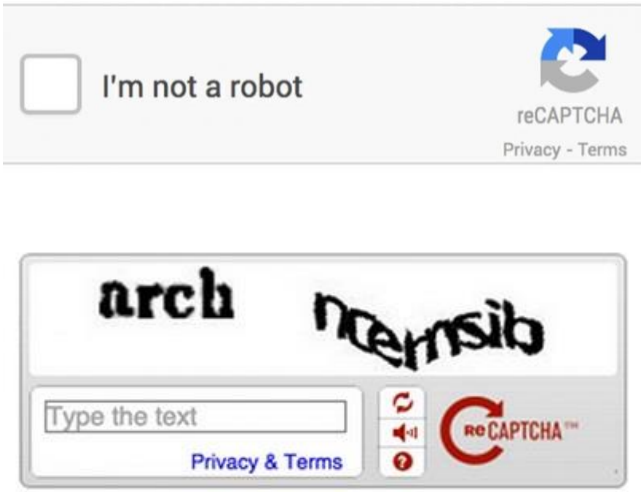
Réseaux de neurones et reconnaissance d'images

Alex Phimanesone

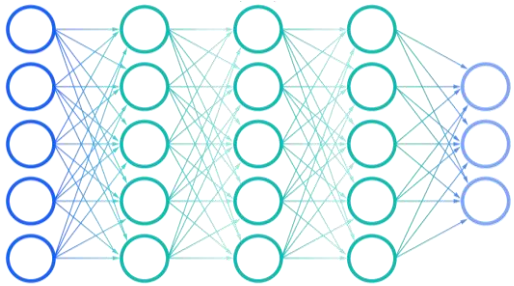
38750

Introduction

Tests captcha



Réseaux de neurones



Reconnaissance faciale
Prévisions météorologiques
Reconnaissance de caractères

Introduction

- Questions de sécurité et de confidentialité

- Comprendre le fonctionnement
- Evaluer les performances
- Mesurer l'ampleur des capacités

- Comment un réseau de neurones peut-il reconnaître un caractère alphanumérique?

Sommaire

I) Fonctionnement des réseaux de neurones

II) Apprentissage supervisé: fondements théoriques

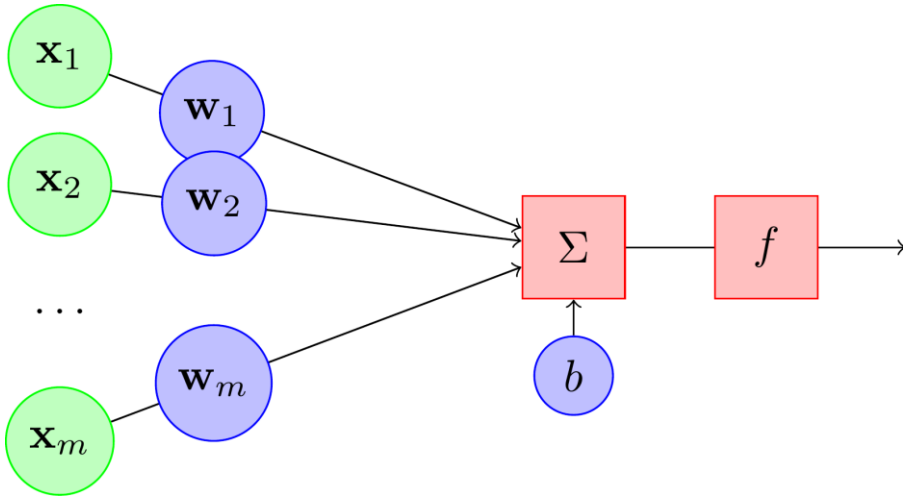
III) Mise en œuvre

IV) Démarche et résultats obtenus

Fonctionnement des réseaux de neurones

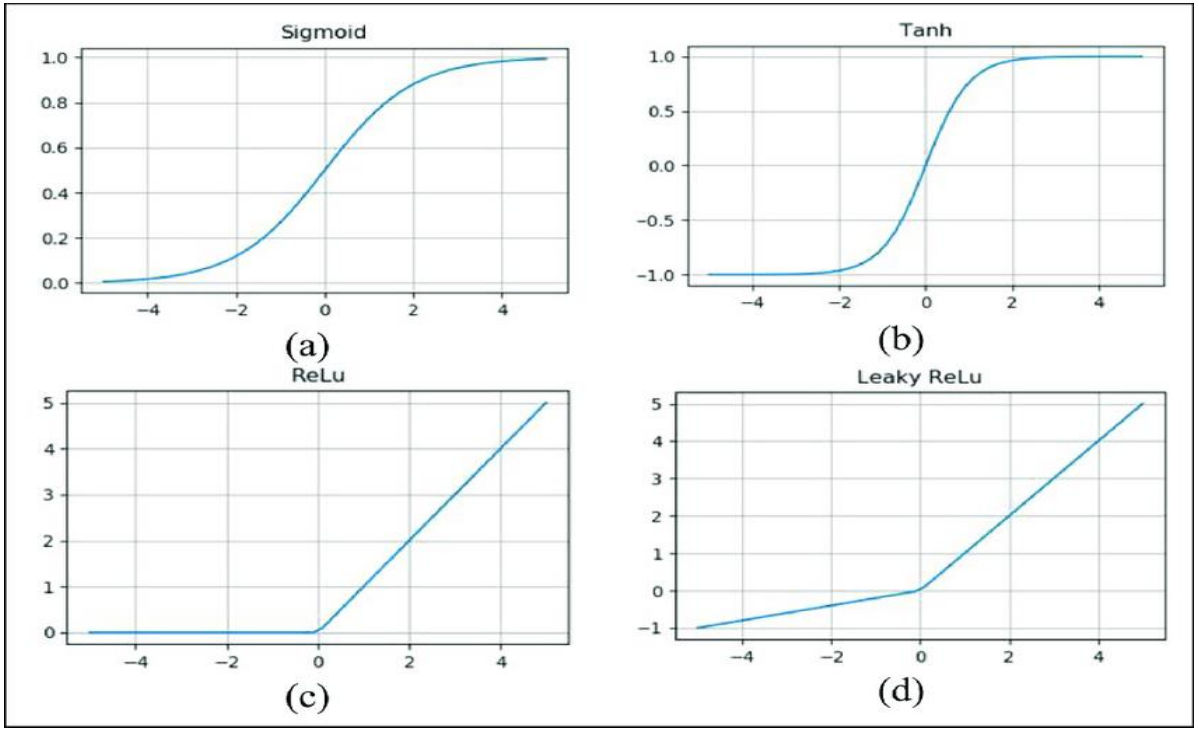
Neurone formel

- m entrées, 1 sortie
- m poids, un biais, une fonction d'activation



- Sortie du neurone: $a = f\left(\sum_{i=1}^{m} x_i w_i - b\right)$

Neurone formel



Sigmoïde :

$$y = \frac{1}{1+e^{-x}}$$

Tanh :

$$y = \tanh(x)$$

ReLu :

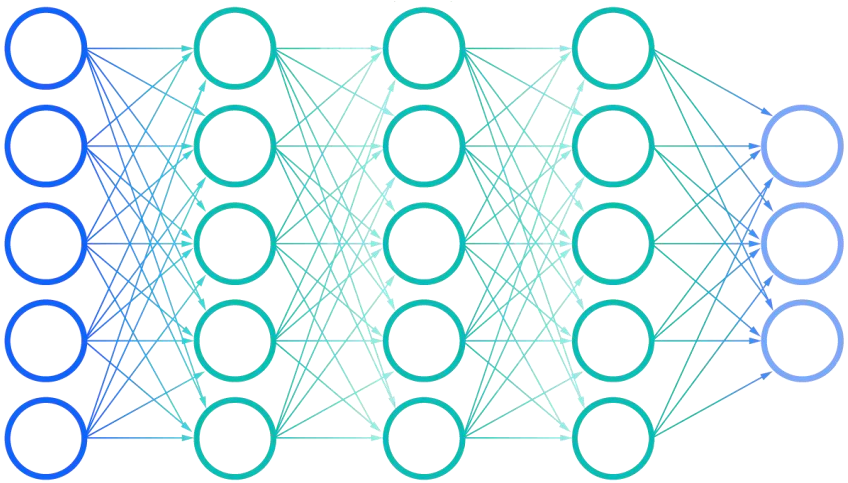
$$y = \max(0, x)$$

Leaky ReLu :

$$y = \max(\varepsilon x, x), \quad \varepsilon = 0.01$$

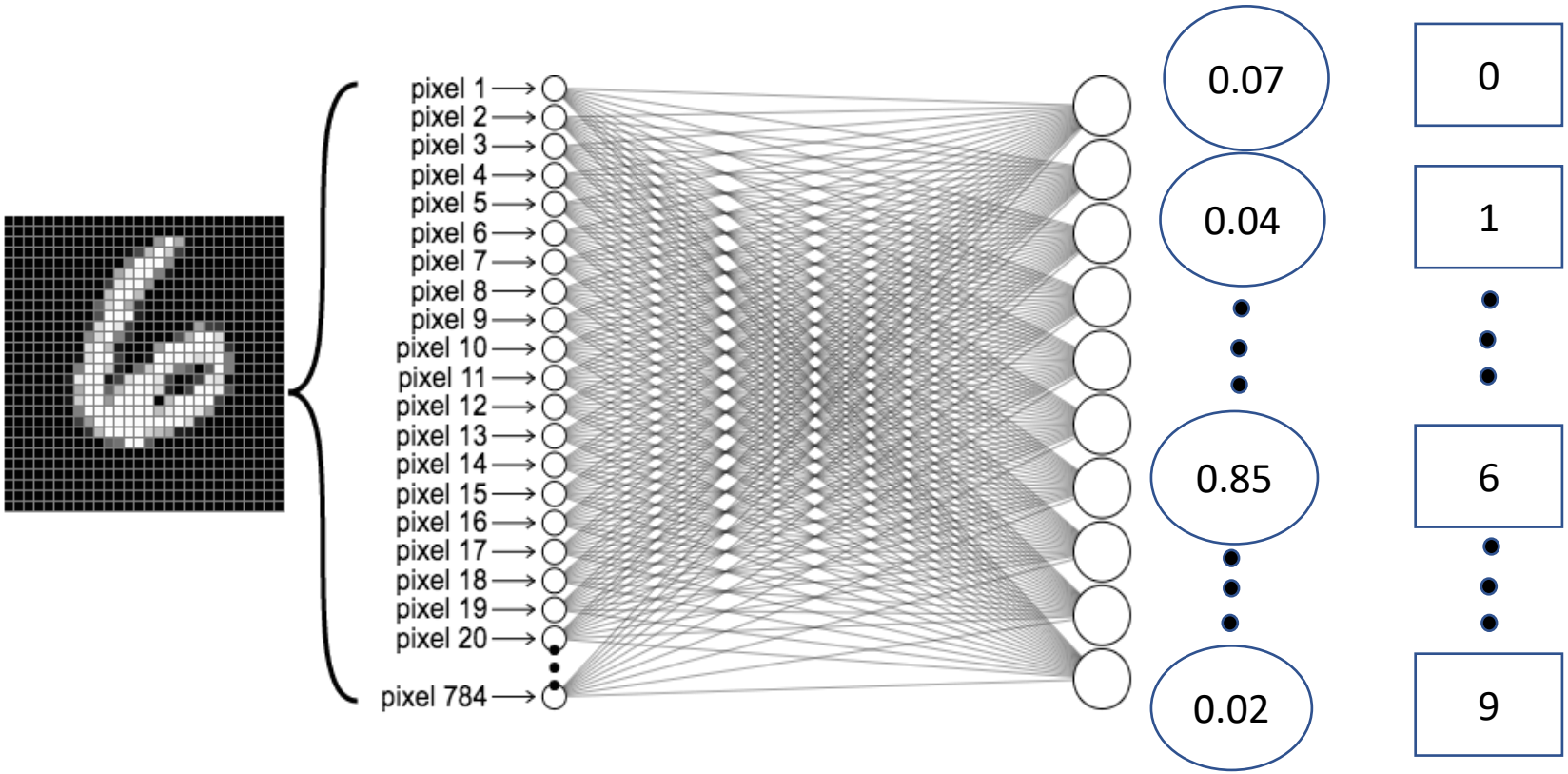
Réseau de neurones

- n entrées, p sorties



- Paramètres du réseau: poids, biais

Utilisation d’un réseau de neurones



Apprentissage supervisé: fondements théoriques

Phase d’entraînement

- Objectif: modifier les paramètres pour améliorer le réseau
- Donner des exemples au réseau

- Ensemble d’entraînement: banque d’exemples
- Epoch: itération sur tous les exemples de l’ensemble d’entraînement

- Définir un algorithme qui, à partir d’un exemple, améliore le réseau

Fonction d'erreur

- Rôle: modéliser l'erreur commise par le réseau

- Exemple: erreur quadratique

$$E(y, \hat{y}) = \sum_{i=1}^{p} (y_i - \hat{y}_i)^2$$

- But: modifier les paramètres du réseau pour minimiser la fonction d'erreur

Descente du gradient

- $f: \mathbb{R}^n \to \mathbb{R}$ différentiable, $a \in \mathbb{R}^n$
- $\exists \delta \in R_+^* : \forall \alpha \in]0, \delta[, f\big(a - \alpha \nabla f(a)\big) \leq f(a)$
- $a_{k+1} = a_k - \alpha \nabla f(a)$
- $Pour\ tout\ i\ \in \{1, \dots, n\}, x_{i,k+1} = x_{i,k} - \alpha \frac{\partial f}{\partial x_i}(a)$

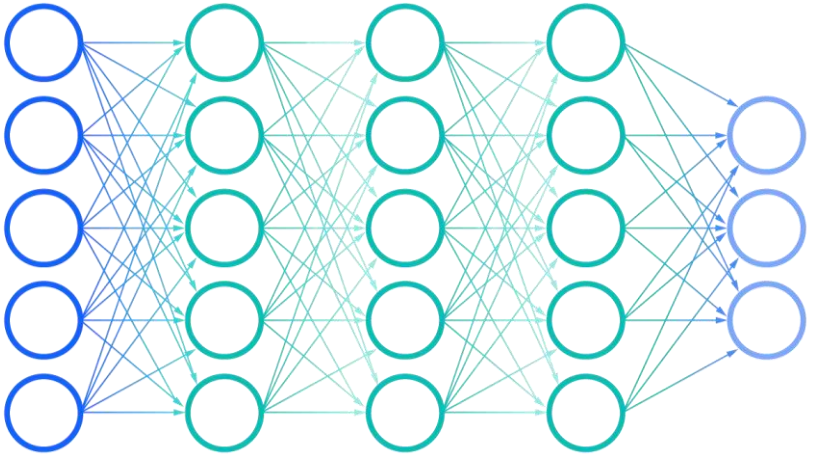
- f : fonction d'erreur E
- Les x_i : les poids et biais du réseau

Rétropropagation du gradient

- $w_{i,j} = w_{i,j} - \alpha \frac{\partial E}{\partial w_{i,j}}$, $b_j = b_j - \alpha \frac{\partial E}{\partial b_j}$
- $w_{\mathrm{i},j} = w_{i,j} + \alpha a_i \Delta[j]$, $b_j = b_j - \alpha \Delta[j]$
- Dernière couche : $\Delta[j] = -f'(p_j) \frac{\partial E}{\partial a_j}$
- Autres couches : $\Delta[j] = f'(p_j) \sum_k w_{j,k} \Delta[k]$

Rétropropagation du gradient

$$\Delta[j] = f'(p_j) \sum_k w_{j,k} \Delta[k]$$



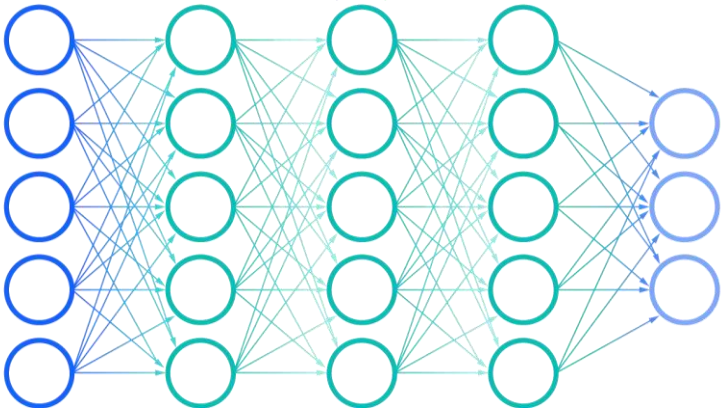
$$w_{\mathrm{i},j} = w_{i,j} + \alpha a_i \Delta[j] \; , \; b_j = b_j - \alpha \Delta[j]$$

Initialisation des paramètres

- Biais initialisés à 0
- Poids initialisés suivant une loi de probabilité

Vanishing gradient problem

$$\Delta[j] = f'(p_j) \sum_k w_{j,k} \Delta[k]$$



Initialisation des paramètres

- Normaliser les entrées du réseau:

$$Pour\ tout\ i\ \in\ \{1,\dots,784\}, x_i = \frac{x_i - x_{\text{moyenne}}}{x_{écart-type}}$$

- Méthodes d'initialisation:

Xavier	He
Sigmoïde, Tanh	Relu, Leaky Relu
$w^l \sim U\left(\left[-\sqrt{\frac{6}{n_{l-1}+n_l}}, \sqrt{\frac{6}{n_{l-1}+n_l}}\right]\right)$	$w^l \sim N\left(0, \frac{2}{n_{l-1}}\right)$

Mise en œuvre

Type neurone et réseau de neurones

- Neurone: m poids, un biais, une fonction d'activation

```
(*                  Types                  *)
type neurone = { poids : float array ; mutable b : float ; f : float -> float } ;;
```

- Réseau de neurones: tableau de tableau de neurones

Structure globale du code

1. Fonctions d’activation
2. Création de réseaux de neurones
3. Propagation avant
4. Fonction de rétropropagation et de modification
5. Entraînement
6. Evaluation

Démarche et résultats obtenus

Démarche

- Objectifs:

- Entraîner des réseaux à reconnaître un chiffre sur une image
- Etudier l'influence des hyper-paramètres et choisir le meilleur réseau

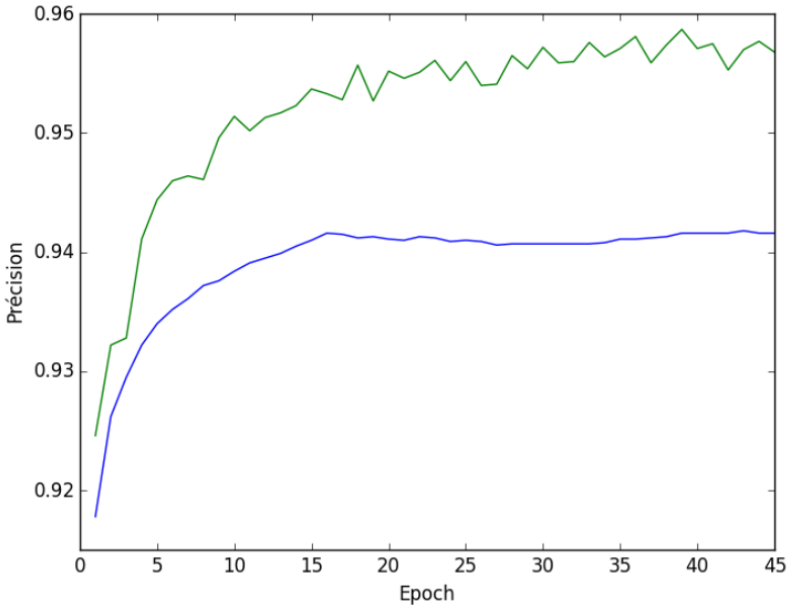
- Entraînement: 50,000 exemples, 45 epoch
- Base de données Emnist
- Evaluation: 10,000 images, $\frac{nombre\ de\ prédictions\ justes}{nombre\ de\ prédictions\ total}$

Ordre de présentation des exemples

- Ordre conservé: mêmes suites de modifications, inaptitude à généraliser

- Désordre
- Ordre conservé

Autres hyper-paramètres:
- Grande dimension
- Erreur quadratique
- Tanh

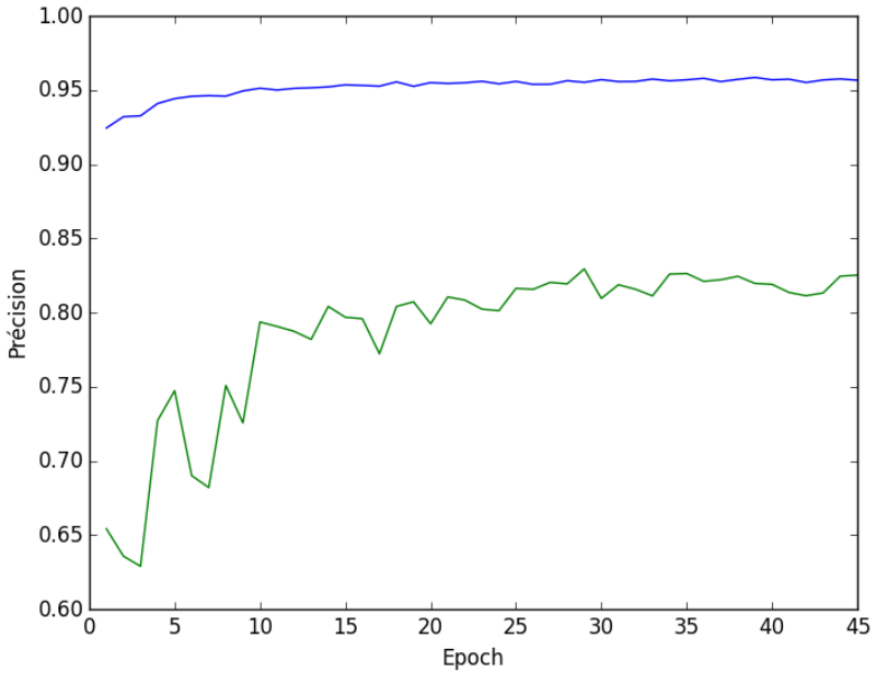


Dimension du réseau

- Plus grande dimension: meilleure capacité d'adaptation et de modélisation

- Grande dimension: [784, 200, 50, 10]
- Petite dimension [784, 100, 10]

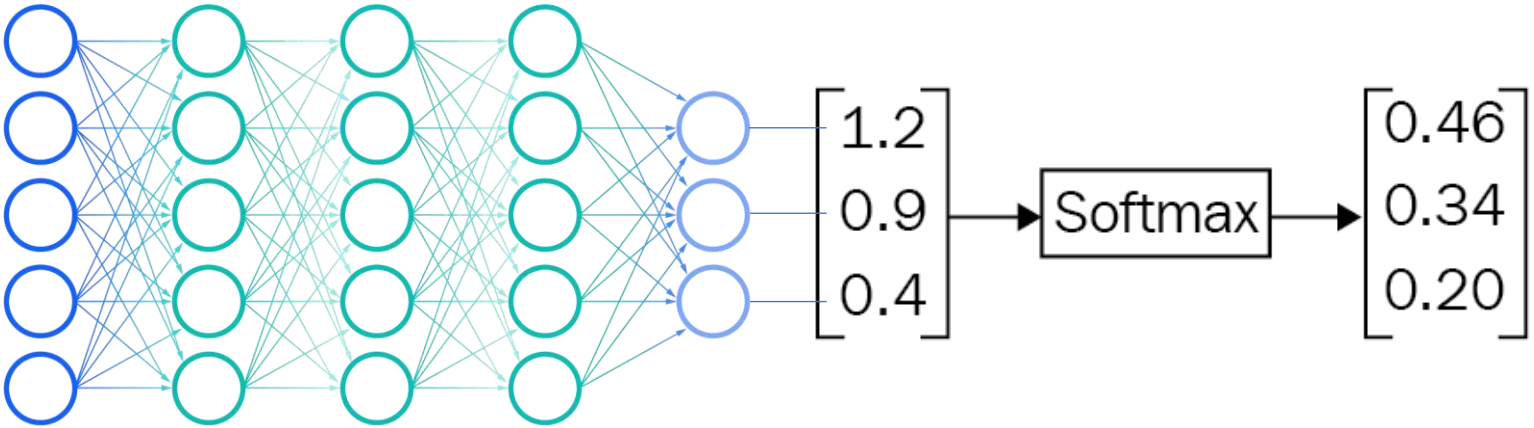
Autres hyper-paramètres:
- Désordre
- Erreur quadratique
- Tanh



Softmax et entropie croisée

$$\forall \begin{pmatrix} y_1 \\ . \\ . \\ y_p \end{pmatrix} \in \mathbb{R}^p, \forall i \in \{1, \dots, p\}, Softmax\left(\begin{pmatrix} y_1 \\ . \\ . \\ y_p \end{pmatrix}\right)_i = \frac{e^{y_i}}{\sum_{j=1}^{p} e^{y_j}}$$

$$E(y, \hat{y}) = -\sum_{i=1}^{p} \hat{y}_i \log(y_i)$$

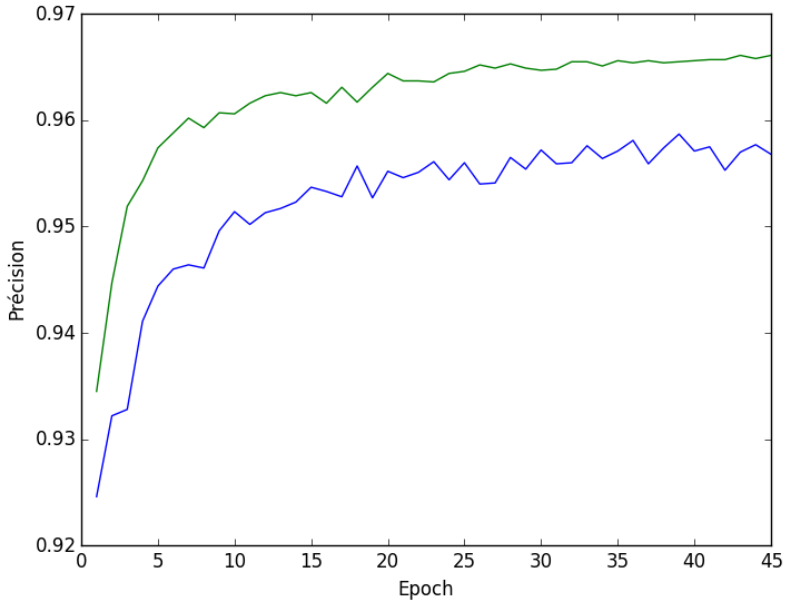


Softmax et entropie croisée

- Softmax et entropie croisée: plus adapté pour de la classification

- Avec softmax, entropie croisée
- Sans softmax, erreur quadratique

Autres hyper-paramètres:
- Désordre
- Grande dimension
- Tanh

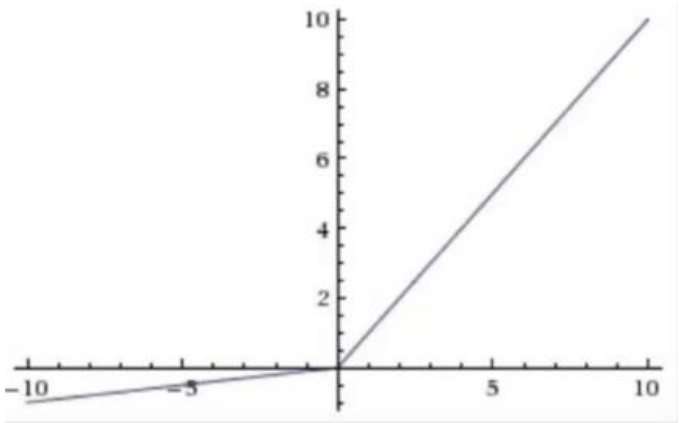


Fonction d'activation

- Comparer Tanh et Leaky ReLu

Leaky Relu:

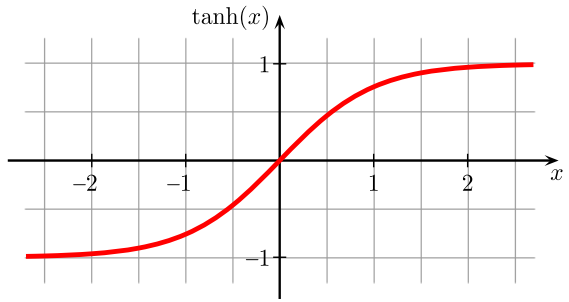
$$y = \max(\varepsilon x, x),$$
$$\varepsilon = 0.01$$



Problèmes de Tanh:

- Bornée
- Dérivée tend vers 0

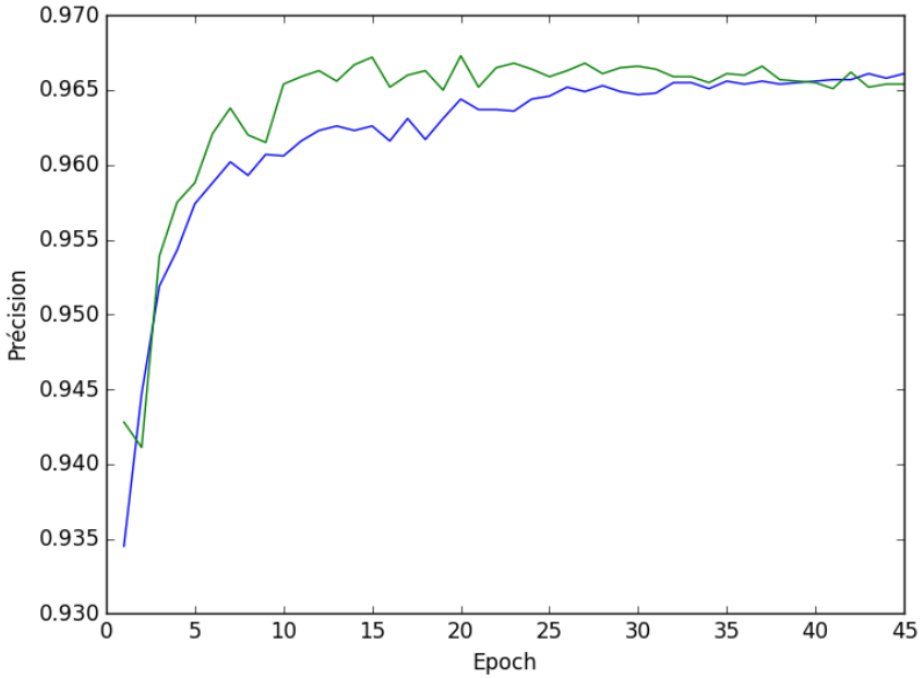
$$\Delta[j] = f'(p_j) \sum_k w_{j,k} \Delta[k]$$



Fonction d'activation

- Leaky Relu
- Tanh

Autres hyper-paramètres:
- Désordre
- Grande dimension
- Avec softmax et entropie croisée



Bilan

Meilleure précision	96.72%
Présentation des exemples	Désordre
Dimension	Grande dimension: [784, 200, 50, 10]
Sortie et fonction d'erreur	Softmax et entropie croisée
Fonction d'activation	Leaky Relu

Conclusion

Annexe

- Démonstration de la descente du gradient (p.32)
- Démonstration de la rétropropagation du gradient (p.33)
- Justification des méthodes de Xavier et de He (p.35)
- Influence des taux d’apprentissage (p.38)
- Non-linéarité des fonctions d’activation (p.39)
- Utilité des biais (p.40)
- Inspiration biologique (p.42)
- Programmes (p.44)

Descente du gradient: démonstration

Soit f une fonction de $\mathbb{R}^n$ dans $\mathbb{R}$ supposée différentiable sur $\mathbb{R}^n$ et a un point de $\mathbb{R}^n$. Supposons que $\nabla f(a) \neq 0$. Montrons que : $\exists \delta \in R_+^* : \forall \alpha \in]0, \delta[, f\big(a - \alpha \nabla f(a)\big) \leq f(a)$.

f est différentiable en a donc il existe une fonction ε qui tend vers 0 en a telle que, pour tout point h au voisinage de $0_{\mathbb{R}^n}$, $f(a+h) = f(a) + df(a) \cdot h + \|h\|\varepsilon(h)$. Ainsi, par définition du gradient, on a, pour tout point h au voisinage de $0_{\mathbb{R}^n}$,

$$f(a+h) - f(a) = \langle \nabla f(a) | h \rangle + \|h\|\varepsilon(h).$$

D'où : pour tout $\alpha \in R_+^*$ assez petit, on a :

$$f\big(a - \alpha \nabla f(a)\big) - f(a) = \alpha \|\nabla f(a)\| \left(-\|\nabla f(a)\| + \varepsilon(-\alpha \nabla f(a))\right)$$

Donc, comme $-\|\nabla f(a)\| < 0$, on a, pour $\alpha \in R_+^*$ assez petit, $f\big(a - \alpha \nabla f(a)\big) - f(a) \leq 0$. Finalement :

$$\exists \delta \in R_+^* : \forall \alpha \in]0, \delta[, f\big(a - \alpha \nabla f(a)\big) \leq f(a).$$

Rétropropagation du gradient: démonstration

On considère un réseau de neurones. On note $w_{i,j}$ le poids entre le neurone i et le neurone j, b_j le biais du neurone j, p_j la pré-activation du neurone j, a_j l'activation du neurone j et f la fonction d'activation. Calculons $\frac{\partial E}{\partial w_{i,j}}$ et $\frac{\partial E}{\partial b_j}$.

D'après la règle de la chaîne, on a : $\frac{\partial E}{\partial w_{i,j}} = \frac{\partial E}{\partial p_j}\frac{\partial p_j}{\partial w_{i,j}}$ d'où $\frac{\partial E}{\partial w_{i,j}} = \frac{\partial E}{\partial p_j} a_i$.

Calculons $\frac{\partial E}{\partial p_j}$.

On a : $\frac{\partial E}{\partial p_j} = \frac{\partial E}{\partial a_j}\frac{\partial a_j}{\partial p_j} = f'(p_j)\frac{\partial E}{\partial a_j}$. Si le neurone j appartient à la dernière couche du reseau, on ne peut simplifier cette expression davantage. Supposons désormais que ce neurone n'appartient pas à la dernière couche.

D'après la règle de la chaîne, on a : $\frac{\partial E}{\partial a_j} = \sum_k \frac{\partial E}{\partial p_k}\frac{\partial p_k}{\partial a_j}$ (où k itère sur tous les neurones de la couche suivante) ie $\frac{\partial E}{\partial a_j} = \sum_k w_{j,k}\frac{\partial E}{\partial p_k}$.

Rétropropagation du gradient: démonstration

Ainsi, en notant pour tout neurone k : $\Delta[k] = -\frac{\partial E}{\partial p_k}$, on a : $\Delta[j] = f'(p_j) \sum_k w_{j,k} \Delta[k]$.

Finalement:

On a : $\frac{\partial E}{\partial w_{i,j}} = -\, a_i\, \Delta[j]$ avec

- Si le neurone j appartient à la dernière couche: $\Delta[j] = -f'(p_j) \frac{\partial E}{\partial a_j}$
- Sinon, $\Delta[j] = f'(p_j) \sum_k w_{j,k} \Delta[k]$ (où k itère sur les neurones de la couche suivante)

On montre de la même manière que: $\frac{\partial E}{\partial b_j} = \Delta[j]$

Méthode de Xavier et de He: justification

Pour éviter le vanishing gradient problem, il faut respecter les critères (empiriques) suivants:

1. A chaque couche l, la moyenne des activations est nulle.
2. La variance des activations est la même à chaque couche.

Montrons que la méthode d'initialisation de He permet de respecter le deuxième critère.

Formulons les hypothèses suivantes : à chaque couche l,

- les poids des neurones, les activations et les pré-activations, sont iid
- les poids et les activations sont mutuellement indépendants
- les poids, ainsi que les pré-activations, sont de moyenne nulle et de loi symétrique par rapport à 0
- les biais sont initialisés à 0.

Supposons que la fonction d'activation est ReLu.

Méthode de Xavier et de He: justification

On considère une couche l. On a, pour tout neurone j dans cette couche, $p_j = \sum_{i=1}^{m} a_i w_{i,j}$. (où m est le nombre d'entrées du neurone j). Ainsi, on a:

$Var(p_j)$

$= \mathrm{m}\, Var(a_1 w_{1,j})$ (d'après les hypothèses d'indépendance et d'identique distribution)

$= m\,[Var(w_{1,j})Var(a_1) + E(w_{1,j})^2\, Var(a_1) + Var(w_{1,j})\, E(a_1)^2]$

$= m\, Var(w_{1,j})(Var(a_1) + E(a_1)^2)$ (par nullité de l'espérance des poids)

$= m\, Var(w_{1,j})\, E(a_1^2)$

De plus, $E(a_1^2) = \int_{-\infty}^{+\infty} {a_1}^2 P(a_1)\, \mathrm{d}a_1 = \int_{-\infty}^{+\infty} max(0, p_1)^2 P(p_1)\, \mathrm{d}p_1 = \frac{1}{2}\int_{-\infty}^{+\infty} {p_1}^2 P(p_1)\, \mathrm{d}p_1 = \frac{1}{2} Var(p_1)$

Les hypothèses d'identique distribution nous permettent d'omettre les indices désignant les neurones.

Méthode de Xavier et de He: justification

Ainsi, en notant n_{l-1} le nombre de neurones dans la couche $l-1$, on a la relation reliant les pré-activations de la couche $l-1$ et celles de la couche l suivante :

$$Var(p^l) = \frac{1}{2} n_{l-1} Var(w^l) Var(p^{l-1})$$

Ainsi, garder la variance constante de couche en couche impose:

$$Var(w^l) = \frac{2}{n_{l-1}}$$

Donc, pour chaque couche l, on choisit: $w^l \sim N\left(0, \frac{2}{n_{l-1}}\right)$

Il suffit d'adapter la démonstration pour prouver le résultat dans le cas où la fonction d'activation est Leaky Relu.

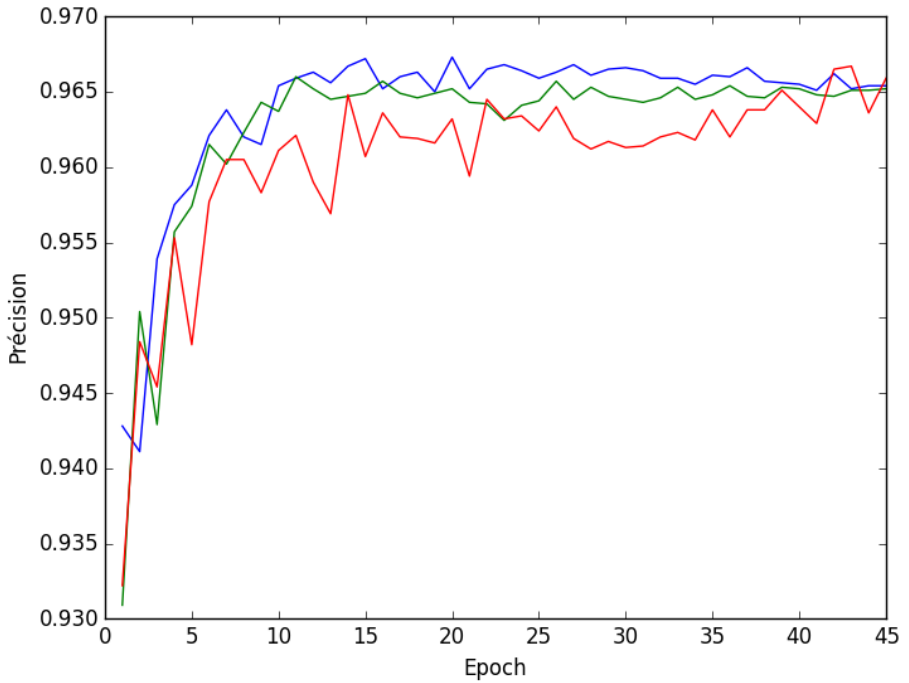
On justifie de la même manière la méthode d'initialisation de Xavier.

Influence des taux d'apprentissage

- Comparaison de différentes évolutions des taux d'apprentissage

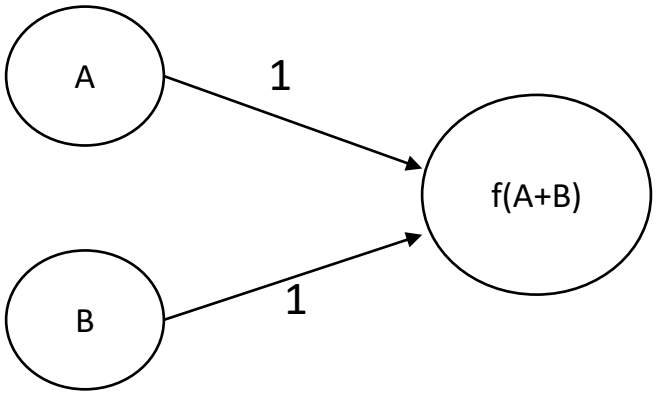
- $\frac{\eta}{1+\delta n}$, de 0.01 à 0.001
- $\frac{\eta}{n^{\delta}}$, 0.01 à 0.001
- Constant: 0.005

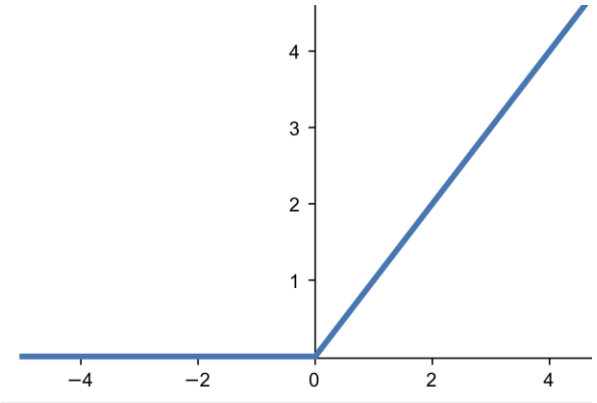
Autres hyper-paramètres:
- Désordre
- Grande dimension
- Avec softmax et entropie croisée
- Leaky Relu



Non-linéarité des fonctions d'activation

- Non-linéarité: capter les interactions entre les entrées

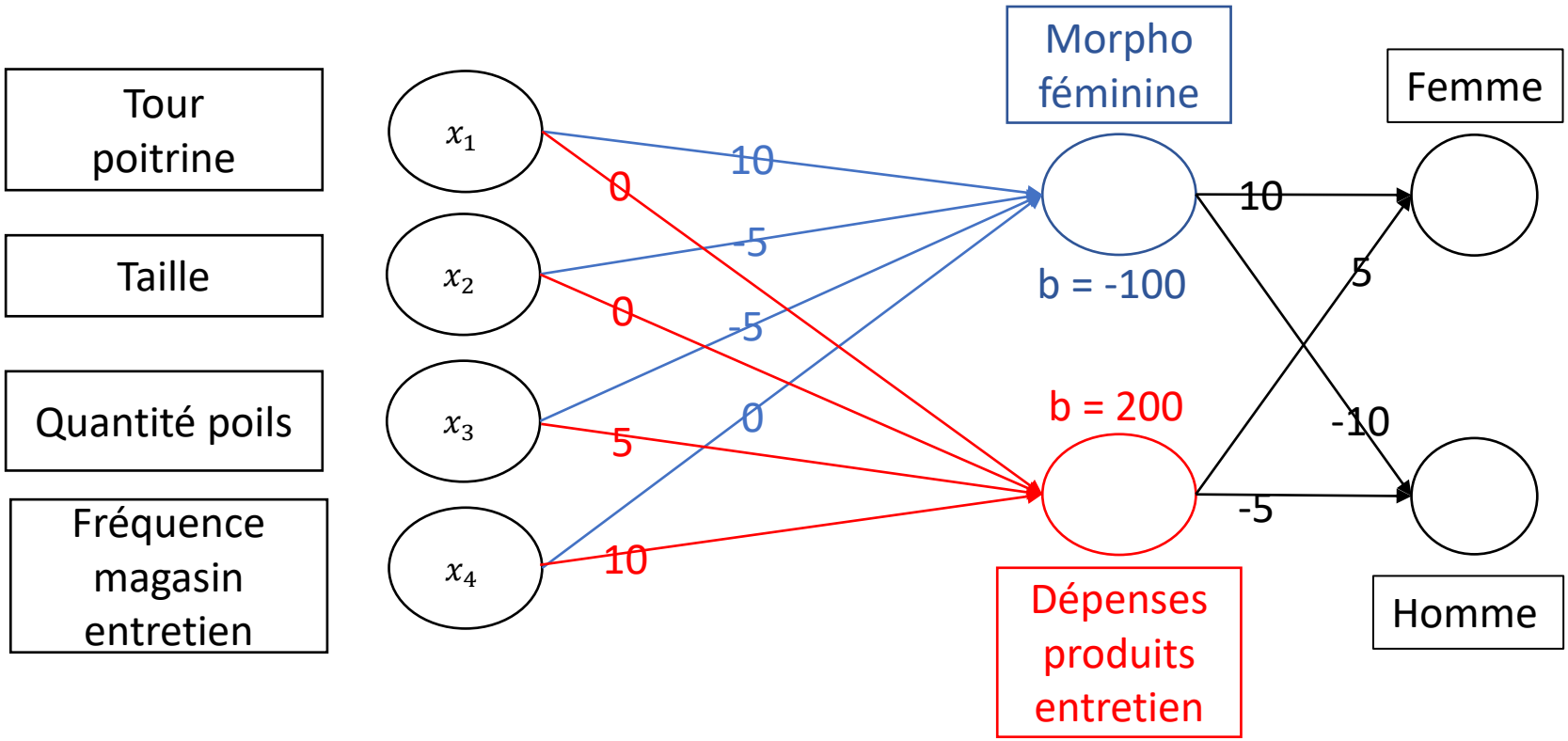




- Supposons que A varie entre -10 et 10

Si B vaut -100	Si B vaut 0
A n'influe pas sur l'activation	A influe sur l'activation

Utilité des biais



Utilité des biais

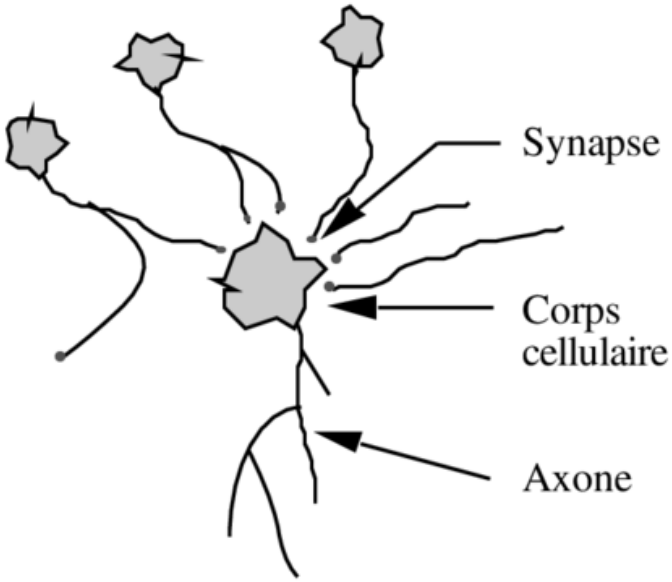
- Fonction d'activation: ReLu

- Le biais b = 200 rend la pré-activation très négative donc désactive le neurone rouge (activité nulle)
- Le biais b = -100 rend la pré-activation très positive donc augmente l'influence du neurone bleu (activité élevée)

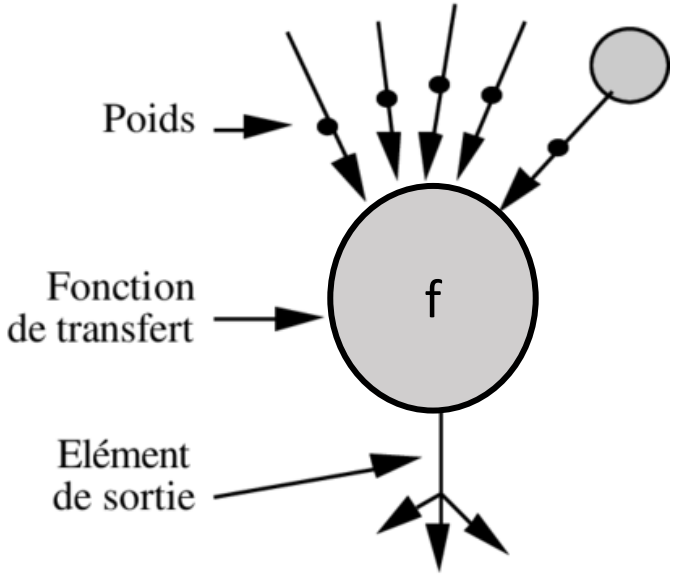
- biais: paramètres supplémentaires pour mieux modéliser la tâche à effectuer

Inspiration biologique

Biologique



Artificiel



Inspiration biologique

Biologique	Artificiel
Dentrites	Entrées
Synapses: neurotransmetteurs excitateurs, neurotransmetteurs inhibiteurs	Poids: Poids positifs, Poids négatifs
Sommation	Sommation
Présence d'une tension seuil réponse proportionnelle à l'entrée	ReLu
Unique sortie communiquée à plusieurs neurones	Unique sortie communiquée à plusieurs neurones

Programmes

```
(*            Imports            *)
open Random ;;



(*            Types            *)
type neurone = { poids : float array ; mutable b : float ; f : float -> float } ;;



(*            1. Calcul et aleatoire            *)


(*renvoie la somme composante par composante des carres des differences*)
let erreur_quadra sortie yd =
  let n = Array.length sortie and e = ref 0. in
    for i =0 to n-1
    do
      e := !e +. (sortie.(i) -. yd.(i)) *. (sortie.(i) -. yd.(i))
    done ;
    !e
;;
```

```
(*retourne l'indice correspondant a la valeur maximale d'un vecteur*)
let maximum v =
  let n = Array.length v in
  let valeur_max = ref v.(0) and res = ref 0 in
    for i = 1 to n-1
    do
      if v.(i) > !valeur_max then
        (valeur_max := v.(i);
         res := i)
    done;
    !res
;;


(*echange les elements dâindices i et j du vecteur v*)
let echange v i j =
  let temp = v.(i) in
    v.(i) <- v.(j) ;
    v.(j) <- temp
;;


(*renvoie la liste des elements de la liste l dans le desordre*)
let desordre l =
```

```
  let n = List.length l and t = Array.of_list l in
  let res = ref [] in
    for i = 0 to n-1
    do
      (*on se donne un indice r dans [[0, n-1-i]]*)
      let r = Random.int (n-i) in
        res := t.(r) :: !res;
        echange t r (n-1-i)
    done;
    !res
;;


let loi_normale_centree sigma =
  let rec loop () =
    let u = Random.float 1.0 and
    v = 1.7156 *. (Random.float 1.0 -. 0.5) in
    let x = u -. 0.449871 and
      y = abs_float v +. 0.386595 in
    let q = x*.x +. y*.(0.19600*.y -. 0.25472*.x) in
      if q > 0.27597 && (q > 0.27846 || v*.v > (-4.0)*.(log u)*.u*.u) then
        loop ()
      else
        sigma *. (v /. u)
```

```
  in
    loop ()
;;


let loi_uniforme_symetrique sup =
  Random.float (2. *. sup) -. sup
;;




(*            2. Fonctions de transfert            *)


(*definition de quelques fonctions de transfert et de leur derivee respective*)
let sigmoide x = 1. /. ( 1. +. exp ( -. x ) ) ;;
let dsigmoide x = exp ( -. x ) /. ( ( 1. +. exp ( -. x ) ) *. ( 1. +. exp ( -. x ) ) ) ;;
let tanh x = ( exp ( x ) -. exp ( -. x ) ) /. ( exp ( x ) +. exp ( -. x ) ) ;;
let dtanh x = 1. -. ( tanh x ) *. ( tanh x ) ;;


(*retourne la fonction Leaky ReLU de parametre alpha*)
```

```
let lrelu alpha = function
  | x when x > 0. -> x
  | x -> alpha *. x
;;


(*renvoie la derivee de lrelu alpha*)
let dlrelu alpha = function
  | x when x > 0. -> 1.
  | _ -> alpha
;;


(*calcule les valeurs prises par f en chacun des points de abcisses*)
let valeurs f abscisses =
  let n = Array.length abscisses in
  let res = Array.make n 0. in
    for j = 0 to n-1
    do
      res.(j) <- f abscisses.(j)
    done;
    res
;;
```

```
(*prend une fonction f en argument:
  si f est lrelu alpha, renvoie 1 et Some alpha
  si f est sigmoide, renvoie 2 et None
  si f est tanh, renvoie 3 et None
  dans les autres cas, affiche une errreur*)
(*le matching de fonctions est delicat donc on se contente de cette verification*)
let identifie f =
  let abscisses = [|-5.; -1.; -0.1; 0.1; 1.; 5. |] in
  let valeursf = valeurs f abscisses in
    if valeursf.(3) = 0.1 && valeursf.(4) = 1. && valeursf.(5) = 5. then
      let alpha = -. valeursf.(0) /. 5. in
        1, Some alpha
    else if valeursf = valeurs sigmoide abscisses then
      2, None
    else if valeursf = valeurs tanh abscisses then
      3, None
    else
      failwith "fonction d'activation inconnue"
;;


(*si f est une fonction connue, renvoie sa derivee, sinon affiche une erreur*)
let derivee f =
```

```
  let id, optional_param = identifie f in
    match id with
      | 1 -> dlrelu (Option.get optional_param)
      | 2 -> dsigmoide
      | 3 -> dtanh
      | _ -> failwith "fonction d'activation inconnue"
;;


(*definition de la fonction softmax*)
let softmax v =
  let n = Array.length v in
  let sum = ref 0. in
    for j = 0 to n-1
    do
      sum := !sum +. exp v.(j)
    done;
    let res = Array.make n 0. in
      for j = 0 to n-1
      do
        res.(j) <- (exp v.(j)) /. !sum
      done;
      res
;;
```

```
(*            3. Creation de reseaux de neurones            *)


let neurone_xavier n sup f =
  let neurone = { poids = Array.make n 0. ; b = 0. ; f = f } in
    for i = 0 to n-1
    do
      neurone.poids.(i) <- loi_uniforme_symetrique sup
    done ;
    neurone
;;


let neurone_he n sigma f =
  let neurone = { poids = Array.make n 0. ; b = 0. ; f = f } in
    for i = 0 to n-1
    do
      neurone.poids.(i) <- loi_normale_centree sigma
    done ;
```

```
    neurone
;;


let make_tsup dim =
  let m = Array.length dim - 1 in
  let tsup = Array.make m 0. in
    for l = 0 to m-1
    do
      tsup.(l) <- sqrt (6. /. (float_of_int (dim.(l) + dim.(l+1))))
    done;
    tsup
;;


let make_tsigma dim =
  let m = Array.length dim - 1 in
  let tsigma = Array.make m 0. in
    for l = 0 to m-1
    do
      tsigma.(l) <- sqrt (2. /. (float_of_int dim.(l)))
    done;
    tsigma
;;
```

```
(*cree un reseau suivant la methode d'initialisation de Xavier*)
let creer_reseau_xavier dim f =
  let m = Array.length dim - 1 in
  let reseau = Array.make m [| |] in
  let tsup = make_tsup dim in
    for l = 0 to m-1
    do
      let couche = Array.make dim.(l+1) (neurone_xavier dim.(l) tsup.(l) f) in
        for j = 0 to Array.length couche - 1
        do
          couche.(j) <- neurone_xavier dim.(l) tsup.(l) f
        done ;
        reseau.(l) <- couche
    done ;
    reseau
;;


(*cree un reseau suivant la methode d'initialisation de He*)
let creer_reseau_he dim f =
  let m = Array.length dim - 1 in
  let reseau = Array.make m [| |] in
```

```
  let tsigma = make_tsigma dim in
    for l = 0 to m-1
    do
      let couche = Array.make dim.(l+1) (neurone_he dim.(l) tsigma.(l) f) in
        for j = 0 to Array.length couche - 1
        do
          couche.(j) <- neurone_he dim.(l) tsigma.(l) f
        done ;
        reseau.(l) <- couche
    done ;
    reseau
;;


(*cree un reseau avec l'initialisatin de Xavier ou l'initialisation de He selon la fonction
d'activation f en argument*)
let creer_reseau dim f =
  let id, _ = identifie f in
    if id = 1 then
      creer_reseau_he dim f
    else if id = 2 || id = 3 then
      creer_reseau_xavier dim f
    else
      failwith "fonction d'activation inconnue"
```

```
;;



(*              4. Propagation avant              *)


(*calcule les valeurs de preactivation d'une couche recevant entree*)
let preactiv_couche couche entree =
  let ne = Array.length entree and ns = Array.length couche in
  let preactivations = Array.make ns 0. in
    for j = 0 to ns - 1
    do
      let poids = couche.(j).poids in
      let somme_ponderee = ref 0. in
        for i = 0 to ne - 1
        do
          somme_ponderee := !somme_ponderee +. poids.(i) *. entree.(i)
        done ;
        preactivations.(j) <-  !somme_ponderee -. couche.(j).b
    done ;
    preactivations
```

```
;;


(*prend un vecteur de preactivations et une fonction d'activation et renvoie le vecteur d'activations*)
let activation preactivations f =
  let n = Array.length preactivations in
  let activations = Array.make n 0. in
    for j = 0 to n-1
    do
      activations.(j) <- f preactivations.(j)
    done;
    activations
;;


(*calcule le vecteur renvoye en sortie d'une couche lorsquâon lui envoie entree*)
let evalue_couche couche entree =
  activation (preactiv_couche couche entree) couche.(0).f
;;


(*calcule les activations de la derniere couche d'un reseau de neurones qui recoit entree*)
```

```
let evalue_reseau reseau entree =
  let m = Array.length reseau in
  let valeur_couche = ref entree in
    for l = 0 to m-1
    do
      valeur_couche := evalue_couche reseau.(l) !valeur_couche
    done ;
    !valeur_couche
;;


(*renvoie le vecteur de predictions calculee par un reseau muni d'une couche softmax
qui recoit entree*)
let prediction reseau entree =
  softmax (evalue_reseau reseau entree)
;;


(*renvoie les activations de la derniere couche ou les predictions d'un reseau selon
l'option renseignee en argument*)
let sortie_obtenue reseau option_reseau entree =
  match option_reseau with
    | "quadra" -> evalue_reseau reseau entree
    | "smce" -> prediction reseau entree
```

```
    | _ -> failwith "option de structure de reseau inconnue"
;;


(*retourne la preactivation et l'activation de chaque neurone d'un reseau qui recoit
entree*)
let preactiv_activ reseau entree =
  let m = Array.length reseau in
  let preactiv = Array.make m [||] in
  let activ = Array.make m [||] in
    preactiv.(0) <- preactiv_couche reseau.(0) entree;
    activ.(0) <- activation preactiv.(0) reseau.(0).(0).f;
    for l = 1 to m-1
    do
      preactiv.(l) <- preactiv_couche reseau.(l) activ.(l-1);
      activ.(l) <- activation preactiv.(l) reseau.(l).(0).f
    done;
    preactiv, activ
;;
```

```
(*            5. Retropropagation du gradient            *)


(*prend une couche de neurones, les deltas associes a cette couche, les activations de
la couche precedente et un taux d'apprentissage eta et modifie la couche de neurone
selon l'algorithme de descente du gradient*)
let modif_couche couche deltas_couche activ_couche_precedente eta =
  let nb_poids = Array.length activ_couche_precedente in
    for j = 0 to Array.length couche - 1
    do
      let neurone = couche.(j) and delta = deltas_couche.(j) in
      let poids = neurone.poids in
        neurone.b <- neurone.b -. eta *. delta;
        for i = 0 to nb_poids - 1
        do
          poids.(i) <- poids.(i) +. eta *. activ_couche_precedente.(i) *. delta
        done
    done
;;


(*a partir d'un exemple, modifie les parametres de reseau avec le taux d'apprentissage
eta et donne l'erreur quadratique entre les activations de la derniere couche et la sortie
desiree (avant modification)*)
```

```
(*on considere que la fonction d'erreur est l'erreur quadratique*)

let retro_quadra reseau exemple eta =

  let m = Array.length reseau in
  let (entree, sortie_desiree) = exemple in

  (*PROPAGATION AVANT*)
  let preactiv, activ = preactiv_activ reseau entree in

  (*PROPAGATION ARRIERE*)
  let deltas = Array.make m [| |] in
    for l = 0 to m-1
    do
      deltas.(l) <- Array.make (Array.length reseau.(l)) 0. ;
    done;
    let derivee_fonction_activation = derivee reseau.(0).(0).f in
      (*calcul des deltas de la derniere couche*)
      for j = 0 to Array.length reseau.(m-1) - 1
      do
        deltas.(m-1).(j) <- - 2. *. (activ.(m-1).(j) -. sortie_desiree.(j))
                            *. (derivee_fonction_activation preactiv.(m-1).(j))
      done;
      (*retropropagation*)
```

```
for l = m-2 downto 1
do
  let couche_suivante = reseau.(l+1) in
  let deltas_couche_suivante = deltas.(l+1)
  and preactivations_couche = preactiv.(l)
  and nb_neurone_couche_suivante = Array.length couche_suivante in
    for j = 0 to Array.length reseau.(l) - 1
    do
      let temp = ref 0. in
        for k = 0 to nb_neurone_couche_suivante - 1
        do
          temp := !temp +. couche_suivante.(k).poids.(j) *. deltas_couche_suivante.(k)
        done;
        deltas.(l).(j) <- (derivee_fonction_activation preactivations_couche.(j)) *. !temp
    done
done;

(*MODIFICATION DES PARAMETRES*)
modif_couche reseau.(0) deltas.(0) entree eta ;
for l = 1 to m-1
do
  modif_couche reseau.(l) deltas.(l) activ.(l-1) eta
done;
```

```
    let sortie_obtenue = activ.(m-1) in
      erreur_quadra sortie_obtenue sortie_desiree
;;


(*a partir d'un exemple, modifie les parametres de reseau avec le taux d'apprentissage
eta et donne l'erreur quadratique entre la prediction et la sortie desiree (avant
modification)*)
(*on considere que le reseau est muni d'une couche de sortie softmax et que la fonction
d'erreur est l'entropie croisee*)

let retro_smce reseau exemple eta =

  let m = Array.length reseau in
  let (entree, sortie_desiree) = exemple in

  (*PROPAGATION AVANT*)
  let preactiv, activ = preactiv_activ reseau entree in
  let prediction = softmax activ.(m-1) in

  (*PROPAGATION ARRIERE*)
  let deltas = Array.make m [| |] in
    for l = 0 to m-1
    do
```

```
    deltas.(l) <- Array.make (Array.length reseau.(l)) 0. ;
  done;
  let derivee_fonction_activation = derivee reseau.(0).(0).f in
    (*calcul des deltas de la derniere couche cachee*)
    for j = 0 to Array.length reseau.(m-1) - 1
    do
      deltas.(m-1).(j) <- -. (prediction.(j) -. sortie_desiree.(j))
                          *. (derivee_fonction_activation preactiv.(m-1).(j))
    done;
    (*retropropagation*)
    for l = m-2 downto 1
    do
      let couche_suivante = reseau.(l+1) in
      let deltas_couche_suivante = deltas.(l+1)
      and preactivations_couche = preactiv.(l)
      and nb_neurone_couche_suivante = Array.length couche_suivante in
        for j = 0 to Array.length reseau.(l) - 1
        do
          let temp = ref 0. in
            for k = 0 to nb_neurone_couche_suivante - 1
            do
              temp := !temp +. couche_suivante.(k).poids.(j) *. deltas_couche_suivante.(k)
            done;
            deltas.(l).(j) <- (derivee_fonction_activation preactivations_couche.(j)) *. !temp
```

```
        done
      done;

      (*MODIFICATION DES PARAMETRES*)
      modif_couche reseau.(0) deltas.(0) entree eta ;
      for l = 1 to m-1
      do
        modif_couche reseau.(l) deltas.(l) activ.(l-1) eta
      done;

      erreur_quadra prediction sortie_desiree
;;

(*execute retro_quadra ou retro_smce selon l'option renseignee en argument*)
let retro_mode option_retro =
  match option_retro with
    | "quadra" -> retro_quadra
    | "smce" -> retro_smce
    | _ -> failwith "option de retropropagation inconnue"
;;
```

```
(*            6. Apprentissage            *)


(*entraine reseau en effectuant 1 epoch avec l'ensemble d'entrainement exemples;
renvoie la liste de couples (numero d'exemple, erreur) pour chacun des exemples*)
let entraine_erreurs_ordre option_retro reseau exemples eta =
  let retro = retro_mode option_retro in
  let reste = ref exemples and donnees = ref [] and numero_ex = ref 1 in
    while !reste <> []
    do
      donnees := (!numero_ex, retro reseau (List.hd !reste) eta) :: !donnees ;
      reste := List.tl !reste ;
      incr numero_ex
    done;
    (List.rev !donnees)
;;


(*melange la liste d'exemples puis execute entraine_erreurs_ordre*)
let entraine_erreurs_desordre option_retro reseau exemples eta =
  entraine_erreurs_ordre option_retro reseau (desordre exemples) eta
;;
```

```
(*execute entraine_erreurs_ordre ou entraine_erreurs_desordre selon l'option
renseignee en argument*)
let entraine_erreurs_mode option_ordre =
  match option_ordre with
    | "ordre" -> entraine_erreurs_ordre
    | "desordre" -> entraine_erreurs_desordre
    | _ -> failwith "option d'entrainement inconnue"
;;




(*            7. Generalisation            *)


(*prend un reseau, un set de generalisaition (contenant des images jamais vues par le
reseau) et renvoie la liste des erreurs*)
let generalisation_erreurs reseau option_reseau gen_set =
  let reste = ref gen_set and resultats = ref [] and numero_test = ref 1 in
    while !reste <> []
    do
      let y = sortie_obtenue reseau option_reseau (fst (List.hd !reste)) in
```

```
      resultats := (!numero_test, erreur_quadra y (snd (List.hd !reste))) :: !resultats ;
      reste := List.tl !reste ;
      incr numero_test
    done;
    (List.rev !resultats)
;;


(*prend un reseau, un set de generalisaition (contenant des images jamais vues par le
reseau) et renvoie la proportion de predictions justes*)
let generalisation_accuracy reseau gen_set =
  let reste = ref gen_set and nb_succes = ref 0 and nb_test = ref 0 in
    while !reste <> []
    do
      incr nb_test;
      let p = prediction reseau (fst (List.hd !reste)) in
      let maxp = maximum p and c = maximum (snd (List.hd !reste)) in
        if maxp = c then
          incr nb_succes;
        reste := List.tl !reste
    done;
    let accuracy = (float_of_int !nb_succes) /. (float_of_int !nb_test) in
      accuracy
;;
```

```
(*              8. Taux d'apprentissage              *)


(*cree un tableau de nb_epoch taux d'apprentissage qui vont de 0.01 a O.OO1 et
suivent une evolution de type harmonique*)
let make_etas1 nb_epoch =
  let delta = 9. /. (float_of_int (nb_epoch - 3)) in
  let etas1 = Array.make nb_epoch 0.01 in
    for t = 4 to nb_epoch
    do
      etas1.(t-1) <- 0.01 /. (1. +. (float_of_int (t-3) *. delta))
    done;
    etas1
;;


(*cree un tableau de nb_epoch taux d'apprentissage qui vont de 0.01 a O.OO1 et
suivent une evolution de type polynomiale*)
let make_etas2 nb_epoch =
```

```
  let delta = log (10.) /. log (float_of_int (nb_epoch - 3)) in
  let etas2 = Array.make nb_epoch 0.01 in
    for t = 4 to nb_epoch
    do
      etas2.(t-1) <- 0.01 /. (float_of_int (t-3) ** delta)
    done;
    etas2
;;


(*cree un tableau de nb_epoch taux d'apprentissage qui valent tous eta*)
let make_etas3 eta nb_epoch = Array.make nb_epoch eta;;


(*cree un tableau de taux d'apprentissage selon l'option renseignee*)
let make_etas option_etas optional_eta =
  match option_etas with
    | "harmonique" -> make_etas1
    | "polynomiale" -> make_etas2
    | "constante" -> make_etas3 (Option.get optional_eta)
    | _ -> failwith "option d'evolution de taux d'apprentissage inconnue"
;;
```

```
(*              9. Sauvegardes et chargements              *)


let chemin = "C:\\Users\\phima\\Documents\\Ecole\\TIPE\\code\\sauvegardes\\";;


(*sauvegarde reseau (sauf la fonction de transfert) dans le fichier .txt nom_de_fichier (si
ce fichier existe deja, il sera ecrase, sinon, il sera cree)*)
let sauver_reseau reseau nom_de_fichier =
  let fichier = open_out (chemin ^ nom_de_fichier ^ ".txt") in
  let m = Array.length reseau in
    output_string fichier(string_of_int m);
    output_string fichier "\n" ;
    for k = 0 to m-1
    do
      let n = Array.length reseau.(k) in
        output_string fichier(string_of_int n) ;
        output_string fichier "\n" ;
        for i = 0 to n-1
        do
          let p = Array.length reseau.(k).(i).poids in
```

```
            output_string fichier(string_of_int p) ;
            output_string fichier"\n" ;
            for j =0  to p-1
            do
              output_string fichier (string_of_float reseau.(k).(i).poids.(j));
              output_string fichier "\n"
            done ;
            output_string fichier (string_of_float reseau.(k).(i).b) ;
            output_string fichier "\n"
        done
    done ;
    close_out fichier
;;


(*renvoie le reseau sauvegarde dans le fichier .txt nom_de_fichier en attribuant a
chaque neurone du reseau la fonction de transfert f*)
let charger_reseau nom_de_fichier f =
  let fichier = open_in (chemin ^ nom_de_fichier ^ ".txt") in
  let m = int_of_string(input_line fichier) in
  let reseau = Array.make m [||] in
    for l = 0 to m-1
    do
      let n = int_of_string(input_line fichier) in
```

```
      reseau.(l) <- Array.make n {poids = [||]; b = 0.; f = f} ;
      for i = 0 to n-1
      do
       let p = int_of_string (input_line fichier) in
        reseau.(l).(i) <- {poids = (Array.make p 0.); b = 0.; f = f} ;
        for j = 0 to p-1
        do
          reseau.(l).(i).poids.(j) <- float_of_string(input_line fichier) ;
        done ;
        reseau.(l).(i).b <- float_of_string(input_line fichier) ;
      done
    done ;
    reseau
;;


(*renvoie la liste dâexemples sauvegardee dans le fichier .txt nom_de_fichier*)
(*Le fichier doit etre de la forme :
  nb_exemples
  ne
  ns
  entree1_0
  entree1_1
  entree1_2
```

```
  sortie1_0
  sortie1_1
  entree2_0
  entree2_1
  entree2_2
  sortie2_0
  sortie2_1
  entree3_0
  etc*)
(*le fichier .txt doit contenir des valeurs non normalisees, la fonction renvoie des valeurs
normalisees*)
let charger_exemples nom_de_fichier =
  let fichier = open_in (chemin ^ nom_de_fichier ^ ".txt") in
  let nb_exemples = int_of_string (input_line fichier) in
  let ne = int_of_string (input_line fichier) and ns = int_of_string (input_line fichier) in
  let mean = 33.31002426147461 and std = 78.56748962402344 in
  let exemples = ref [] in
    for k = 1 to nb_exemples
    do
      (*lecture de l'entree de l'exemple courant*)
      let entree = Array.make ne 0. in
        for i = 0 to ne - 1
        do
          let valeur = float_of_string (input_line fichier) in
```

```
            entree.(i) <- (valeur -. mean) /. std;
          done ;
          (*lecture de la sortie desiree de l'exemple courant*)
          let sortie = Array.make ns 0. in
           for s = 0 to ns - 1
           do
             sortie.(s) <- float_of_string (input_line fichier)
           done ;
           exemples := (entree, sortie) :: !exemples
      done ;
      close_in fichier;
      List.rev !exemples
;;


(*ecrit dans un fichier .txt la liste de couples (numero, erreur)*)
(*cela permettra de tracer le graphe correspondant*)
let sauver_erreurs liste_erreurs nom_de_fichier =
  let fichier = open_out (chemin ^ nom_de_fichier ^ ".txt") in
  let reste = ref liste_erreurs in
    while !reste <> []
    do
      output_string fichier (string_of_int (fst(List.hd !reste)) ^ " : ") ;
      output_string fichier (string_of_float (snd(List.hd !reste)) ^ "\n") ;
```

```
      reste := List.tl !reste
    done ;
    close_out fichier
;;


(*sauvegarde une accuracy*)
let sauver_accuracy accuracy nom_de_fichier =
  let fichier = open_out (chemin ^ nom_de_fichier ^ ".txt") in
    output_string fichier (string_of_float accuracy ^ "\n") ;
    close_out fichier
;;




(*              10. Obtention de resultats              *)


(*entraine reseau sur train_set avec les taux d'apprentissage etas selon les modes
option_reseau et option_ordre et stocke les reseaux et erreurs obtenus a chaque epoch
dans un dossier nom_dossier*)
let entrainement reseau option_reseau train_set option_ordre etas nom_dossier =
```

```
  let nb_epoch = Array.length etas in
  let entraine_erreurs = entraine_erreurs_mode option_ordre in
    for n = 0 to nb_epoch - 1
    do
      let nom_fichier_donnees =  nom_dossier ^ "\\" ^ (nom_dossier ^ "_donnees_epoch"
^ string_of_int (n+1))
      and nom_fichier_reseau = nom_dossier ^ "\\" ^ (nom_dossier ^ "_reseau_epoch" ^
string_of_int (n+1)) in
        sauver_erreurs (entraine_erreurs option_reseau reseau train_set etas.(n))
nom_fichier_donnees;
        sauver_reseau reseau nom_fichier_reseau
    done
;;


(*pour chacun des nb_epoch reseaux dans nom_dossier, lui attribue la fonction f, et
stocke ses performances sur gen_set dans nom_dossier*)
let performances nom_dossier nb_epoch f option_reseau gen_set =
  for n = 0 to nb_epoch - 1
  do
    let nom_reseau = nom_dossier ^ "\\" ^ (nom_dossier ^ "_reseau_epoch" ^
string_of_int (n+1)) in
    let reseau = charger_reseau nom_reseau f in
    let nom_fichier_erreurs = nom_dossier ^ "\\" ^ (nom_dossier ^ "_erreurs_epoch" ^
```

```
string_of_int (n+1))
    and nom_fichier_accuracy = nom_dossier ^ "\\" ^ (nom_dossier ^ "_accuracy_epoch"
^ string_of_int (n+1)) in
      sauver_erreurs (generalisation_erreurs reseau option_reseau gen_set)
nom_fichier_erreurs;
      sauver_accuracy (generalisation_accuracy reseau gen_set) nom_fichier_accuracy
  done
;;


(*identifie une fonction f et cree une string correspondante*)
let write_function f =
  let id, optional_param = identifie f in
    match id with
      | 1 -> "lrelu " ^ (string_of_float (Option.get optional_param))
      | 2 -> "sigmoide"
      | 3 -> "tanh"
      | _ -> failwith "fonction inconnue"
;;


(*ecrit tous les hyper-parametres du reseau, de la phase d'entrainement et de la phase
de test dans un fichier .txt, dans nom_dossier*)
```

```
let write_hp dim f option_retro train_set option_ordre nb_epoch option_etas optional_eta
gen_set nom_dossier =

  let titre = nom_dossier ^ ": hyper-parametres" in
  let string_f = write_function f in
  let nb_train = string_of_int (List.length train_set) in
  let string_option_ordre = "presentation des exemples: " ^ option_ordre in
  let nb_gen = string_of_int (List.length gen_set) in
  let string_nb_epoch = string_of_int (nb_epoch) in
  let option_creation_reseau =
    let id, _= identifie f in
      if id = 1 then
        "he"
      else if id = 2 || id = 3 then
        "xavier"
      else
        failwith "fonction d'activation inconnue"
  in
  let string_option_creation_reseau = "initialisation des poids: mode " ^
option_creation_reseau in
  let string_option_etas =
    if option_etas = "harmonique" then
      "decroissance harmonique"
    else if option_etas = "polynomiale" then
```

```
    "decroissance polynomiale"
  else if option_etas = "constante" then
    "constante"
  else
    failwith "option d'evolution de taux d'apprentissage inconnue"
 in
 let string_dim = ref "dimensions du reseau: " in
 let p = Array.length dim in
  for k = 0 to p-1
  do
   string_dim := !string_dim ^ (string_of_int dim.(k) ^ " ")
  done;

  let fichier = open_out (chemin ^ nom_dossier ^ "\\" ^ nom_dossier ^ "_hyper-
parametres" ^ ".txt") in
   output_string fichier (titre ^ "\n\n\n") ;
   output_string fichier (!string_dim ^ "\n") ;
   if option_retro = "smce" then
    output_string fichier ("a cela s'ajoute la couche de sortie du softmax" ^ "\n") ;
   output_string fichier ("la fonction d'activation est : " ^ string_f ^ "\n") ;
   if option_retro = "quadra" then
    output_string fichier ("la fonction d'erreur est l'erreur quadratique\n")
   else if option_retro = "smce" then
    output_string fichier ("la fonction d'erreur est l'entropie croisee\n")
```

```
      else
        failwith "configuration de reseau inconnue" ;
      output_string fichier (string_option_creation_reseau ^ "\n") ;
      output_string fichier ("biais initialises a 0" ^ "\n\n") ;
      output_string fichier ("taille training set: " ^ nb_train ^ "\n") ;
      output_string fichier (string_option_ordre ^ "\n") ;
      output_string fichier ("taille generalization set: " ^ nb_gen ^ "\n\n") ;
      output_string fichier ("nombre d'epoch: " ^ string_nb_epoch ^ "\n") ;
      output_string fichier ("evolution des taux d'apprentissage: " ^ string_option_etas);
      if optional_eta <> None then
        output_string fichier ("\n" ^ "valeur du taux d'apprentissage: " ^ (string_of_float
(Option.get optional_eta)));

      close_out fichier
;;


(*arguments:
  1. dimensions du reseau
  2. la fonction de transfert
  3. configuration du reseau: quadra ou smce
  4. nom du training set
  5. ordre ou desordre
  6. nombre d'epoch
```

```
  7. mode d'evolution des taux d'apprentissage
  8. optional_eta, pour le cas evolution constante
  9. nom du generalisation set
  10. nom du dossier ou seront stockees toutes les donnees*)
(*cree le reseau, l'entraine, stocke ses donnees d'entrainement et ses performances et
note tous les hyper-parametres dans nom_dossier*)
(*il faut creer le dossier nom_dossier au prealable*)
let god dim f option_reseau nom_train_set option_ordre nb_epoch option_etas
optional_eta nom_gen_set nom_dossier =
  let train_set = charger_exemples nom_train_set and gen_set = charger_exemples
nom_gen_set in
  let etas = make_etas option_etas optional_eta nb_epoch in
  let reseau = creer_reseau dim f in
    write_hp dim f option_reseau train_set option_ordre nb_epoch option_etas
optional_eta gen_set nom_dossier;
    entrainement reseau option_reseau train_set option_ordre etas nom_dossier;
    performances nom_dossier nb_epoch f option_reseau gen_set
;;




(*            11. Commandes            *)
```

```
(*arguments:
  1. dimensions du reseau
  2. la fonction de transfert
  3. configuration du reseau: quadra ou smce
  4. nom du training set
  5. ordre ou desordre
  6. nombre d'epoch
  7. mode d'evolution des taux d'apprentissage
  8. optional_eta, pour le cas evolution constante
  9. nom du generalisation set
  10. nom du dossier ou seront stockees toutes les donnees*)

(*
god [|784; 200; 50; 10|] (lrelu 0.01)  "smce" "digits_train_set" "desordre" 45
"harmonique" None "digits_gen_set" "god5" ;;
*)
```

```
clear all
close all
clc

b_digits = load('emnist-digits.mat')
b_train = b_digits.dataset.train
b_images = b_train.images
b_labels = b_train.labels

fileid = fopen('C:\Users\phima\Documents\Ecole\TIPE\code\d_70k.txt','w')

nb_test = 70000

fprintf(fileid,'%d', nb_test)
fprintf(fileid,'\n')
fprintf(fileid,'%d', 784)
fprintf(fileid,'\n')
fprintf(fileid,'%d', 10)
fprintf(fileid,'\n')
for i = (1):(nb_test)
    for j = 1:784
        fprintf(fileid,'%f', b_images(i,j))
        fprintf(fileid,'\n')
    end
```

```
    for j = 0:9
        if j == b_labels(i)
            fprintf(fileid,'%f', 1.0)
            fprintf(fileid,'\n')
        else
            fprintf(fileid,'%f', 0.0)
            fprintf(fileid,'\n')
        end
    end
end
```

```
import matplotlib.pyplot as plt
import os

path = "U:\\tipe\\Accuracies\\"

def quick_sort(L):
    if not L: return []
    else:
        pivot = L[0]
        l1 = []
        l2 = []
        for i in L[1:]:
            if int((i[55:]).replace(".txt", ""))< int((pivot[55:]).replace(".txt", "")):
                l1.append(i)
            else:
                l2.append(i)
        return quick_sort(l1) + [pivot] + quick_sort(l2)

def graphe(epoch):
    files = []
    acc = []

    for r, d, f in os.walk(path):
        for file in f:
```

```
            if '.txt' in file:
                files.append(os.path.join(r, file))
    files = quick_sort(files)
    print(files)
    #on lit chacun des fichiers listÃŠs pour crÃŠer une liste contenant les valeurs
d'accuracies
    for i in range(epoch):
        f = open(files[i], "r")
        acc.append(float(f.read()))
        f.close()

    epoch = range(1, epoch + 1 )

    plt.plot(epoch, acc)
    plt.xlabel("Epoch")
    plt.ylabel("Accuracy")
    plt.show()
    return files
```