Réseaux neuronaux simples et la régression logistique

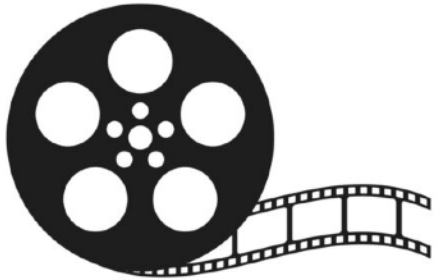
TALN semaine 5

Thanks to Dan Jurafsky for slides and inspiration!

Plan pour aujourd'hui

1. **Classification du texte**
2. **Unités neuronales**
3. **Architectures de réseaux neuronaux simples et régression logistique**
4. **Modèles d'entraînement et rétro-propagation**
5. ***Exercices de groupe***

Classification du texte



Critique de film positive ou négative ?

- unbelievably disappointing
- Full of zany characters and richly applied satire, and some great plot twists
- this is the greatest screwball comedy ever filmed
- It was pathetic. The worst part about it was the boxing scenes.

Classification du texte

- **D’autres exemples:**
  - Classification de critique
  - Détection de spam
  - Analyse des sentiments
  - Détection de l'auteur
  - Assignation de sujet

... Et la liste est longue !

Classification du texte

La classification du texte est...

La tâche d'attribuer une étiquette de classe à un document ou à un texte.

Input:
- a document d
- a fixed set of classes $C = \{c_1, c_2, \ldots, c_J\}$

Output: a predicted class $c \in C$

Classification du texte

La classification du texte est...

La tâche d'attribuer une étiquette de classe à un document ou à un texte.

Input:
- a document d <- “This was the greatest comedy of the year”
- a fixed set of classes $C = \{c_1, c_2, \ldots c_J\}$ <- {Positif, Neutre, Negatif}

Output: a predicted class $c \in C$

Classification du texte

À quoi devrait ressembler notre algorithme de classification ?

Input

Mais “This was the greatest comedy of the year”

C = {Positif, Negatif}

Classifier

Heuristic/rule-based

```
If “greatest” in d:
        Return positive
Else if “worst” in d:
        Return negative
```

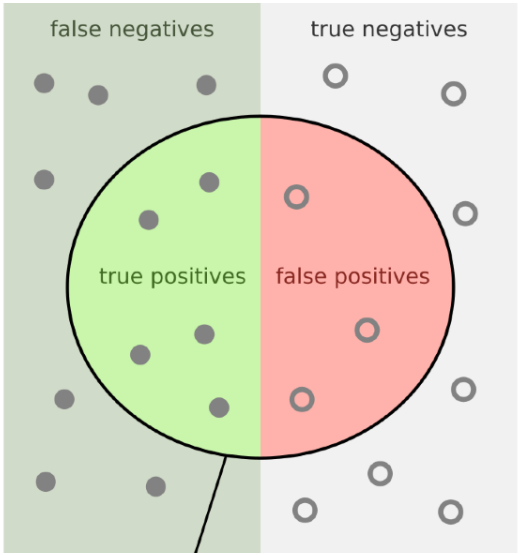
Output

c = Positif

Précision et rappel

- Les classificateurs heuristiques ont tendance à avoir une grande **précision** mais un très faible **rappel**.

- L'exactitude du classificateur est mesurée avec précision et rappel :

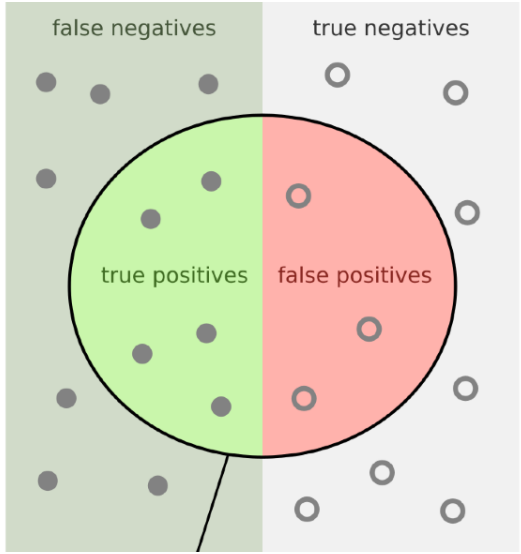


Rappel = TP / (**FN** +TP) Précision = TP / (**FP** +TP)

Précision et rappel

score F1 est la moyenne harmonique des deux :

$$F_1 = \frac{2}{\mathrm{recall}^{-1} + \mathrm{precision}^{-1}} = 2\frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}} = \frac{2\mathrm{TP}}{2\mathrm{TP} + \mathrm{FP} + \mathrm{FN}}$$



How many relevant items are retrieved?

Recall =

How many retrieved items are relevant?

Precision =

Rappel = TP / (**FN** +TP) Précision = TP / (**FP** +TP)

Apprentissage supervisé

Une alternative aux classificateurs heuristiques est d'utiliser l'apprentissage automatique ***pour apprendre un algorithme plutôt que de définir un algorithme*** qui classera correctement les textes.

Nous le faisons en utilisant l'**apprentissage supervisé** à l'aide de données de formation correctement **étiquetées** avec les classes souhaitées. Nous essaierons ensuite d'apprendre un ensemble de **caractéristiques partagées** à travers les documents pour chaque classe.

Input:

- a document d
- a fixed set of classes $C = \{c_1, c_2, \ldots, c_J\}$
- A training set of m hand-labeled documents $(d_1,c_1),\ldots,(d_m,c_m)$

Output:

- a learned classifier $\gamma:d \rightarrow c$

Apprentissage supervisé

Il existe de nombreux types de classificateurs appris :

Naive Bayes

Régression logistique (ce cours)

Réseaux neuronaux (ce cours)

...

Même les LLM affinés ou par apprentissage éphémère contextuel avec LLMs

Tous ont besoin de données d'entraînement étiquetées pour faire la classification

Unité de réseau neuronal

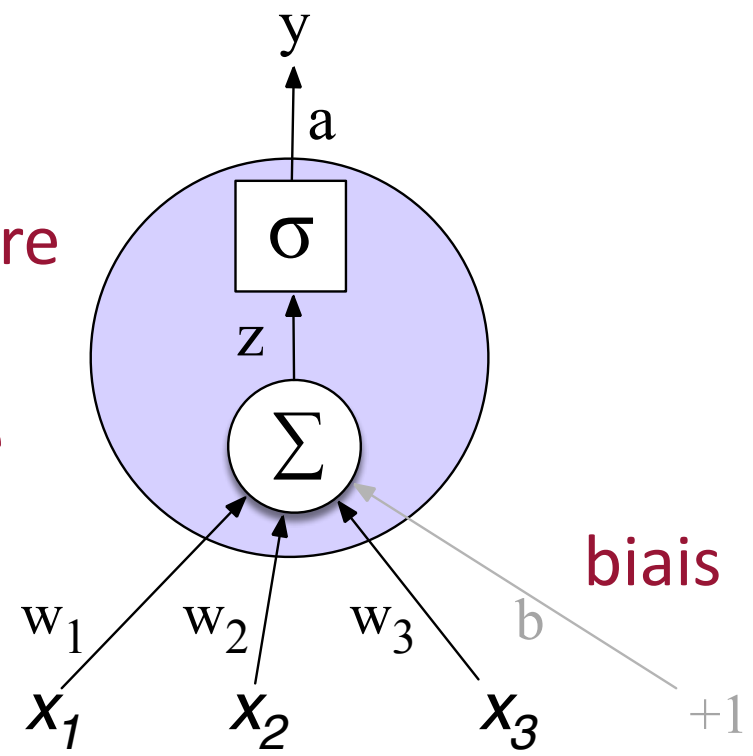
Valeur de sortie

Transformation non linéaire

Somme pondérée

Poids

Couche d'entrée



Unité de réseau neuronal

Prendre la somme pondérée des entrées, plus le biais

$$z = b + \sum_i w_i x_i$$

$$z = w \cdot x + b$$

Au lieu d'utiliser simplement z, nous appliquerons une fonction d'activation non linéaire f :

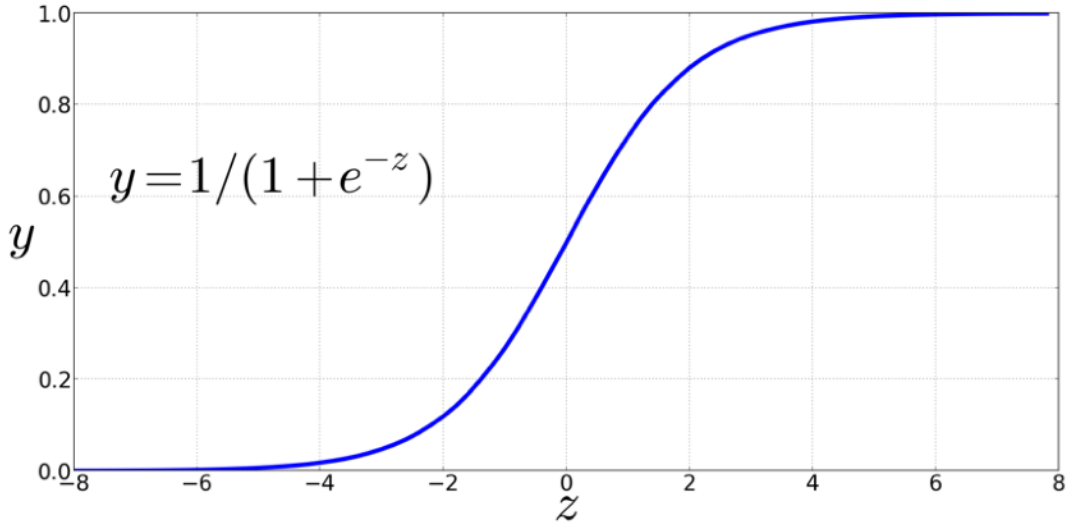
$$y = a = f(z)$$

Fonctions d'activation non linéaires

La fonction sigmoïde pour la régression logistique :

Sigmoïde

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



Fonction finale calculé par l'unité

$$y = \sigma(w \cdot x + b) = \frac{1}{1 + \exp(-(w \cdot x + b))}$$

Unité de réseau neuronal encore une fois

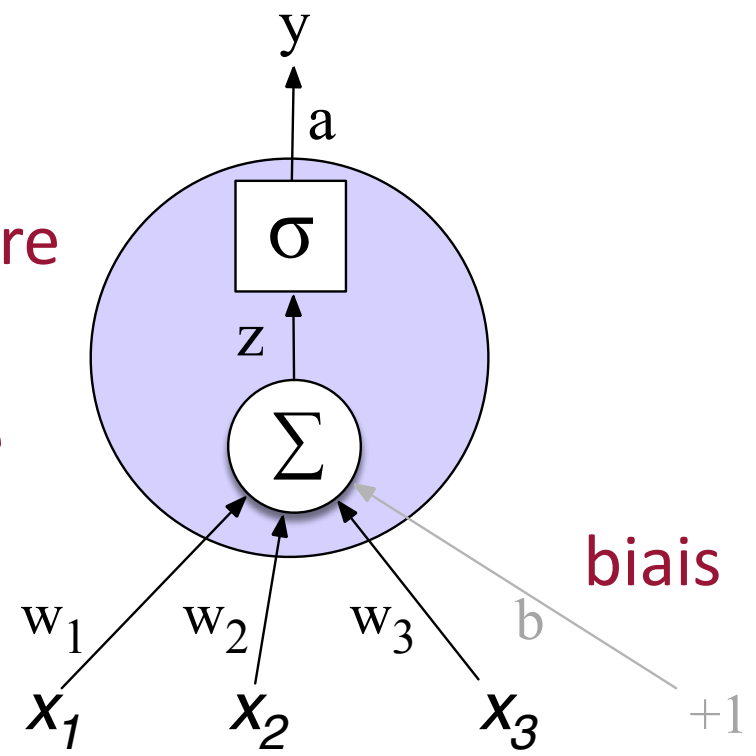
Valeur de sortie

Transformation non linéaire

Somme pondérée

Poids

Couche d'entrée



Exemple

Supposons qu'une unité ait :

w = [0.2,0.3,0.9]

b = 0.5

Que se passe-t-il avec l'entrée x :

x = [0.5,0.6,0.1]

$$y = \sigma(w \cdot x + b) = \frac{1}{1 + e^{-(w \cdot x + b)}} =$$

An example

Supposons qu'une unité ait :

w = [0.2,0.3,0.9]

b = 0.5

Que se passe-t-il avec l'entrée x :

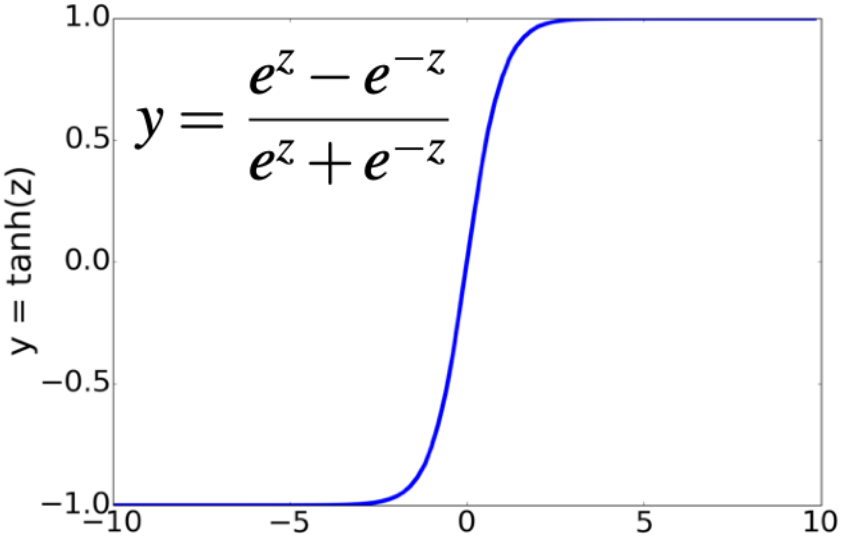
x = [0.5,0.6,0.1]

$$y = \sigma(w \cdot x + b) = \frac{1}{1 + e^{-(w \cdot x + b)}} =$$

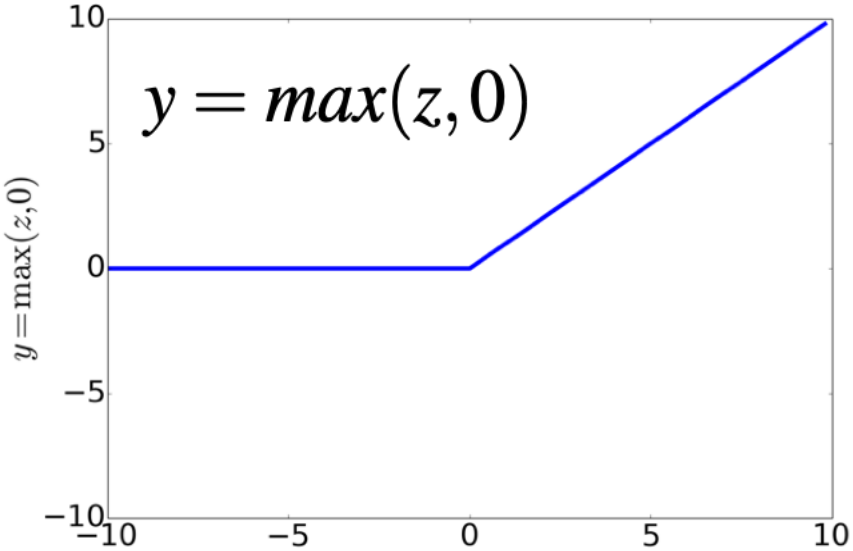
$$\frac{1}{1 + e^{-(.5*.2+.6*.3+.1*.9+.5)}} = \frac{1}{1 + e^{-0.87}} = .70$$

Fonctions d'activation non linéaires autres que sigmoïde

Les plus communs:



tanh

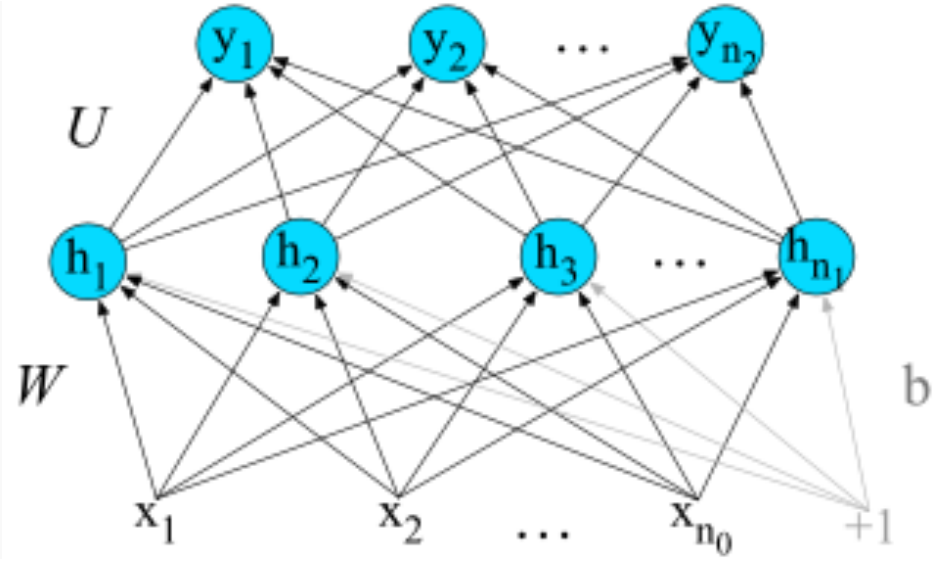


ReLU

Rectified Linear Unit

Réseaux neuronaux Feedforward (FNN)

Peut également être appelé perceptrons multicouches (ou MLP) pour des raisons historiques



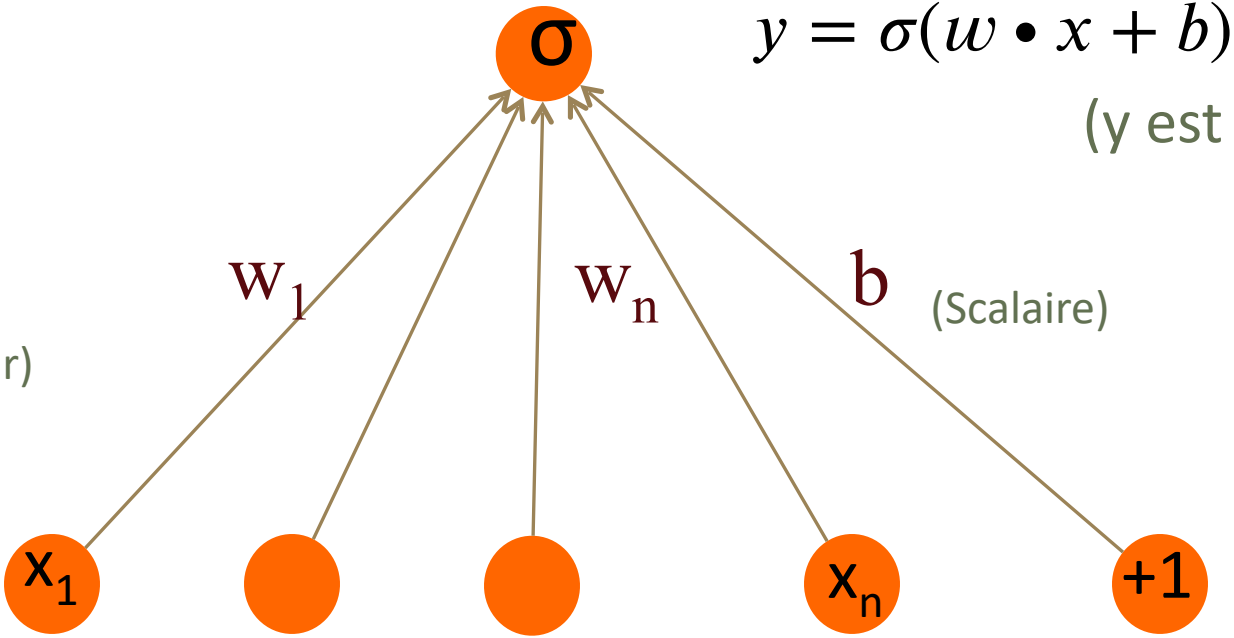
Régression logistique binaire avec un réseau à 1 couche

(Nous ne comptons pas la couche d'entrée dans le nombre de couches!)

Couche de sortie (Noeud σ)

$$y = \sigma(w \bullet x + b)$$

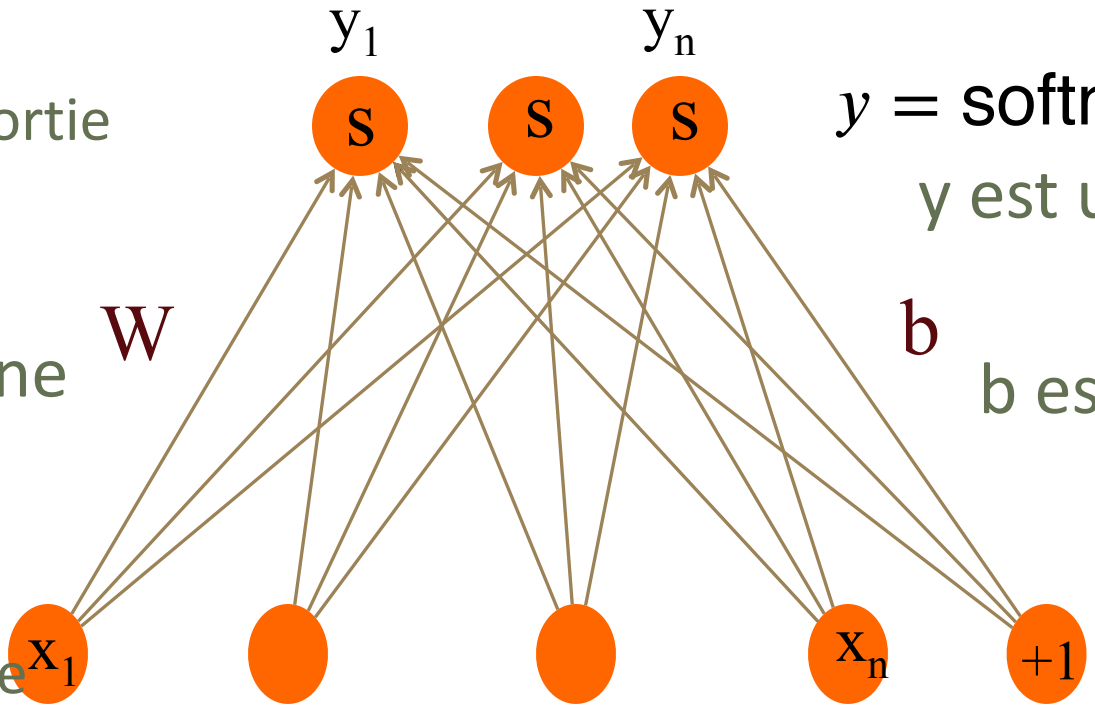
(y est scalaire)



Couche d'entrée vecteur x

Régression logistique multinomiale avec un réseau à 1 couche

Réseau à une couche entièrement connecté



Couche de sortie

$$y = \text{softmax}(Wx + b)$$

y est un vecteur

W

W est une matrice

b

b est un vecteur

Couche d'entrée

x est un vecteur

Softmax : une généralisation du sigmoïde

Pour un vecteur z de dimensionnalité k, le softmax est :

$$\text{softmax}(z) = \left[\frac{\exp(z_1)}{\sum_{i=1}^{k} \exp(z_i)}, \frac{\exp(z_2)}{\sum_{i=1}^{k} \exp(z_i)}, \ldots, \frac{\exp(z_k)}{\sum_{i=1}^{k} \exp(z_i)} \right]$$

$$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_{j=1}^{k} \exp(z_j)} \quad 1 \leq i \leq k$$

Exemple:

$$z = [0.6, 1.1, -1.5, 1.2, 3.2, -1.1]$$

$$\text{softmax}(z) = [0.055, 0.090, 0.006, 0.099, 0.74, 0.010]$$

Réseau à 2 couches avec sortie scalaire (classification binaire)

y est scalaire

$$y = \sigma(z)$$
$$z = Uh$$

Couche de sortie
(Noeuds σ)

U

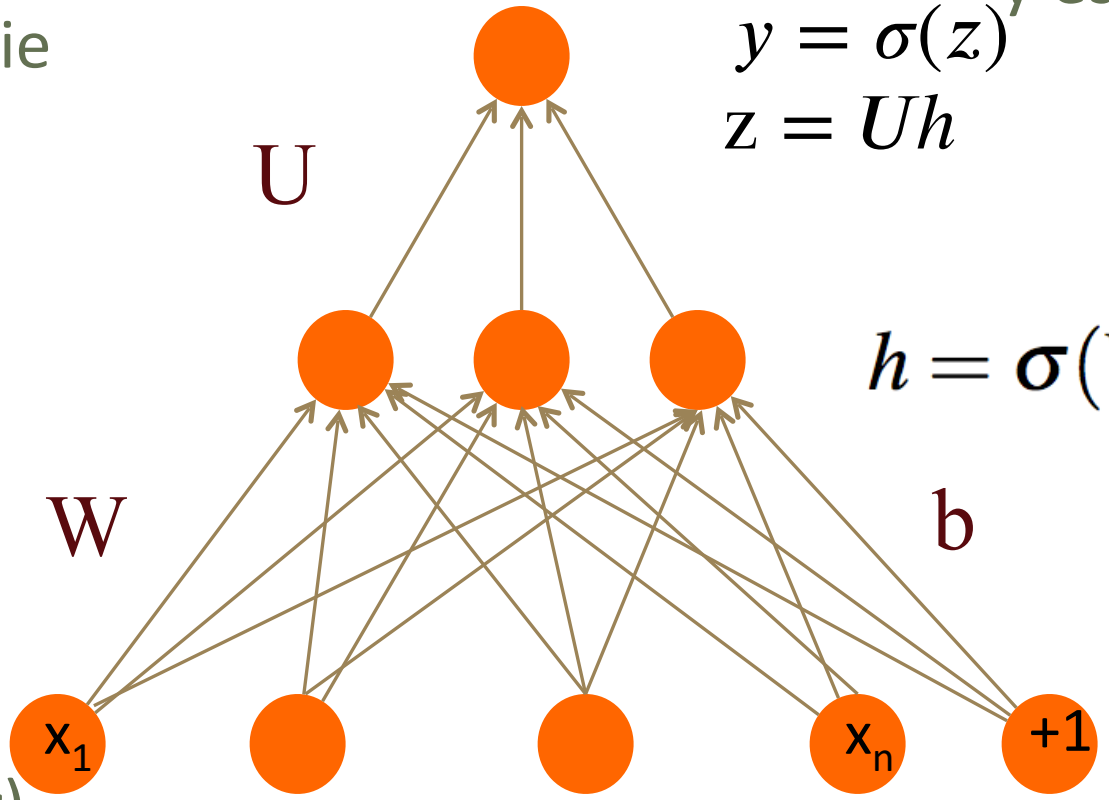
Unités cachées
(Noeuds σ)

$$h = \sigma(Wx + b)$$

Pourrait être
ReLU ou tanh

W

b



Couche d'entrée
(X est un vecteur)

Réseau à 2 couches avec sortie scalaire (classification binaire)

y est scalaire

$$y = \sigma(z)$$
$$z = Uh$$

Couche de sortie
(Noeuds σ)

U

Unités cachées
(Noeuds σ)

$$h = \sigma(Wx + b)$$

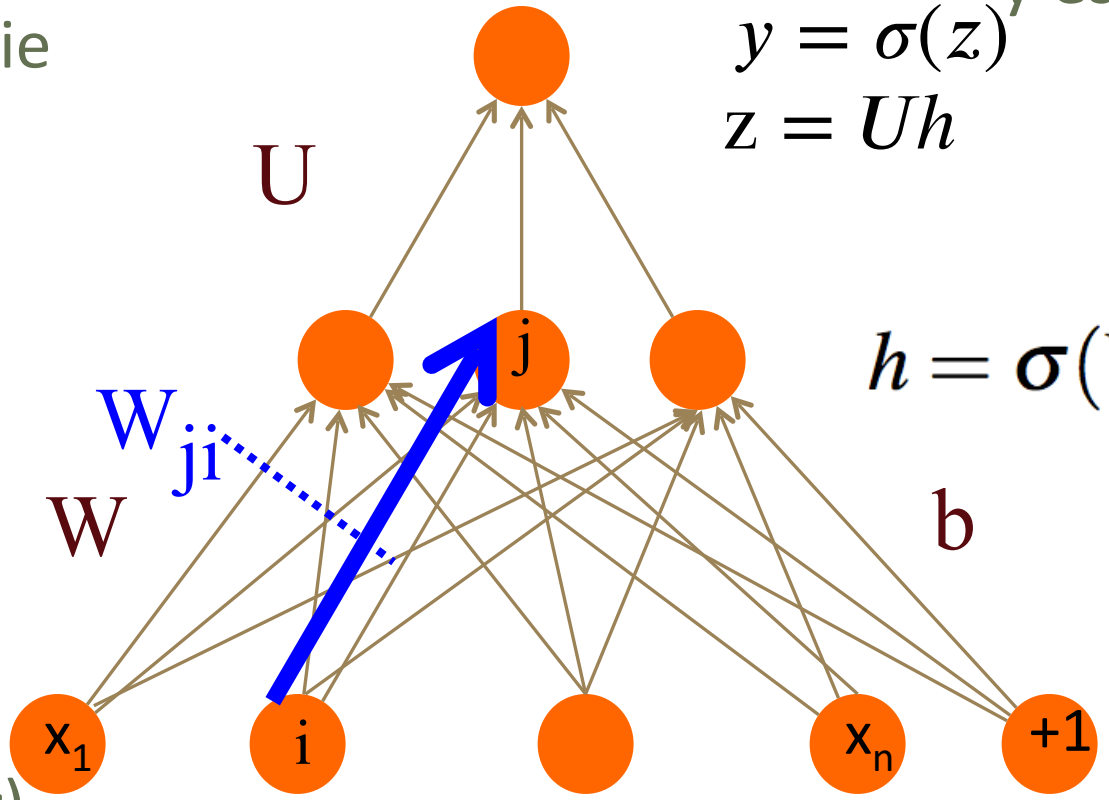
Pourrait être
ReLU ou tanh

W_{ji}

W

b

Couche d'entrée
(X est un vecteur)



Réseau à 2 couches avec sortie scalaire (classification binaire)

y est scalaire

$$y = \sigma(z)$$
$$z = Uh$$

Couche de sortie
(Noeuds σ)

U

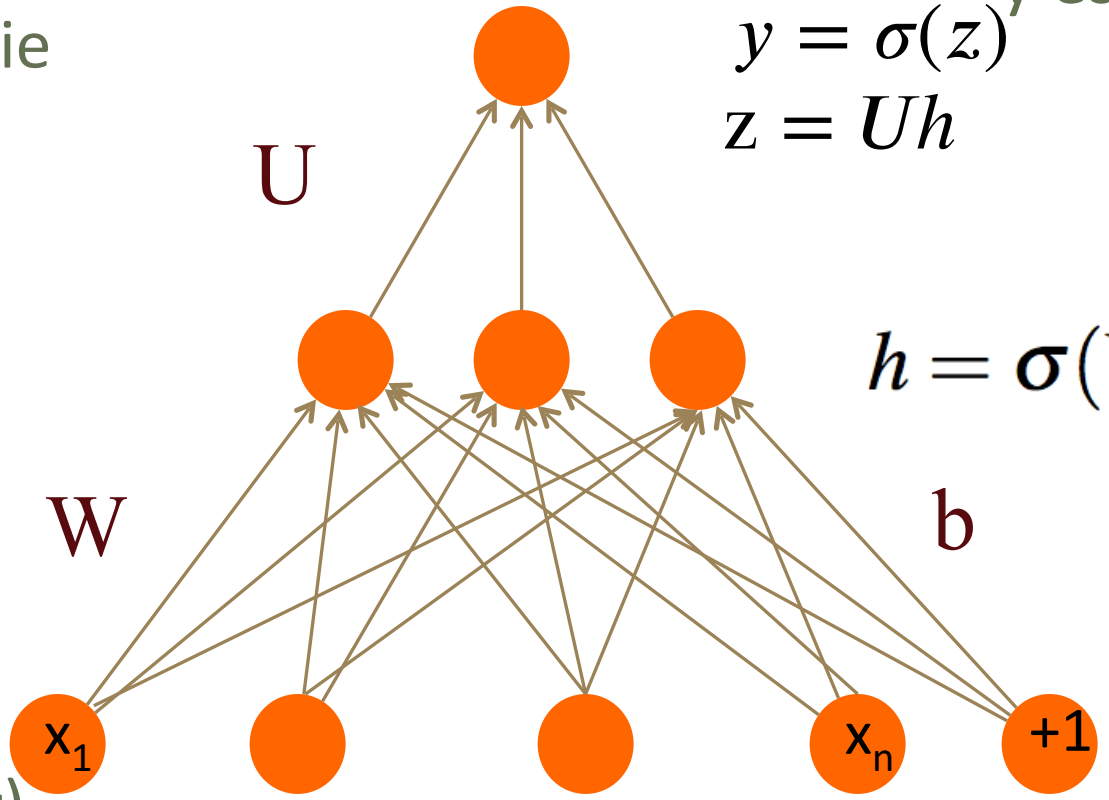
Unités cachées
(Noeuds σ)

$$h = \sigma(Wx + b)$$

Pourrait être
ReLU ou tanh

W

b



Couche d'entrée
(X est un vecteur)

Réseau à 2 couches avec sortie softmax (multinomiale)

Couche de sortie
(Noeuds softmax)

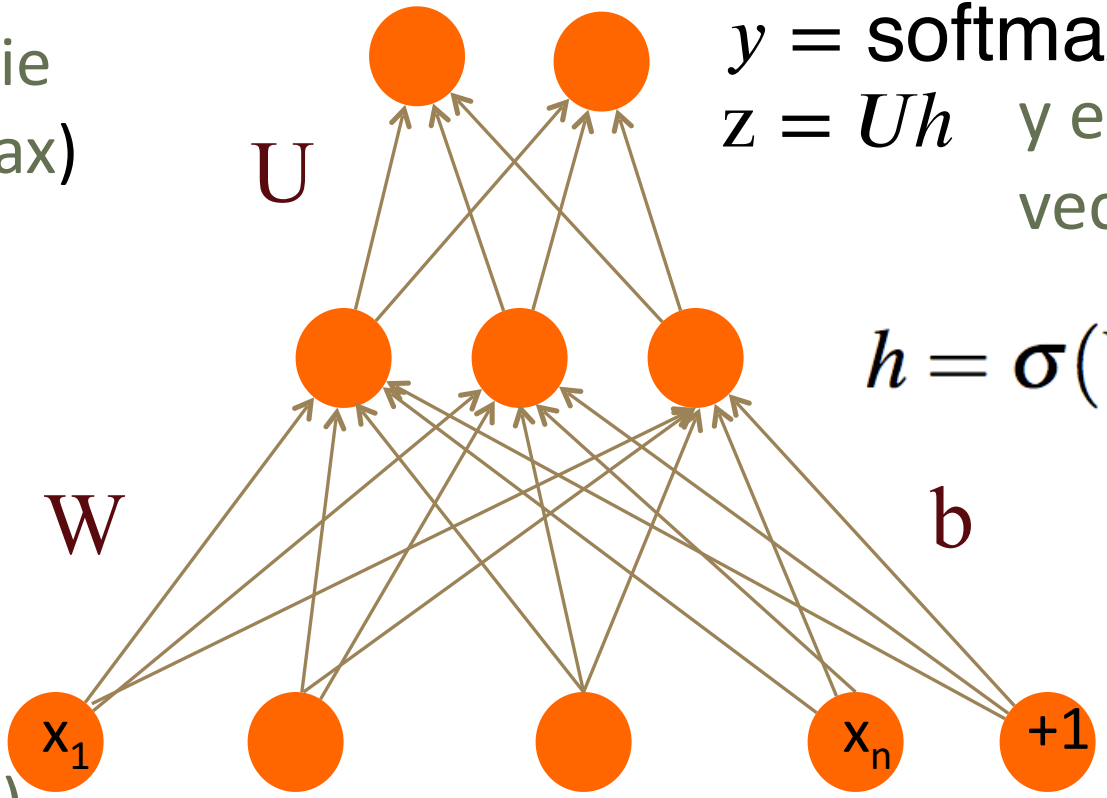
$$y = \text{softmax}(z)$$
$$z = Uh$$

y est un vecteur

Unités cachées
(Noeuds σ)

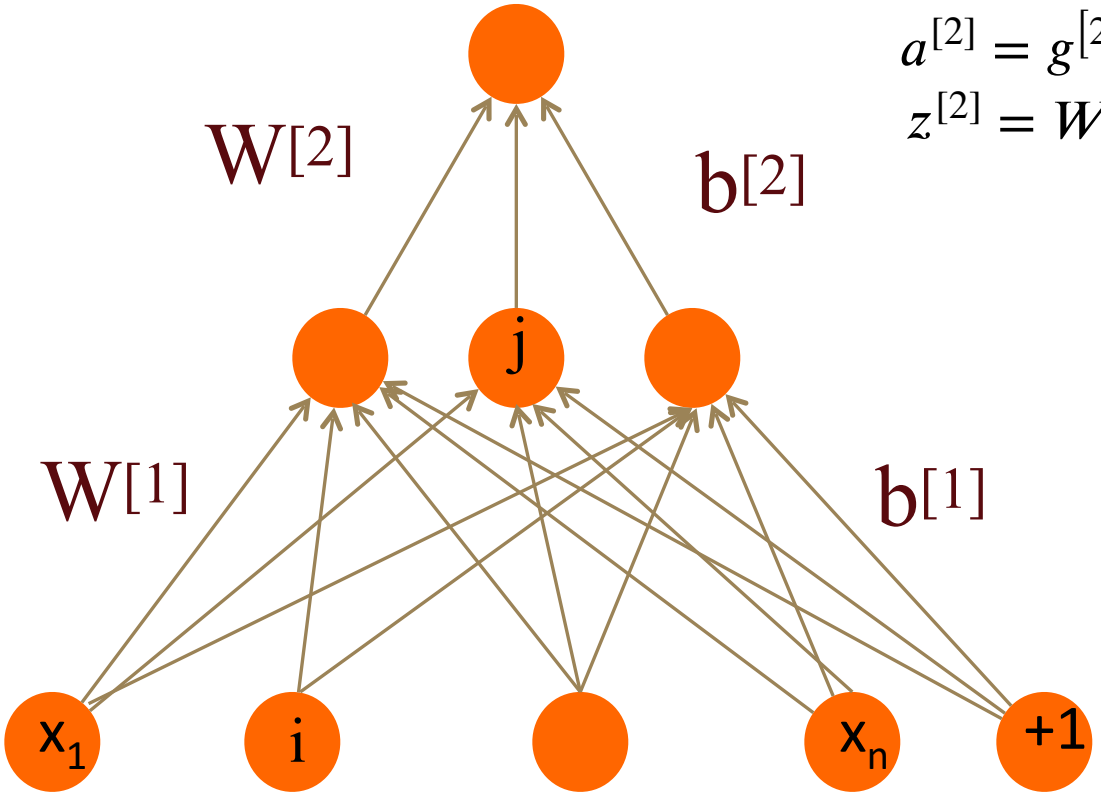
$$h = \sigma(Wx + b)$$

Pourrait être ReLU ou tanh



Couche d'entrée
(X est un vecteur)

Notation multicouche



$$y = a^{[2]}$$

$$a^{[2]} = g^{[2]}(z^{[2]})$$ Sigmoide ou softmax

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

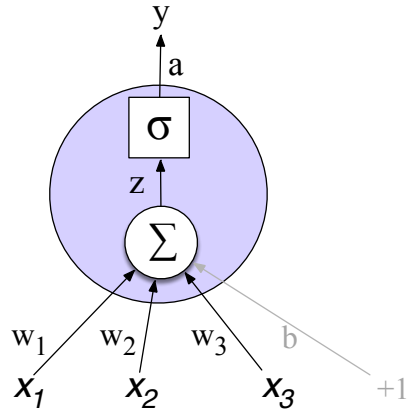
$$a^{[1]} = g^{[1]}(z^{[1]})$$ ReLU

$$z^{[1]} = W^{[1]}a^{[0]} + b^{[1]}$$

$a^{[0]}$

Notation multicouche

$$
\begin{aligned}
z^{[1]} &= W^{[1]} a^{[0]} + b^{[1]} \\
a^{[1]} &= g^{[1]}(z^{[1]}) \\
z^{[2]} &= W^{[2]} a^{[1]} + b^{[2]} \\
a^{[2]} &= g^{[2]}(z^{[2]}) \\
\hat{y} &= a^{[2]}
\end{aligned}
$$



for i **in** 1..n

$$
\begin{aligned}
z^{[i]} &= W^{[i]} a^{[i-1]} + b^{[i]} \\
a^{[i]} &= g^{[i]}(z^{[i]}) \\
\hat{y} &= a^{[n]}
\end{aligned}
$$

Replacing the bias unit

Let's switch to a notation without the bias unit

Just a notational change

1. Add a dummy node $a_0=1$ to each layer
2. Its weight w_0 will be the bias
3. So input layer $a^{[0]}_0=1$,
   - And $a^{[1]}_0=1$, $a^{[2]}_0=1$,…

Remplacement du terme de biais

Au lieu de :

$$x = x_1, x_2, \ldots, x_{n0}$$

$$h = \sigma(Wx + b)$$

$$h_j = \sigma\left(\sum_{i=1}^{n_0} W_{ji}x_i + b_j\right)$$

Nous allons faire ceci :

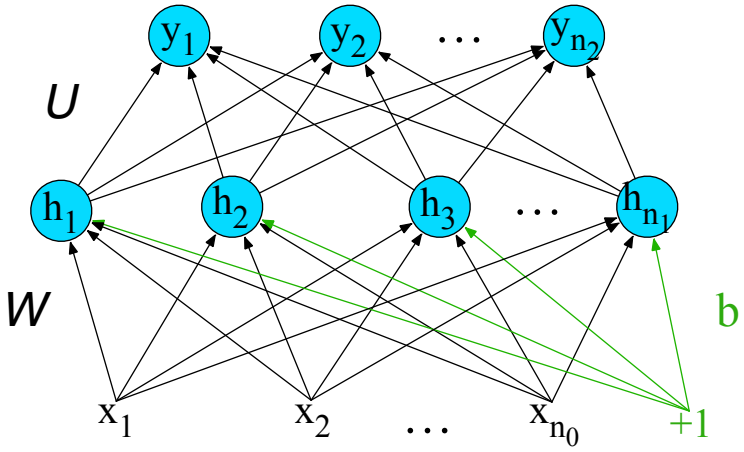
$$x = x_0, x_1, x_2, \ldots, x_{n0}$$

$$h = \sigma(Wx)$$

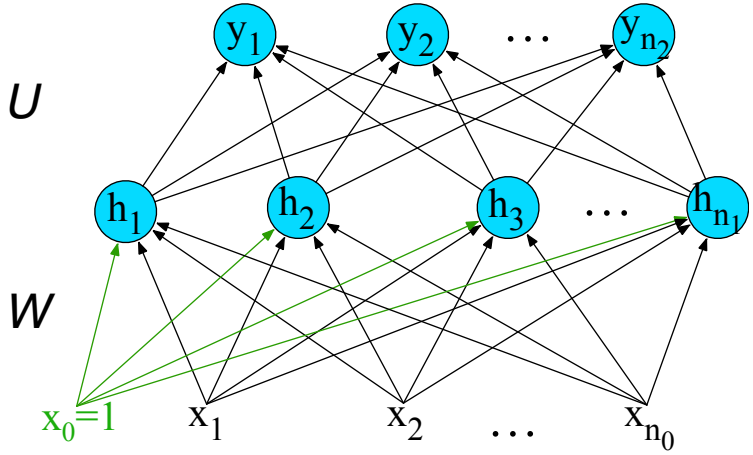
$$\sigma\left(\sum_{i=0}^{n_0} W_{ji}x_i\right)$$

Remplacement du terme de biais

Au lieu de :



Nous allons faire ceci :



Cas d'utilisation pour les réseaux feedforward

Considérons 2 exemples de tâches (simplifiées) :

1. Classification du texte
2. Modélisation du langage

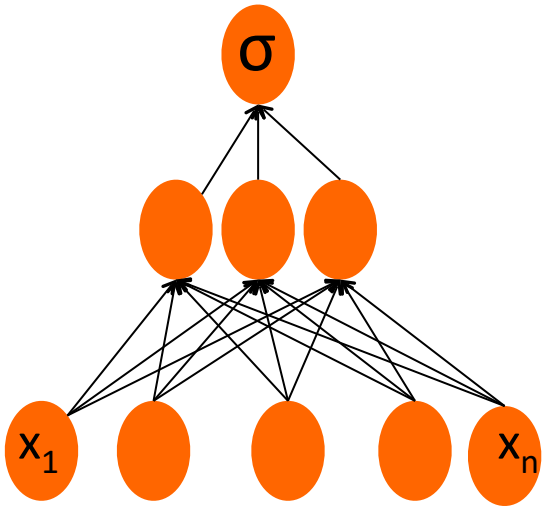
Les systèmes de pointe utilisent des architectures neuronales plus puissantes, mais des modèles simples sont utiles à considérer !

1. Classification: Analyse de sentiments

Nous pourrions faire exactement ce que nous avons fait avec la régression logistique

La couche d'entrée sont des caractéristiques binaires comme avant (ex. Contient le mot “great”)

La couche de sortie est 0 (neg) ou 1 (pos)

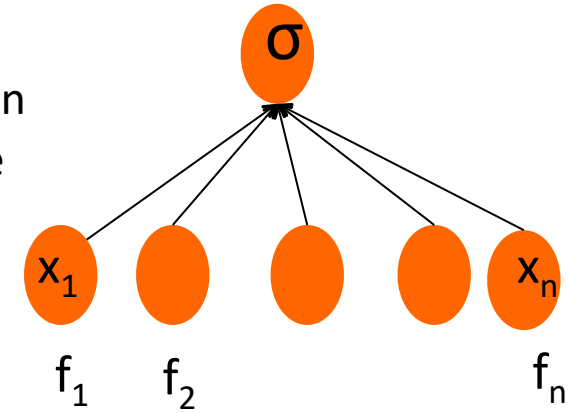


Exemples de caractéristiques d'entrée

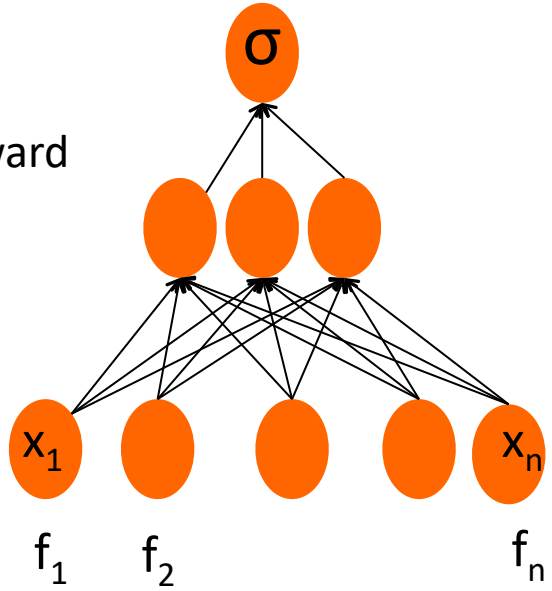
Var	Definition
x_1	count(positive lexicon) $\in$ doc)
x_2	count(negative lexicon) $\in$ doc)
x_3	$\begin{cases} 1 & \text{if } \text{“no”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$
x_4	count(1st and 2nd pronouns $\in$ doc)
x_5	$\begin{cases} 1 & \text{if } \text{“!”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$
x_6	log(word count of doc)

Réseau Feedforward pour classification binaire

Régression logistique



Réseau Feedforward à 2 couches



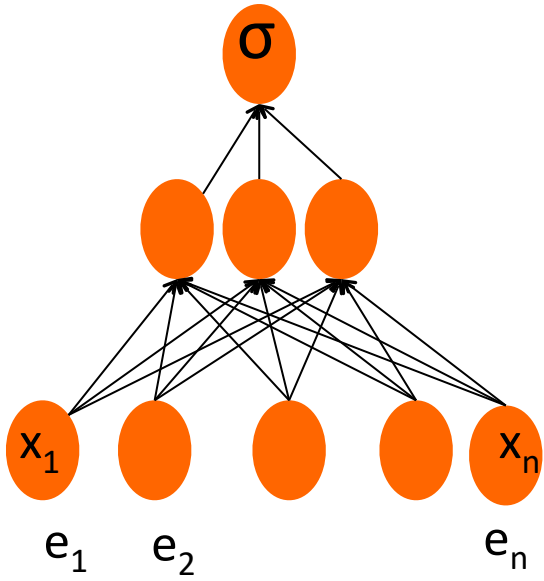
Il suffit d'ajouter une couche cachée à la régression logistique. Permet au réseau d'utiliser des interactions non linéaires entre les caractéristiques qui peuvent (ou non) améliorer les performances.

Encore mieux : l'apprentissage de la représentation (embeddings)

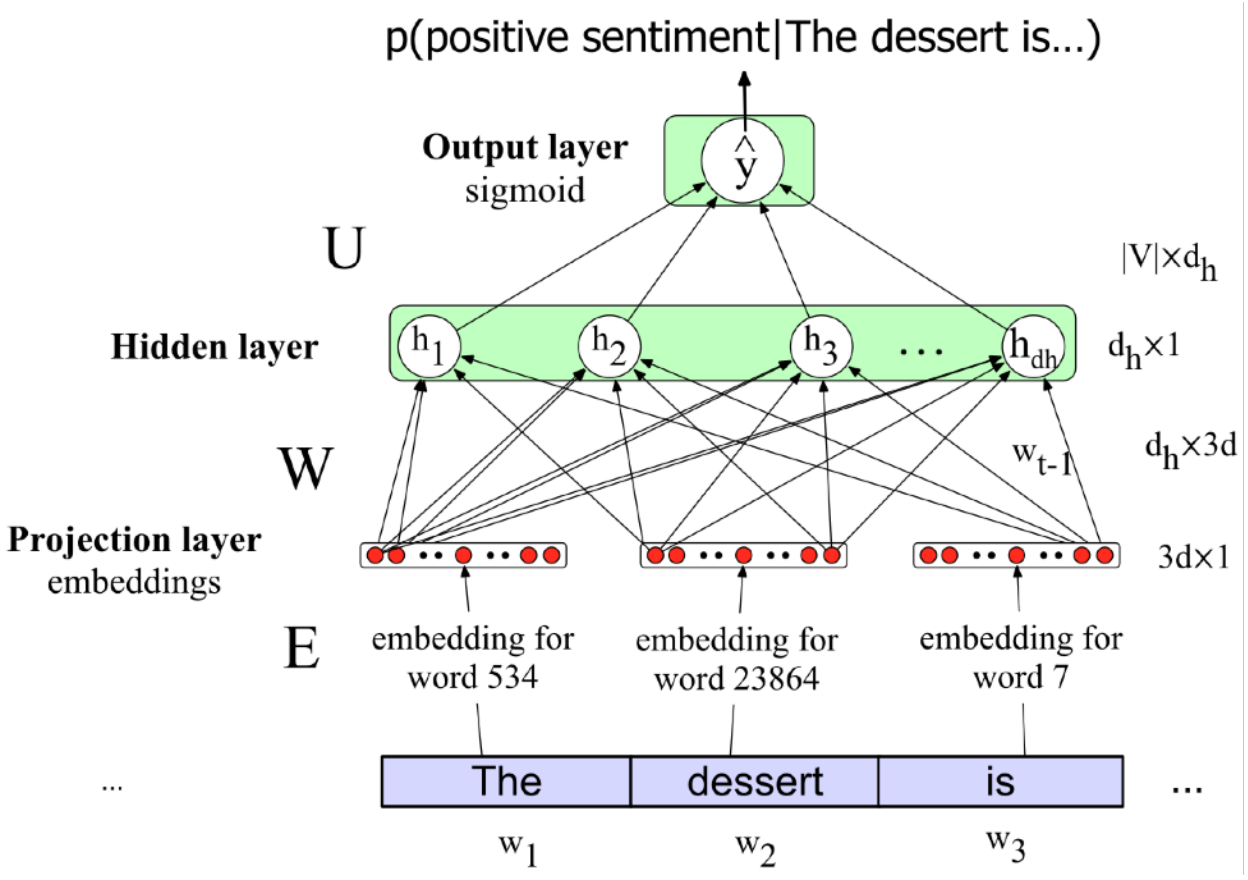
Le véritable pouvoir de l'apprentissage profond vient de la capacité d'apprendre les caractéristiques des données.

Au lieu d'utiliser des caractéristiques pré-déterminé par un humain pour la classification,

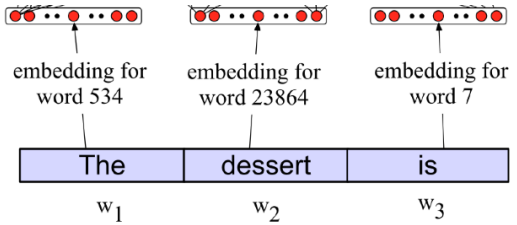
utilisez des représentations apprises comme des embeddings !



Classification neuronale avec embeddings comme caractéristiques d'entrée !



Problème : les textes sont de tailles différentes



Suppose une longueur de taille fixe (3) !

Un peu irréaliste.

Quelques solutions simples (des solutions plus sophistiquées plus tard)

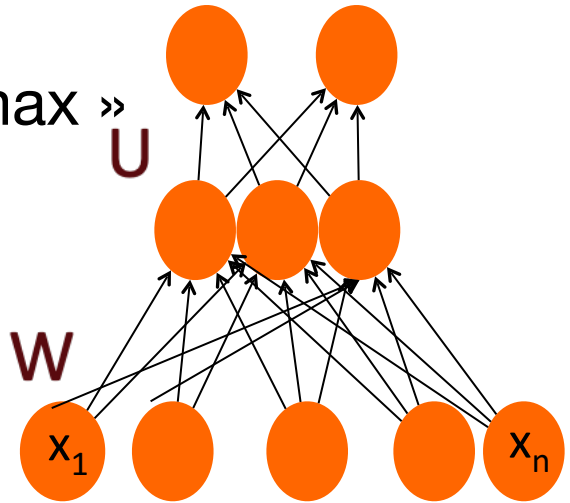
1. Faites de l'entrée la longueur du texte le plus long
   - Si c'est plus court, alors pad avec embeddings de zero
   - Tronquez si vous obtenez des textes plus longs au moment du test
2. Créez une seule “embedding de phrase" (la même dimensionnalité qu'un mot) pour représenter tous les mots
   - Prendre la moyenne de toutes les embeddings de mots
   - Prendre l’élément maximum sur chaque dimention de toutes les embedding de mots

Rappel : Sorties multiclasses

Que se passe-t-il si vous avez plus de deux classes de sortie ?

- Ajoutez plus d'unités de sortie (une pour chaque classe)
- Et utilisez une « couche softmax »

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} \quad 1 \leq i \leq D$$



2. Modélisation du language avec FNN

Modélisation du langage : Calculer de la probabilité du mot suivant dans une séquence donnée conditionné sur un certain contexte.

- Nous avons vu des LM basés sur les N-grams
- Mais les LM neuronaux surpassent de loin les LM n-gram

Les LM neuronaux de pointe sont basés sur des technologies de réseau neuronal plus puissantes comme les Transformers

Mais de simples FNN peuvent presque aussi bien faire !

Modèles de langage simple de FNN

Tâche : prédire le prochain mot w_t

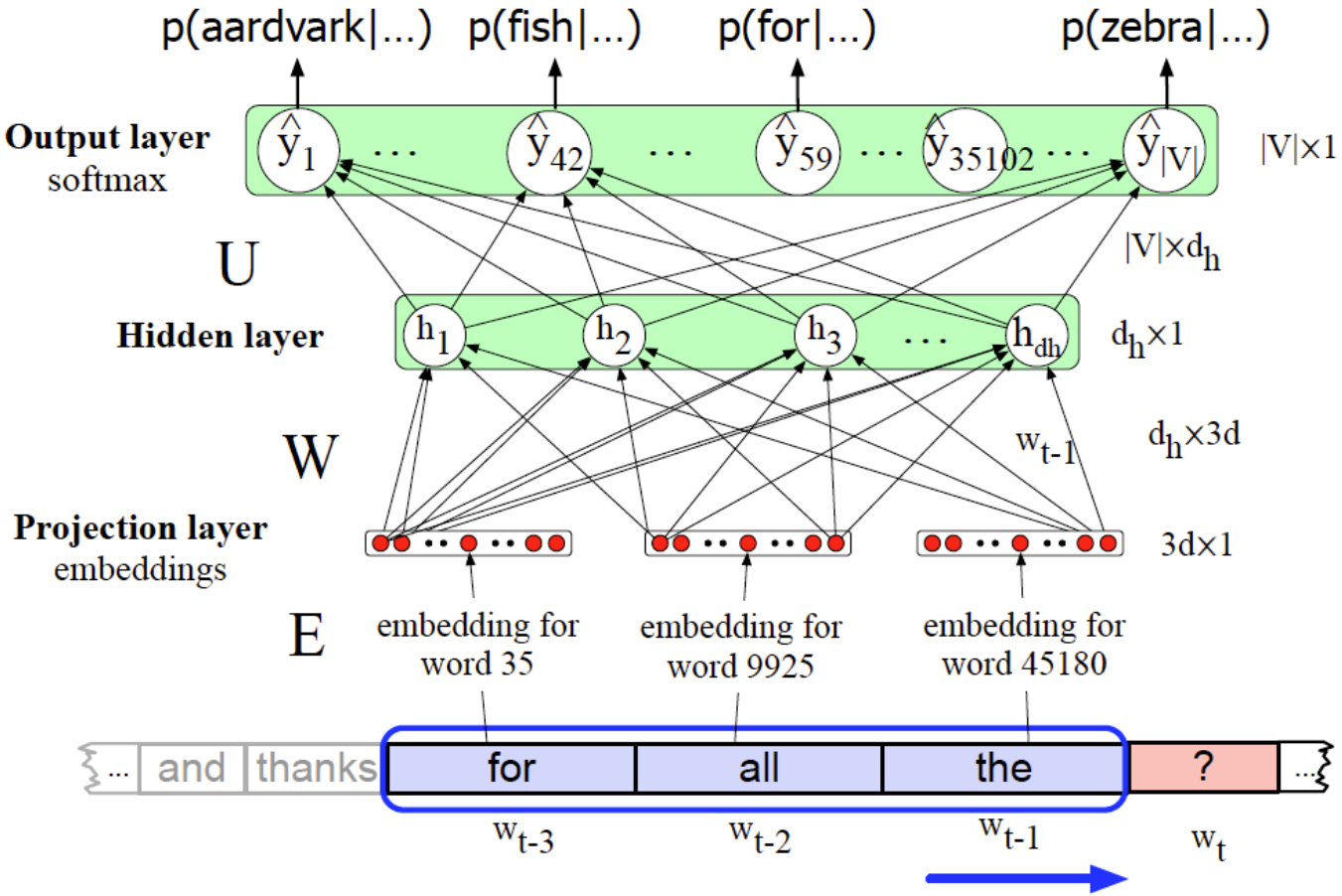
Donné les mots précédents w_{t-1}, w_{t-2}, w_{t-3}, **…**

Problème : Maintenant, nous avons affaire à des séquences de longueur arbitraire.

Solution: Fenêtres coulissantes (de longueur fixe)

$$P(w_t|w_1^{t-1}) \approx P(w_t|w_{t-N+1}^{t-1})$$

Modèles de langage simple de FNN



Pourquoi les LM neuronaux fonctionnent mieux que les LM N-gram

Données de formation :

Nous avons vu : I have to make sure that the cat gets fed.

Jamais vu : dog gets fed

Données de test :

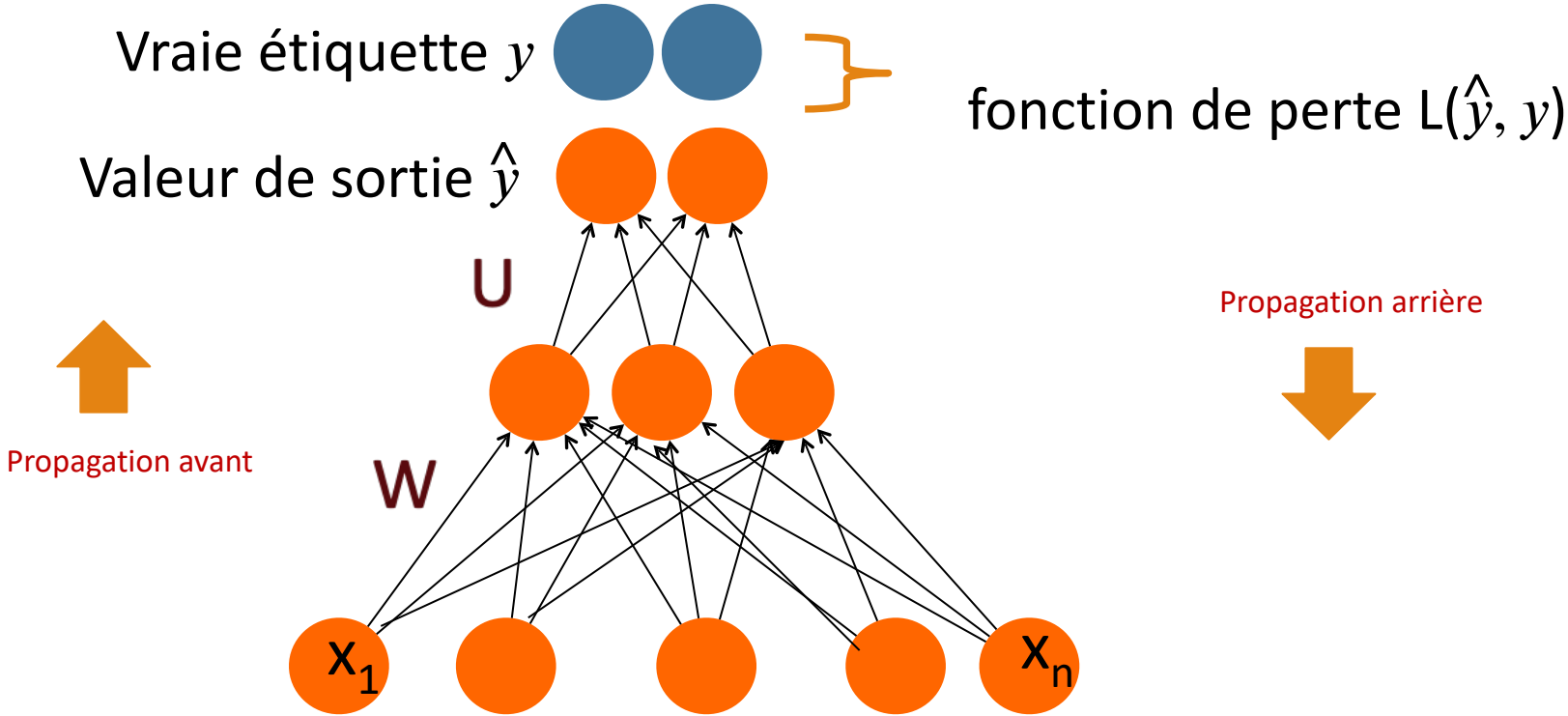
I forgot to make sure that the dog gets ___

Modèle n-gram ne peut pas prédire "fed"!

LM neuronal peut utiliser la similitude des embeddings de "cat" et "dog" pour generaliser et predire “fed” après dog

Super, mais comment entraîner ou adapter un modèle de réseau neuronal aux données ?

Intuition : entraîner un réseau à 2 couches



Instance d'entraînement

Intuition : entraîner un réseau à 2 couches

Pour chaque tuple d'entraînement (x, y)
- Exécutez le calcul *avant* pour trouver l'estimation $\hat{y}$
- Exécutez le calcul *arrière* pour mettre à jour les poids :
  - Pour chaque nœud de sortie
    - Calculer la perte L entre le vrai y et la valeur estimée $\hat{y}$
    - Pour chaque poid w de la couche cachée à la couche de sortie, mettre à jour les poids
  - Pour chaque nœud caché
    - Évaluez le montant du blâme qu'il mérite pour la réponse actuelle
    - Pour chaque poid w de la couche d'entrée à la couche cachée, mettre à jour les poids

Fonction de perte pour la régression logistique binaire

Une mesure de l'écart entre la valeur prédite et l'étiquette

Erreur entropie croisée (Cross entropy loss) pour la régression logistique:

$$L_{\text{CE}}(\hat{y}, y) = -\log p(y|x) = -[y \log \hat{y} + (1-y)\log(1-\hat{y})]$$

$$= -[y \log \sigma(w \cdot x + b) + (1-y)\log(1-\sigma(w \cdot x + b))]$$

Descente de gradient pour les mises à jour de poids

Utilisez la dérivée de la fonction de perte relative aux poids $\frac{d}{dw} L(f(x; w), y)$

Pour nous dire comment ajuster les poids pour chaque élément d'entraînement

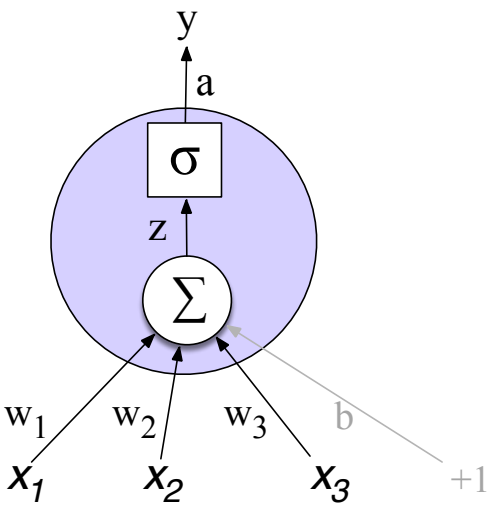
- Déplacez-les dans la direction opposée du gradient

$$w^{t+1} = w^t - \eta \frac{d}{dw} L(f(x; w), y)$$

D'où vient ce dérivé ?

la règle de dérivation en chaîne

$$f(x) = u(v(x)) \qquad \frac{df}{dx} = \frac{du}{dv} \cdot \frac{dv}{dx}$$



Dérivé de la somme pondérée

Dérivé de la fonction d'activation

Dérivé de la perte

$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial z} \frac{\partial z}{\partial w_i}$$

Comment trouver ce gradient pour chaque poids du réseau ?

Les dérivés sur la diapositive précédente ne donnent que les mises à jour pour une couche de poids : la dernière !

Qu'en est-il du reste du réseau?

- Plusieurs couches, différentes fonctions d'activation?

Solution:

- Graphe de calcul et la différenciation vers l'arrière !

Pourquoi les graphes de calculs

Pour l'entraînement, nous avons besoin de la dérivée de la perte par rapport à chaque poids dans chaque couche du réseau

- Mais la perte n'est calculée qu'à la toute fin du réseau !

Solution: **Rétropropagation des erreurs** (Rumelhart, Hinton, Williams, 1986)

- **Backprop** est un cas particulier de différenciation vers l'arrière qui repose sur des graphes de calculs.

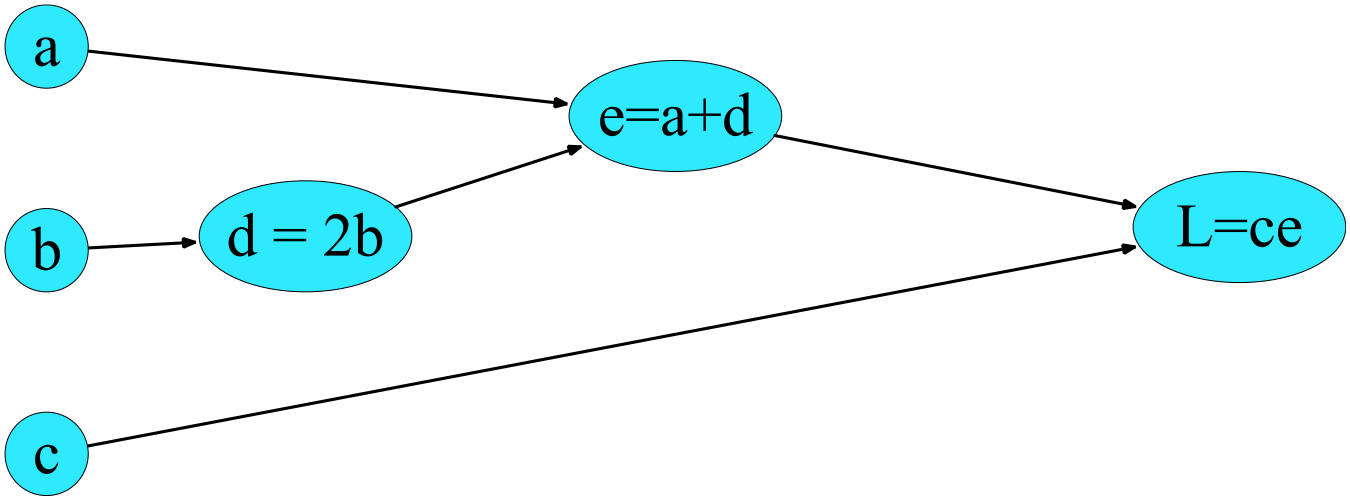
Graphes de calculs

Un graphe de calcul représente le processus de calcul d'une expression mathématique

Exemple: $L(a,b,c) = c(a+2b)$

Calculs:

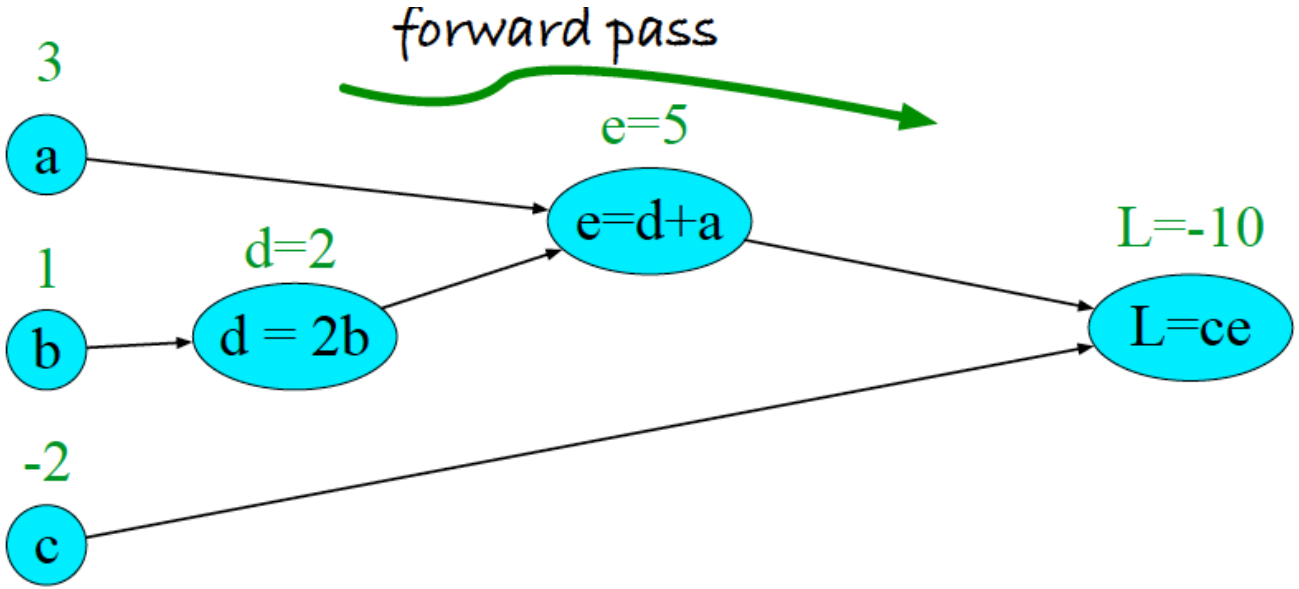
$$d = 2 * b$$
$$e = a + d$$
$$L = c * e$$



Exemple: $L(a,b,c) = c(a+2b)$

Calculs:

$$d = 2*b$$
$$e = a+d$$
$$L = c*e$$



Différenciation arrière dans les graphes de calculs

L'importance du graphe de calcul provient de la passe arrière

Celui-ci est utilisé pour calculer les dérivés dont nous aurons besoin pour la mise à jour du poids.

Exemple $L(a,b,c) = c(a+2b)$

$$
\begin{aligned}
d &= 2*b \\
e &= a+d \\
L &= c*e
\end{aligned}
$$

Ce que nous voulons: $\frac{\partial L}{\partial a}$, $\frac{\partial L}{\partial b}$, and $\frac{\partial L}{\partial c}$

Le derivé $\frac{\partial L}{\partial a}$, Nous dit combien un petit changement dans *a* affect *L*.

Règle de dérivation en chaîne

Calcul de la dérivée d'une fonction composée:

$$f(x) = u(v(x)) \qquad \frac{df}{dx} = \frac{du}{dv} \cdot \frac{dv}{dx}$$

$$f(x) = u(v(w(x))) \qquad \frac{df}{dx} = \frac{du}{dv} \cdot \frac{dv}{dw} \cdot \frac{dw}{dx}$$

Exemple $L(a,b,c) = c(a+2b)$

$$
\begin{aligned}
d &= 2*b \\
e &= a+d \\
L &= c*e
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial L}{\partial c} &= e \\
\frac{\partial L}{\partial a} &= \frac{\partial L}{\partial e}\frac{\partial e}{\partial a} \\
\frac{\partial L}{\partial b} &= \frac{\partial L}{\partial e}\frac{\partial e}{\partial d}\frac{\partial d}{\partial b}
\end{aligned}
$$

Exemple $L(a,b,c) = c(a+2b)$

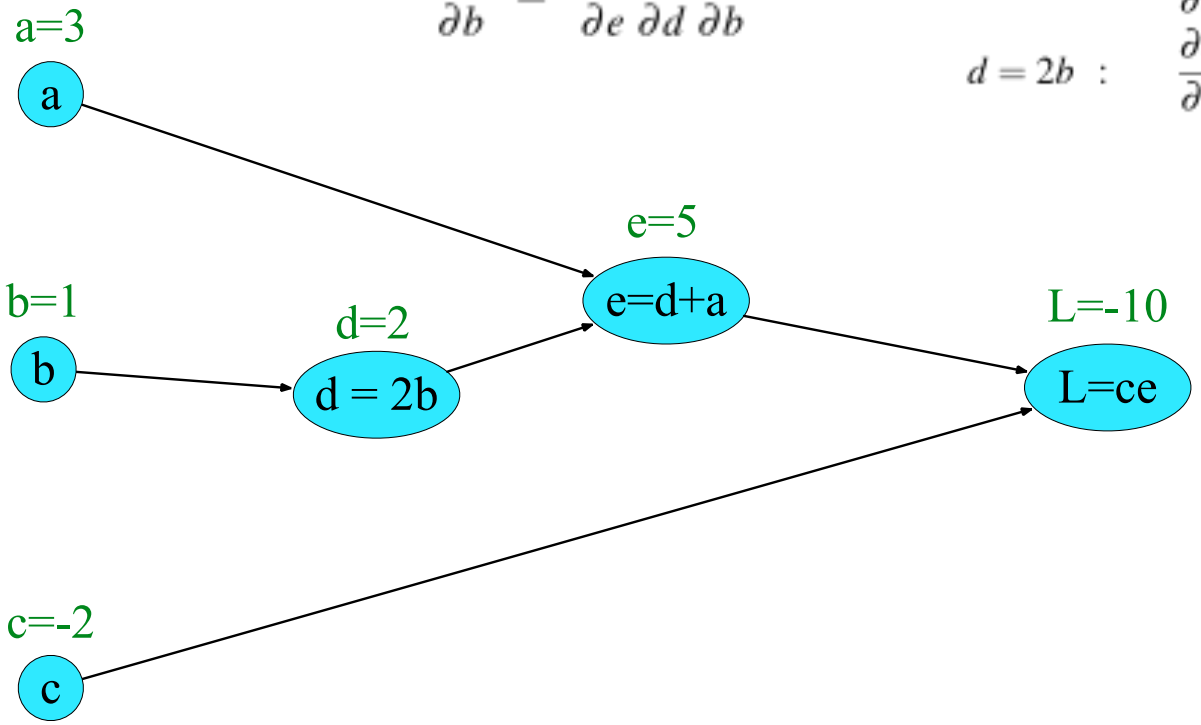
$$
\begin{aligned}
d &= 2*b \\
e &= a+d \\
L &= c*e
\end{aligned}
\qquad
\begin{aligned}
\frac{\partial L}{\partial a} &= \frac{\partial L}{\partial e}\frac{\partial e}{\partial a} \\
\frac{\partial L}{\partial b} &= \frac{\partial L}{\partial e}\frac{\partial e}{\partial d}\frac{\partial d}{\partial b}
\end{aligned}
$$

$$
\begin{aligned}
L = ce &: \quad \frac{\partial L}{\partial e} = c, \frac{\partial L}{\partial c} = e \\
e = a+d &: \quad \frac{\partial e}{\partial a} = 1, \frac{\partial e}{\partial d} = 1 \\
d = 2b &: \quad \frac{\partial d}{\partial b} = 2
\end{aligned}
$$

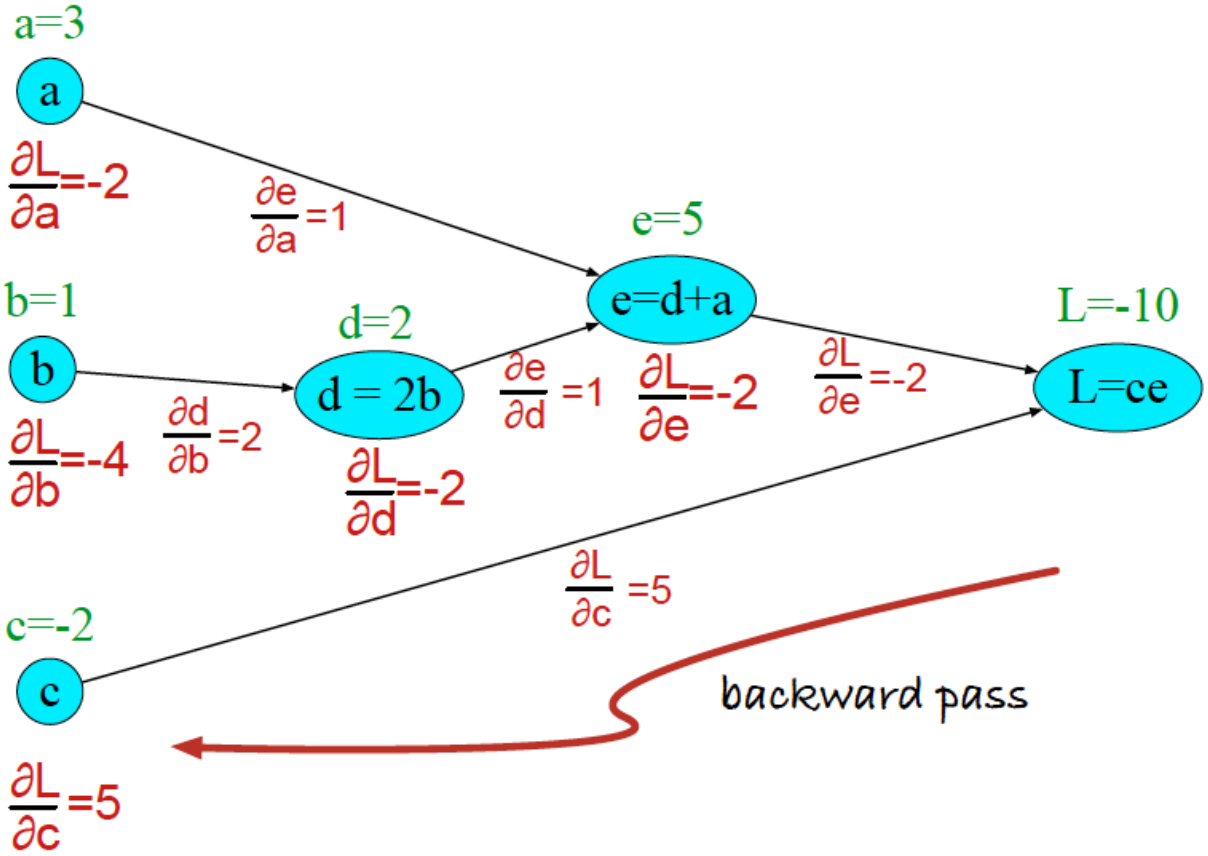
Exemple

$$\frac{\partial L}{\partial a} = \frac{\partial L}{\partial e}\frac{\partial e}{\partial a}$$
$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial e}\frac{\partial e}{\partial d}\frac{\partial d}{\partial b}$$

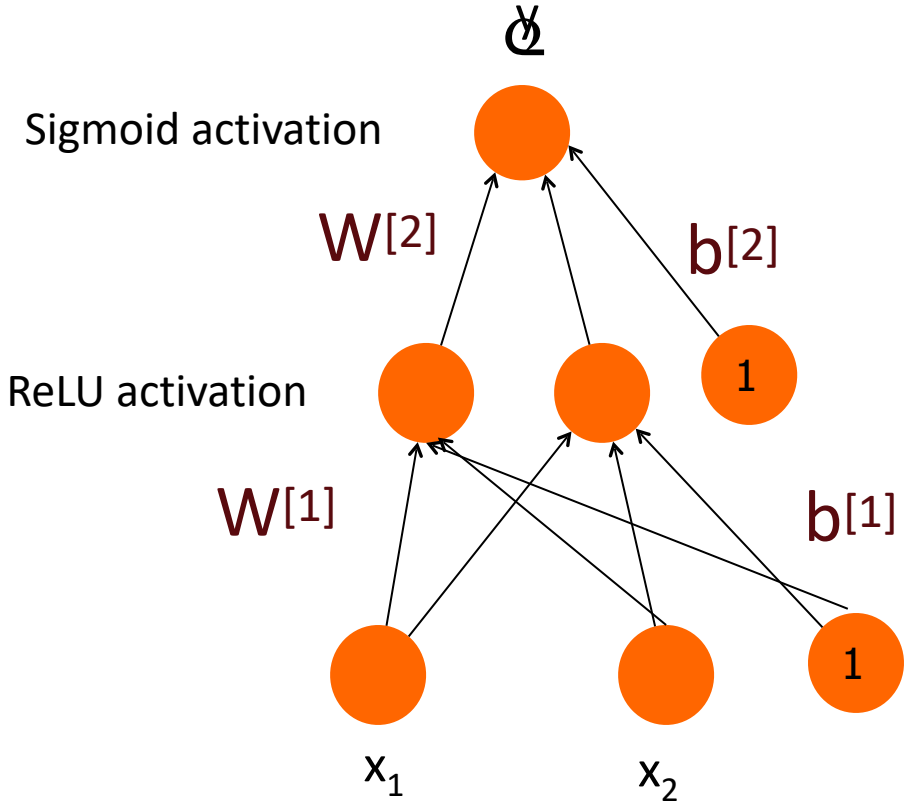
$$L = ce : \quad \frac{\partial L}{\partial e} = c, \frac{\partial L}{\partial c} = e$$
$$e = a + d : \quad \frac{\partial e}{\partial a} = 1, \frac{\partial e}{\partial d} = 1$$
$$d = 2b : \quad \frac{\partial d}{\partial b} = 2$$



Exemple

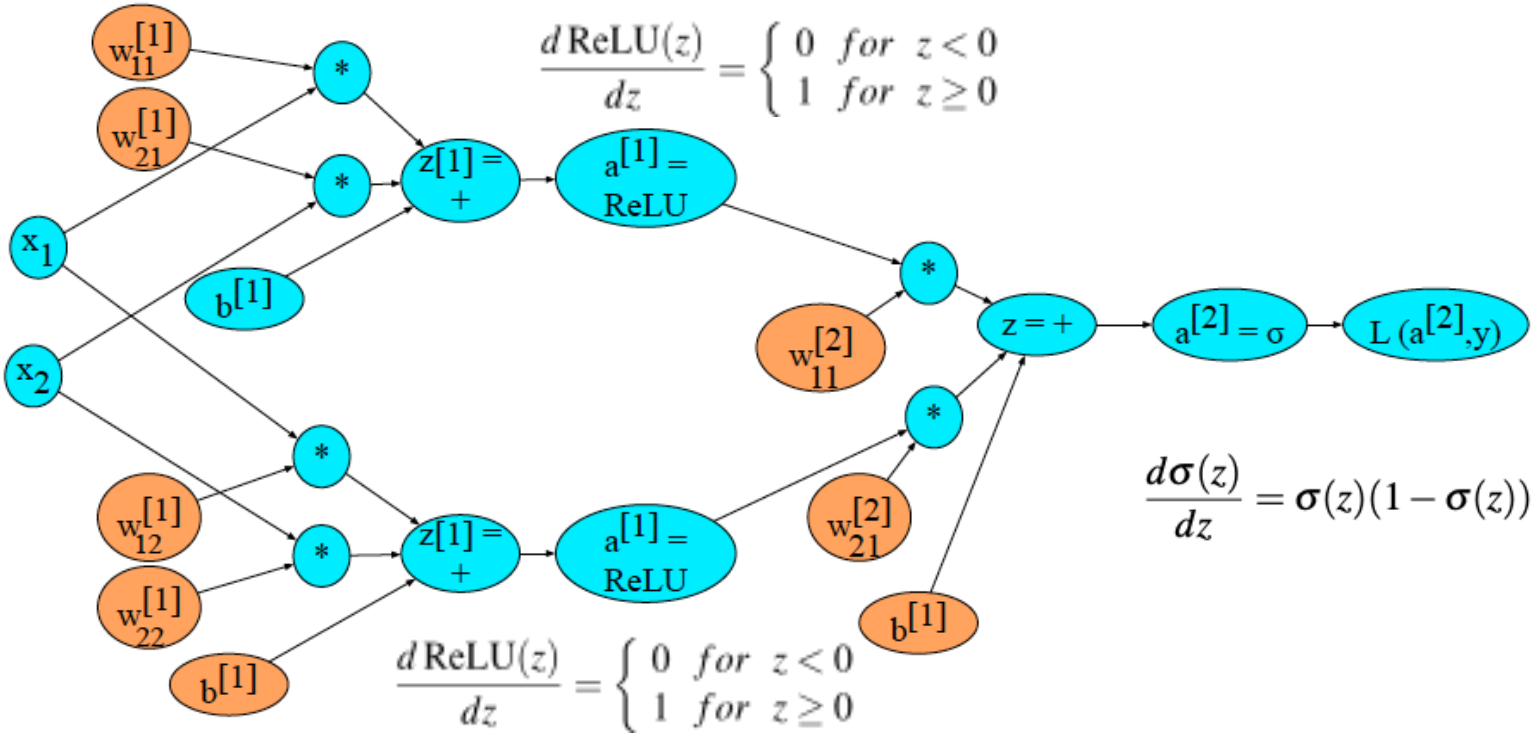

a=3
a
∂L/∂a=-2
∂e/∂a=1
e=5
e=d+a
b=1
b
∂L/∂b=-4
∂d/∂b=2
d=2
d = 2b
∂L/∂d=-2
∂e/∂d=1
∂L/∂e=-2
∂L/∂e=-2
L=-10
L=ce
∂L/∂c=5
c=-2
c
backward pass
∂L/∂c=5

Différenciation arrière sur un réseau à deux couches



$$
\begin{aligned}
z^{[1]} &= W^{[1]}\mathbf{x} + b^{[1]} \\
a^{[1]} &= \mathrm{ReLU}(z^{[1]}) \\
z^{[2]} &= W^{[2]}a^{[1]} + b^{[2]} \\
a^{[2]} &= \sigma(z^{[2]}) \\
\hat{y} &= a^{[2]}
\end{aligned}
$$

Différenciation arrière sur un réseau à deux couches



Résumé

Pour l'entraînement, nous avons besoin de la dérivée de la perte par rapport aux poids dans toutes les couches du réseau

- Mais la perte n'est calculée qu'à la toute fin du réseau !

Solution: **Différenciation vers l'arrière**

Étant donné un graphe de calcul et les dérivées de toutes les fonctions qu'il confère, nous pouvons automatiquement calculer la dérivée de la perte par rapport à chaque poid.

Rappel : Descente de gradient pour les mises à jour de poids

Utilisez la dérivée de la fonction de perte par rapport aux poids $\frac{d}{dw} L(f(x; w), y)$

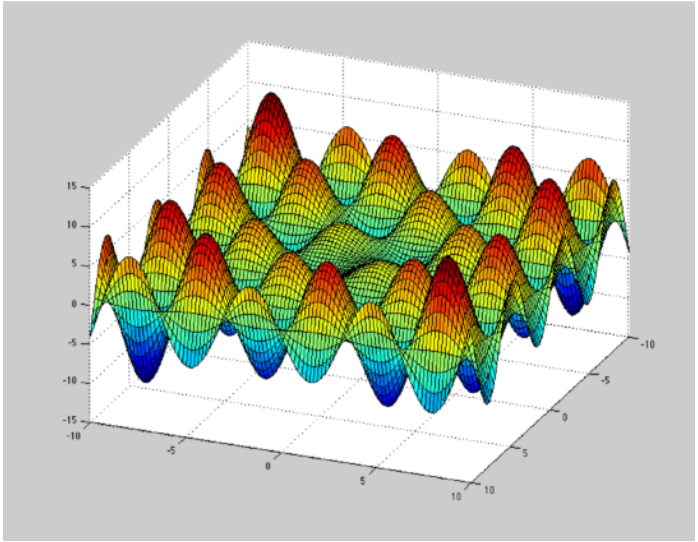
Pour nous dire comment ajuster les poids pour chaque élément d'entraînement

- Déplacez les poids dans la direction opposée du gradient

$$w^{t+1} = w^t - \eta \frac{d}{dw} L(f(x; w), y)$$

Optimisation

- L'optimisation de NN est un problème non convexe (c'est-à-dire qu'il existe de nombreuses optimisations locales)
- Alors, comment pouvons-nous essayer de trouver le meilleur et d'éviter le surajustement (overfitting)?



Hyperparamètres

Les poids (W, b) sont les paramètres du modèle qui seront mis à jour pendant l'entraînement; les hyperparamètres sont ceux que nous définissons dès le début et qui ne changent pas mais affectent le graphe de calcul.

Hyperparamètres

Voici quelques hyperparamètres qui affecteront l'optimisation du modèle :

- pas du gradient (Learning rate)
- Nombre d’epochs
- Taille de minibatch
- Nombre de couches
- Taille de des couches cachées (Hidden layer sizes)
- …

[pause - 15 minutes]

Essayons des modèles NN !

Mettez-vous en équipe!

Ouvrez les exercices/week 5 dans votre dossier de cours et commencez à écrire/exécuter du code !