Guide de l'étudiant

APP1– S7

Réseaux de neurones artificiels

Faculté de génie
Université de Sherbrooke

Hiver 2021

Note : En vue d'alléger le texte, le masculin est utilisé pour désigner les femmes et les hommes.

Document guide
Rédigé par François Grondin et Marc-Antoine Maheux, Hiver 2021

Table des matières

1 Activités pédagogiques et compétences

GRO720: Réseaux de neurones artificiels

1. Maîtriser la théorie de l'apprentissage machine supervisé.
2. Concevoir un réseau de neurones entièrement connecté à plusieurs couches.
3. Utiliser les stratégies d'optimisation pour entraîner un réseau de neurones.

2 Synthèse de l'évaluation

La note attribuée aux activités pédagogiques de l'APP est une note individuelle, sauf pour le rapport d'APP qui est une note par équipe de deux. L'évaluation porte sur les compétences figurant dans la description des activités pédagogiques de l'APP à la section 1. Ces compétences, ainsi que la pondération de chacune d'entre elles dans l'évaluation de l'unité, sont:

Activité pédagogique	**GRO720**		
Compétences	**C1**	**C2**	**C3**
Rapport d'APP et validation	50	50	50
Évaluation sommative	100	100	100
Évaluation finale	100	100	100

Tableau 1: Pondération des points

Le pointage obtenu est ensuite utilisé pour attribuer des cotes selon cette grille:

E	D	D+	C-	C	C+	B-	B	B+	A-	A	A+
<50%	50%	53.5%	57%	60.5%	64%	67.5%	71%	74.5%	78%	81.5%	85%

Tableau 2: Grille de note selon le pointage

3 Qualités de l'ingénieur

Les qualités de l'ingénieur visées par cette unité d'APP sont les suivantes. D'autres qualités peuvent être présentes sans être visées ou évaluées dans cette unité d'APP.

	Q01	Q02	Q03	Q04	Q05	Q06	Q07	Q08	Q09	Q10	Q11	Q12
Touchée	×	×			×							
Évaluée	×	×			×							

Tableau 3: Qualités de l'ingénieur visées

Les qualités de l'ingénieur sont les suivantes. Pour une description détaillée des qualités et leur provenance, consultez le lien suivant: `http://www.usherbrooke.ca/genie/etudiants-actuels/au-baccalaureat/bcapg/`.

Qualité	**Libellé**
Q01	Connaissances en génie
Q02	Analyse de problèmes
Q03	Investigation
Q04	Conception
Q05	Utilisation d'outils d'ingénierie
Q06	Travail individuel et en équipe
Q07	Communication
Q08	Professionnalisme
Q09	Impact du génie sur la société et l'environnement
Q10	Déontologie et équité
Q11	Économie et gestion de projets
Q12	Apprentissage continu

Tableau 4: Liste des qualités de l'ingénieur

4 Énoncé de la problématique

Vous avec été embauché comme stagiaire dans une entreprise qui conçoit des applications mobiles. L'entreprise aimerait développer un logiciel qui permet de valider la solution d'une grille de Sudoku. L'idée est de permettre à l'utilisateur de prendre en photo une grille remplie à la main (par exemple dans un journal), reconnaître les chiffres imprimés et ceux inscrits à la main, exécuter un algorithme pour résoudre le Sudoku, et valider que les chiffres inscrits à la main correspondent à la solution. Les ingénieurs possèdent déjà un engin de reconnaissance des caractères imprimés, qui permet de reconnaître les chiffres initiaux qui sont fournis pour résoudre le puzzle. Cependant, l'engin de reconnaissance n'est pas en mesure de classifier adéquatement les chiffres manuscrits puisque la forme et le style des chiffres changent selon l'écriture de chaque personne. On vous demande donc de concevoir et entraîner un réseau de neurones qui pourra effectuer cette tâche grâce à l'ensemble de données MNIST qui regroupe des milliers d'images de chiffres écrits à la main entre 0 et 9 (pour le besoin du Sudoku, seuls les chiffres de 1 à 9 sont utiles, mais l'entreprise aimerait éventuellement réutiliser le système pour d'autres applications dans lesquelles le chiffre 0 est aussi présent). Cet ensemble est séparé en sous-ensembles d'entraînement, de validation et de test. L'équipe d'ingénierie possède déjà un algorithme qui permet de segmenter les chiffres inscrits sur la grille et de les présenter un à un comme une image de 28×28 pixels de niveaux de gris au système de reconnaissance de chiffres que vous devrez concevoir. Une fois les chiffres identifiés, une autre fonction permet de valider si la solution au Sudoku est la bonne. La figure suivante illustre ce qui devra être accompli par le réseau de neurones. Dans cet exemple, les chiffres sont identifiés un à un grâce au réseau de neurones, et le résultat final permet d'identifier deux chiffres erronés par rapport à la solution.

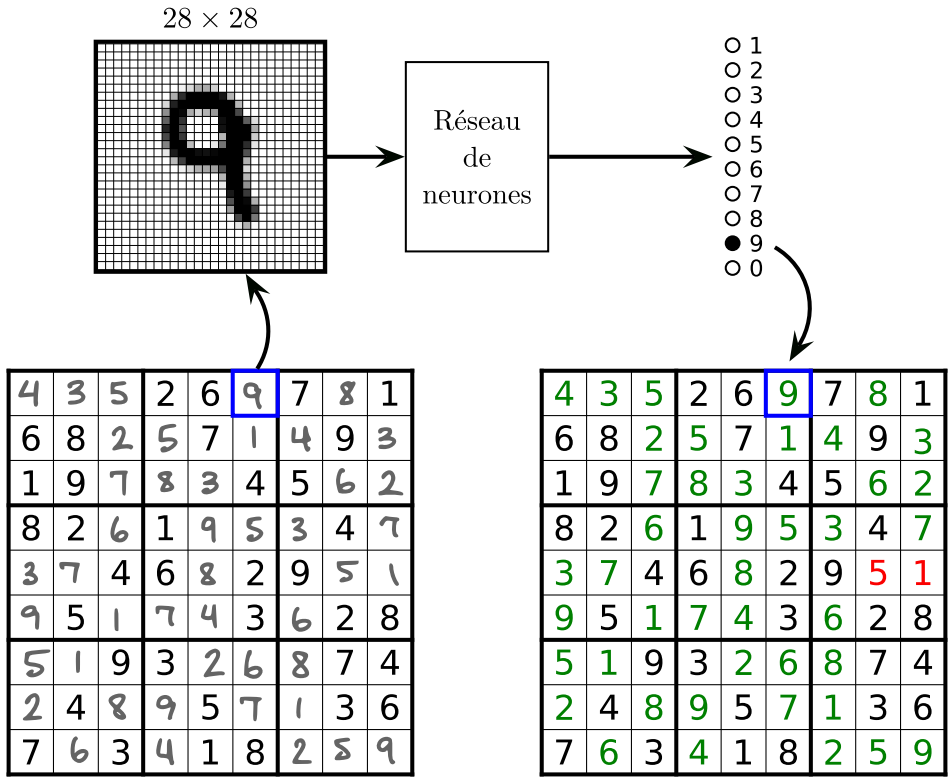
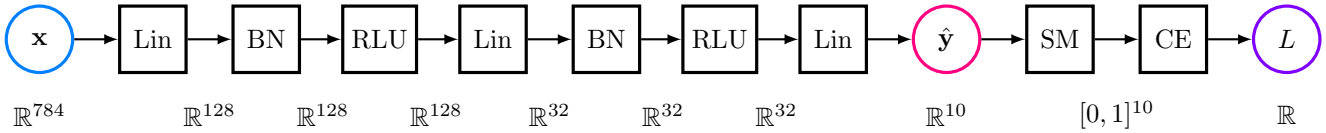


Figure 1: Reconnaissance des chiffres manuscripts dans la grille du Sudoku avec un réseau de neurones

Vous allez devoir réaliser un réseau de neurones à plusieurs couches entièrement connectées. Chaque image de 28×28 pixels sera présentée au réseau. L'image sera d'abord réorganisée en un vecteur possédant 784 dimensions (tous les pixels sont concatenés sur une seule dimension), qui sera envoyé vers une couche linéaire pour générer un vecteur de sortie à 128 dimensions. Ce vecteur sera ensuite normalisé par une couche de normalisation de lot, puis traité par un rectificateur. Le nouveau vecteur de 128 dimensions sera envoyé vers une nouvelle couche linéaire pour produire un nouveau vecteur à 32 dimensions, avant d'être à nouveau normalisé et rectifié. Finalement, le vecteur de 32 dimensions sera envoyé à une dernière couche linéaire qui produira un vecteur à 10 dimensions. Ce vecteur sera

ensuite comparé à la cible grâce à une fonction de coût qui consiste d'une couche de softmax en cascade avec une fonction de coût d'entropie croisée.



Bien qu'il existe déjà certaines librairies logicielles permettant de faire de l'apprentissage profond (e.g. PyTorch, TensorFlow, Keras), l'entreprise aimerait que vous conceviez une librairie logicielle maison dans le langage Python qui repose entièrement sur le module Numpy qui est déjà présent dans la plupart de leurs solutions logicielles. Le stagiaire précédent a déjà mis en place les classes génériques suivantes:

- `Dataset`
- `DatasetLoader`
- `Layer`
- `Loss`
- `Metrics`
- `Network`
- `Optimizer`
- `Trainer`

Votre tâche consistera d'abord à analyser et décortiquer le code déjà en place. Du code a également été réalisé pour charger l'ensemble de données (`Dataset:MnistDataset`) et effectuer l'entraînement (`Trainer:MnistTrainer`). Il manque cependant plusieurs classes que vous devrez créer et qui hériteront des classes génériques déjà en place:

- `Layer:FullyConnectedLayer`
- `Layer:BatchNormalization`
- `Layer:ReLU`
- `Loss:CrossEntropyLoss`
- `Optimizer:SgdOptimizer`

Une fois ces classes programmées et testées individuellement, vous allez devoir créer un réseau avec la classe `Network` qui implémente l'architecture proposée pour la reconnaissance de chiffres. Ce réseau devra ensuite être entraîné et validé afin de démontrer qu'il accomplit sa tâche de classification avec une précision acceptable. Vous devez concevoir votre code pour pouvoir effectuer l'entraînement avec descente stochastique de gradient avec mini-lots.

5 Connaissances nouvelles

Connaissances déclaratives: Quoi

- Ensemble de données d'entraînement, de validation et de test
- Tâches de classification et de régression
- Phénomènes de sous-apprentissage et de surapprentissage
- Techniques de régularisation
- Fonctions d'activations non-linéaires: sigmoïde, tangente hyperbolique, rectificateur, fonction expoentielle normalisée
- Fonctions de coût: erreur quadratique moyenne, entropie croisée, divergence de Kullback-Leibler
- Optimisation avec descente de gradient
- Rétropropagation d'erreur
- Architecture d'un réseau de neurones entièrement connecté

Connaissances procédurales: Comment

- Séparer un ensemble de données en sous-ensembles
- Appliquer des méthodes de régularisation pour réduire le phénomène de surapprentissage
- Calculer manuellement les gradients pour les fonctions d'activation, les fonctions de coût et les couches d'un réseau de neurones
- Programmer une application qui utilise la descente de gradient pour entraîner un réseau de neurones
- Mesurer les performances d'un réseau de neurones entraîné pour accomplir une tâche particulière

Connaissances conditionnelles: Quand

- Choisir entre une approche de classification ou de régression pour une tâche donnée
- Choisir quand utiliser chaque fonction de coût

6 Guide de lecture

6.1 Références obligatoires

Voici le livre de références qui sera à l'étude pour cet APP:

The Hundred-Page Machine Learning Book de Andriy Burkov (`http://themlbook.com/`)

Ce livre est disponible en version française, et sera utilisé également pour les APPs de GRO721 et GRO722. Des notes de cours sont également disponibles à la fin de ce guide à la section 15.

6.2 Séquence d'étude suggérée

6.2.1 Préparation au 1er procédural

- Livre – Chapitre 1: Introduction, pp. 1–8
- Livre – Chapitre 2: Notations et définitions, pp. 9–21
- Livre – Chapitre 4: Anatomie d'un algorithme d'apprentissage, pp. 39–46
- Livre – Chapitre 5: Fondamentaux de l'apprentissage machine, pp. 49–68

6.2.2 Préparation au 1er laboratoire

- Notes – Section 15.4: Optimisation par descente de gradient

6.2.3 Préparation au 2e procédural

- Notes – Section 15.1: Concepts mathématiques
- Notes – Section 15.2: Fonctions non-linéaires
- Notes – Section 15.3: Fonctions de coûts

6.2.4 Préparation au 2e laboratoire

- Notes – Section 15.5: Couches
- Notes – Section 15.6: Vérifications diligentes

7 Logiciels

Pour cet APP, vous aurez besoin d'installer l'environnement de Python 3, disponible gratuitement en ligne pour tous les systèmes d'exploitation (`www.python.org`). Il est également suggéré d'utiliser le gestionnaire de progiciel Pip pour installer les progiciels suivants: Numpy, Matplotlib, Tqdm. Il est recommandé d'utiliser l'IDE PyCharm (`https://www.jetbrains.com/fr-fr/pycharm/`).

8 Sommaire des activités liées à l'unité

Semaine 1

- Tutorat d'ouverture
- Activité procédurale 1

Semaine 2

- Activité en laboratoire 1
- Activité procédurale 2
- Activité en laboratoire 2

Semaine 3

- Validation de la problématique

Semaine 4

- Tutorat de fermeture
- Évaluation sommative

9 Productions à remettre

- Validation pratique de la solution en laboratoire. Vous devrez présenter votre solution en équipe lors de cette rencontre. **La présence des deux membres de l'équipe est obligatoire.**
- Rapport d'APP à remettre avant 9h00 le jour du deuxième tutorat. Les consignes de réaction de l'unité d'APP sont données à la section 9.2.

Tout retard sur la remise de livrable entraîne une pénalité de 20% par jour.

9.1 Formation des équipes

La taille des équipes pour l'APP est fixée à 2. Aucune exception ne sera accordée. Veuillez utiliser le formulaire disponible sur le site de la session pour inscrire votre équipe.

9.2 Livrables

Votre solution à la problématique sera évaluée lors d'une séance de validation, mais surtout par un rapport qui décrira vos choix technologiques et le code que vous aurez développé. On attend donc une paire de livrables (rapport et code) par équipe de 2 étudiants.

Rapport

Le rapport, de 10 pages **au maximum**, devra contenir:

- Une description des couches de neurones implémentées
- Une description de la fonction de coût implémentée
- Une description de l'optimisation implémentée
- Un schéma bloc illustrant l'architecture du cadre logiciel et le rôle de chaque classe
- Les courbes de la fonction de coût et la précision pour les ensembles d'entraînement et de validation en fonction des époques d'entraînement

Fichiers de code

Vous devez également remettre le code complet de votre solution. Vous pouvez inclure seulement les fichiers que vous avez modifiés (ou ajoutés) aux fichiers fournis pour l'APP.

Procédure de dépôt

Vous devrez combiner en une seule archive ZIP votre rapport **en format PDF** et vos fichiers de code pour le dépôt. Le lien vers la plateforme de dépôt sera disponible sur le site web de la session.

10 Évaluations

10.1 Rapport d'APP

Le contenu du rapport sera évalué selon cette pondération:

Activité pédagogique	**GRO720**		
Compétences	**C1**	**C2**	**C3**
Rapport	**35**	**35**	**35**
Couches de neurones	15	15	–
Entièrement connecté	*5*	*5*	–
Normalisation de lot	*5*	*5*	–
Rectificateur	*5*	*5*	–
Fonction de coût	5	5	–
Entropie croisée binaire	*5*	*5*	–
Optimisation	5	–	5
Descente de gradient stochastique avec mini-lots	*5*	–	*5*
Schéma bloc de l'architecture	10	15	–
Courbes d'entraînement	–	–	30
Fonction de coût	–	–	*15*
Précision	–	–	*15*
Validation	**15**	**15**	**15**
Total	**50**	**50**	**50**

Tableau 5: Pondération des points

10.2 Validation

Votre solution sera également validée en personne lors de la séance de validation. Elle sera évaluée de façon critériée. Vous n'avez rien à ajouter au rapport en lien avec la validation, mais le pointage résultant de l'évaluation y sera ajouté lors de la correction. Voici la grille d'évaluation pour la validation.

10.3 Évaluation sommative

L'évaluation sommative porte sur tous les objectifs d'apprentissage de l'unité. C'est un examen théorique qui se fera **sans documentation**. Des informations pertinentes seront ajoutées en annexe à l'examen.

10.4 Évaluation finale

L'évaluation finale se fera par activité pédagogique et portera sur tous les objectifs d'apprentissage de cette activité pédagogique. C'est un examen théorique qui se fera **sans documentation**. Des informations pertinentes seront ajoutées en annexe à l'examen.

Qualité	**Q01**	**Q02**	**Q05**	**Q05**
Compétence	**C1**	**C3**	**C2**	**C3**
Pondération	**15**	**7**	**15**	**8**
Excellent (4) – 100%	Connaît adéquament les équations pour les couches de neurones et les fonctions de coût	Applique efficament la procédure d'entraînement d'un réseau de neurones et analyse parfaitement les résultats	L'entraînement avec optimisation par descente de gradient est parfaitement implémenté	Les modules pour constituer le réseau de neurones sont parfaitement implémentés
Cible (3) – 85%	Connaît la plupart des équations pour les couches de neurones et les fonctions de coût	Applique la procédure d'entraînement d'un réseau de neurones et analyse correctement les résultats	L'entraînement avec optimisation par descente de gradient est correctement implémenté	Les modules pour constituer le réseau de neurones sont correctement implémentés
Seuil (2) – 60%	Connaît les équations essentielles pour les couches de neurones et les fonctions de coût, avec quelques erreurs mineures	Applique en bonne partie la procédure d'entraînement d'un réseau de neurones et analyse les résultats avec quelques erreurs mineures	L'entraînement avec optimisation par descente de gradient est partiellement implémenté (les fonctionnalités essentielles sont présentes)	Les modules pour constituer le réseau de neurones sont partiellement implémentés (les fonctionnalités essentielles sont présentes)
Insuffisant (1) – 25%	Connaît superficiellement les équations pour les couches de neurones et les fonctions de coût, avec plusieurs erreurs importantes	Applique minimalement la procédure d'entraînement d'un réseau de neurones et analyse les résultats avec plusieurs erreurs importantes	L'entraînement avec optimisation par descente de gradient est partiellement implémenté (certaines fonctionnalités essentielles sont absentes)	Les modules pour constituer le réseau de neurones sont partiellement implémentés (les fonctionnalités essentielles sont absentes)
Non initié (0) – 0%	Ne connaît pas les équations pour les couches de neurones et les fonctions de coût	N'applique pas la procédure d'entraînement d'un réseau de neurone et n'est pas en mesure d'analyser les résultats	L'entraînement avec optimisation par descente de gradient n'est pas implémenté	Les modules pour constituer le réseau de neurones ne sont pas implémentés

Tableau 6: Grille de validation

11 Formation à la pratique procédurale 1

11.1 Buts de l'activité

- Se familiariser avec les étapes à suivre pour organiser un ensemble de données et entraîner un modèle par apprentissage supervisé.
- Identifier une tâche comme étant une tâche de classification ou de régression.
- Comprendre les phénomènes de surapprentissage et sous-apprentissage, et identifier les stratégies pour y remédier.
- Connaître et appliquer les métriques de validation lors d'une tâche de classification.

11.2 Guide de lecture

The Hundred-Page Machine Learning Book

- Chapitre 1: Introduction, pp. 1–8
- Chapitre 2: Notations et définitions, pp. 9–21
- Chapitre 4: Anatomie d'un algorithme d'apprentissage, pp. 39–46
- Chapitre 5: Fondamentaux de l'apprentissage machine, pp. 49–68

11.3 Contenu

Question 1 – Vous devez concevoir un réseau de neurones qui sera en mesure de déterminer si une photo contient un chat ou un chien. On vous fournit un ensemble de données contenant $10,000$ photos, la moitié étant des photos de chats et l'autre moitié des photos de chiens. Décrivez les étapes à suivre pour préparer et effectuer l'entraînement d'un réseau de neurones qui accomplira cette tâche.

Question 2 – Voici plusieurs tâches à accomplir. S'agit-il d'une tâche de classification ou de régression?

1. Déterminer si un extrait sonore contient de la parole humaine ou du bruit ambiant.
2. Prédire la valeur d'un titre en bourse en fonction de l'historique du titre au cours de la dernière année.
3. Prédire le mot suivant dans une phrase en fonction des mots qui le précèdent.
4. Estimer la température moyenne à l'extérieur selon le jour de l'année et la latitude et longitude d'un lieu sur le globe.
5. Prédire le nombre de locuteurs dans un extrait audio.

Question 3 – Voici l'erreur obtenue sur les sous-ensembles d'entraînement et de validation en fonction de la complexité du modèle:

Expliquez les phénomènes observés dans les zones A et B. Quelle serait les solutions possibles pour maintenir une complexité élevée dans la zone B mais réduire l'erreur observée avec le sous-ensemble de test?

Question 4 – Nous avons entraîné le réseau de neurones qui effectue une classification pour déterminer si un segment audio d'une seconde contient de la parole humaine ou non. Le réseau génère une sortie entre 0 et 1: une valeur proche de 0 indique que le segment ne contient pas de voix, et une valeur proche de 1 indique qu'il contient de la parole. Voici la distribution des sorties du réseau en fonction des entrées durant la validation:
Si le seuil est fixé à 0.5, déterminer le nombre de vrais positifs, de vrais négatifs, de faux positifs et de faux négatifs, la matrice de confusion, la justesse, la précision et le rappel.

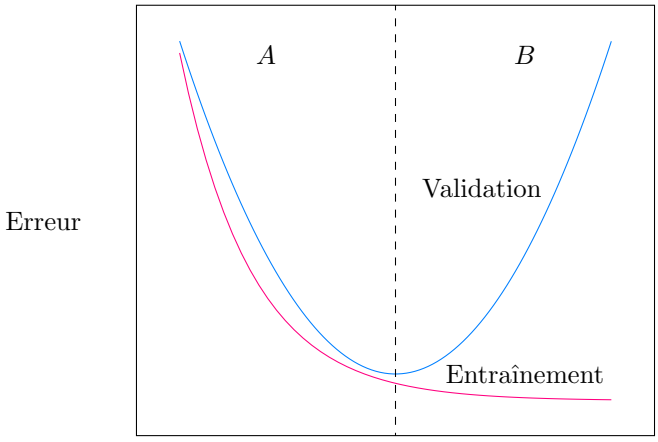


Figure 2: Erreur pour les ensembles de données de validation et d'entraînement

#	Entrée	Sortie
1	Parole humaine	0.20
2	Son d'un avion	0.90
3	Parole humaine	0.99
4	Parole humaine	0.75
5	Son d'une voiture	0.01
6	Musique	0.10
7	Parole humaine	0.80
8	Bruit	0.10
9	Parole humaine	0.70
10	Son de cloche	0.70

Tableau 7: Distribution des sorties du réseau en fonction des entrées durant la validation

12 Formation à la pratique en laboratoire 1

12.1 Buts de l'activité

Dans ce laboratoire, vous allez réaliser un code Python pour expérimenter avec l'algorithme de descente de gradient, et par la même occasion vous aurez l'opportunité de vous familiariser avec le progiciel Numpy qui sera utilisé pour résoudre la problématique.

12.2 Guide de lecture

Notes de cours

- Section 15.4: Optimisation par descente de gradient

12.3 Consignes

Nous allons explorer comment il est possible d'inverser une matrice à l'aide de l'approche de descente de gradient. Supposons que nous avons une matrice carrée $\mathbf{A} \in \mathbb{R}^{N \times N}$:

$$\mathbf{A} = \begin{bmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,N} \\ a_{2,1} & a_{2,2} & \dots & a_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N,1} & a_{N,2} & \dots & a_{N,N} \end{bmatrix} \tag{1}$$

Nous cherchons à obtenir la matrice $\mathbf{B} \in \mathbb{R}^{N \times N}$, de sorte que $\mathbf{BA} = \mathbf{I}$, où $\mathbf{I} \in \mathbb{R}^{N \times N}$ est la matrice d'identité. Nous avons donc:

$$\mathbf{B} = \begin{bmatrix} b_{1,1} & b_{1,2} & \dots & b_{1,N} \\ b_{2,1} & b_{2,2} & \dots & b_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ b_{N,1} & b_{N,2} & \dots & b_{N,N} \end{bmatrix} \tag{2}$$

$$\mathbf{I} = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix} \tag{3}$$

Il est possible de définir la fonction de coût suivante que nous cherchons à minimiser. Ici l'opérateur $\|\dots\|_2^2$ implique que tous les éléments de la matrice sont mis au carré et additionnés ($\|\mathbf{X}\|_2^2 = \sum_{i=1}^{N} \sum_{j=1}^{N} x_{i,j}^2$):

$$L = \|\mathbf{BA} - \mathbf{I}\|_2^2 \tag{4}$$

Le gradient correspond à:

$$\frac{\partial L}{\partial \mathbf{B}} = 2(\mathbf{BA} - \mathbf{I})\mathbf{A}^T \tag{5}$$

Et donc l'optimisation se résume à estimer de manière itérative la matrice $\mathbf{B}$:

$$\mathbf{B}^t = \mathbf{B}^{t-1} - \mu \frac{\partial L}{\partial \mathbf{B}} \tag{6}$$

12.3.1 Matrice 3×3

Commencez par écrire le code pour calculer l'inverse de la matrice suivante:

$$\mathbf{A} = \begin{bmatrix} 3 & 4 & 1 \\ 5 & 2 & 3 \\ 6 & 2 & 2 \end{bmatrix} \tag{7}$$

Les fonctions suivantes seront particulièrement utiles:

- numpy.array
- numpy.identity
- numpy.matmul
- numpy.transpose

Si l'optimisation s'effectue correctement, le résultat devrait se rapprocher de la solution suivante:

$$\mathbf{A}^{-1} = \begin{bmatrix} -\frac{1}{12} & -\frac{1}{4} & +\frac{5}{12} \\ +\frac{1}{3} & +0 & -\frac{1}{6} \\ -\frac{1}{12} & +\frac{3}{4} & +\frac{-7}{12} \end{bmatrix} \tag{8}$$

Effectuez l'optimisation avec les valeurs de μ suivantes pour 1000 itérations:

- $\mu = 0.005$
- $\mu = 0.001$
- $\mu = 0.01$

Calculez le coût pour chaque itération et afficher le coût en fonction de l'itération sur un graphique en utilisant le progiciel Matplotlib. Vérifier que l'inversion a fonctionné en multipliant avec la matrice initiale afin de vérifier que la matrice identitaire est obtenue. Laquelle de ces valeurs de pas mène à la bonne solution après 1000 itérations? Que se passe-t-il avec les autres valeurs?

12.3.2 Matrice 6×6

Modifiez votre code pour procéder à l'inversion de la matrice suivante:

$$\mathbf{A} = \begin{bmatrix} 3 & 4 & 1 & 2 & 1 & 5 \\ 5 & 2 & 3 & 2 & 2 & 1 \\ 6 & 2 & 2 & 6 & 4 & 5 \\ 1 & 2 & 1 & 3 & 1 & 2 \\ 1 & 5 & 2 & 3 & 3 & 3 \\ 1 & 2 & 2 & 4 & 2 & 1 \end{bmatrix} \tag{9}$$

Ajustez le pas μ et le nombre d'itérations au besoin. Afficher le coût en fonction de l'itération sur un graphique et vérifier que la matrice identitaire est obtenue avec la matrice inverse estimée.

12.3.3 Matrice 4×4

Modifiez votre code pour inverser la matrice suivante:

$$\mathbf{A} = \begin{bmatrix} 2 & 1 & 1 & 2 \\ 1 & 2 & 3 & 2 \\ 2 & 1 & 1 & 2 \\ 3 & 1 & 4 & 1 \end{bmatrix} \tag{10}$$

Que se passe-t-il? Pourquoi?

13 Formation à la pratique procédurale 2

13.1 Buts de l'activité

- Se familiariser avec les fonctions d'activation non-linéaires et être en mesure de calculer leur dérivée.
- Se familiariser avec les fonctions de coûts et être en mesure de calculer leur dérivée.
- Calculer la propagation des gradients dans un perceptron.

13.2 Guide de lecture

The Hundred-Page Machine Learning Book

- Section 6.1: Réseaux de neurones, pp. 69–72

Notes de cours

- Section 15.1: Concepts mathématiques
- Section 15.2: Fonctions non-linéaires
- Section 15.3: Fonctions de coûts

13.3 Contenu

Question 1 – Définissez mathématiquement les fonctions de coûts suivantes, et calculez leurs dérivées: a) Redresseur; b) Sigmoïde; c) Tangente hyperbolique; Lorsque possible, exprimez la dérivée en fonction de la sortie y.

Question 2 – Définissez mathématiquement les fonctions de coûts suivantes pour un vecteur de sortie avec N éléments, et calculez leurs dérivées: a) Erreur quadratique moyenne; b) Entropie croisée; c) Divergence de Kullback-Leibler; En supposant que $\hat{y}_n > 0 \ \forall n \in \{1, 2, \ldots, N\}$, qu'arrive-t-il à la valeur de la fonction coût de Divergence de Kullback-Leibler lorsque $y_n = 0$?

Question 3 – Supposons que nous avons le perceptron suivant:

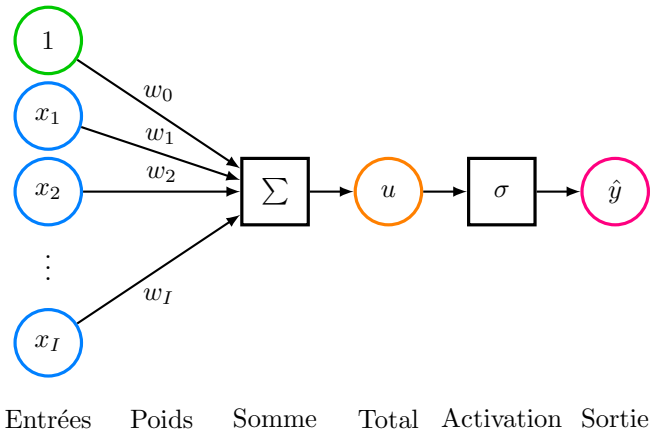


Figure 3: Architecture du perceptron

Nous avons $I = 3$ entrées ($x_1 = 1$, $x_2 = 3$, $x_3 = -1$), et la cible correspond à $y = 1$. La fonction d'activation $\sigma(u)$ correspond à une fonction sigmoïde. Lequel de ces ensembles de paramètres permet de prédire une sortie qui minimise l'erreur quadratique moyenne? Évaluez également le gradient des paramètres w_0, w_1, w_2 et w_3 de la fonction de coût pour chaque ensemble.

Ensemble A: $w_0 = 3$, $w_1 = 2$, $w_2 = -1$, $w_3 = 1$
Ensemble B: $w_0 = -2$, $w_1 = 3$, $w_2 = 2$, $w_3 = 0$

14 Formation à la pratique en laboratoire 2

14.1 Buts de l'activité

Ce laboratoire permettra de se familiariser avec l'entraînement d'un réseau de neurones à quelques couches, et offrira des pistes de solution pour résoudre la problématique.

14.2 Guide de lecture

Notes de cours

- Section 15.1: Concepts mathématiques
- Section 15.2: Fonctions non-linéaires
- Section 15.3: Fonctions de coûts
- Section 15.4: Optimisation par descente de gradient
- Section 15.5: Couches
- Section 15.6: Vérifications diligentes

14.3 Consignes

Nous voulons concevoir un réseau de neurones qui permettra de classifier des points sur une image en fonction de deux zones. La figure suivante illustre la classification à effectuer avec l'ensemble d'entraînement. Si les coordonnées d'un point se situe dans la zone de gauche du logo (en rouge), ce point doit être associé à la classe 0, tandis que s'il est dans la zone de droite (en bleu), il doit être associé à la classe 1.

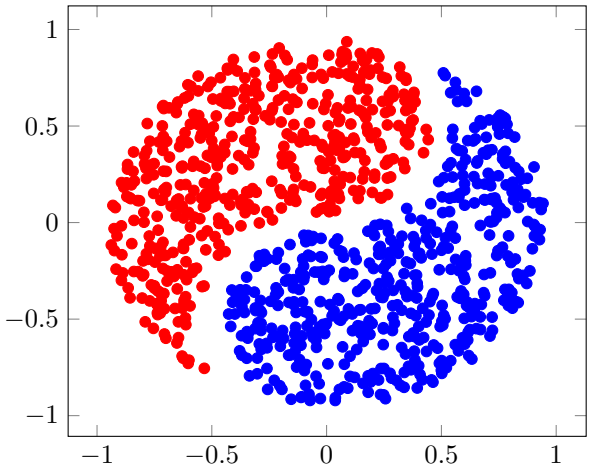


Figure 4: Ensemble d'entraînement pour la classification

Pour accomplir cette tâche, vous allez devoir réaliser un réseau de neurones composé d'une couche linéaire, une activation par rectificateur, une autre couche linéaire et une autre activation par rectificateur, et finalement une dernière couche linéaire et une activation par sigmoïde. Il s'agit d'une tâche de classification binaire, et donc nous utiliserons l'entropie croisée binaire comme fonction de coût à minimiser. Vous allez entraîner votre réseau de neurones à l'aide des données contenues dans le fichier CSV `train.csv`, qui contient la coordonnée x, y et la classe du point (0 ou 1). Vous utiliserez les points dans le fichier `val.csv` pour faire la validation (donc vous assurer qu'il n'y a pas de surapprentissage) et finalement les points dans le fichier `test.csv` pour mesurer les performances de votre réseau.

On peut observer la propagation du gradient dans le réseau pour faire la mise à jour des paramètres par descente de gradient. Vous pouvez vous référer aux notes de cours à la section 15.5 pour obtenir les expressions pour les gradients.

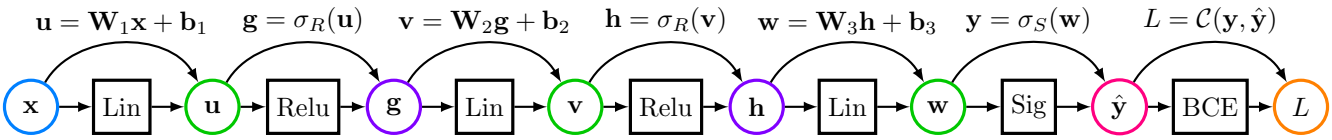


Figure 5: Couches du réseau et prédiction de la sortie

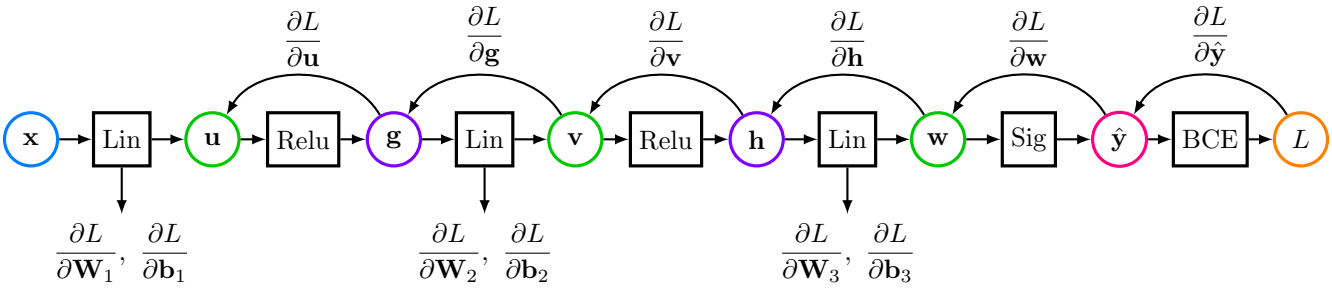


Figure 6: Couches du réseau et rétropropagation de l'erreur

Vous allez devoir coder les fonctions suivantes en utilisant le progiciel Numpy:

- `fully_connected_forward(W, b, X)`: Calcule l'inférence d'une couche linéaire avec en entrée le vecteur $\mathbf{X} \in \mathbb{R}^{N \times I}$ et les paramètres $\mathbf{W} \in \mathbb{R}^{N \times I \times J}$ et $\mathbf{b} \in \mathbb{R}^{N \times J}$.
- `fully_connected_backward(W, b, X, output_grad)`: Calcule les gradients $\frac{\partial L}{\partial \mathbf{x}} \in \mathbb{R}^{N \times I}$, $\frac{\partial L}{\partial \mathbf{W}} \in \mathbb{R}^{N \times I \times J}$ et $\frac{\partial L}{\partial \mathbf{b}} \in \mathbb{R}^{N \times J}$ en fonction du vecteur en entrée $\mathbf{X} \in \mathbb{R}^{N \times I}$ durant la phase d'inférence, des paramètres $\mathbf{W} \in \mathbb{R}^{N \times I \times J}$ et $\mathbf{b} \in \mathbb{R}^{N \times J}$, et du gradient à la sortie $\frac{\partial L}{\partial \mathbf{y}} \in \mathbb{R}^{N \times J}$.
- `relu_forward(X)`: Calcule l'inférence d'une fonction d'activation rectificateur avec en entrée le vecteur $\mathbf{X} \in \mathbb{R}^{N \times I}$.
- `relu_backward(X, output_grad)`: Calcul le gradient $\frac{\partial L}{\partial \mathbf{x}} \in \mathbb{R}^{N \times I}$ en fonction du vecteur en entrée $\mathbf{X} \in \mathbb{R}^{N \times I}$ durant la phase d'inférence, et du gradient à la sortie $\frac{\partial L}{\partial \mathbf{y}} \in \mathbb{R}^{N \times I}$.
- `sigmoid_forward(X)`: Calcule l'inférence d'une fonction d'activation sigmoïde avec en entrée le vecteur $\mathbf{X} \in \mathbb{R}^{N \times I}$.
- `sigmoid_backward(X, output_grad)`: Calcule le gradient $\frac{\partial L}{\partial \mathbf{x}} \in \mathbb{R}^{N \times I}$ en fonction du vecteur en entrée $\mathbf{X} \in \mathbb{R}^{N \times I}$ durant la phase d'inférence, et du gradient à la sortie $\frac{\partial L}{\partial \mathbf{y}} \in \mathbb{R}^{N \times I}$.
- `bce_forward(x, target)`: Calcule l'erreur pour une fonction de coût d'entropie croisée binaire entre la prédiction et la cible.
- `bce_backward(x, target)`: Calcule le gradient pour une fonction de coût d'entropie croisée binaire à partir de la prédiction et de la cible.

Une fois ces couches implémentées, utilisez la fonction `test()` qui est fournie pour valider votre code. Notez que la méthode de vérification diligente avec calcul du gradient est employée ici pour faire cette validation.

Modifiez ensuite la fonction `train()` afin d'y ajouter les étapes du calcul du gradient et de la descente de gradient. Ensuite, vérifier que votre réseau est en mesure d'effectuer un surapprentissage d'un petit ensemble de données. Finalement, lancer l'entraînement et observer l'évolution de la fonction de coût et de la précision au fil du temps.

15 Notes de cours

Cette section introduit certains concepts non-converts ou couverts rapidement dans l'ouvrage de référence proposé pour ce module.

15.1 Concepts mathématiques

Voici d'abord quelques rappels concernant plusieurs concepts mathématiques qui seront utiles pour analyser et entraîner un réseau de neurones. L'expression log utilisée ici fait référence au logarithme naturel ($\log(e^x) = x$).

15.1.1 Dérivées

Voici un rappel de quelques dérivées utiles. Notez que $f'(x) = \frac{d}{dx}f(x)$ et $g'(x) = \frac{d}{dx}g(x)$.

$$\frac{d}{dx}cx = c \tag{11}$$

$$\frac{d}{dx}x^c = cx^{c-1} \tag{12}$$

$$\frac{d}{dx}\log(x) = \frac{1}{x} \tag{13}$$

$$\frac{d}{dx}\exp(x) = \exp(x) \tag{14}$$

$$\frac{d}{dx}f(g(x)) = f'(g(x))g'(x) \tag{15}$$

$$\frac{d}{dx}f(x)g(x) = f'(x)g(x) + f(x)g'(x) \tag{16}$$

$$\frac{d}{dx}\frac{f(x)}{g(x)} = \frac{f'(x)g(x) - f(x)g'(x)}{g(x)^2} \tag{17}$$

15.1.2 Limites

Lorsqu'une expression est indéfinie (par exemple 0 divisé par 0), il est souvent utile d'utiliser la règle de l'Hôpital:

$$\lim_{x \to c}\frac{f(x)}{g(x)} = \lim_{x \to c}\frac{f'(x)}{g'(x)}. \tag{18}$$

15.1.3 Gradient

Si on définit le vecteur $\mathbf{x} \in \mathbb{R}^N$ comme suit:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}. \tag{19}$$

Alors le gradient d'une expression $f(\mathbf{x}) \in \mathbb{R}$ se définit ainsi:

$$\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial f(\mathbf{x})}{\partial x_1} \\ \frac{\partial f(\mathbf{x})}{\partial x_2} \\ \vdots \\ \frac{\partial f(\mathbf{x})}{\partial x_N} \end{bmatrix}. \tag{20}$$

L'expression $\frac{\partial}{\partial x}$ représente la dérivée partielle en fonction de x. On peut également définir un gradient en fonction d'une matrice. Si nous avons la matrice $\mathbf{X} \in \mathbb{R}^{N \times M}$ suivante:

$$\mathbf{X} = \begin{bmatrix} x_{1,1} & x_{1,2} & \dots & x_{1,M} \\ x_{2,1} & x_{2,2} & \dots & x_{2,M} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N,1} & x_{N,2} & \dots & x_{N,M} \end{bmatrix}. \tag{21}$$

Alors le gradient d'une expression $f(\mathbf{X}) \in \mathbb{R}$ se définit comme suit:

$$\frac{\partial f(\mathbf{X})}{\partial \mathbf{X}} = \begin{bmatrix} \frac{\partial f(\mathbf{X})}{\partial x_{1,1}} & \frac{\partial f(\mathbf{X})}{\partial x_{1,2}} & \dots & \frac{\partial f(\mathbf{X})}{\partial x_{1,M}} \\ \frac{\partial f(\mathbf{X})}{\partial x_{2,1}} & \frac{\partial f(\mathbf{X})}{\partial x_{2,2}} & \dots & \frac{\partial f(\mathbf{X})}{\partial x_{2,M}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f(\mathbf{X})}{\partial x_{N,1}} & \frac{\partial f(\mathbf{X})}{\partial x_{N,2}} & \dots & \frac{\partial f(\mathbf{X})}{\partial x_{N,M}} \end{bmatrix}. \tag{22}$$

15.2 Fonctions non-linéaires

Voici les principales fonctions non-linéaires qui seront utilisées dans ce module.

15.2.1 Rectificateur

Un rectificateur transforme une entrée $x \in \mathbb{R}$ vers une sortie $y \in \mathbb{R}^+$. L'équation suivante représente la courbe non-linéaire d'un rectificateur. Notez que la non-linéarité vient de la portion nulle de la courbe lorsque la valeur d'entrée est négative.

$$y = \max(0, x) = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases} \tag{23}$$

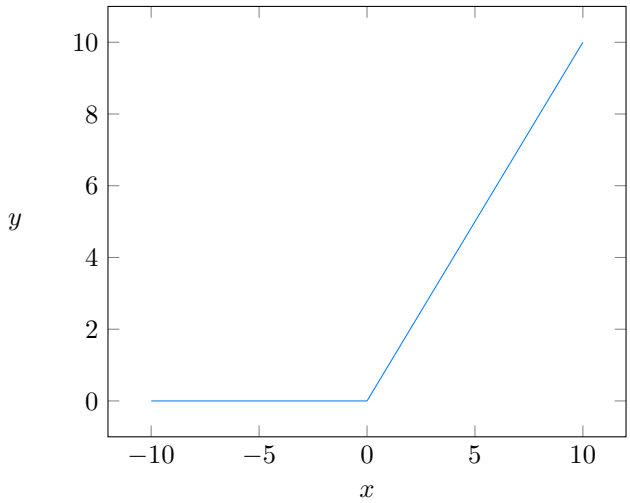


Figure 7: Fonction de rectificateur

La dérivée de cette fonction correspond à:

$$\frac{\partial y}{\partial x} = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases}. \tag{24}$$

15.2.2 Fonction sigmoïde

La fonction sigmoïde projette la valeur d'entrée $x \in \mathbb{R}$ vers une valeur $y \in [0, 1]$.

$$y = \frac{1}{1 + \exp(-x)} \tag{25}$$

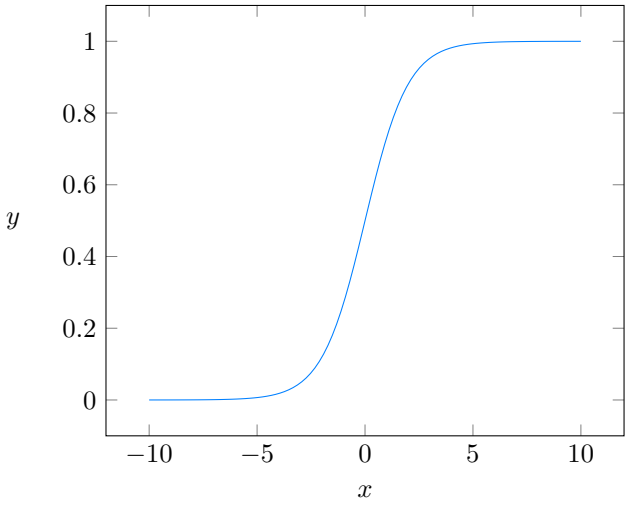


Figure 8: Fonction sigmoïde

La dérivée de cette fonction correspond à:

$$\frac{\partial y}{\partial x} = (1 - y)y. \tag{26}$$

15.2.3 Fonction tangente hyperbolique

La fonction tangente hyperbolique est similaire à la fonction sigmoïde, mais elle permet de projeter la valeur d'entrée $x \in \mathbb{R}$ vers une valeur $y \in [-1, +1]$.

$$y = \tanh(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)} \tag{27}$$

La dérivée de cette fonction correspond à:

$$\frac{\partial y}{\partial x} = 1 - y^2. \tag{28}$$

15.3 Fonctions de coûts

Entraîner un réseau de neurones revient à adapter ses paramètres afin de minimiser une fonction de coût.

15.3.1 Erreur quadratique moyenne

Cette fonction de coût est habituellement utilisée lorsqu'on tente de résoudre un problème de **régression**. Si on considère que la variable $y \in \mathbb{R}$ représente la cible et que $\hat{y} \in \mathbb{R}$ représente la valeur prédite, alors la fonction de coût $L \in \mathbb{R}^+$ est définie comme suit:

$$L = (\hat{y} - y)^2. \tag{29}$$

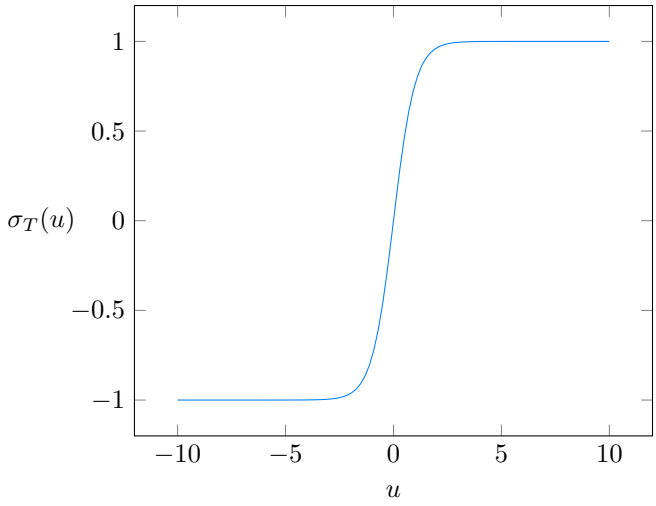


Figure 9: Fonction tangente hyperbolique

Il est facile de constater que lorsque les deux valeurs sont égales, cette fonction de coût est égale à zéro, et qu'elle tend vers l'infini plus la différence entre les valeurs est élevée. La dérivée $\frac{\partial L}{\partial \hat{y}} \in \mathbb{R}$ correspond à l'expression suivante:

$$\frac{\partial L}{\partial \hat{y}} = 2(\hat{y} - y). \tag{30}$$

Il est également possible de calculer une fonction de coût $L \in \mathbb{R}^+$ pour un vecteur cible $\mathbf{y} \in \mathbb{R}^J$ et une prédiction $\hat{\mathbf{y}} \in \mathbb{R}^J$:

$$L = \|\hat{\mathbf{y}} - \mathbf{y}\|_2^2 = \sum_{j=1}^{J} (\hat{y}_j - y_j)^2 \tag{31}$$

Le gradient $\frac{\partial L}{\partial \hat{\mathbf{y}}} \in \mathbb{R}^J$ correspond à l'expression suivante:

$$\frac{\partial L}{\partial \hat{\mathbf{y}}} = 2(\hat{\mathbf{y}} - \mathbf{y}). \tag{32}$$

15.3.2 Entropie croisée

Cette fonction de coût est habituellement utilisée lorsqu'on tente de résoudre un problème de **classification**. On considère que nous avons généralement N classes, et donc que le vecteur cible $\mathbf{y} \in [0, 1]^N$ aura tous ses éléments à 0 à l'exception d'un élément qui aura une valeur de 1 et correspondra à la classe active. La prédiction sera donc un vecteur $\hat{\mathbf{y}} \in [0, 1]^N$, et la fonction de coût correspond à:

$$L = -\sum_{j=1}^{J} y_j \log \hat{y}_j. \tag{33}$$

Notez que dans le cas spécifique d'une classification binaire (deux classes), nous pouvons simplifier le vecteur qui aurait normalement deux dimensions par un scalaire, car nous assumons que la somme des deux éléments est de 1. Dans ce cas précis, la fonction de coût correspond à:

$$L = -y \log \hat{y} - (1 - y) \log(1 - \hat{y}). \tag{34}$$

Le gradient pour l'entropie croisée correspond à:

$$\frac{\partial L}{\partial \hat{\mathbf{y}}} = -\mathbf{y} \oslash \hat{\mathbf{y}}. \tag{35}$$

L'opérateur $\oslash$ signifie qu'on divise les vecteurs élément par élément. Dans le cas de l'entropie croisée binaire, le gradient est en fait une dérivée qui correspond à:

$$\frac{\partial L}{\partial \hat{y}} = -\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}} \tag{36}$$

15.3.3 Divergence de Kullback-Leibler

Cette fonction de coût permet de comparer deux distributions probabilistes, représentées par les vecteurs de cible et de prédiction $\mathbf{y}$ et $\hat{\mathbf{y}}$. On assume que nous avons des fonctions de masse normalisées de sorte que $\sum_{j=1}^{J} y_j = 1$ et $\sum_{j=1}^{J} \hat{y}_j = 1$. La fonction de coût correspond donc à l'expression suivante:

$$L = \sum_{j=1}^{J} y_j \log\left(\frac{y_j}{\hat{y}_j}\right). \tag{37}$$

Remarquez que cette fonction de coût converge vers une valeur nulle lorsque les deux distributions sont identiques $y_j = \hat{y}_j \ \forall j \in \{1, 2, \ldots, J\}$. Le gradient correspond à l'expression suivante:

$$\frac{\partial L}{\partial \hat{\mathbf{y}}} = -\mathbf{y} \oslash \hat{\mathbf{y}} \tag{38}$$

Il est intéressant de constater que le gradient est identique à celui de l'entropie croisée. C'est le cas car en fait les deux fonctions de coûts différent seulement par une valeur constante:

$$\sum_{j=1}^{J} y_j \log\left(\frac{y_j}{\hat{y}_j}\right) = -\sum_{j=1}^{J} y_j \log \hat{y}_j + \sum_{j=1}^{J} y_j \log y_j = -\sum_{j=1}^{J} y_j \log \hat{y}_j + C_0 \tag{39}$$

15.4 Optimisation par descente de gradient

L'optimisation par descente de gradient est une méthode qui permet de trouver un minimum local pour une fonction différentiable. Il s'agit de sélectionner un point de départ aléatoire, calculer le gradient à cet endroit, et ensuite calculer un nouveau point en soustrayant le gradient du point initial. En appliquant cette opération itérativement, on finit par atteindre un point qui se situe dans un minimum local.

Il est possible d'illustrer ce concept avec une analogie toute simple: imaginez que vous êtes perdu en montagne et qu'il y a un brouillard assez intense qui vous empêche de voir à plus de 5-10 mètres devant vous. Vous voulez rejoindre le village qui se situe dans la vallée. Puisque vous n'avez pas de vue d'ensemble du relief, la solution la plus efficace consiste à vous diriger dans la direction où le terrain possède la plus grande pente descendante, et corriger votre trajectoire au fur et à mesure en suivant cette pente, en espérant qu'éventuellement vous atteindrez de cette façon la vallée.

D'un point de vue mathématique, l'optimisation par descente de gradient se définit ainsi:

$$\mathbf{v}_{t+1} = \mathbf{v}_t - \mu \frac{\partial L}{\partial \mathbf{v}}\Big|_{\mathbf{v}_t}. \tag{40}$$

Le vecteur $\mathbf{v}_t$ contient les paramètres à l'itération t, le gradient évalué au point $\mathbf{v}_t$ est représenté par l'expression $\frac{\partial L}{\partial \mathbf{v}}\Big|_{\mathbf{v}_t}$, et le point suivant $\mathbf{v}_{t+1}$ est mis à jour à l'aide du pas μ. Prenons un exemple simple pour bien comprendre l'algorithme. Supposons que nous cherchons à minimiser la fonction de coût suivante:

$$L(v) = \frac{1}{10}v^4 - \frac{1}{2}v^3 - 5v^2 + 50v \tag{41}$$

Dans cet exemple, il n'y a qu'un seul élément dans le vecteur $\mathbf{v}$, c'est-à-dire v, et donc le gradient $\frac{\partial L}{\partial \mathbf{v}}$ est en fait la dérivée $\frac{\partial L}{\partial v}$. Nous évaluons donc l'expression suivante:

$$\frac{\partial L(v)}{\partial v} = \frac{4}{10}v^3 - \frac{3}{2}v^2 - 10v + 50. \tag{42}$$

Supposons que nous choisissons le point de départ à $v_0 = 8$ et $L(v_0) = 233.6$, et que nous avons un pas de $\mu = 0.04$.

$$
\begin{aligned}
v_1 &= v_0 - \mu \frac{\partial L}{\partial v}\Big|_{v_0} = +8.00 - 0.04 \cdot +78.8 = +4.85 & v_{11} &= v_{10} - \mu \frac{\partial L}{\partial v}\Big|_{v_{10}} = -4.82 - 0.04 \cdot +18.5 = -5.56 \\
v_2 &= v_1 - \mu \frac{\partial L}{\partial v}\Big|_{v_1} = +4.85 - 0.04 \cdot +11.8 = +4.37 & v_{12} &= v_{11} - \mu \frac{\partial L}{\partial v}\Big|_{v_{11}} = -5.56 - 0.04 \cdot -9.6 = -5.18 \\
v_3 &= v_2 - \mu \frac{\partial L}{\partial v}\Big|_{v_2} = +4.37 - 0.04 \cdot +11.0 = +3.93 & v_{13} &= v_{12} - \mu \frac{\partial L}{\partial v}\Big|_{v_{12}} = -5.18 - 0.04 \cdot +6.1 = -5.42 \\
v_4 &= v_3 - \mu \frac{\partial L}{\partial v}\Big|_{v_3} = +3.93 - 0.04 \cdot +11.8 = +3.46 & v_{14} &= v_{13} - \mu \frac{\partial L}{\partial v}\Big|_{v_{13}} = -5.42 - 0.04 \cdot -3.5 = -5.28 \\
v_5 &= v_4 - \mu \frac{\partial L}{\partial v}\Big|_{v_4} = +3.46 - 0.04 \cdot +14.0 = +2.90 & v_{15} &= v_{14} - \mu \frac{\partial L}{\partial v}\Big|_{v_{14}} = -5.28 - 0.04 \cdot +2.2 = -5.37 \\
v_6 &= v_5 - \mu \frac{\partial L}{\partial v}\Big|_{v_5} = +2.90 - 0.04 \cdot +18.1 = +2.18 & v_{16} &= v_{15} - \mu \frac{\partial L}{\partial v}\Big|_{v_{15}} = -5.37 - 0.04 \cdot -1.3 = -5.31 \\
v_7 &= v_6 - \mu \frac{\partial L}{\partial v}\Big|_{v_6} = +2.18 - 0.04 \cdot +25.3 = +1.16 & v_{17} &= v_{16} - \mu \frac{\partial L}{\partial v}\Big|_{v_{16}} = -5.31 - 0.04 \cdot +0.8 = -5.34 \\
v_8 &= v_7 - \mu \frac{\partial L}{\partial v}\Big|_{v_7} = +1.16 - 0.04 \cdot +37.0 = -0.31 & v_{18} &= v_{17} - \mu \frac{\partial L}{\partial v}\Big|_{v_{17}} = -5.34 - 0.04 \cdot -0.5 = -5.33 \\
v_9 &= v_8 - \mu \frac{\partial L}{\partial v}\Big|_{v_8} = -0.31 - 0.04 \cdot +53.0 = -2.43 & v_{19} &= v_{18} - \mu \frac{\partial L}{\partial v}\Big|_{v_{18}} = -5.33 - 0.04 \cdot +0.3 = -5.34 \\
v_{10} &= v_9 - \mu \frac{\partial L}{\partial v}\Big|_{v_9} = -2.43 - 0.04 \cdot +59.7 = -4.82 & v_{20} &= v_{19} - \mu \frac{\partial L}{\partial v}\Big|_{v_{19}} = -5.34 - 0.04 \cdot -0.2 = -5.33
\end{aligned}
\tag{43}
$$

Voici le graphique de l'erreur $L(v)$ en fonction de v, et les différents points sélectionnés durant l'optimisation:

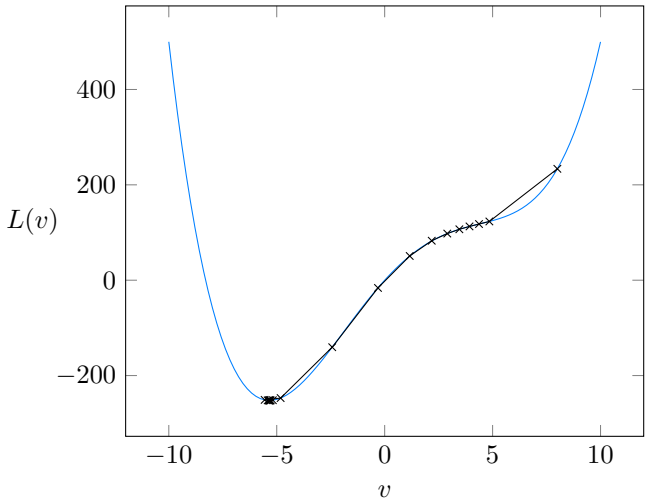


Figure 10: Erreur en fonction de v et descente de gradient

On voit clairement que l'approche par descente de gradient permet de trouver un minimum local (qui par chance est également un minimum global dans cet exemple) itérativement.

Dans un cas d'apprentissage profond supervisé, la fonction de coût correspond à l'équation suivante pour l'ensemble d'entraînement $\mathbb{X} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$:

$$L(\mathbb{X}) = \frac{1}{N} \sum_i^N L(f(\mathbf{x}_i), y_i) \tag{44}$$

Le vecteur $\mathbf{x}_i$ contient les caractéristiques de la donnée i de l'ensemble d'entraînement $\mathbb{X}$. La fonction L correspond à la fonction de coût entre la valeur estimée et la cible. En règle générale, L correspond à l'erreur quadratique moyenne dans un cas de régression et à l'entropie croisée dans un cas de classification. La fonction $f(\mathbf{x})$ représente le réseau de neurones.

La descente de gradient d'un réseau de neurones contenant K paramètres $\mathbf{w}^{(j)}$ peut être définie comme suit :

$$\mathbf{w}_{t+1}^{(j)} = \mathbf{w}_{t+1}^{(j)} - \mu \frac{\partial L(\mathbb{X})}{\partial \mathbf{w}^{(j)}}\bigg|_{\mathbf{w}_t^{(j)}} \tag{45}$$

Ce calcul doit être effectué pour tous les paramètres $\mathbf{w}^{(j)}$ du réseau de neurones. Ainsi, il est possible de remarquer que la descente de gradient peut demander beaucoup de calculs si les constantes N et K sont élevées. Pour atténuer l'effet de K, le calcul de $\frac{\partial L(\mathbb{X})}{\partial \mathbf{w}^{(j)}}\Big|_{\mathbf{w}_t^{(j)}}$ est fait efficacement à l'aide de rétropropagation de l'erreur. Avec la descente de gradient, il est nécessaire de calculer $L(f(\mathbf{x}_i), y_i)$ pour toutes les valeurs de i pour faire une seulement mise à jour des paramètres $\mathbf{w}^{(j)}$. Ceci a longtemps été un problème ralentissant l'entraînement des réseaux de neurones.

Il n'est cependant pas nécessaire de calculer $\frac{\partial L(\mathbb{X})}{\partial \mathbf{w}^{(j)}}\Big|_{\mathbf{w}_t^{(j)}}$ car il est possible de l'approximer par $\frac{\partial L(\mathbb{B}_t)}{\partial \mathbf{w}^{(j)}}\Big|_{\mathbf{w}_t^{(j)}}$. L'ensemble $\mathbb{B}_t$ contient M données choisies aléatoirement dans l'ensemble d'entraînement $\mathbb{X}$. Le contenu de l'ensemble $\mathbb{B}_t = \{(\mathbf{x}_i, y_i)\}_{i=1}^{M}$ change à chaque mise à jour de paramètres $\mathbf{w}^{(j)}$. Dans ce cas, la fonction de coût et la descente de gradient deviennent :

$$L(\mathbb{B}_t) = \frac{1}{M}\sum_{i}^{M} L(f(\mathbf{x}_i), y_i) \tag{46}$$

$$\mathbf{w}_{t+1}^{(j)} = \mathbf{w}_{t+1}^{(j)} - \mu \frac{\partial L(\mathbb{B})}{\partial \mathbf{w}^{(j)}}\bigg|_{\mathbf{w}_t^{(j)}} \tag{47}$$

Intuitivement, $L(\mathbb{X})$ peut être vue comme le calcul de la moyenne de $L(f(\mathbf{x}_i), y_i)$ sur l'ensemble d'entraînement $\mathbb{X}$. Par conséquent, $L(\mathbb{B}_t)$ est l'approximation de cette moyenne en utilisant M données. Ainsi, il faut juste calculer M fois au lieu de N fois la fonction de coût $L(f(\mathbf{x}_i), y_i)$ pour effectuer une mise à jour des paramètres $\mathbf{w}^{(j)}$. Ceci permet d'accélérer grandement l'entraînement des réseaux de neurones.

Si $M = 1$, l'algorithme se nomme descente de gradient stochastique. Si $1 < M < N$, l'algorithme se nomme descente de gradient stochastique avec mini-lots. Sinon, c'est la descente de gradient. Une valeur de M trop petite ne permet pas d'estimer avec précision $L(\mathbb{X})$, donc ceci nuit à la convergence de l'algorithme. Plus M est élevé plus l'estimation de $L(\mathbb{X})$ est bonne, mais lorsque M est trop grand et qu'une bibliothèque logicielle d'apprentissage profond est utilisée, il est possible de manquer de mémoire vive, car le calcul de $L(\mathbb{B}_t)$ se fait en parallèle. De façon générale, la valeur de M est entre 32 et 256.

15.5 Couches

Il existe de nombreuses familles de couches de neurones. Nous allons analyser en détails les couches qui seront nécessaires pour réaliser la problématique. En général pour une couche de neurones, on parle de la propagation vers l'avant quand on calcule la prédiction en fonction des entrées. Lorsqu'on fait l'entraînement du réseau, on parle de propagation vers l'arrière quand on calcule le gradient de l'entrée en fonction du gradient de la sortie.

Voici un exemple avec quatre couches (C_1, C_2, C_3 et C_4) qui génèrent une sortie $\hat{\mathbf{y}}$ à partir de l'entrée $\mathbf{x}$. Nous définissons également les variables intermédiaires $\mathbf{h}_1$, $\mathbf{h}_2$ et $\mathbf{h}_3$. Finalement, la fonction de coût $\mathcal{C}$ génère l'erreur L.

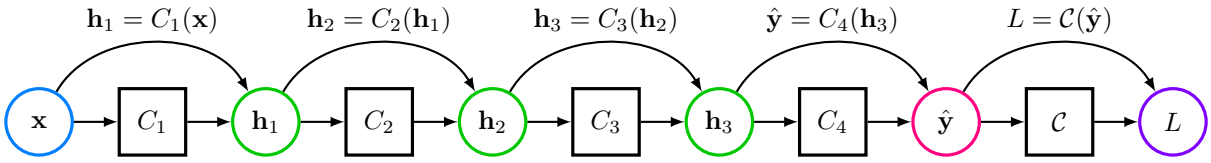


Figure 11: Propagation vers l'avant

Lorsqu'on effectue la propagation vers l'arrière du gradient, il est possible de calculer le gradient pour les paramètres de chaque couche de neurones ($\frac{\partial L}{\partial \boldsymbol{\theta}_1}$, $\frac{\partial L}{\partial \boldsymbol{\theta}_2}$, $\frac{\partial L}{\partial \boldsymbol{\theta}_3}$ et $\frac{\partial L}{\partial \boldsymbol{\theta}_4}$), et d'utiliser ensuite ces gradients pour faire la mise à jour des paramètres grâce à une méthode d'optimisation de descente de gradient.

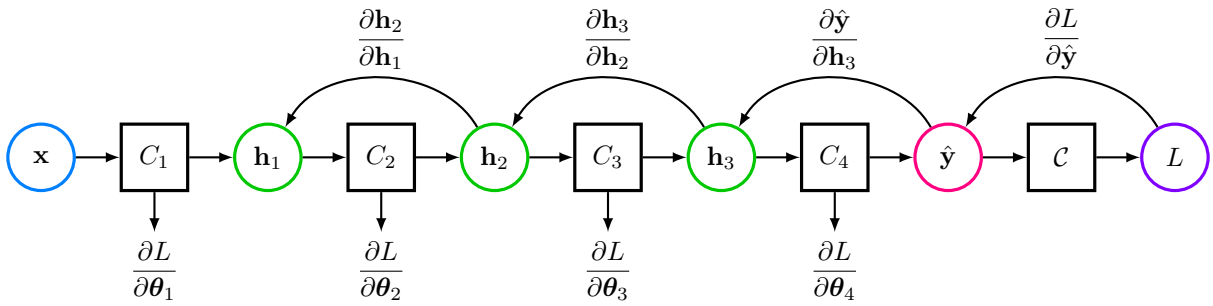


Figure 12: Propagation vers l'arrière et gradient de chaque paramètre

Par exemple, si on désire obtenir le gradient des paramètres $\boldsymbol{\theta}_3$ de la couche C_3, on doit calculer le gradient de l'erreur L en fonction de la sortie $\hat{\mathbf{y}}$, c'est-à-dire $\frac{\partial L}{\partial \hat{\mathbf{y}}}$. On obtient ensuite le gradient de l'erreur L en fonction du vecteur intermédiaire $\hat{\mathbf{h}}_3$. Finalement, puisqu'on connait le lien entre le gradient en sortie de la couche C_3 et le gradient des paramètres (qu'on appelle $\hat{C}_3$), on peut calculer $\frac{\partial L}{\partial \hat{\boldsymbol{\theta}}_3}$. On applique ensuite une descente de gradient.

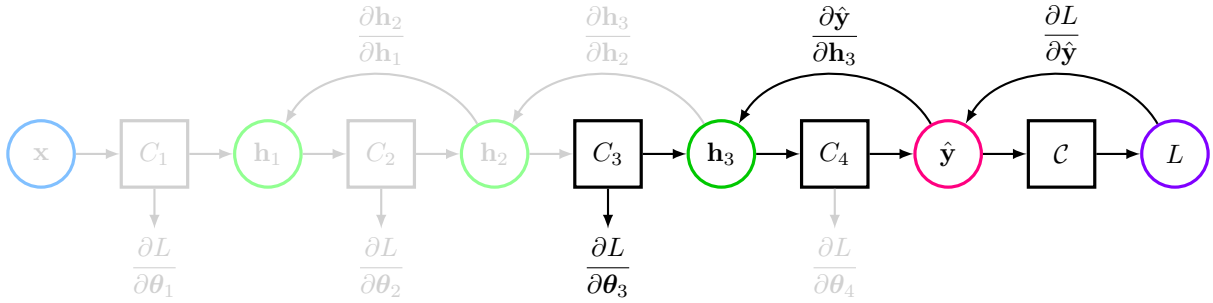


Figure 13: Propagation vers l'arrière et gradient pour le paramètre $\boldsymbol{\theta}_3$

$$\frac{\partial L}{\partial \boldsymbol{\theta}_3} = \hat{C}_3 \left(\frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{h}_3} \frac{\partial L}{\partial \hat{\mathbf{y}}} \right) \tag{48}$$

$$\boldsymbol{\theta}_3 \leftarrow \boldsymbol{\theta}_3 - \mu \frac{\partial L}{\partial \boldsymbol{\theta}_3} \tag{49}$$

15.5.1 Linéaire

La couche linéaire est également souvent nommée la couche entièrement connectée. Dans cette architecture illustré à la figure 14, il y a I entrées qui sont connectées à J sorties, pour un total de IJ paramètres. Il est également possible d'ajouter un biais de J paramètres supplémentaires. Les paramètres peuvent être représentés sous forme matricielle, avec l'expression $\mathbf{W} \in \mathbb{R}^{J \times I}$ qui représente les poids qui relient le vecteur d'entrée $\mathbf{x} \in \mathbb{R}^{I \times 1}$ au vecteur de sortie $\mathbf{y} \in \mathbb{R}^{J \times 1}$, et l'expression $\mathbf{b} \in \mathbb{R}^{J \times 1}$ qui contient les poids pour le biais.

$$y_j = \sum_{i=1}^{I} w_{j,i} x_i + b_j, \ \mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b} \tag{50}$$

Ensuite, si on considère que la sortie de la couche linéaire est connectée à une fonction de coût $\mathcal{C}$ et génère l'erreur L, on peut constater avec la démonstration à la figure 15 que le gradient $\frac{\partial L}{\partial \mathbf{y}} \in \mathbb{R}^{J \times 1}$ peut se propager dans le réseau et être exprimé par rapport à l'entrée, sous la forme $\frac{\partial L}{\partial \mathbf{x}} \in \mathbb{R}^{I \times 1}$. Il s'agit d'une propriété importante, car il est ainsi possible de propager le gradient d'une couche à l'autre lorsque plusieurs couches sont connectées en cascade.

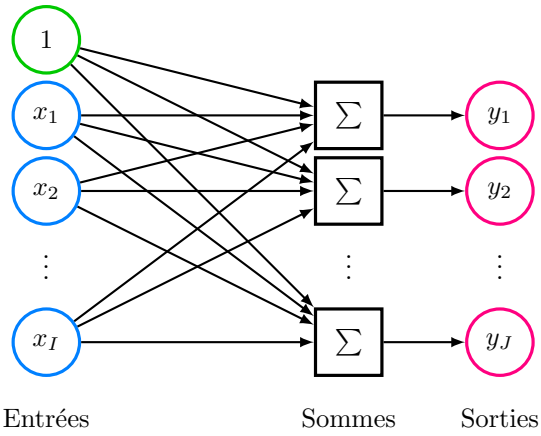


Figure 14: Architecture d'une couche linéaire

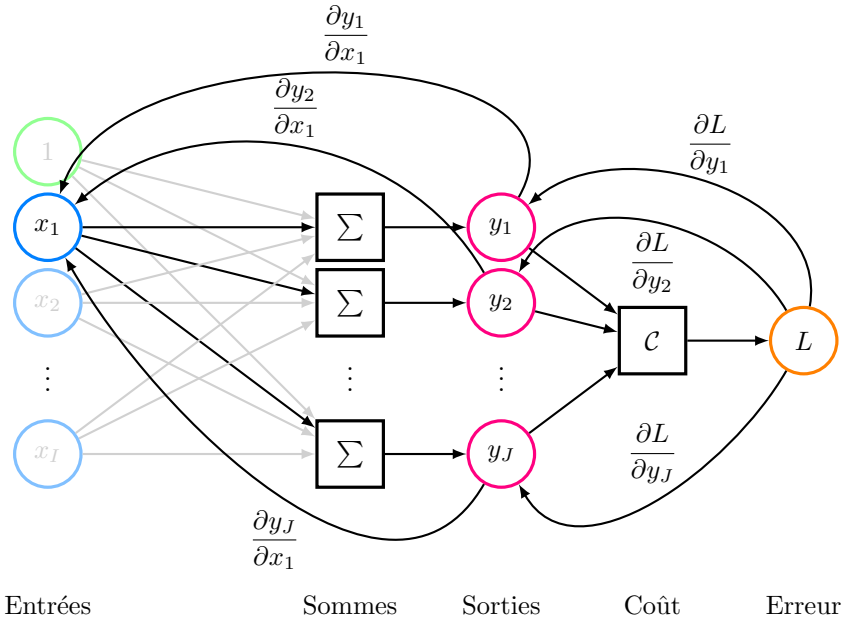


Figure 15: Rétropropagation du gradient pour une couche linéaire

On peut calculer la propagation du gradient élément par élément, mais il est également possible de la convertir sous forme matricielle:

$$\frac{\partial L}{\partial x_i} = \sum_{j=1}^{J} \frac{\partial y_j}{\partial x_i} \frac{\partial L}{\partial y_j} = \sum_{j=1}^{J} w_{j,i} \frac{\partial L}{\partial y_j}, \quad \frac{\partial L}{\partial \mathbf{x}} = \mathbf{W}^T \frac{\partial L}{\partial \mathbf{y}}. \tag{51}$$

Il est ensuite possible de calculer les gradients pour les paramètres $\mathbf{W}$ et $\mathbf{b}$.

$$\frac{\partial L}{\partial w_{j,i}} = x_i \frac{\partial L}{\partial y_j}, \quad \frac{\partial L}{\partial \mathbf{W}} = \frac{\partial L}{\partial \mathbf{y}} \mathbf{x}^T \tag{52}$$

$$\frac{\partial L}{\partial b_j} = \frac{\partial L}{\partial y_j}, \quad \frac{\partial L}{\partial \mathbf{b}} = \frac{\partial L}{\partial \mathbf{y}} \tag{53}$$

Les valeurs des paramètres doivent être initialisés aléatoirement, tout en demeurant dans un interval approprié pour assurer une stabilité numérique. Pour ce faire, on utilise habituellement la méthode de Xavier, qui attribue des valeurs aléatoires à chaque paramètre en suivant une distribution gaussienne avec une moyenne nulle, et une variance qui dépend du nombre d'entrées (I) et de sorties (J):

$$w_{j,i} \sim \mathcal{N}(0, \frac{2}{I+J}) \tag{54}$$

$$b_j \sim \mathcal{N}(0, \frac{2}{J}) \tag{55}$$

15.5.2 Normalisation de lot

Une couche de normalisation vise à normaliser la distribution statistique pour accélérer l'apprentissage dans un réseau de neurones. La figure 16 montre l'effet de cette normalisation sur la distribution statistique. De manière générale, ce type de couche est placé avant les fonctions d'activation. La couche accepte un vecteur $\mathbf{x} \in \mathbb{R}^N$ et retourne un vecteur de même dimension $\mathbf{y} \in \mathbb{R}^N$.

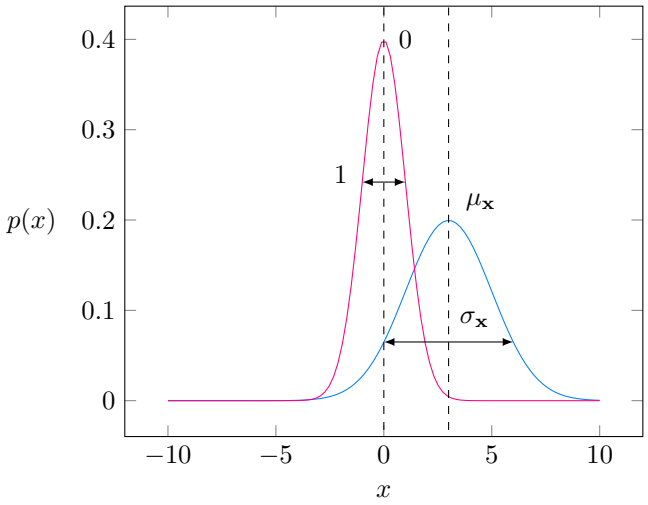


Figure 16: Normalisation de la moyenne et de la variance

À l'entraînement, la normalisation s'effectue à l'aide des propriétés statistiques du lot. La moyenne et la variance sont calculées à l'aide des équations suivantes. L'expression $(\dots)^{\circ 2}$ signifie que chaque terme du vecteur est portée à la puissance au carré.

$$\boldsymbol{\mu}_B = \frac{1}{M} = \sum_{i=1}^{M} \mathbf{x}_i \tag{56}$$

$$\boldsymbol{\sigma}_B^2 = \frac{1}{M} \sum_{i=1}^{M} (\mathbf{x}_i - \boldsymbol{\mu}_B)^{\circ 2} \tag{57}$$

La prochaine étape est de normaliser $\mathbf{x}$ à l'aide de l'équation suivante.

$$\hat{\mathbf{x}} = (\mathbf{x} - \boldsymbol{\mu}_B) \oslash \sqrt{(\boldsymbol{\sigma}_B^2 + \epsilon)} \tag{58}$$

Il est possible que le réseau préfère ajuster plus finement la normalisation selon l'impact sur la fonction de coût. Pour cette raison, les vecteurs $\boldsymbol{\gamma}$ et $\boldsymbol{\beta}$ peuvent être appris par descente de gradient pour ajuster la normalisation.

$$\mathbf{y} = \boldsymbol{\gamma} \circ \hat{\mathbf{x}} + \boldsymbol{\beta} \tag{59}$$

Lors de l'étape d'inférence, cette couche utilise la moyenne et la variance de tout l'ensemble de données d'entraînement. Ces statistiques sont donc sont calculées à l'aide d'un filtre passe-bas lors de l'entraînement qui s'applique aux moyennes et variances obtenues pour chaque lot.

$$\boldsymbol{\mu}|_t = (1-\alpha)\boldsymbol{\mu}|_{t-1} + \alpha\boldsymbol{\mu}_B \tag{60}$$

$$\boldsymbol{\sigma}^2|_t = (1-\alpha)\boldsymbol{\sigma}^2|_{t-1} + \alpha\boldsymbol{\sigma}_B^2 \tag{61}$$

Au moment de l'inférence, la couche calcule sa sortie comme suit:

$$\hat{\mathbf{x}} = (\mathbf{x} - \boldsymbol{\mu}|_T) \oslash \sqrt{\boldsymbol{\sigma}^2|_T + \epsilon} \tag{62}$$

$$\mathbf{y} = \boldsymbol{\gamma} \circ \hat{\mathbf{x}} + \boldsymbol{\beta} \tag{63}$$

Les gradients pour l'entraînement sont les suivants.

$$\frac{\partial L}{\partial \hat{\mathbf{x}}_i} = \frac{\partial L}{\partial \mathbf{y}_i} \circ \boldsymbol{\gamma} \tag{64}$$

$$\frac{\partial L}{\partial \boldsymbol{\sigma}_B^2} = \sum_{i=1}^{M} \frac{\partial L}{\partial \hat{\mathbf{x}}_i} \circ (\mathbf{x}_i - \boldsymbol{\mu}_B) \circ -\frac{1}{2}(\boldsymbol{\sigma}_B^2 + \epsilon)^{\circ -3/2} \tag{65}$$

$$\frac{\partial L}{\partial \boldsymbol{\mu}_B} = \left(-\sum_{i=1}^{M} \frac{\partial L}{\partial \hat{\mathbf{x}}_i} \oslash \sqrt{(\boldsymbol{\sigma}_B^2 + \epsilon)} \right) + \frac{-2}{M} \frac{\partial L}{\partial \boldsymbol{\sigma}_B^2} \circ \sum_{i=1}^{M} (\mathbf{x}_i - \boldsymbol{\mu}_B) \tag{66}$$

$$\frac{\partial L}{\partial \mathbf{x}_i} = \frac{\partial L}{\partial \hat{\mathbf{x}}_i} \oslash \sqrt{(\boldsymbol{\sigma}_B^2 + \epsilon)} + \frac{2}{M} \frac{\partial L}{\partial \boldsymbol{\sigma}_B^2} (\mathbf{x}_i - \boldsymbol{\mu}_B) + \frac{1}{M} \frac{\partial L}{\partial \boldsymbol{\mu}_B} \tag{67}$$

$$\frac{\partial L}{\partial \boldsymbol{\gamma}} = \sum_{i=1}^{M} \frac{\partial L}{\partial \mathbf{y}_i} \circ \frac{\partial L}{\partial \hat{\mathbf{x}}_i} \tag{68}$$

$$\frac{\partial L}{\partial \boldsymbol{\beta}} = \sum_{i=1}^{M} \frac{\partial L}{\partial \mathbf{y}_i} \tag{69}$$

Notez qu'au début de l'entraînement le paramètre $\boldsymbol{\gamma}$ est initialisé avec le vecteur $\mathbf{1}$ (on met tous les éléments à 1) et le paramètre $\boldsymbol{\beta}$ avec le vecteur $\mathbf{0}$ (tous les éléments sont zéro).

15.5.3 Softmax

Une couche de softmax permet de générer un vecteur qui s'approche d'un vecteur 1 parmi n. Dans ce type de vecteur, une seule entrée correspond à une valeur de 1, et les autres éléments sont à 0. Une couche de softmax accepte un vecteur $\mathbf{x} \in \mathbb{R}^N$ et retourn un vecteur $\mathbf{y} \in [0,1]^N$. L'idée est de calculer la valeur exponentielle de chaque élément, et normaliser par la sommes des valeurs exponentielles. Un élément qui possède une valeur plus grande que celles des autres éléments tendera vers une valeur de 1 tandis que les autres éléments tenderont vers 0. La couche softmax se calcule comme ceci:

$$y_j = \frac{\exp(x_j)}{\sum_{i=1}^{I} \exp(x_i)}, \ \mathbf{y} = \frac{\exp(\mathbf{x})}{\|\exp(\mathbf{x})\|_1} \tag{70}$$

Il est possible de calculer les gradients sous la forme suivante:

$$\frac{\partial L}{\partial x_i} = \sum_{j=1}^{J} \frac{\partial y_j}{\partial x_i} \frac{\partial L}{\partial y_j} = \sum_{j=1}^{J} d_{i,j} \frac{\partial L}{\partial y_j}, \ \frac{\partial L}{\partial \mathbf{x}} = \mathbf{D} \frac{\partial L}{\partial \mathbf{y}} \tag{71}$$

$$d_{i,j} = \begin{cases} -y_i y_j & i \neq j \\ y_j(1-y_j) & i = j \end{cases}, \ \mathbf{D} = \begin{bmatrix} d_{1,1} & d_{1,2} & \dots & d_{1,J} \\ d_{2,1} & d_{2,2} & \dots & d_{2,J} \\ \vdots & \vdots & \ddots & \vdots \\ d_{I,1} & d_{I,2} & \dots & d_{I,J} \end{bmatrix} \tag{72}$$

15.6 Vérifications diligentes

Lors de l'implémentation d'un réseau de neurones, il est parfois difficile de vérifier si le code fonctionne correctement. Les vérifications diligentes suivantes permettent de valider l'implémentation.

15.6.1 Vérification du calcul du gradient des fonctions mathématiques

La valeur de la différence finie centrée est approximativement égale au calcul analytique du gradient pour des petites valeurs de h. Cette vérification diligente permet de vérifier que le gradient est calculé correctement. Si la fonction utilise en entrée un vecteur et retourne un scalaire, la différence finie centrée d'une fonction $f(\mathbf{x})$ pour la dimension i du vecteur x est définie par l'équation suivante.

$$\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}_i}\Big|_{\mathbf{x}} \approx \frac{f(\mathbf{x} + \mathbf{1_i}h) - f(\mathbf{x} - \mathbf{1_i}h))}{2h} \tag{73}$$

Le vecteur $\mathbf{1_i}$ contient la valeur 0 pour toutes les dimensions, sauf la dimension i qui contient la valeur 1. Si la fonction reçoit en entrée un vecteur et retourne un vecteur, la différence finie centrée d'une fonction $\mathbf{f}(\mathbf{x})$ pour la dimension i du vecteur x est définie par l'équation suivante.

$$\frac{\partial \mathbf{f}(\mathbf{x})}{\partial \mathbf{x}_i}\Big|_{\mathbf{x}} \approx \sum_j \left[\frac{\mathbf{f}(\mathbf{x} + \mathbf{1_i}h) - \mathbf{f}(\mathbf{x} - \mathbf{1_i}h))}{2h}\right]_j \tag{74}$$

La notation $[...]_j$ indique la dimension j du vecteur résultant de l'opération entre les crochets.

15.6.2 Vérification de la valeur de la fonction de coût dans le cas de classification

Après une initialisation aléatoire des paramètres du réseau, il faut vérifier que la valeur de l'entropie croisée est approximativement égale à $-ln(\frac{1}{N})$ où N est le nombre de classes. Une erreur dans l'implémentation des couches ou de la fonction de coût pourrait être la cause d'un écart important de la valeur de l'entropie croisée.

15.6.3 Surapprentissage sur un petit lot de données

Lorsque l'implémentation est valide et que les paramètres d'apprentissage sont bien choisis, il est possible de faire du surapprentissage sur un petit lot de données. Si ce n'est pas le cas, ceci indique qu'il y a possiblement une erreur dans la structure du réseau et/ou dans l'interprétation des données d'entrées ou de sorties.

15.6.4 Visualisation des courbes d'apprentissage

En visualisant les courbes d'apprentissage, il est possible de savoir si le bon taux d'apprentissage est choisi. La figure 17 montre l'effet du taux d'apprentissage au long de l'entraînement.

- Un taux d'apprentissage trop élevé diverge, car les sauts effectués par la descente de gradient sont trop grands.
- Un taux d'apprentissage bas permet de converger vers la bonne solution, mais très lentement.
- Un taux d'apprentissage élevé se rapproche de la bonne solution, mais conserve un biais parce que les sauts effectués par la descente de gradient sont grands, donc l'algorithme tourne autour de cette solution.
- Un bon taux d'apprentissage permet de converger rapidement vers la bonne solution.

Une des manières pour s'assurer de converger rapidement vers une bonne solution sans trouver le meilleur taux d'apprentissage est d'utiliser un taux d'apprentissage légèrement élevé et de le faire diminuer au fil des époques pour permettre de converger vers la bonne solution.

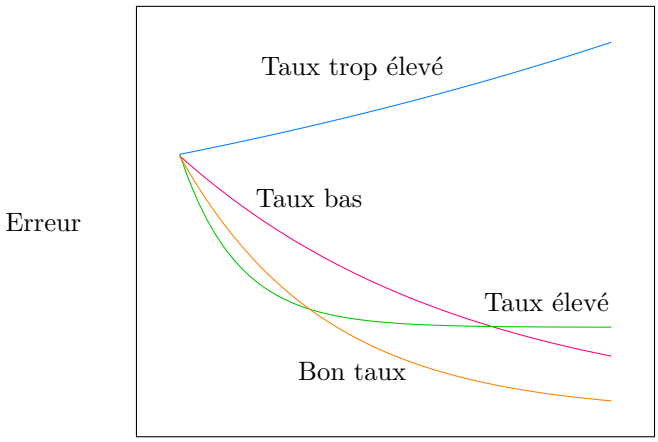


Figure 17: Effet du taux d'apprentissage au long de l'entraînement