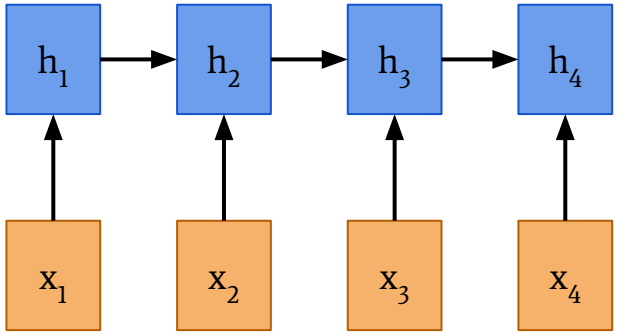
L'attention

Julien Romero

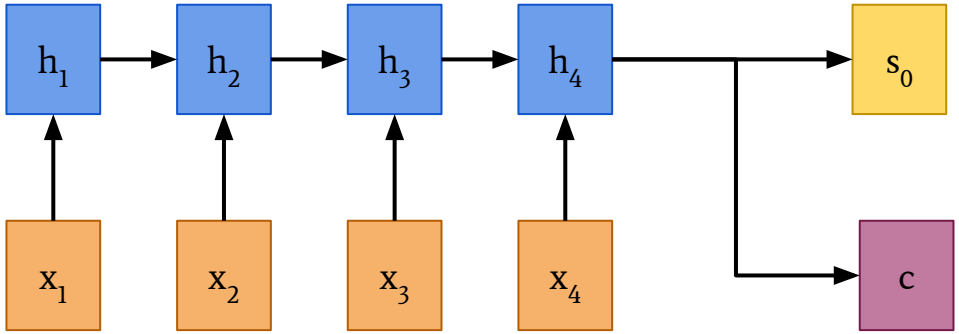
Retour sur seq2seq

Entrée : $x_1, x_2, ..., x_T$
Sortie : $y_1, y_2, ..., y_T$
Encodeur : $h_t = f_W(x_i, h_{t-1})$



Entrée : $x_1, x_2, ..., x_T$
Sortie : $y_1, y_2, ..., y_T$
Encodeur : $h_t = f_W(x_i, h_{t-1})$

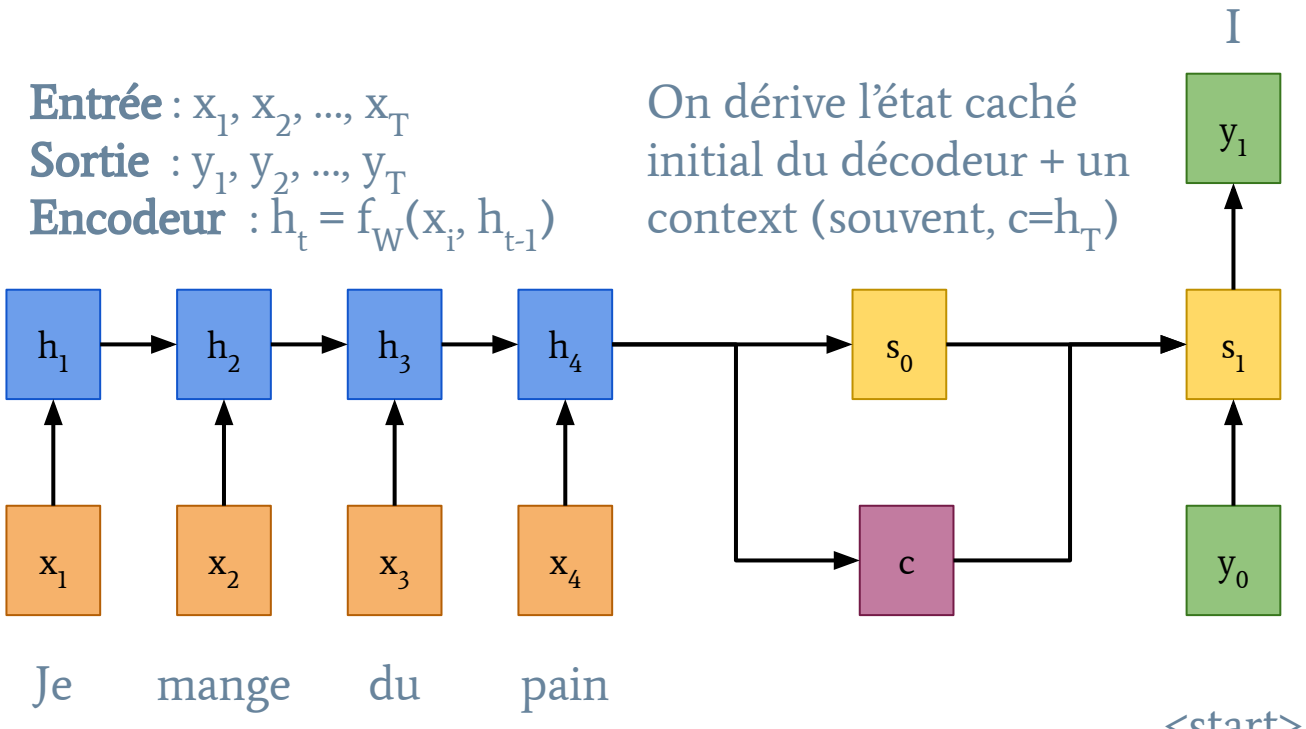
On dérive l'état caché initial du décodeur + un context (souvent, $c=h_T$)



Decodeur : $s_t = g_U(y_{t-1}, s_{t-1}, c)$

Entrée : $x_1, x_2, ..., x_T$
Sortie : $y_1, y_2, ..., y_T$
Encodeur : $h_t = f_W(x_i, h_{t-1})$

On dérive l'état caché initial du décodeur + un context (souvent, $c=h_T$)

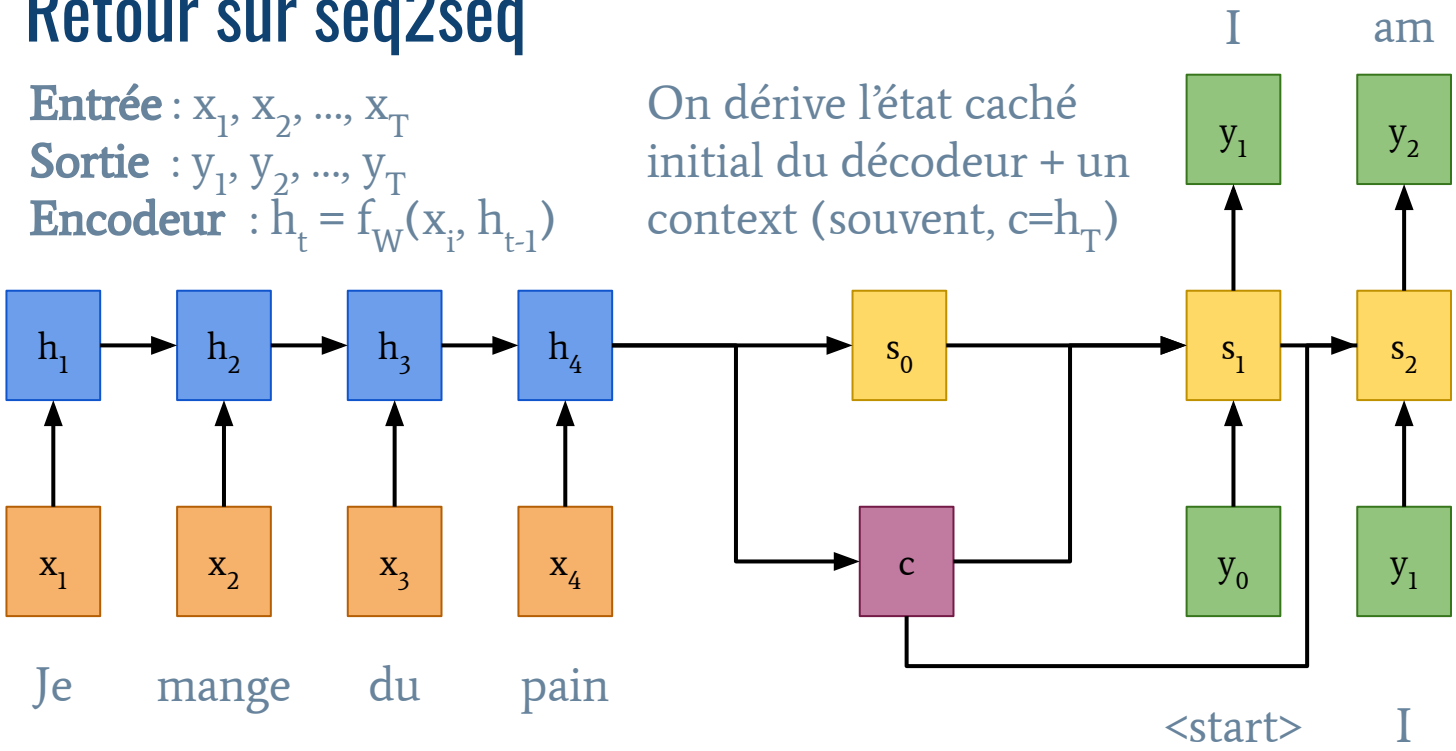


Decodeur : $s_t = g_U(y_{t-1}, s_{t-1}, c)$

Retour sur seq2seq

Entrée : $x_1, x_2, ..., x_T$
Sortie : $y_1, y_2, ..., y_T$
Encodeur : $h_t = f_W(x_i, h_{t-1})$

On dérive l'état caché initial du décodeur + un context (souvent, $c=h_T$)

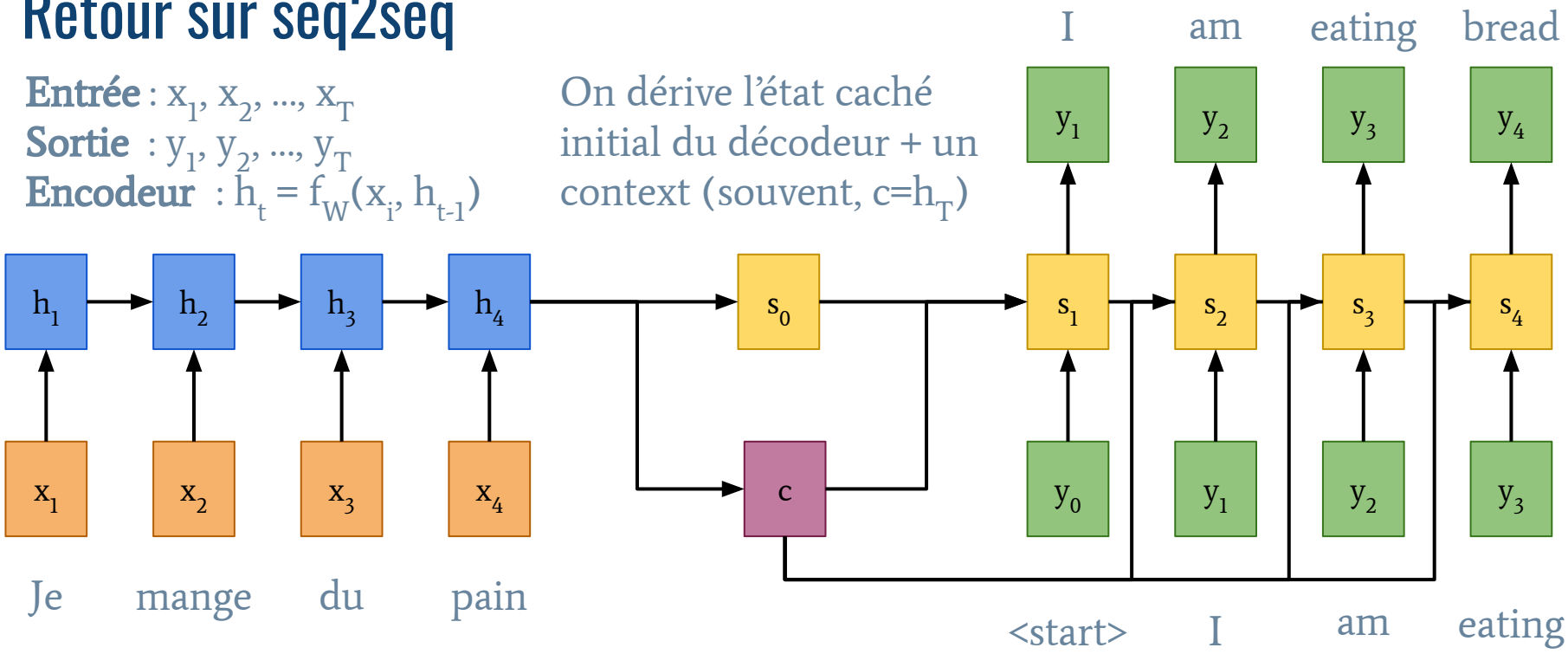


Decodeur : $s_t = g_U(y_{t-1}, s_{t-1}, c)$

Retour sur seq2seq

Entrée : $x_1, x_2, ..., x_T$
Sortie : $y_1, y_2, ..., y_T$
Encodeur : $h_t = f_W(x_i, h_{t-1})$

On dérive l'état caché initial du décodeur + un context (souvent, $c=h_T$)

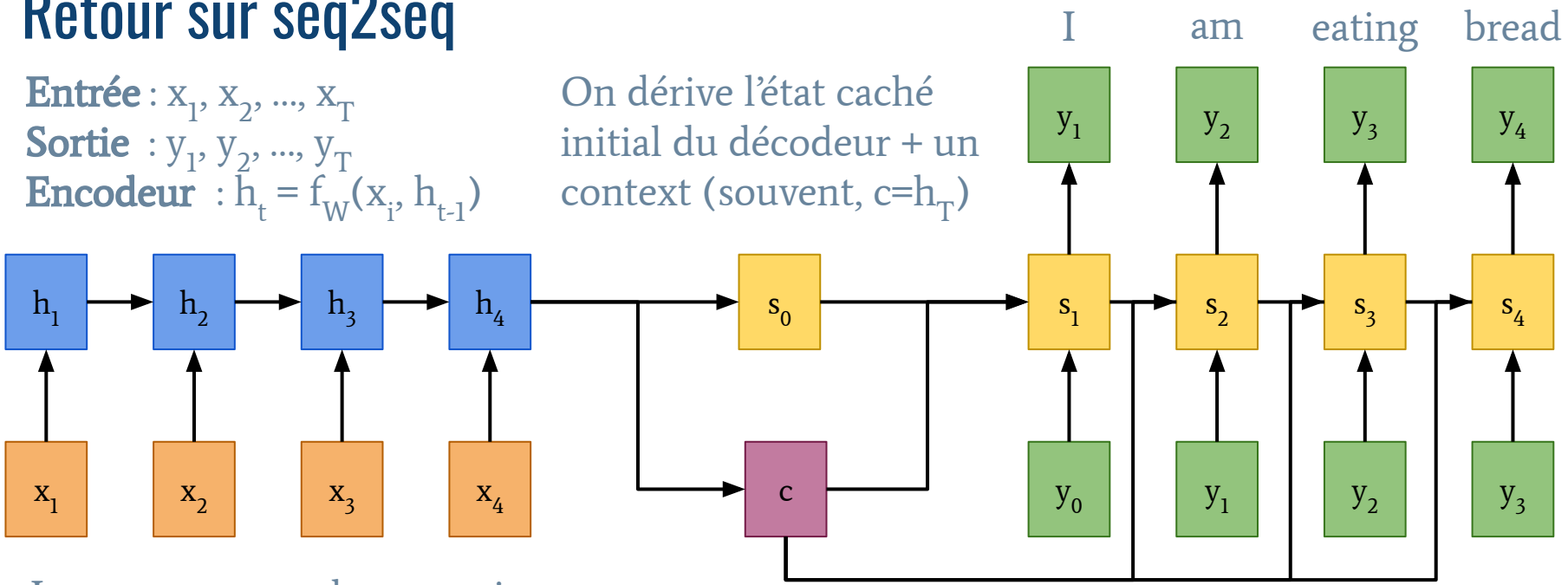


Decodeur : $s_t = g_U(y_{t-1}, s_{t-1}, c)$

Retour sur seq2seq

Entrée : $x_1, x_2, ..., x_T$
Sortie : $y_1, y_2, ..., y_T$
Encodeur : $h_t = f_W(x_i, h_{t-1})$

On dérive l'état caché initial du décodeur + un context (souvent, $c=h_T$)



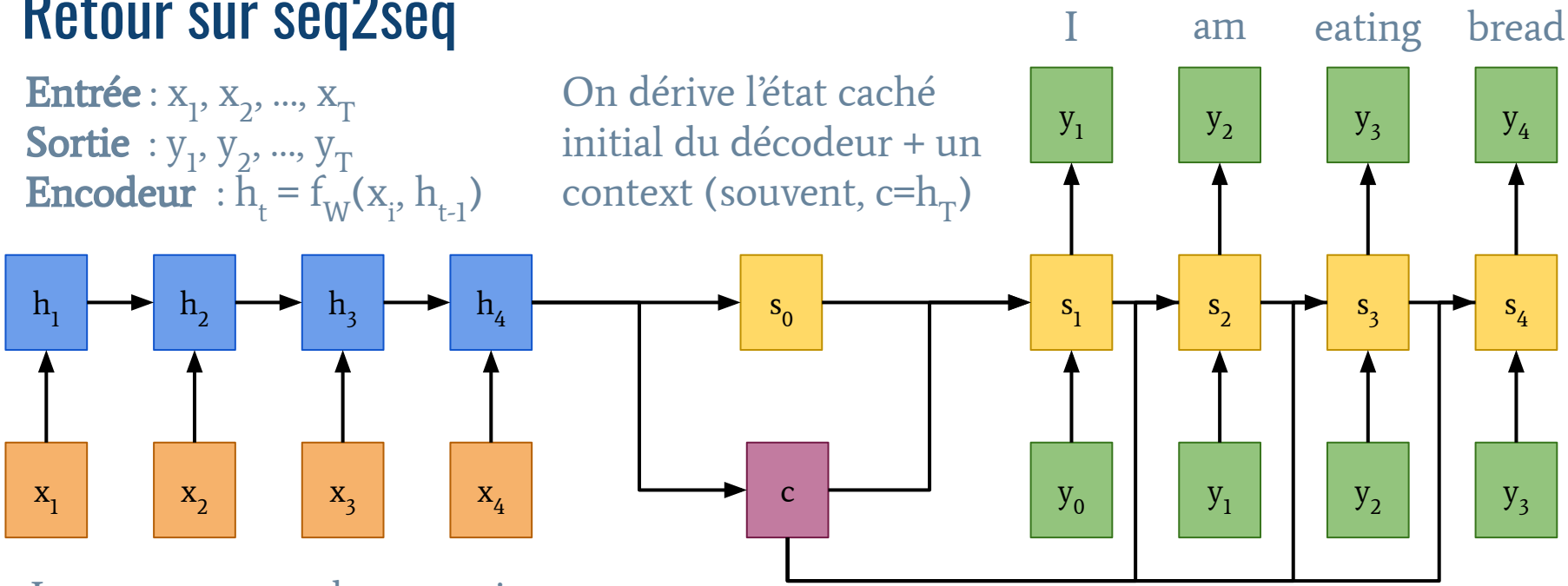
Problème : c est un goulot d'étranglement

Retour sur seq2seq

Decodeur : $s_t = g_U(y_{t-1}, s_{t-1}, c)$

Entrée : $x_1, x_2, ..., x_T$
Sortie : $y_1, y_2, ..., y_T$
Encodeur : $h_t = f_W(x_i, h_{t-1})$

On dérive l'état caché initial du décodeur + un context (souvent, $c=h_T$)



Problème : c est un goulot d'étranglement

Idée : recalculer un nouveau c à chaque étape

Couche d'attention

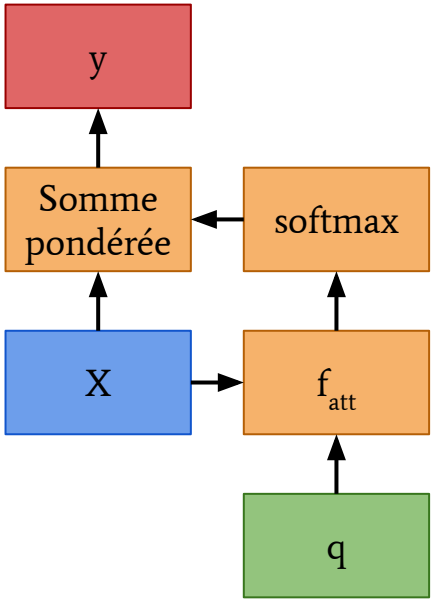
Idée : On calcule une distribution de probabilité pour déterminer sur quelle partie de l'entrée il faut se concentrer en fonction d'une requête.

Entrées/paramètres :

- **Le vecteur de la requête :** q (dim. D_Q)
- **Le vecteur d'entrée :** X (dim N_XxD_X)
- **Une fonction de similarité** : f_{att}

Calculs :

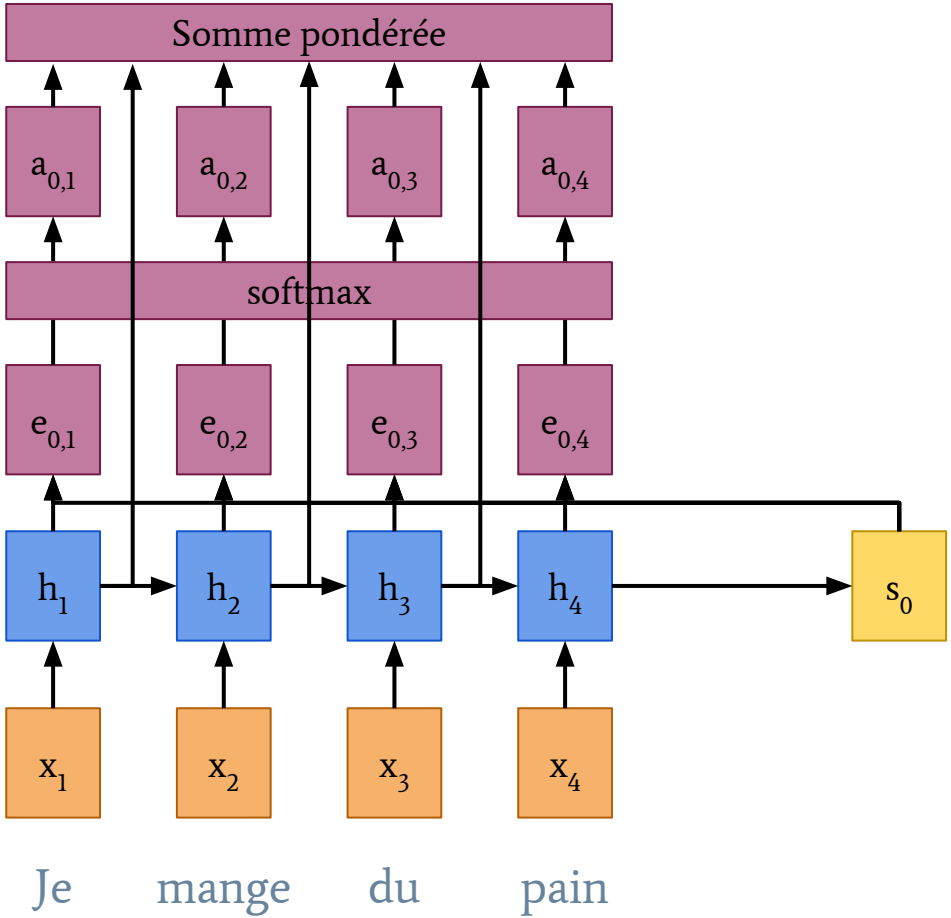
- **Le vecteur de similarité** : e, $e_i=f_{att}(q, X_i)$
- **Les poids de l'attention :** a = softmax(e)
- **Le vecteur de sortie :** $y = \sum_i a_i X_i$

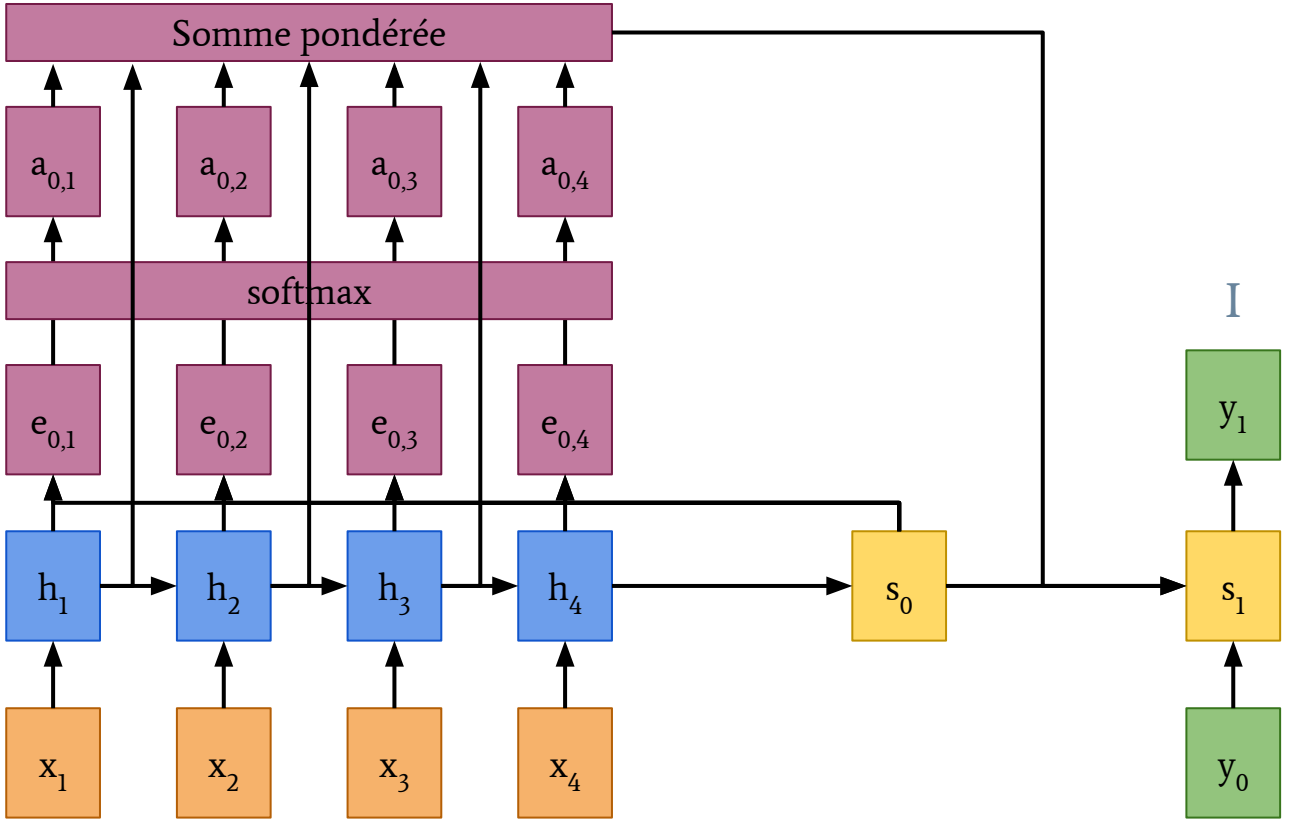


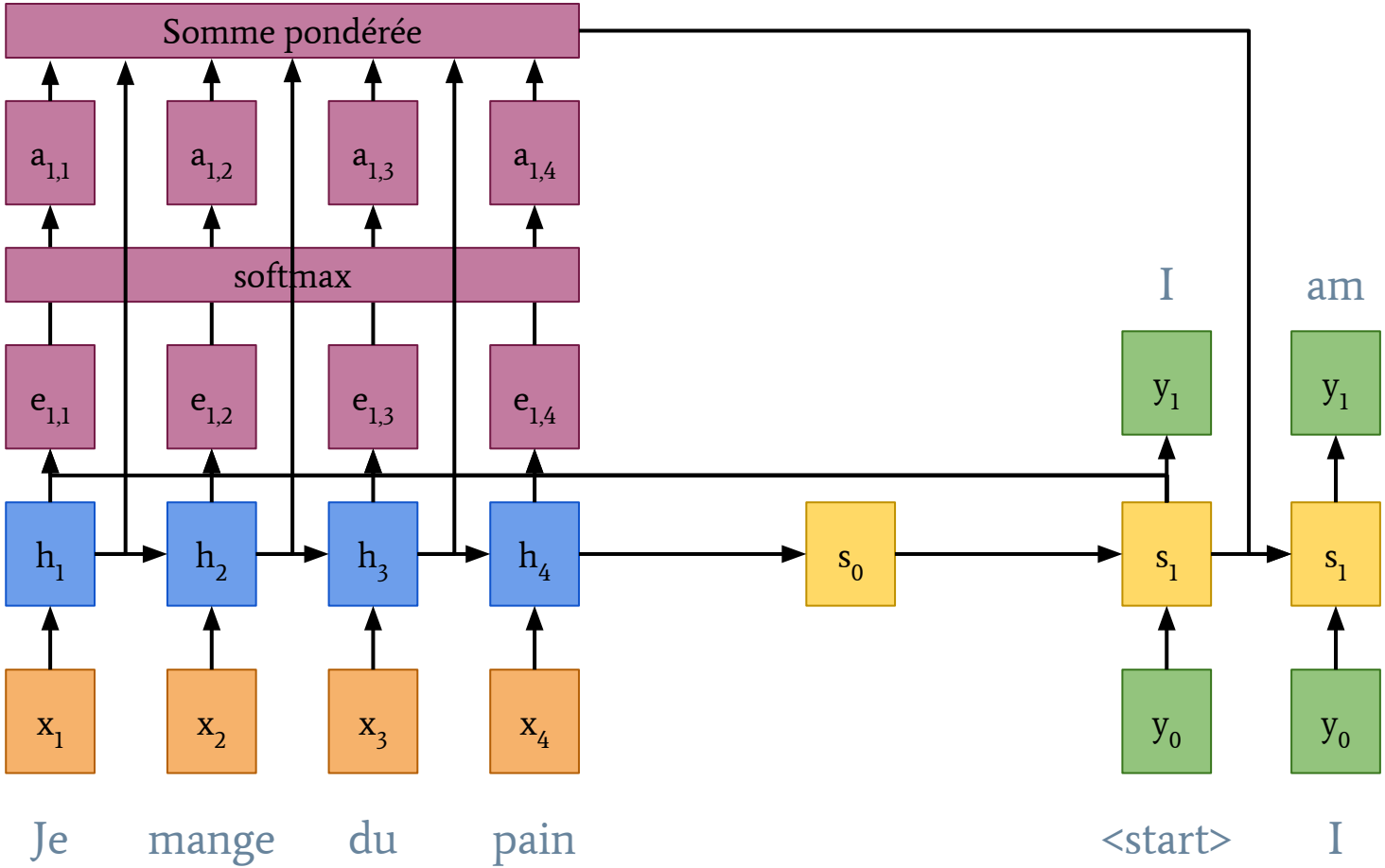
Couche d'attention - Fonction de similarité

- Les paramètres se **seulement** souvent ici.
- Exemples classique :
  - Produit scalaire (pas de paramètre)
  - Produit scalaire normalisé : $e_i = \frac{q \cdot X_i}{\sqrt{D_Q}}$
  - Produit scalaire + fonction linéaire (voir après)

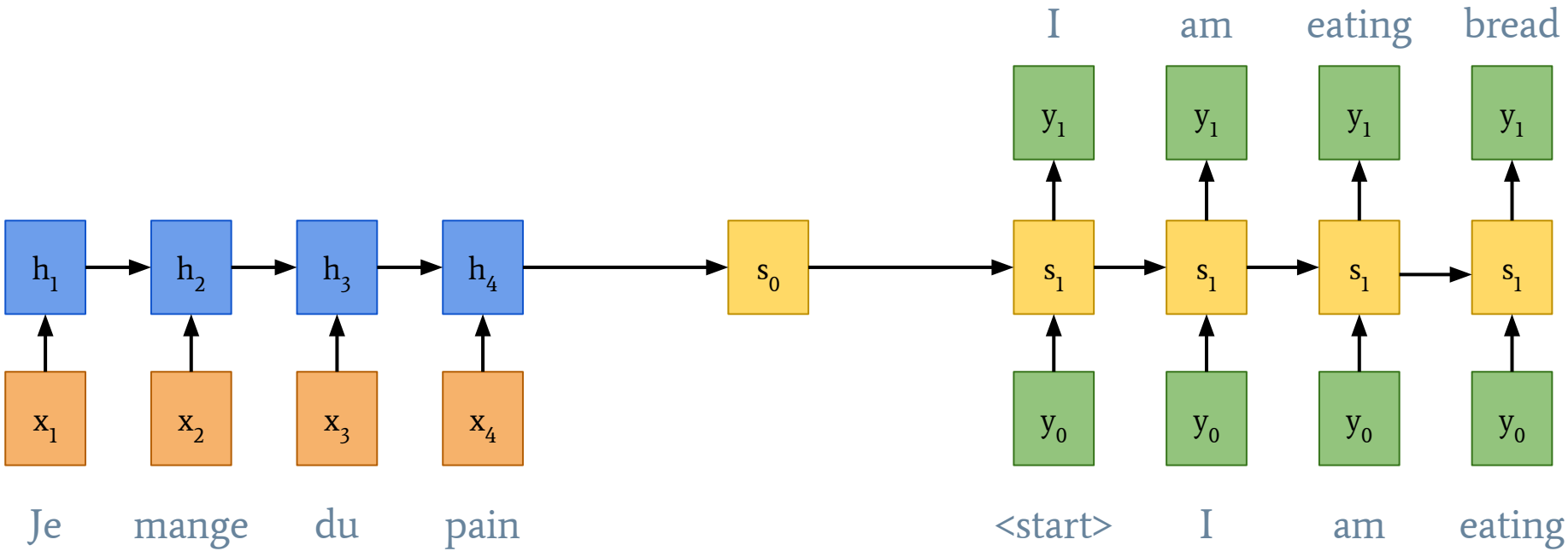
(Origine de la racine carrée. Si a est un vecteur avec toutes ses composantes égales, on a $|a| = \sqrt{\sum_i a^2} = a\sqrt{D_Q}$. La dimension augmente artificiellement la norme. Si la norme est trop grande, le produit scalaire aussi et le softmax sature)


Somme pondérée
$a_{0,1}$
$a_{0,2}$
$a_{0,3}$
$a_{0,4}$
softmax
$e_{0,1}$
$e_{0,2}$
$e_{0,3}$
$e_{0,4}$
h_1
h_2
h_3
h_4
s_0
x_1
x_2
x_3
x_4
Je
mange
du
pain

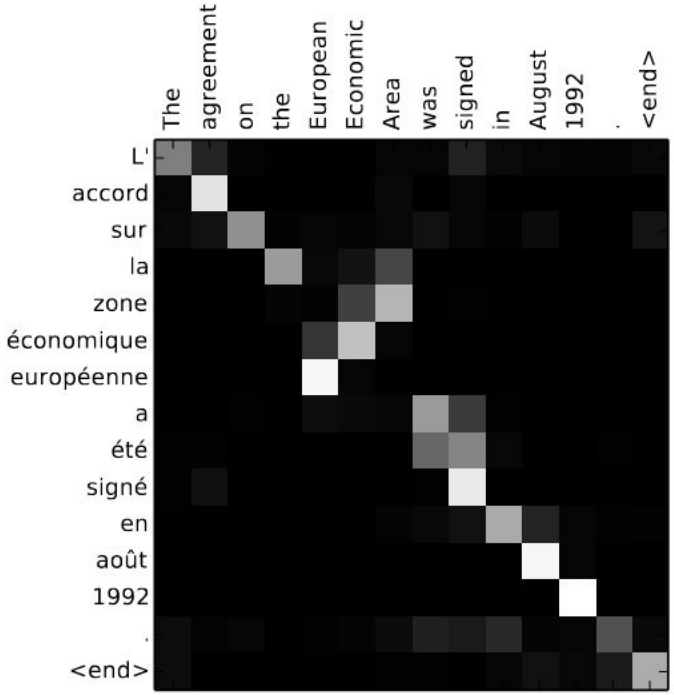

Somme pondérée
$a_{0,1}$
$a_{0,2}$
$a_{0,3}$
$a_{0,4}$
softmax
$e_{0,1}$
$e_{0,2}$
$e_{0,3}$
$e_{0,4}$
h_1
h_2
h_3
h_4
s_0
s_1
x_1
x_2
x_3
x_4
y_1
y_0
I
Je
mange
du
pain
<start>


Somme pondérée
$a_{1,1}$
$a_{1,2}$
$a_{1,3}$
$a_{1,4}$
softmax
$e_{1,1}$
$e_{1,2}$
$e_{1,3}$
$e_{1,4}$
h_1
h_2
h_3
h_4
x_1
x_2
x_3
x_4
Je
mange
du
pain
s_0
s_1
y_0
y_1
<start>
I
s_1
y_0
y_1
I
am

On répète jusqu'à la fin



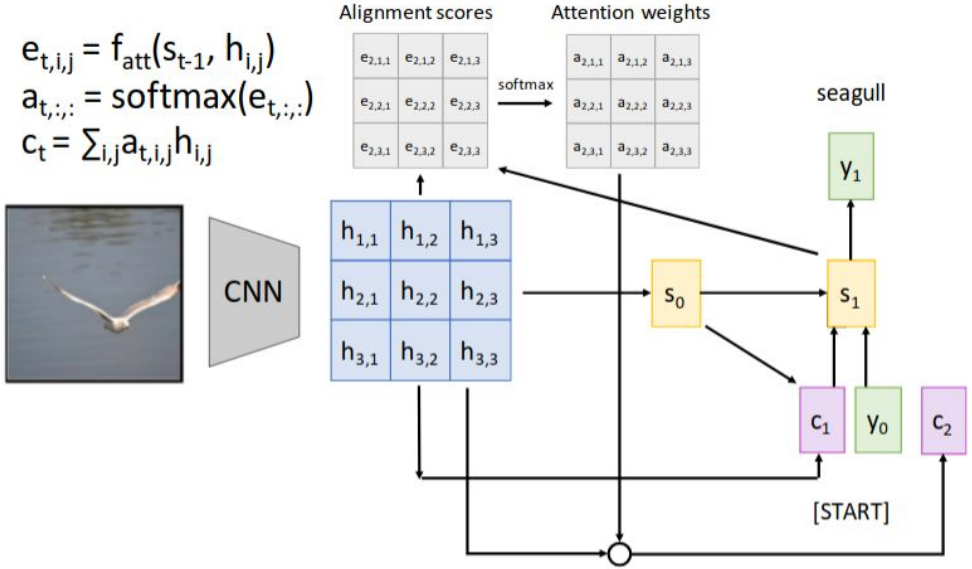
Les poids d'attention sont facilement interprétables



Bahdanau, D. (2014). Neural machine translation by jointly learning to align and translate

On peut faire la même chose avec des images

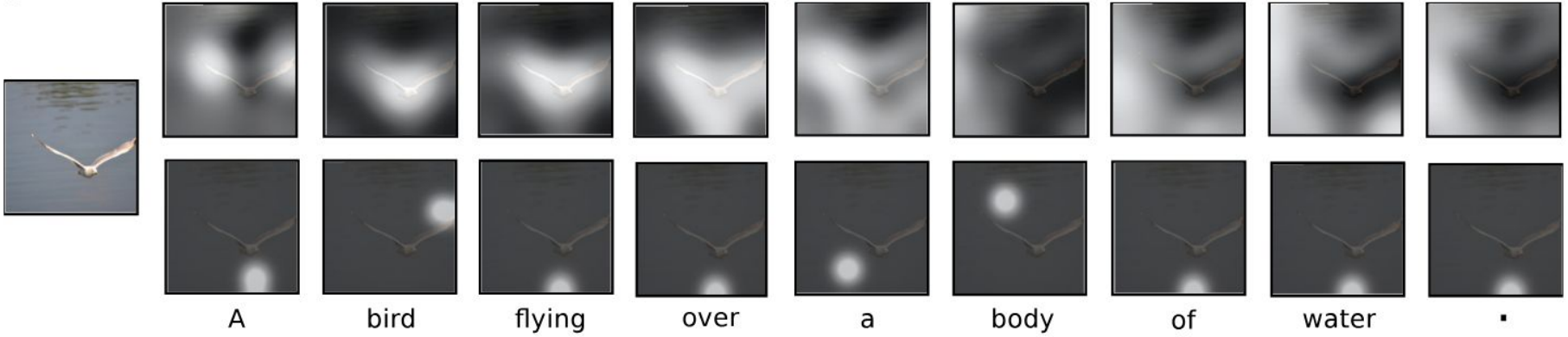
On utilise la sortie des convolutions



Xu, K. (2015). Show, attend and tell: Neural image caption generation with visual attention

On peut faire la même chose avec des images

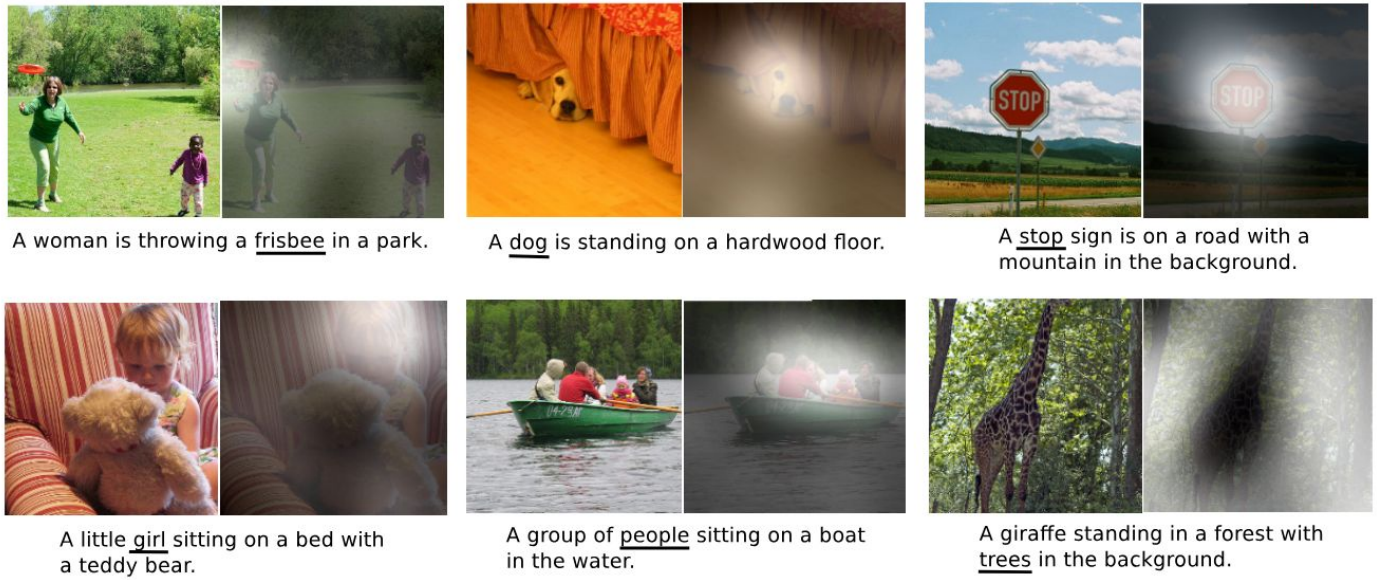
Figure 2. Attention over time. As the model generates each word, its attention changes to reflect the relevant parts of the image. "soft" (top row) vs "hard" (bottom row) attention. (Note that both models generated the same captions in this example.)



Xu, K. (2015). Show, attend and tell: Neural image caption generation with visual attention

On peut faire la même chose avec des images

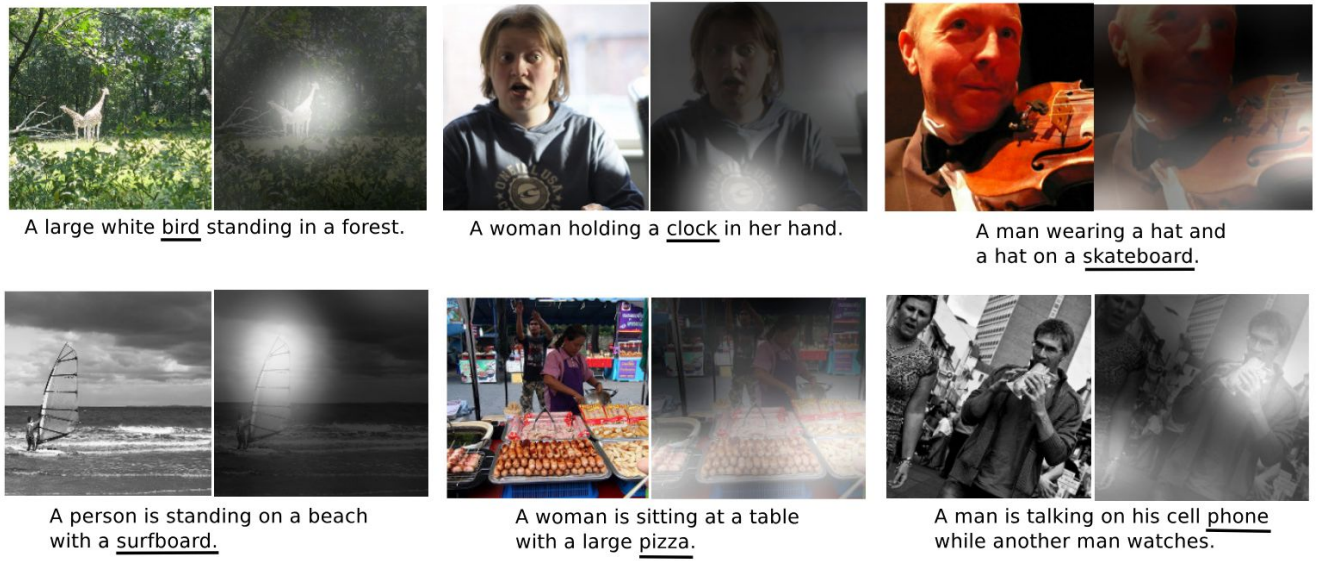
Figure 3. Examples of attending to the correct object (*white* indicates the attended regions, *underlines* indicated the corresponding word)



Xu, K. (2015). Show, attend and tell: Neural image caption generation with visual attention

On peut faire la même chose avec des images

Figure 5. Examples of mistakes where we can use attention to gain intuition into what the model saw.



Xu, K. (2015). Show, attend and tell: Neural image caption generation with visual attention

Couche d'attention - Ajouter de la complexité

Entrées/paramètres :

- **Le vecteur de la requête :** q (dim. D_Q)
- **Le vecteur d'entrée :** X (dim $N_X x D_X$)
- **Une fonction de similarité** : f_{att}

Calculs :

- **Le vecteur de similarité** : e, $e_i = f_{att}(q, X_i)$
- **Les poids de l'attention :** $a = \text{softmax}(e)$
- **Le vecteur de sortie :** $y = \sum_i a_i X_i$

Couche d'attention - La requête peut être une matrice

Entrées/paramètres :

- **La matrices des requêtes :** Q (dim. $N_Q x D_Q$)
- **Le vecteur d'entrée :** X (dim $N_X x D_Q$)
- **Une fonction de similarité** : Produits scalaires

Calculs :

- **Le vecteur de similarité** : $E = QX^T / \text{sqrt}(D_Q)$
- **Les poids de l'attention :** A = softmax(E, dim=1)
- **Le vecteur de sortie :** y = AX

$E_{i,j} = Q_i \cdot X_j / \text{sqrt}(D_Q)$

On compare chaque entrée avec chaque requête

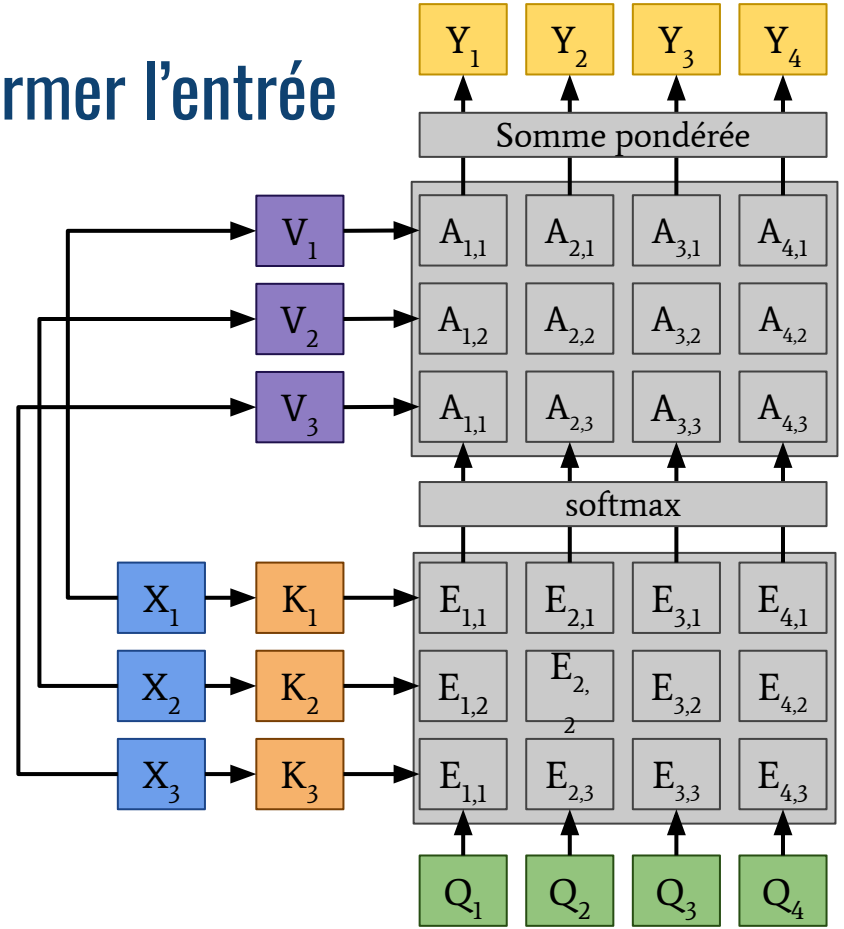
Couche d'attention - On peut transformer l'entrée

Entrées/paramètres :

- La matrices des requêtes : Q (dim. N_QxD_Q)
- Le vecteur d'entrée : X (dim N_XxD_X)
- Une fonction de similarité : Produits scalaires
- Matrice des clefs : W_K (D_XxD_Q)
- Matrice des valeurs : W_V(D_XxD_V)

Calculs :

- Matrice des clefs : $K = XW_K$
- Matrice des valeurs : $V = XW_V$
- Le vecteur de similarité : $E = QK^T / \text{sqrt}(D_Q)$
- Les poids de l'attention : A = softmax(E, dim=1)
- Le vecteur de sortie : y = AV



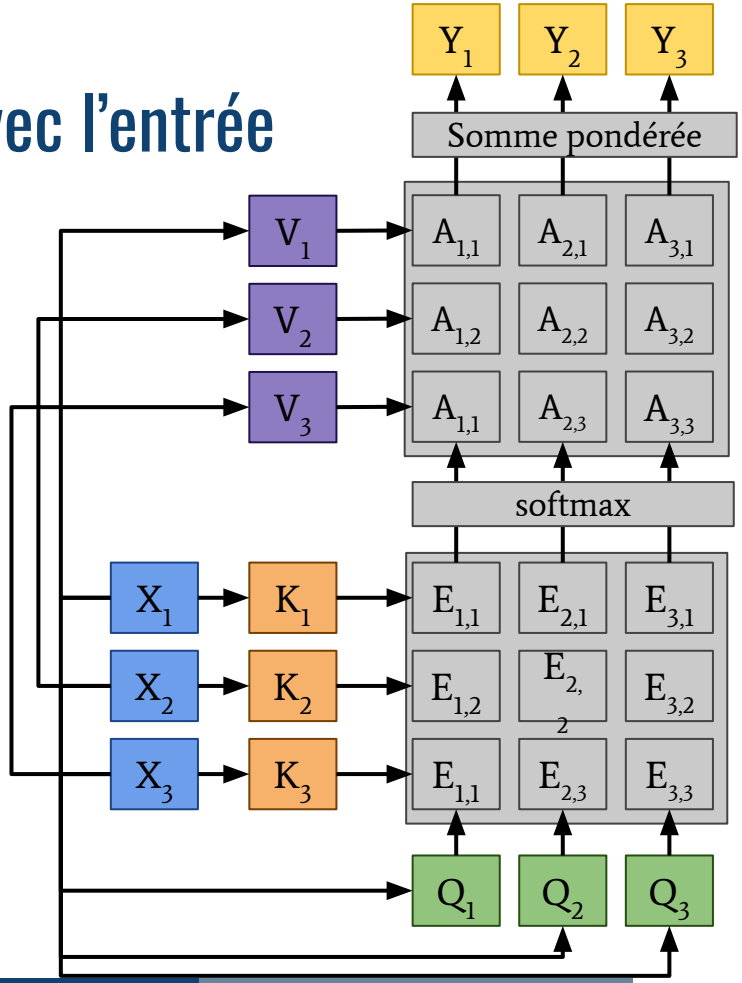
Self attention - La requête est calculée avec l'entrée

Entrées/paramètres :

- Le vecteur d'entrée : X (dim N_XxD_X)
- Une fonction de similarité : Produits scalaires
- Matrice des clefs : W_K (D_XxD_Q)
- Matrice des valeurs : W_V(D_XxD_V)
- La matrices des requêtes : W_Q (dim. D_XxD_Q)

Calculs :

- Vecteurs de requête : $Q = XW_Q$
- Matrice des clefs : $K = XW_K$
- Matrice des valeurs : $V = XW_V$
- Le vecteur de similarité : $E = QK^T/ sqrt(D_Q)$
- Les poids de l'attention : A = softmax(E, dim=1)
- Le vecteur de sortie : y = AV



Self attention - Remarque sur la position

La self-attention est invariante par une permutation

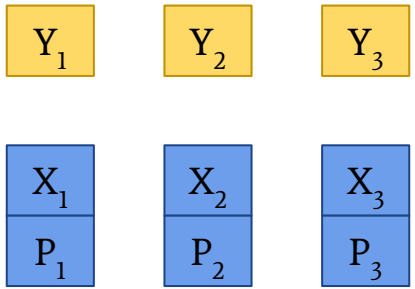
=> Changer la position de chaque entrée change la position de la sortie de la même manière.

Self attention - Remarque sur la position

La self-attention est invariante par une permutation

=> Changer la position de chaque entrée change la position de la sortie de la même manière.

Solution : encoder la position dans l'entrée (c.f. Cours suivant)

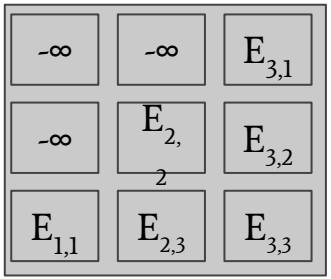


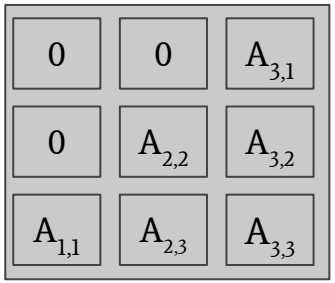
Self attention - Remarque sur le futur

- Pendant l'entraînement, on ne peut pas toujours supposé que l'on a accès aux entrées futures
  - Mais on veut quand même faire une multiplication de matrice (parallélisable)

Self attention - Remarque sur le futur

- Pendant l'entraînement, on ne peut pas toujours supposé que l'on a accès aux entrées futures
  - Mais on veut quand même faire une multiplication de matrice (parallélisable)
- Solution : Cacher manuellement des morceaux de matrices (Masked self attention)





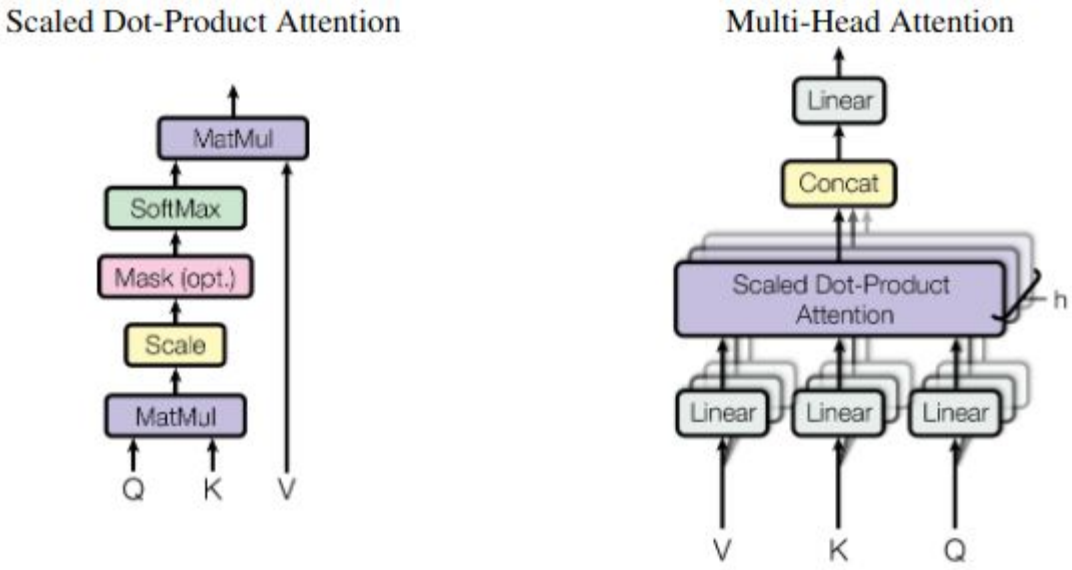
Multihead - Multiplier les couches d'attention sans augmenter la profondeur.

Si : $Attention(Q, K, V) = softmax(\frac{QK^T}{D_K})V$

Alors : $Multihead(Q, K, V) = Concat(head_1, \ldots, head_h)W^O$

Avec : $head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$

Multihead - Multiplier les couches d'attention sans augmenter la profondeur.



Vaswani, A. (2017). Attention is all you need

RNN vs Self-Attention

RNN

+ Fonctionne avec de longues séquences
+ Peu de paramètres
- Non parallélisable

Self-attention

+ Fonctionne avec de longues séquences
+ Hautement parallélisable : chaque sortie est calculée en parallèle
- Demande beaucoup de mémoire

La suite...

La self-attention nous emmène à Transformers qui nous emmène à BERT, GPT, ChatGPT, DeepSeek, ...

On verra tout ça dans le prochain cours.

En résumé

- L'attention est un mécanisme permettant de se concentrer sur une partie précise de l'entrée
- Plusieurs variantes :
  - Self-attention
  - Masked self-attention
  - Multi-head