[Apprentissage Automatique]
Réseaux de Neurones Convolutionnels

Anthony Larcher, thanks to Loïc Barrault

anthony.larcher@univ-lemans.fr
Le Mans Université

1er décembre 2020

Plan

- La convolution
  - Définitions et (un peu) d'arithmétique
- CNN pour le texte
  - du RNN au CNN
  - Convolution pour le texte
  - Applications
    - Classification
    - Traduction automatique

Sources principales

- "Convolutional Neural Network (for NLP)", R. Socher
  - `http://cs224d.stanford.edu/`
- "A guide to convolution arithmetic for deep learning", V. Dumoulin & F. Visin
  - `https://arxiv.org/abs/1603.07285`

La convolution

- Définition générale (convolution discrète, 1 dimension)

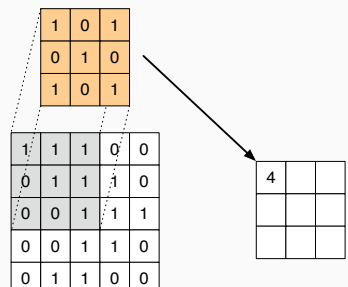
$$(f * g)(n) = \sum_{m=-M}^{M} f(n-m)g(m)$$

- Très utile pour extraire des caractéristiques d'une image
- Exemple en 2 dimensions
  - filtre :

1	0	1
0	1	0
1	0	1

  image :

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0



- filtre = **kernel**
- image : **input feature map**

La convolution

- Définition générale (convolution discrète, 1 dimension)

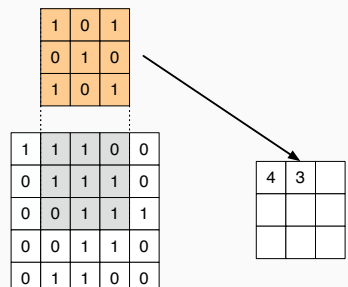
$$(f * g)(n) = \sum_{m=-M}^{M} f(n-m)g(m)$$

- Très utile pour extraire des caractéristiques d'une image
- Exemple en 2 dimensions
  - filtre :

1	0	1
0	1	0
1	0	1

  image :

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0



- filtre = **kernel**
- image : **input feature map**

La convolution

- Définition générale (convolution discrète, 1 dimension)

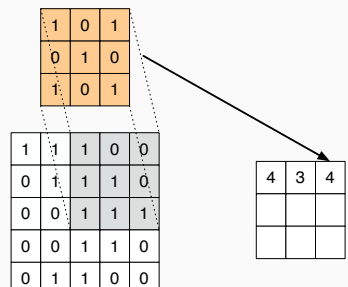
$$(f * g)(n) = \sum_{m=-M}^{M} f(n-m)g(m)$$

- Très utile pour extraire des caractéristiques d'une image
- Exemple en 2 dimensions
  - filtre :

1	0	1
0	1	0
1	0	1

  - image :

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0



- filtre = **kernel**
- image : **input feature map**

La convolution

- Définition générale (convolution discrète, 1 dimension)

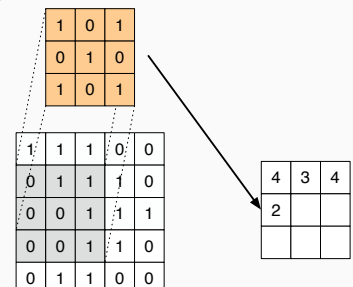
$$(f * g)(n) = \sum_{m=-M}^{M} f(n-m)g(m)$$

- Très utile pour extraire des caractéristiques d'une image
- Exemple en 2 dimensions
  - filtre :

1	0	1
0	1	0
1	0	1

  image :

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0



- filtre = **kernel**
- image : **input feature map**

La convolution

- Définition générale (convolution discrète, 1 dimension)

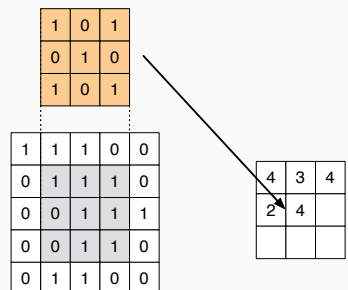
$$(f * g)(n) = \sum_{m=-M}^{M} f(n-m)g(m)$$

- Très utile pour extraire des caractéristiques d'une image
- Exemple en 2 dimensions
  - filtre :

1	0	1
0	1	0
1	0	1

  image :

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0



- filtre = **kernel**
- image : **input feature map**

La convolution

- Définition générale (convolution discrète, 1 dimension)

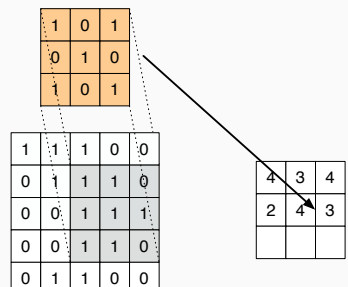
$$(f * g)(n) = \sum_{m=-M}^{M} f(n-m)g(m)$$

- Très utile pour extraire des caractéristiques d'une image
- Exemple en 2 dimensions
  - filtre :

1	0	1
0	1	0
1	0	1

  - image :

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0



- filtre = **kernel**
- image : **input feature map**

La convolution

- Définition générale (convolution discrète, 1 dimension)

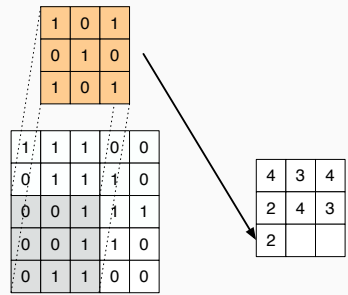
$$(f * g)(n) = \sum_{m=-M}^{M} f(n-m)g(m)$$

- Très utile pour extraire des caractéristiques d'une image
- Exemple en 2 dimensions
  - filtre :

1	0	1
0	1	0
1	0	1

  - image :

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0



- filtre = **kernel**
- image : **input feature map**

La convolution

- Définition générale (convolution discrète, 1 dimension)

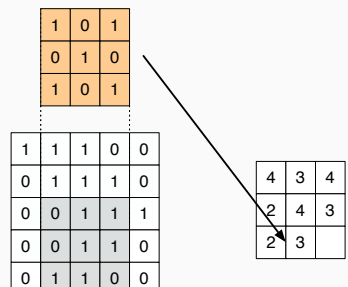
$$(f * g)(n) = \sum_{m=-M}^{M} f(n-m)g(m)$$

- Très utile pour extraire des caractéristiques d'une image
- Exemple en 2 dimensions
  - filtre :

1	0	1
0	1	0
1	0	1

  image :

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0



- filtre = **kernel**
- image : **input feature map**

La convolution

- Définition générale (convolution discrète, 1 dimension)

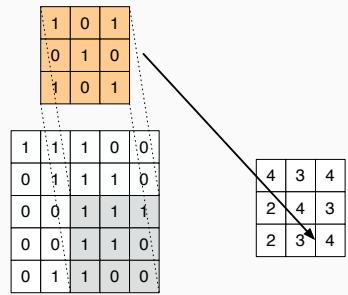
$$(f * g)(n) = \sum_{m=-M}^{M} f(n-m)g(m)$$

- Très utile pour extraire des caractéristiques d'une image
- Exemple en 2 dimensions
  - filtre :

1	0	1
0	1	0
1	0	1

  - image :

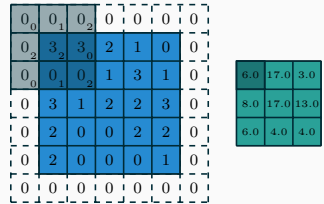
1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0



- filtre = **kernel**
- image : **input feature map**

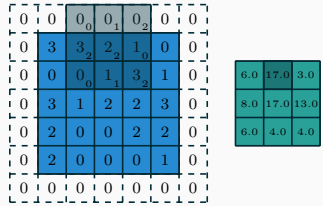
CNN : formalisation

- Soit :
  - $n \equiv$ nombre de cartes de caractéristiques en sortie
  - $m \equiv$ nombre de cartes de caractéristiques en entrée
  - $k_j \equiv$ taille du noyau sur l'axe j
- Les propriétés suivantes affectent la taille de sortie o_j d'une couche convolutionnelle selon l'axe j :
  - i_j : taille de l'entrée selon l'axe j (**input feature map)**
  - k_j : taille du noyau selon l'axe j (**kernel**)
  - s_j : pas selon l'axe j (**stride**)
  - p_j : remplissage de 0 selon l'axe j (**padding**)
  - Exemple ci-contre :
    - $i_1 = i_2 = 5$, $k_1 = k_2 = 3$, $s_1 = s_2 = 2$, $p_1 = p_2 = 1$
  - Figures issues de [Dumoulin and Visin, 2016]
  - Voir aussi `https://github.com/vdumoulin/conv_arithmetic`



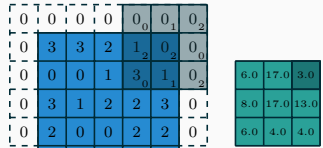
CNN : formalisation

- Soit :
  - $n \equiv$ nombre de cartes de caractéristiques en sortie
  - $m \equiv$ nombre de cartes de caractéristiques en entrée
  - $k_j \equiv$ taille du noyau sur l'axe j
- Les propriétés suivantes affectent la taille de sortie o_j d'une couche convolutionnelle selon l'axe j :
  - i_j : taille de l'entrée selon l'axe j (**input feature map)**
  - k_j : taille du noyau selon l'axe j (**kernel**)
  - s_j : pas selon l'axe j (**stride**)
  - p_j : remplissage de 0 selon l'axe j (**padding**)
  - Exemple ci-contre :
    - $i_1 = i_2 = 5$, $k_1 = k_2 = 3$, $s_1 = s_2 = 2$, $p_1 = p_2 = 1$
  - Figures issues de [Dumoulin and Visin, 2016]
  - Voir aussi `https://github.com/vdumoulin/conv_arithmetic`



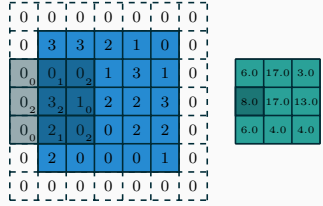
CNN : formalisation

- Soit :
  - $n \equiv$ nombre de cartes de caractéristiques en sortie
  - $m \equiv$ nombre de cartes de caractéristiques en entrée
  - $k_j \equiv$ taille du noyau sur l'axe j
- Les propriétés suivantes affectent la taille de sortie o_j d'une couche convolutionnelle selon l'axe j :
  - i_j : taille de l'entrée selon l'axe j (**input feature map)**
  - k_j : taille du noyau selon l'axe j (**kernel**)
  - s_j : pas selon l'axe j (**stride**)
  - p_j : remplissage de 0 selon l'axe j (**padding**)
  - Exemple ci-contre :
    - $i_1 = i_2 = 5$, $k_1 = k_2 = 3$, $s_1 = s_2 = 2$, $p_1 = p_2 = 1$
  - Figures issues de [Dumoulin and Visin, 2016]
  - Voir aussi `https://github.com/vdumoulin/conv_arithmetic`



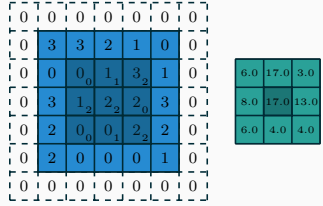
CNN : formalisation

- Soit :
  - $n \equiv$ nombre de cartes de caractéristiques en sortie
  - $m \equiv$ nombre de cartes de caractéristiques en entrée
  - $k_j \equiv$ taille du noyau sur l'axe j
- Les propriétés suivantes affectent la taille de sortie o_j d'une couche convolutionnelle selon l'axe j :
  - i_j : taille de l'entrée selon l'axe j (**input feature map)**
  - k_j : taille du noyau selon l'axe j (**kernel**)
  - s_j : pas selon l'axe j (**stride**)
  - p_j : remplissage de 0 selon l'axe j (**padding**)
  - Exemple ci-contre :
    - $i_1 = i_2 = 5$, $k_1 = k_2 = 3$, $s_1 = s_2 = 2$, $p_1 = p_2 = 1$
  - Figures issues de [Dumoulin and Visin, 2016]
  - Voir aussi `https://github.com/vdumoulin/conv_arithmetic`



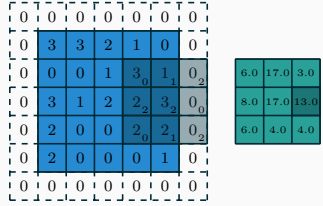
CNN : formalisation

- Soit :
  - $n \equiv$ nombre de cartes de caractéristiques en sortie
  - $m \equiv$ nombre de cartes de caractéristiques en entrée
  - $k_j \equiv$ taille du noyau sur l'axe j
- Les propriétés suivantes affectent la taille de sortie o_j d'une couche convolutionnelle selon l'axe j :
  - i_j : taille de l'entrée selon l'axe j (**input feature map)**
  - k_j : taille du noyau selon l'axe j (**kernel**)
  - s_j : pas selon l'axe j (**stride**)
  - p_j : remplissage de 0 selon l'axe j (**padding**)
  - Exemple ci-contre :
    - $i_1 = i_2 = 5$, $k_1 = k_2 = 3$, $s_1 = s_2 = 2$, $p_1 = p_2 = 1$
  - Figures issues de [Dumoulin and Visin, 2016]
  - Voir aussi `https://github.com/vdumoulin/conv_arithmetic`



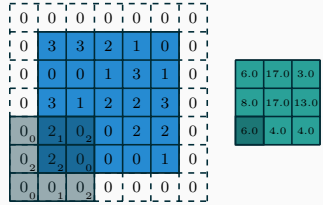
CNN : formalisation

- Soit :
  - $n \equiv$ nombre de cartes de caractéristiques en sortie
  - $m \equiv$ nombre de cartes de caractéristiques en entrée
  - $k_j \equiv$ taille du noyau sur l'axe j
- Les propriétés suivantes affectent la taille de sortie o_j d'une couche convolutionnelle selon l'axe j :
  - i_j : taille de l'entrée selon l'axe j (**input feature map)**
  - k_j : taille du noyau selon l'axe j (**kernel**)
  - s_j : pas selon l'axe j (**stride**)
  - p_j : remplissage de 0 selon l'axe j (**padding**)
  - Exemple ci-contre :
    - $i_1 = i_2 = 5$, $k_1 = k_2 = 3$, $s_1 = s_2 = 2$, $p_1 = p_2 = 1$
  - Figures issues de [Dumoulin and Visin, 2016]
  - Voir aussi `https://github.com/vdumoulin/conv_arithmetic`



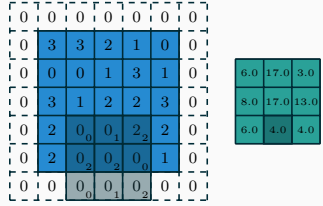
CNN : formalisation

- Soit :
  - $n \equiv$ nombre de cartes de caractéristiques en sortie
  - $m \equiv$ nombre de cartes de caractéristiques en entrée
  - $k_j \equiv$ taille du noyau sur l'axe j
- Les propriétés suivantes affectent la taille de sortie o_j d'une couche convolutionnelle selon l'axe j :
  - i_j : taille de l'entrée selon l'axe j (**input feature map)**
  - k_j : taille du noyau selon l'axe j (**kernel**)
  - s_j : pas selon l'axe j (**stride**)
  - p_j : remplissage de 0 selon l'axe j (**padding**)
  - Exemple ci-contre :
    - $i_1 = i_2 = 5$, $k_1 = k_2 = 3$, $s_1 = s_2 = 2$, $p_1 = p_2 = 1$
  - Figures issues de [Dumoulin and Visin, 2016]
  - Voir aussi `https://github.com/vdumoulin/conv_arithmetic`



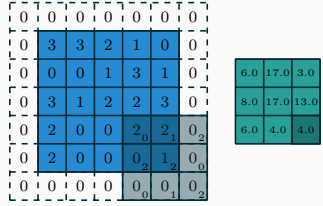
CNN : formalisation

- Soit :
  - $n \equiv$ nombre de cartes de caractéristiques en sortie
  - $m \equiv$ nombre de cartes de caractéristiques en entrée
  - $k_j \equiv$ taille du noyau sur l'axe j
- Les propriétés suivantes affectent la taille de sortie o_j d'une couche convolutionnelle selon l'axe j :
  - i_j : taille de l'entrée selon l'axe j (**input feature map)**
  - k_j : taille du noyau selon l'axe j (**kernel**)
  - s_j : pas selon l'axe j (**stride**)
  - p_j : remplissage de 0 selon l'axe j (**padding**)
  - Exemple ci-contre :
    - $i_1 = i_2 = 5$, $k_1 = k_2 = 3$, $s_1 = s_2 = 2$, $p_1 = p_2 = 1$
  - Figures issues de [Dumoulin and Visin, 2016]
  - Voir aussi `https://github.com/vdumoulin/conv_arithmetic`



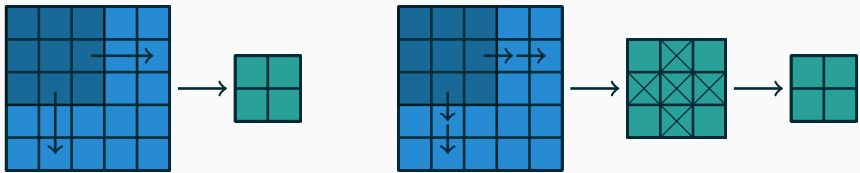
CNN : formalisation

- Soit :
  - $n \equiv$ nombre de cartes de caractéristiques en sortie
  - $m \equiv$ nombre de cartes de caractéristiques en entrée
  - $k_j \equiv$ taille du noyau sur l'axe j
- Les propriétés suivantes affectent la taille de sortie o_j d'une couche convolutionnelle selon l'axe j :
  - i_j : taille de l'entrée selon l'axe j (**input feature map)**
  - k_j : taille du noyau selon l'axe j (**kernel**)
  - s_j : pas selon l'axe j (**stride**)
  - p_j : remplissage de 0 selon l'axe j (**padding**)
  - Exemple ci-contre :
    - $i_1 = i_2 = 5$, $k_1 = k_2 = 3$, $s_1 = s_2 = 2$, $p_1 = p_2 = 1$
  - Figures issues de [Dumoulin and Visin, 2016]
  - Voir aussi `https://github.com/vdumoulin/conv_arithmetic`



Le pas (**stride**)

- Une autre manière de voir le pas (**stride**)



- On effectue la convolution par pas de 1
- mais on ne retient qu'une valeur tous les $s = 2$ éléments

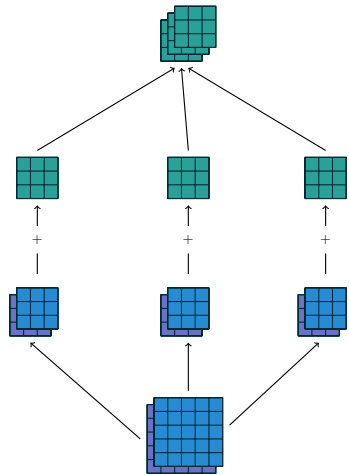
Convolution

- 2 cartes de caractéristiques $\mathbf{FM}_1$ et $\mathbf{FM}_2$ en entrée
- 3 cartes de caractéristiques en sortie
- Collection de noyaux $\mathbf{w}$: 3 x 2 x 3 x 3

- $\mathbf{FM}_1$ convoluée avec **kernel** $\mathbf{w}_{1,1}$
- $\mathbf{FM}_2$ convoluée avec **kernel** $\mathbf{w}_{1,2}$
- les résultats sont **sommés** élément par élément

$\rightarrow$ on obtient la première carte de sortie

- on répète pour les autres noyaux (**kernel**)

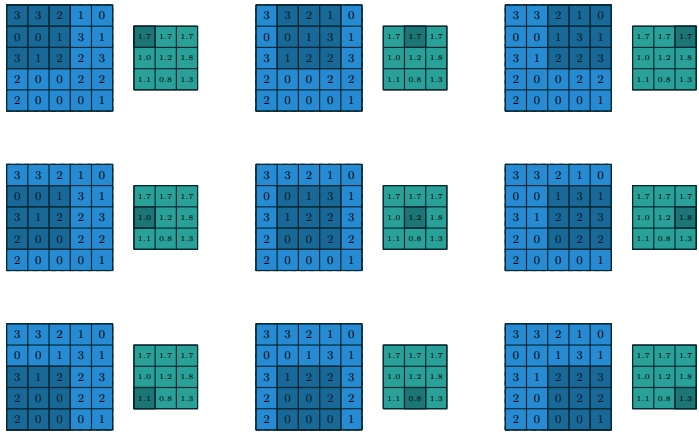


Regroupement (**Pooling**)

- Permet de réduire la taille des cartes de caractéristiques
- Permet de **résumer** des sous-régions
- Associé à une fonction (généralement non linéaire)
  - Maximum : **max-pooling**
  - Moyenne : **average-pooling**

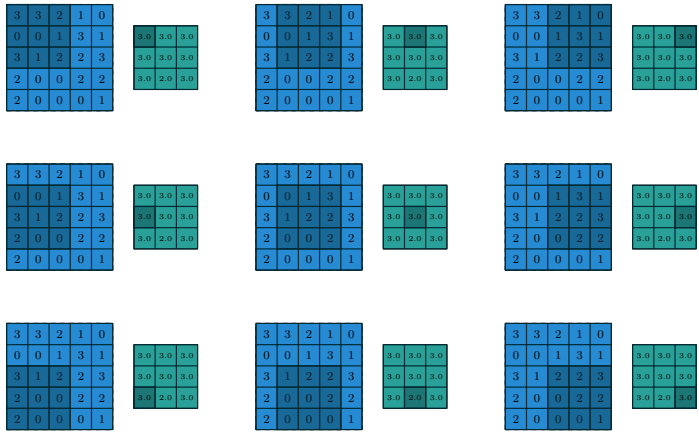
- Les propriétés suivantes affectent la taille de sortie o_j d'une couche de regroupement selon l'axe j :
  - i_j : taille de l'entrée selon l'axe j (**input feature map)**
  - k_j : taille de la fenêtre de pooling selon l'axe j (**window size**)
  - s_j : pas selon l'axe j (**stride**)

Regroupement : moyenne (**average pooling**)



- **average pooling** en 2 dimensions : fenêtre 3 x 3, entrée 5 x 5, par pas de 1 x 1

Regroupement : maximum (**max pooling**)



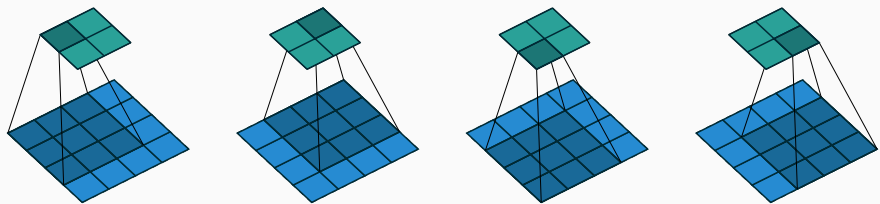
- **max pooling** en 2 dimensions : fenêtre 3 x 3, entrée 5 x 5, par pas de 1 x 1

Convolution : propriétés

Cas le plus simple :

- $p = 0$: pas de padding, $s = 1$: par pas de 1
- La taille de sortie est définie par :

$$o = (i - k) + 1 \text{ avec } i \text{ la taille de l'entrée, } k \text{ la taille du noyau}$$



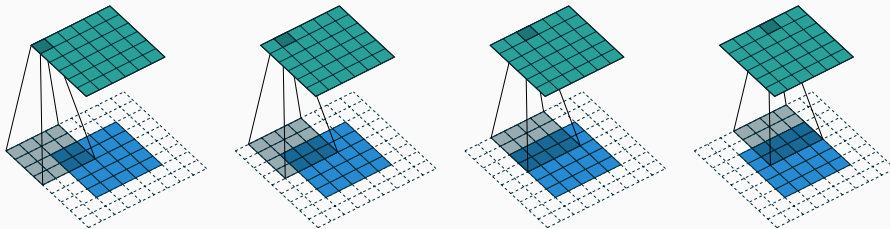
- Ex : noyau $k = 3$ x 3, entrée $i = 4$ x 4

Convolution : propriétés

Cas simple + padding :

- padding p : ajout de $p = 2$ zéros à la bordure, $s = 1$: par pas de 1
- La taille de sortie est définie par :

$$o = (i - k) + 2p + 1 \text{ avec } i \text{ la taille de l'entrée, } k \text{ la taille du noyau}$$



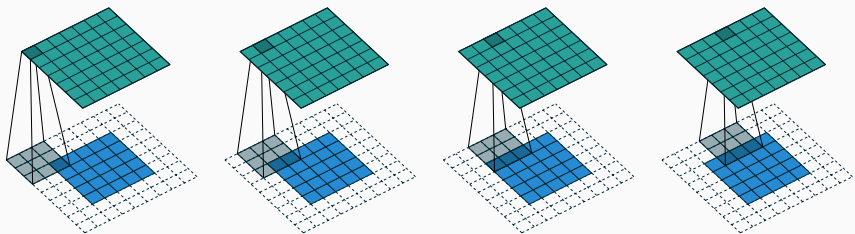
- Ex : noyau $k = 4$ x 4, entrée $i = 5$ x 5, padding $p = 2$ x 2
- Note : résultat plus grand que l'entrée (dans cet exemple)

Convolution : propriétés

Cas simple + padding complet :

- padding p : ajout de $p = k - 1$ zéros à la bordure, $s = 1$: par pas de 1
- La taille de sortie est définie par :

$$o = i + 2(k-1) - (k-1) = \text{i} + (\text{k-1})$$



- Ex : noyau $k = 3$ x 3, entrée $i = 5$ x 5, padding $p = 2$ x 2
- Toutes les superpositions possibles entre le noyau et l'entrée sont considérées

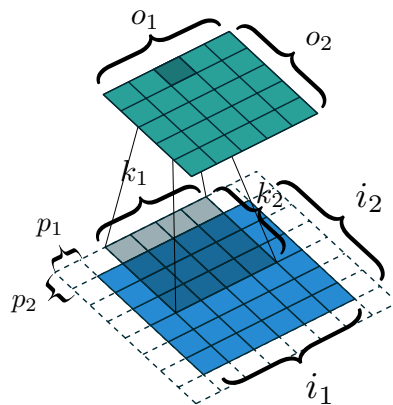
Convolution : propriétés

Question :

- Comment obtenir une sortie de même taille que l'entrée (en gardant un pas de 1) ?

À faire chez vous :

- i_1, i_2, k_1 et k_2 sont fixés
- le pas $s = 1$ (**stride**) est fixé
- Déterminer $p1$ et $p2$

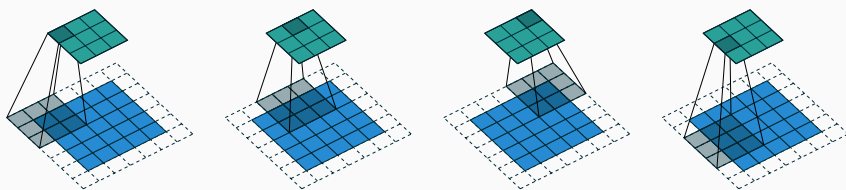


Convolution : propriétés

Pas > 1 + padding :

- padding p, pas s
- La taille de sortie est définie par :

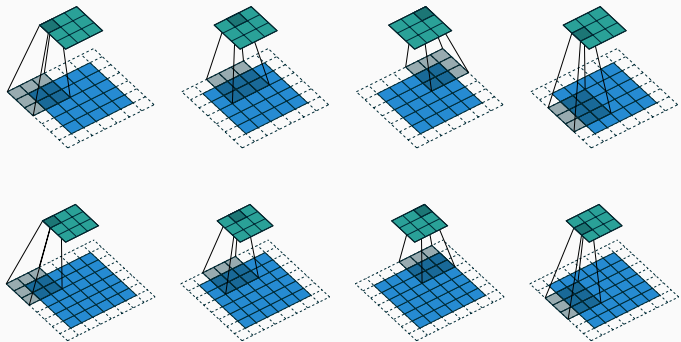
$$o = \left\lfloor \frac{i+2p-k}{s} \right\rfloor + 1$$



- Ex : noyau $k = 3$ x 3, entrée $i = 5$ x 5, padding $p = 1$ x 1, pas $s = 2$ x 2

Convolution : propriétés

Pair / impair



- Note : la dernière ligne et la colonne de droite de l'entrée ne sont pas traitées !
- Malgré les tailles d'entrée différentes, les sorties sont de même tailles (noyau fixé)

Convolution : pour aller plus loin

- "A guide to convolution arithmetic for deep learning", V. Dumoulin & F. Visin

- Que se passe-t-il si on veut aller dans l'autre sens ?

→ Notions de convolution transposée

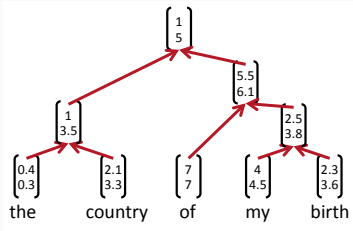
→ Déconvolution

- Visualisation `http://scs.ryerson.ca/~aharley/vis/conv/`

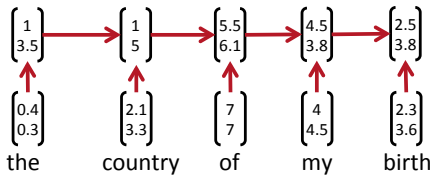
CNN pour le texte

Du RNN au CNN

réseau récursif



réseau récurrent



nécessite un analyser pour obtenir la structure de l'arbre (syntaxe)

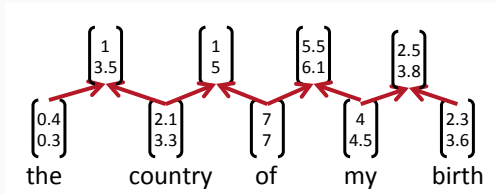
nécessite un préfixe (contexte gauche) pour capturer l'information
capture (souvent) trop d'information sur le dernier mot $\rightarrow$ biRNN

Du RNN au CNN

- Et si on calculait les vecteurs pour chaque segment de la phrase ?
- Exemple : "the country of my birth"
  - taille 2 : "the country", "country of", "of my", "my birth"
  - taille 3 : "the country of", "country of my", "of my birth",
  - taille 4 : "the country of my", "country of my birth"
- peu importe la grammaticalité
- pas besoin d'analyseur
- pas de justification linguistique

Convolution pour le texte

- Première couche : calcul des vecteurs pour les bigrammes



- Même opération que pour le RNN, mais pour chaque paire de mots

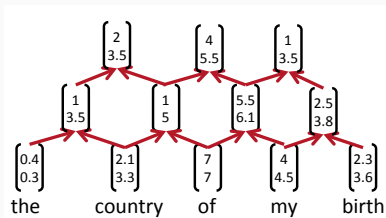
$$p = tanh\left(\mathbf{W}\begin{bmatrix} c_1 \\ c_2 \end{bmatrix} + \mathbf{b}\right)$$

- convolution sur le vecteurs de mots

→ Les poids **W** et **b** sont partagés (nombre de paramètres réduits)

Convolution pour le texte

- Plusieurs solutions pour calculer les couches de plus haut niveau
- Idée simple : répéter le processus avec des poids différents

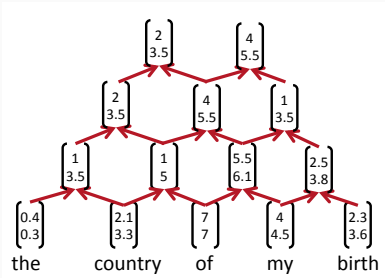


$$p = tanh\left(\mathbf{W}^{(2)}\begin{bmatrix} c_1 \\ c_2 \end{bmatrix} + \mathbf{b}^{(2)}\right)$$

- Simple à comprendre et implémenter, pas nécessairement la meilleure solution

Convolution pour le texte

- Plusieurs solutions pour calculer les couches de plus haut niveau
- Idée simple : répéter le processus avec des poids différents

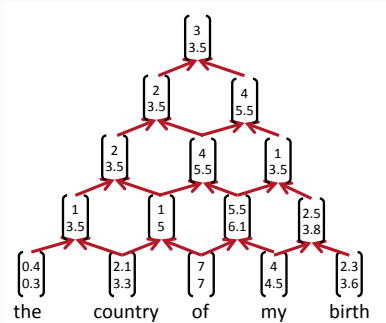


$$p = tanh\left(\mathbf{W}^{(3)}\begin{bmatrix} c_1 \\ c_2 \end{bmatrix} + \mathbf{b}^{(3)}\right)$$

- Simple à comprendre et implémenter, pas nécessairement la meilleure solution

Convolution pour le texte

- Plusieurs solutions pour calculer les couches de plus haut niveau
- Idée simple : répéter le processus avec des poids différents

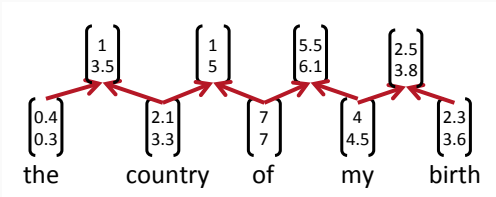


$$p = tanh\left(\mathbf{W}^{(4)}\begin{bmatrix} c_1 \\ c_2 \end{bmatrix} + \mathbf{b}^{(4)}\right)$$

- Simple à comprendre et implémenter, pas nécessairement la meilleure solution

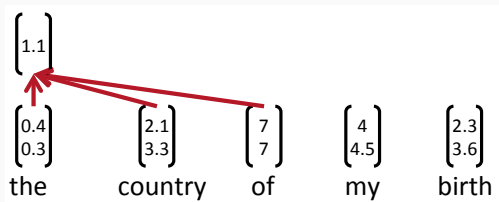
CNN avec une seule couche

- Une couche convolutionnelle + une opération de groupement (**pooling**)
- cf. [Collobert et al., 2011] et [Kim, 2014]
- Formalisation
  - Embeddings : $\mathbf{x}_i \in \mathbb{R}^k$
  - Phrase : $\mathbf{x}_{1:n} = \mathbf{x}_1 \oplus \mathbf{x}_2 \oplus \cdots \oplus \mathbf{x}_n$ [vecteurs concaténés]
  - Concaténation d'embeddings dans un intervalle : $\mathbf{x}_{i:i+j}$
  - Filtre pour la convolution : $\mathbf{w} \in \mathbb{R}^{hk}$ [s'étend sur une fenêtre de h mots]
    - Note : le filtre $\mathbf{w}$ est un vecteur !
  - Pourrait-être 2 (comme précédemment)



CNN avec une seule couche

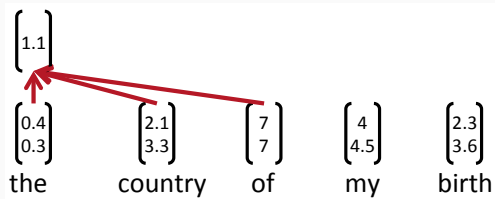
- Une couche convolutionnelle + une opération de groupement (**pooling**)
- cf. [Collobert et al., 2011] et [Kim, 2014]
- Formalisation
  - Embeddings : $\mathbf{x}_i \in \mathbb{R}^k$
  - Phrase : $\mathbf{x}_{1:n} = \mathbf{x}_1 \oplus \mathbf{x}_2 \oplus \cdots \oplus \mathbf{x}_n$ [vecteurs concaténés]
  - Concaténation d'embeddings dans un intervalle : $\mathbf{x}_{i:i+j}$
  - Filtre pour la convolution : $\mathbf{w} \in \mathbb{R}^{hk}$ [s'étend sur une fenêtre de h mots]
    - Note : le filtre $\mathbf{w}$ est un vecteur !
  - Pourrait-être 2 (comme précédemment) ou plus, 3 par exemple :



CNN avec une seule couche

- Calcul du paramètre pour le CNN :

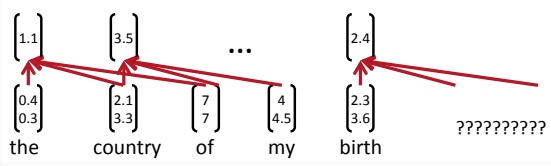
$$c_i = f(\mathbf{W}^\top \mathbf{x}_{i:i+h-1} + \mathbf{b})$$



CNN avec une seule couche

- Le filtre **w** est appliqué à toutes les fenêtres possibles (vecteurs concaténés) :
- Phrase : $\mathbf{x}_{1:n} = \mathbf{x}_1 \oplus \mathbf{x}_2 \oplus \cdots \oplus \mathbf{x}_n$ [vecteurs concaténés]
- Fenêtres de taille h possibles : $\{\mathbf{x}_{1:h}, \mathbf{x}_{2:h+1}, \ldots, \mathbf{x}_{n-h+1:n}\}$

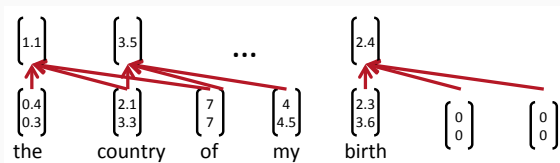
  Le résultat : carte de caractéristiques (**feature map)**
  - $\mathbf{c} = [c_1, c_2, \ldots, c_{n-h+1}] \in \mathbb{R}^{n-h+1}$



CNN avec une seule couche

- Le filtre **w** est appliqué à toutes les fenêtres possibles (vecteurs concaténés) :
- Phrase : $\mathbf{x}_{1:n} = \mathbf{x}_1 \oplus \mathbf{x}_2 \oplus \cdots \oplus \mathbf{x}_n$ [vecteurs concaténés]
- Fenêtres de taille h possibles : $\{\mathbf{x}_{1:h}, \mathbf{x}_{2:h+1}, \ldots, \mathbf{x}_{n-h+1:n}\}$

  Le résultat : carte de caractéristiques (**feature map)**
  - $\mathbf{c} = [c_1, c_2, \ldots, c_{n-h+1}] \in \mathbb{R}^{n-h+1}$



Couche de regroupement (**pooling**)

- Nouvelle brique pour les réseaux de neurones : **pooling**
- En particulier : couche de max pooling **temporel**
- Idée : capture l'activation la plus importante à travers le temps
- À partir d'une carte de caractéristiques $\mathbf{c} = [c_1, c_2, \ldots, c_{n-h+1}] \in \mathbb{R}^{n-h+1}$
  $\rightarrow$ Conserver une seule valeur : $\hat{c} = max(\mathbf{c})$
- mais on a besoin de plus de valeurs !

Couche de regroupement (**pooling**)

- Nouvelle brique pour les réseaux de neurones : **pooling**
- En particulier : couche de max pooling **temporel**
- Idée : capture l'activation la plus importante à travers le temps
- À partir d'une carte de caractéristiques $\mathbf{c} = [c_1, c_2, \ldots, c_{n-h+1}] \in \mathbb{R}^{n-h+1}$
  $\rightarrow$ Conserver une seule valeur : $\hat{c} = max(\mathbf{c})$
- mais on a besoin de plus de valeurs !
  $\rightarrow$ Solution 1 : utiliser plusieurs filtres $\mathbf{w}$

Couche de regroupement (**pooling**)

- Nouvelle brique pour les réseaux de neurones : **pooling**
- En particulier : couche de max pooling **temporel**
- Idée : capture l'activation la plus importante à travers le temps
- À partir d'une carte de caractéristiques $\mathbf{c} = [c_1, c_2, \ldots, c_{n-h+1}] \in \mathbb{R}^{n-h+1}$
  - $\rightarrow$ Conserver une seule valeur : $\hat{c} = max(\mathbf{c})$
- mais on a besoin de plus de valeurs !
  - $\rightarrow$ Solution 1 : utiliser plusieurs filtres $\mathbf{w}$
    - Par exemple des filtres $\mathbf{w}$ de tailles différentes (h)

Couche de regroupement (**pooling**)

- Nouvelle brique pour les réseaux de neurones : **pooling**
- En particulier : couche de max pooling **temporel**
- Idée : capture l'activation la plus importante à travers le temps
- À partir d'une carte de caractéristiques $\mathbf{c} = [c_1, c_2, \ldots, c_{n-h+1}] \in \mathbb{R}^{n-h+1}$
  - $\rightarrow$ Conserver une seule valeur : $\hat{c} = max(\mathbf{c})$
- mais on a besoin de plus de valeurs !
  - $\rightarrow$ Solution 1 : utiliser plusieurs filtres $\mathbf{w}$
    - Par exemple des filtres $\mathbf{w}$ de tailles différentes (h)
    - Grâce au **max pooling**, la taille de $\mathbf{c}$ ne change rien.

Couche de regroupement (**pooling**)

- Nouvelle brique pour les réseaux de neurones : **pooling**
- En particulier : couche de max pooling **temporel**
- Idée : capture l'activation la plus importante à travers le temps
- À partir d'une carte de caractéristiques $\mathbf{c} = [c_1, c_2, \ldots, c_{n-h+1}] \in \mathbb{R}^{n-h+1}$
  - $\rightarrow$ Conserver une seule valeur : $\hat{c} = max(\mathbf{c})$
- mais on a besoin de plus de valeurs !
  - $\rightarrow$ Solution 1 : utiliser plusieurs filtres $\mathbf{w}$
    - Par exemple des filtres $\mathbf{w}$ de tailles différentes (h)
    - Grâce au **max pooling**, la taille de $\mathbf{c}$ ne change rien.
    - On peut utiliser des filtres qui regardent les unigrams, les bigrams, les trigrams, etc.

Couche de regroupement (**pooling**)

- Nouvelle brique pour les réseaux de neurones : **pooling**
- En particulier : couche de max pooling **temporel**
- Idée : capture l'activation la plus importante à travers le temps
- À partir d'une carte de caractéristiques $\mathbf{c} = [c_1, c_2, \ldots, c_{n-h+1}] \in \mathbb{R}^{n-h+1}$
  - $\rightarrow$ Conserver une seule valeur : $\hat{c} = max(\mathbf{c})$
- mais on a besoin de plus de valeurs !
  - $\rightarrow$ Solution 1 : utiliser plusieurs filtres $\mathbf{w}$
  - $\rightarrow$ Solution 2 : (idée) utiliser plusieurs canaux

Couche de regroupement (**pooling**)

- Nouvelle brique pour les réseaux de neurones : **pooling**
- En particulier : couche de max pooling **temporel**
- Idée : capture l'activation la plus importante à travers le temps
- À partir d'une carte de caractéristiques $\mathbf{c} = [c_1, c_2, \ldots, c_{n-h+1}] \in \mathbb{R}^{n-h+1}$
  - $\rightarrow$ Conserver une seule valeur : $\hat{c} = max(\mathbf{c})$
- mais on a besoin de plus de valeurs !
  - $\rightarrow$ Solution 1 : utiliser plusieurs filtres $\mathbf{w}$
  - $\rightarrow$ Solution 2 : (idée) utiliser plusieurs canaux
    - Initialiser les embeddings avec un modèle pré-entraîné (word2vec ou Glove)

Couche de regroupement (**pooling**)

- Nouvelle brique pour les réseaux de neurones : **pooling**
- En particulier : couche de max pooling **temporel**
- Idée : capture l'activation la plus importante à travers le temps
- À partir d'une carte de caractéristiques $\mathbf{c} = [c_1, c_2, \ldots, c_{n-h+1}] \in \mathbb{R}^{n-h+1}$
  - $\rightarrow$ Conserver une seule valeur : $\hat{c} = max(\mathbf{c})$
- mais on a besoin de plus de valeurs !
  - $\rightarrow$ Solution 1 : utiliser plusieurs filtres $\mathbf{w}$
  - $\rightarrow$ Solution 2 : (idée) utiliser plusieurs canaux
    - Initialiser les embeddings avec un modèle pré-entraîné (word2vec ou Glove)
    - Commencer avec 2 copies

Couche de regroupement (**pooling**)

- Nouvelle brique pour les réseaux de neurones : **pooling**
- En particulier : couche de max pooling **temporel**
- Idée : capture l'activation la plus importante à travers le temps
- À partir d'une carte de caractéristiques $\mathbf{c} = [c_1, c_2, \ldots, c_{n-h+1}] \in \mathbb{R}^{n-h+1}$
  - $\rightarrow$ Conserver une seule valeur : $\hat{c} = max(\mathbf{c})$
- mais on a besoin de plus de valeurs !
  - $\rightarrow$ Solution 1 : utiliser plusieurs filtres $\mathbf{w}$
  - $\rightarrow$ Solution 2 : (idée) utiliser plusieurs canaux
    - Initialiser les embeddings avec un modèle pré-entraîné (word2vec ou Glove)
    - Commencer avec 2 copies
    - Mettre à jour une copie par backpropagation, garder l'autre inchangée

Couche de regroupement (**pooling**)

- Nouvelle brique pour les réseaux de neurones : **pooling**
- En particulier : couche de max pooling **temporel**
- Idée : capture l'activation la plus importante à travers le temps
- À partir d'une carte de caractéristiques $\mathbf{c} = [c_1, c_2, \ldots, c_{n-h+1}] \in \mathbb{R}^{n-h+1}$
  - $\rightarrow$ Conserver une seule valeur : $\hat{c} = max(\mathbf{c})$
- mais on a besoin de plus de valeurs !
  - $\rightarrow$ Solution 1 : utiliser plusieurs filtres $\mathbf{w}$
  - $\rightarrow$ Solution 2 : (idée) utiliser plusieurs canaux
    - Initialiser les embeddings avec un modèle pré-entraîné (word2vec ou Glove)
    - Commencer avec 2 copies
    - Mettre à jour une copie par backpropagation, garder l'autre inchangée
    - Les deux canaux sont ajoutés à $\mathbf{c}$ avant le max pooling

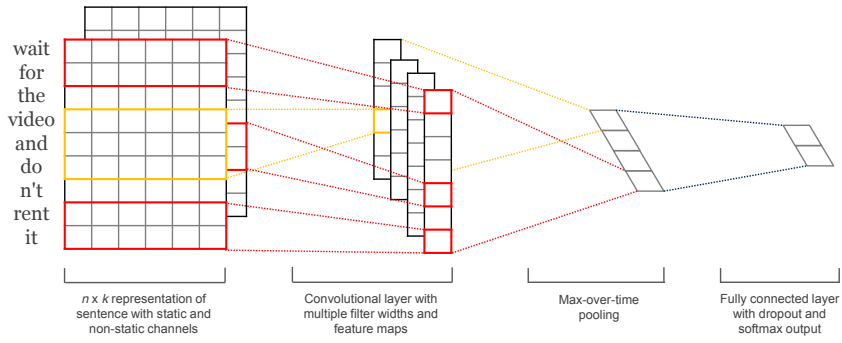
Classification après un CNN à une seule couche

- Couche de convolution suivie d'une couche de max pooling
- Pour obtenir les vecteurs de caractéristiques finaux : $\mathbf{z} = [\hat{c}_1, \hat{c}_2, \ldots, \hat{c}_m]$
  $\rightarrow$ avec m filtres $\mathbf{w}$
- Simple fonction softmax en sortie :

$$y = softmax(\mathbf{W}^{(s)}\mathbf{z} + \mathbf{b})$$

$$\text{avec } softmax(a_i) = \frac{e^{a_i}}{\sum_m e^{a_m}}$$

Figure issue de Kim, 2014



- séquence de n mots (pouvant être complétée par des 0 si taille petite)
- chacun représenté par un vecteur de taille k

Dropout 1/2

- Idée : inhiber certaines caractéristiques de $\mathbf{z}$ de manière aléatoire
- Vecteur masque $\mathbf{r}$ de variables aléatoires suivant une loi de Bernouilli

$\rightarrow$ probabilité p : hyperparamètre (à déterminer empiriquement)

- Rappel : variable de Bernouilli :

$$P(X = x) = \begin{cases} p \text{ si } x = 1 \\ 1 - p \text{ si } x = 0 \\ 0 \text{ sinon} \end{cases}$$

- On ignore certaines caractéristiques pendant l'entraînement :

$$y = softmax\left(\mathbf{W}^{(s)}(\mathbf{r} \circ \mathbf{z}) + \mathbf{b}\right)$$

$\rightarrow$ empêche le sur-apprentissage

Dropout 2/2

$$y = softmax\left(\mathbf{W}^{(s)}(\mathbf{r} \circ \mathbf{z}) + \mathbf{b}\right)$$

- Pendant l'entraînement :
  - gradients rétropropagés uniquement dans un sous-ensemble d'éléments où $\mathbf{r}_i = 1$
- En phase de test : pas de dropout, donc $\mathbf{z}$ est plus grand

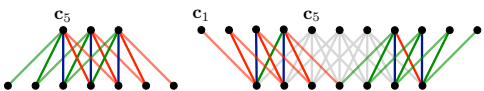
$\rightarrow$ On ajuste le vecteur final avec la probabilité p :

$$\mathbf{W}^{(s)} = p\mathbf{W}^{(s)}$$

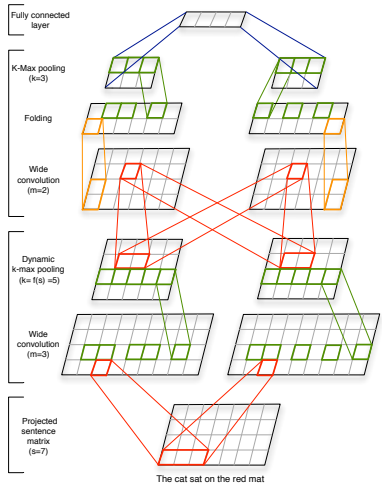
- [Kim, 2014] :
  - 2 à 4% d'amélioration de la précision
  - possibilité d'entraîner de très grands réseaux sans sur-apprentissage.

CNN : Variantes

- Convolution réductrice ou amplificatrice :

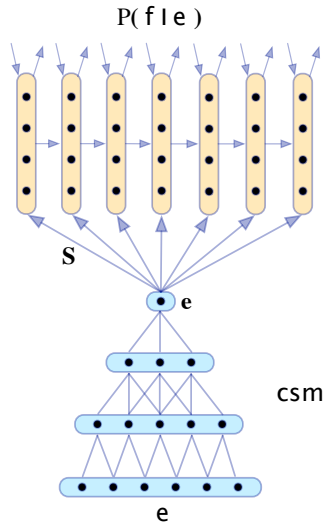


- Regroupements complexes sur les séquences (**pooling**)
- Réseaux plus profonds
- [Kalchbrenner et al., 2014]



CNN : application à la traduction automatique

- Un des premiers succès pour la traduction automatique
- CNN encode la phrase source
- RNN pour le décodage
- [Kalchbrenner and Blunsom, 2013]



Comparaison des modèles

- Sac de mots :
  - très bon système de base pour les problèmes de classification (surprenant !)
  - Surtout si suivi de quelques couches (profondeur)
- CNN :
  - bon pour la classification + facile à paralléliser sur des GPUs
  - comment incorporer des annotations au niveau sous-phrastique ?
  - nécessite de compléter avec des 0 si la phrase est courte
  - difficile à interpréter
- Réseaux récursifs :
  - linguistiquement plausible, **mais** nécessite un arbre syntaxique
- RNN :
  - Cognitivement plausible : lecture de gauche à droite
  - pas les meilleurs résultats en classification
  - améliorations avec les unités à portes (LSTM, GRU)

References I

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. P. (2011).
Natural language processing (almost) from scratch.
CoRR, abs/1103.0398.

Dumoulin, V. and Visin, F. (2016).
A guide to convolution arithmetic for deep learning.

Kalchbrenner, N. and Blunsom, P. (2013).
Recurrent continuous translation models.
Seattle. Association for Computational Linguistics.

Kalchbrenner, N., Grefenstette, E., and Blunsom, P. (2014).
A convolutional neural network for modelling sentences.
CoRR, abs/1404.2188.

References II

Kim, Y. (2014).
Convolutional neural networks for sentence classification.
CoRR, abs/1408.5882.