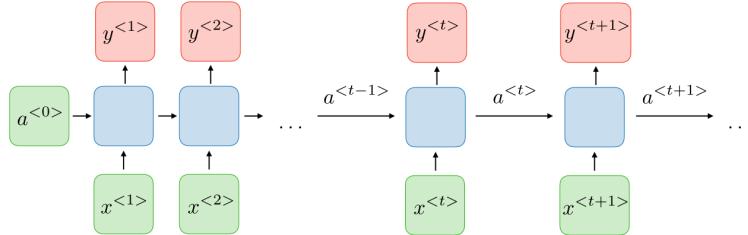
**Inception Network** – Cette architecture utilise des modules d'*inception* et a pour but de tester toute sorte de configuration de convolution pour améliorer sa performance en diversifiant ses attributs. En particulier, elle utilise l'astuce de la convolution  $1 \times 1$  pour limiter sa complexité de calcul.

\* \* \*

## 2 Réseaux de neurones récurrents

### 2.1 Vue d'ensemble

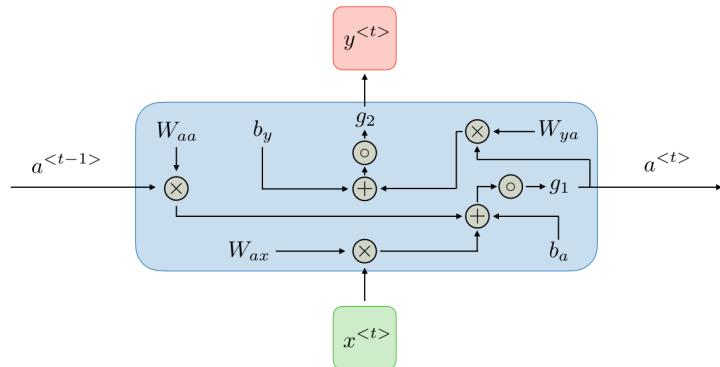
**Architecture d'un RNN traditionnel** – Les réseaux de neurones récurrents (en anglais *recurrent neural networks*), aussi appelés RNNs, sont une classe de réseaux de neurones qui permettent aux prédictions antérieures d'être utilisées comme entrées, par le biais d'états cachés (en anglais *hidden states*). 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"> <li>- Prend en compte des entrées de toute taille</li> <li>- La taille du modèle n'augmente pas avec la taille de l'entrée</li> <li>- Les calculs prennent en compte les infos antérieures</li> <li>- Les coefficients sont indépendants du temps</li> </ul> | <ul style="list-style-type: none"> <li>- Le temps de calcul est long</li> <li>- Difficulté d'accéder à des informations d'un passé lointain</li> <li>- Impossibilité de prendre en compte des informations futures un état donné</li> </ul> |

**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<br>$T_x = T_y = 1$              |              | Réseau de neurones traditionnel |
| Un à plusieurs<br>$T_x = 1, T_y > 1$    |              | Génération de musique           |
| Plusieurs à un<br>$T_x > 1, T_y = 1$    |              | Classification de sentiment     |
| Plusieurs à plusieurs<br>$T_x = T_y$    |              | Reconnaissance d'entité         |
| Plusieurs à plusieurs<br>$T_x \neq T_y$ |              | 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 :

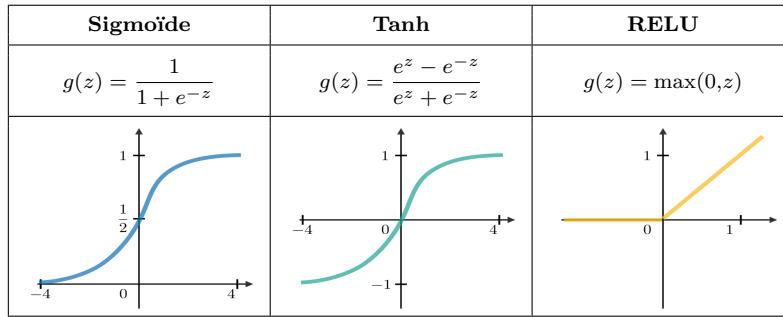
$$\mathcal{L}(\hat{y}, y) = \sum_{t=1}^{T_y} \mathcal{L}(\hat{y}^{<t>}, y^{<t>})$$

□ **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 :

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

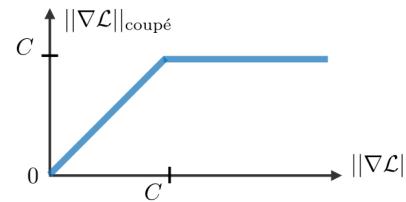
## 2.2 Dépendances à long terme

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



□ **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.

□ **Coupe 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.



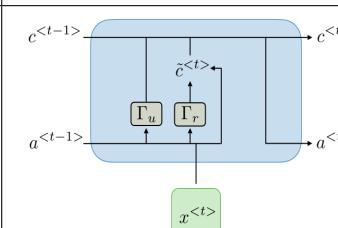
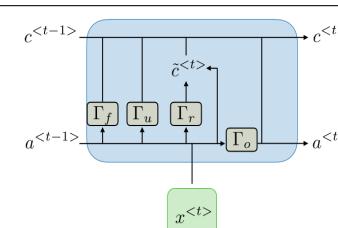
□ **Types de porte** – Pour remédier au problème du gradient qui disparaît, certains types de porte sont spécifiquement utilisés dans des variantes de RNNs et ont un but bien défini. Les portes sont souvent notées  $\Gamma$  et sont telles que :

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

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

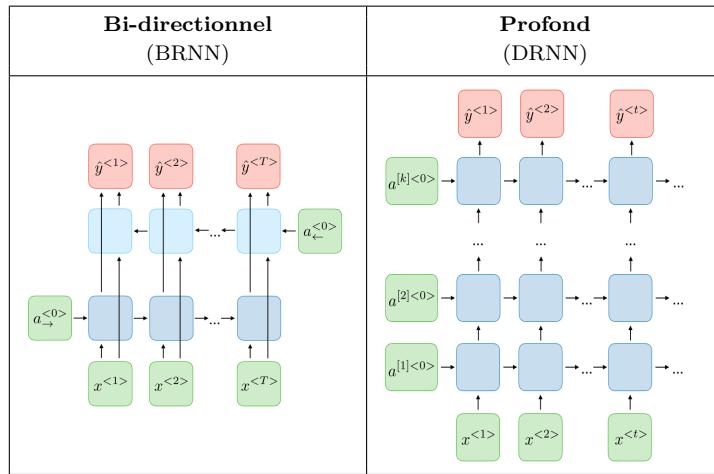
| Type de porte                    | Rôle   | Utilisée dans |
|----------------------------------|--|---------------|
| Porte d'actualisation $\Gamma_u$ | Dans quelle mesure le passé devrait être important ? | GRU, LSTM     |
| Porte de pertinence $\Gamma_r$   | Enlever les informations précédentes ?               | GRU, LSTM     |
| Porte d'oubli $\Gamma_f$         | Enlever une cellule ?                                | LSTM          |
| Porte de sortie $\Gamma_o$       | 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) appaissent 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)   |
|-------------------|---|---|
| $\tilde{c}^{<t>}$ | $\tanh(W_c[\Gamma_r * a^{<t-1>}, x^{<t>}] + b_c)$                                     | $\tanh(W_c[\Gamma_r * a^{<t-1>}, x^{<t>}] + b_c)$                                     |
| $c^{<t>}$         | $\Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) * c^{<t-1>}$                             | $\Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>}$                                   |
| $a^{<t>}$         | $c^{<t>}$   | $\Gamma_o * c^{<t>}$  |
| Dépendances       |  |  |

Remarque : le signe  $*$  dénote le produit de Hadamard entre deux vecteurs.

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



## 2.3 Apprentissage de la représentation de mots

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

### 2.3.1 Motivation et notations

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

| Représentation binaire  | Représentation du mot                                    |
|---|--|
|   |  |
| - Noté $o_w$<br>- Approche naïve, pas d'information de similarité | - Noté $e_w$<br>- Prend en compte la similarité des mots |

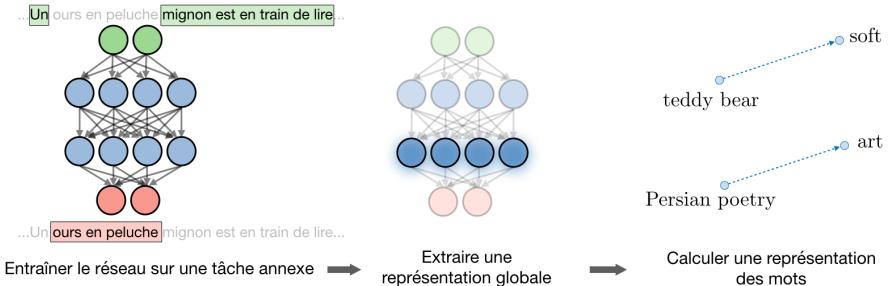
□ **Matrice de représentation** – Pour un mot donné  $w$ , la matrice de représentation (en anglais *embedding matrix*)  $E$  est une matrice qui relie une représentation binaire  $o_w$  à sa représentation correspondante  $e_w$  de la manière suivante :

$$e_w = E o_w$$

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

### 2.3.2 Représentation de mots

□ **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  $\theta_t$  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 classificateurs 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  $\theta$  ont dans un modèle, la représentation du mot final  $e_w^{(\text{final})}$  est donnée par :

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

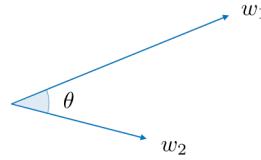
*Remarque : les composantes individuelles de la représentation d'un mot n'est pas nécessairement facilement interprétable.*

## 2.4 Comparaison de mots

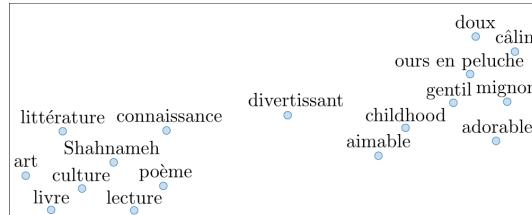
□ **Similarité cosinus** – La similarité cosinus (en anglais *cosine similarity*) entre les mots  $w_1$  et  $w_2$  est donnée par :

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

*Remarque :  $\theta$  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.



## 2.5 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 :

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

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

## 2.6 Traduction machine

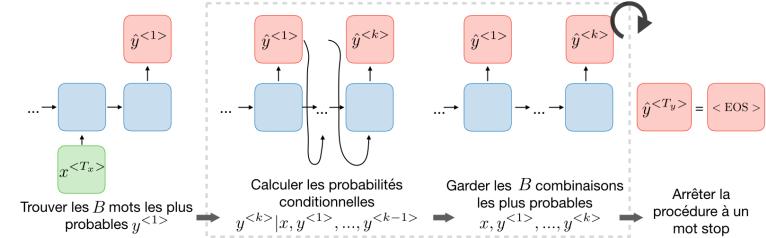
□ **Vue d'ensemble** – Un modèle de traduction machine est similaire à un modèle de langage ayant un auto-encodeur placé en amont. Pour cette raison, ce modèle est souvent surnommé modèle conditionnel de langage.

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  $\alpha$  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<br>- Régulariser<br>- 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 :

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

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

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

\* \* \*

## 2.7 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 :

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

### 3 Petites astuces

#### 3.1 Traitement des données

**Augmentation des données** – Les modèles d'apprentissage profond ont typiquement besoin de beaucoup de données afin d'être entraînés convenablement. Il est souvent utile de générer plus de données à partir de celles déjà existantes à l'aide de techniques d'augmentation de données. Celles les plus souvent utilisées sont résumées dans le tableau ci-dessous. À partir d'une image, voici les techniques que l'on peut utiliser :

| Original                  | Symétrie axiale   | Rotation  | Recadrage aléat.   |
|---------------------------|---|---|--|
|                           |   |   |  |
| - Image sans aucune modif | - Symétrie par rapport à un axe pour lequel le sens de l'image est conservé | - Rotation avec un petit angle<br>- Reproduit une calibration imparfaite de l'horizon | - Concentration aléatoire sur une partie de l'image<br>- Plusieurs rognements aléatoires peuvent être faits à la suite |

| Ch. de couleur   | Addition de bruit  | Perte d'info.  | Ch. de contraste  |
|--|--|--|---|
|  |  |  |   |
| - Nuances de RGB sont légèrement changées<br>- Capture le bruit qui peut survenir avec de l'exposition lumineuse | - Addition de bruit<br>- Plus de tolérance envers la variation de la qualité de l'entrée | - Parties de l'image ignorées<br>- Imité des pertes potentielles de parties de l'image | - Changement de luminosité<br>- Contrôle la différence de l'exposition dû à l'heure de la journée |

**Normalisation de lot** – La normalisation de lot (en anglais *batch normalization*) est une étape qui normalise le lot  $\{x_i\}$  avec un choix de paramètres  $\gamma, \beta$ . En notant  $\mu_B, \sigma_B^2$  la moyenne et la variance de ce que l'on veut corriger du lot, on a :

$$x_i \leftarrow \gamma \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} + \beta$$

Ceci est couramment fait après un fully connected/couche de convolution et avant une couche non-linéaire. Elle vise à permettre d'avoir de plus grands taux d'apprentissages et de réduire la dépendance à l'initialisation.

#### 3.2 Entrainer un réseau de neurones

##### 3.2.1 Définitions

**Epoch** – Dans le contexte de l'entraînement d'un modèle, l'*epoch* est un terme utilisé pour réferer à une itération où le modèle voit tout le training set pour mettre à jour ses coefficients.

**Gradient descent sur mini-lots** – Durant la phase d'entraînement, la mise à jour des coefficients n'est souvent basée ni sur tout le training set d'un coup à cause de temps de calculs coûteux, ni sur un seul point à cause de bruits potentiels. À la place de cela, l'étape de mise à jour est faite sur des mini-lots, où le nombre de points dans un lot est un paramètre que l'on peut régler.

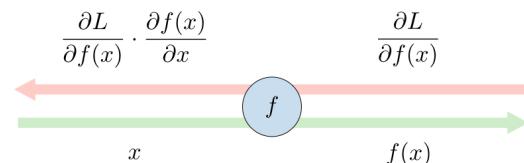
**Fonction de loss** – Pour pouvoir quantifier la performance d'un modèle donné, la fonction de loss (en anglais *loss function*)  $L$  est utilisée pour évaluer la mesure dans laquelle les sorties vraies  $y$  sont correctement prédites par les prédictions du modèle  $z$ .

**Entropie croisée** – Dans le contexte de la classification binaire d'un réseau de neurones, l'entropie croisée (en anglais *cross-entropy loss*)  $L(z,y)$  est couramment utilisée et est définie par :

$$L(z,y) = -[y \log(z) + (1-y) \log(1-z)]$$

##### 3.2.2 Recherche de coefficients optimaux

**Backpropagation** – La backpropagation est une méthode de mise à jour des coefficients d'un réseau de neurones en prenant en compte les sorties vraies et désirées. La dérivée par rapport à chaque coefficient  $w$  est calculée en utilisant la règle de la chaîne.

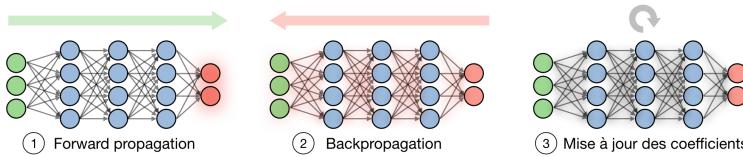


En utilisant cette méthode, chaque coefficient est mis à jour par :

$$w \leftarrow w - \alpha \frac{\partial L(z,y)}{\partial w}$$

**Mise à jour les coefficients** – Dans un réseau de neurones, les coefficients sont mis à jour par :

- Étape 1 : Prendre un lot de training data et effectuer une forward propagation pour calculer le loss.
- Étape 2 : Backpropaguer le loss pour obtenir le gradient du loss par rapport à chaque coefficient.
- Étape 3 : Utiliser les gradients pour mettre à jour les coefficients du réseau.



### 3.3 Réglage des paramètres

#### 3.3.1 Initialisation des coefficients

**Initialisation de Xavier** – Au lieu de laisser les coefficients s'initialiser de manière purement aléatoire, l'initialisation de Xavier permet d'avoir des coefficients initiaux qui prennent en compte les caractéristiques uniques de l'architecture.

**Apprentissage par transfert** – Entrainer un modèle d'apprentissage profond requiert beaucoup de données et beaucoup de temps. Il est souvent utile de profiter de coefficients pré-entraînés sur des données énormes qui ont pris des jours/semaines pour être entraînés, et profiter de cela pour notre cas. Selon la quantité de données que l'on a sous la main, voici différentes manières d'utiliser cette méthode :

| Taille du training | Illustration | Explication   |
|--------------------|--------------|---|
| Petite             |              | Gèle toutes les couches, entraîne les coefficients du softmax   |
| Moyenne            |              | Gèle la plupart des couches, entraîne les coefficients des dernières couches et du softmax                              |
| Grande             |              | Entraîne les coefficients des couches et du softmax en initialisant les coefficients sur ceux qui ont été pré-entraînés |

#### 3.3.2 Optimisation de la convergence

**Taux d'apprentissage** – Le taux d'apprentissage (en anglais *learning rate*), souvent noté  $\alpha$  ou  $\eta$ , indique la vitesse à laquelle les coefficients sont mis à jour. Il peut être fixe ou variable. La

méthode actuelle la plus populaire est appelée Adam, qui est une méthode faisant varier le taux d'apprentissage.

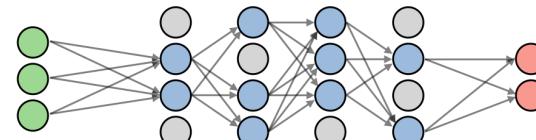
**Taux d'apprentissage adaptatifs** – Laisser le taux d'apprentissage varier pendant la phase d'entraînement du modèle peut réduire le temps d'entraînement et améliorer la qualité de la solution numérique optimale. Bien que la méthode d'Adam est la plus utilisée, d'autres peuvent aussi être utiles. Les différentes méthodes sont récapitulées dans le tableau ci-dessous :

| Méthode  | Explication  | Mise à jour de $w$                                   | Mise à jour de $b$  |
|----------|--|--|---|
| Momentum | - Amortit les oscillations<br>- Amélioration par rapport à la méthode SGD<br>- 2 paramètres à régler     | $w - \alpha v_{dw}$                                  | $b - \alpha v_{db}$   |
| RMSprop  | - Root Mean Square propagation<br>- Accélère l'algorithme d'apprentissage en contrôlant les oscillations | $w - \alpha \frac{dw}{\sqrt{s_{dw}}}$                | $b \leftarrow b - \alpha \frac{db}{\sqrt{s_{db}}}$                |
| Adam     | - Adaptive Moment estimation<br>- Méthode la plus populaire<br>- 4 paramètres à régler                   | $w - \alpha \frac{v_{dw}}{\sqrt{s_{dw}} + \epsilon}$ | $b \leftarrow b - \alpha \frac{v_{db}}{\sqrt{s_{db}} + \epsilon}$ |

Remarque : parmi les autres méthodes existantes, on trouve Adadelta, Adagrad et SGD.

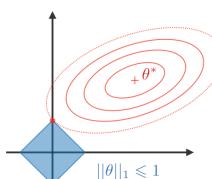
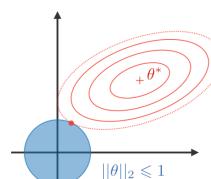
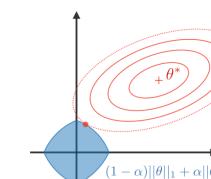
### 3.4 Régularisation

**Dropout** – Le dropout est une technique qui est destinée à empêcher le sur-ajustement sur les données de training en abandonnant des unités dans un réseau de neurones avec une probabilité  $p > 0$ . Cela force le modèle à éviter de trop s'appuyer sur un ensemble particulier de features.



Remarque : la plupart des frameworks d'apprentissage profond paramètrent le dropout à travers le paramètre 'garder'  $1 - p$ .

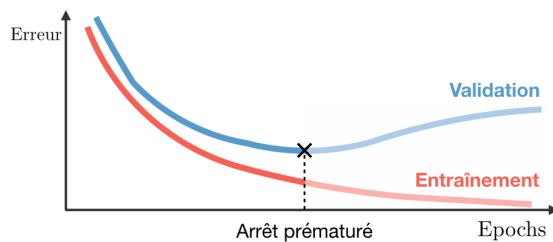
**Régularisation de coefficient** – Pour s'assurer que les coefficients ne sont pas trop grands et que le modèle ne sur-ajuste pas sur le training set, on utilise des techniques de régularisation sur les coefficients du modèle. Les techniques principales sont résumées dans le tableau suivant :

| LASSO   | Ridge   | Elastic Net  |
|---|---|--|
| - Réduit les coefficients à 0<br>- Bon pour la sélection de variables             | Rend les coefficients plus petits   | Compromis entre la sélection de variables et la réduction de la taille des coefficients  |
|  |  |   |
| $\dots + \lambda \ \theta\ _1$<br>$\lambda \in \mathbb{R}$                        | $\dots + \lambda \ \theta\ _2^2$<br>$\lambda \in \mathbb{R}$                      | $\dots + \lambda \left[ (1 - \alpha) \ \theta\ _1 + \alpha \ \theta\ _2^2 \right]$<br>$\lambda \in \mathbb{R}, \alpha \in [0,1]$ |

|              | Gradient numérique   | Gradient analytique   |
|--------------|--|---|
| Formule      | $\frac{df}{dx}(x) \approx \frac{f(x+h) - f(x-h)}{2h}$  | $\frac{df}{dx}(x) = f'(x)$  |
| Commentaires | <ul style="list-style-type: none"> <li>- Coûteux ; le loss doit être calculé deux fois par dimension</li> <li>- Utilisé pour vérifier l'exactitude d'une implémentation analytique</li> <li>- Compromis dans le choix de h entre pas trop petit (instabilité numérique) et pas trop grand (estimation du gradient approximatif)</li> </ul> | <ul style="list-style-type: none"> <li>- Résultat 'exact'</li> <li>- Calcul direct</li> <li>- Utilisé dans l'implémentation finale</li> </ul> |

★ ★ ★

□ **Arrêt prématué** – L'arrêt prématué (en anglais *early stopping*) est une technique de régularisation qui consiste à stopper l'étape d'entraînement dès que le loss de validation atteint un plateau ou commence à augmenter.



### 3.5 Bonnes pratiques

□ **Surapprentissage d'un mini-lot** – Lorsque l'on débuge un modèle, il est souvent utile de faire de petits tests pour voir s'il y a un gros souci avec l'architecture du modèle lui-même. En particulier, pour s'assurer que le modèle peut être entraîné correctement, un mini-lot est passé dans le réseau pour voir s'il peut sur-ajuster sur lui. Si le modèle ne peut pas le faire, cela signifie que le modèle est soit trop complexe ou pas assez complexe pour être sur-ajusté sur un mini-lot.

□ **Vérification de gradient** – La méthode de la vérification de gradient (en anglais *gradient checking*) est utilisée durant l'implémentation d'un backward pass d'un réseau de neurones. Elle compare la valeur du gradient analytique par rapport au gradient numérique au niveau de certains points et joue un rôle de vérification élémentaire.