Compte Rendu

IA et Optimisation

TD

MATHIEU Elyes

–

5A Polytech Dijon

Patrick Bard

2024–2025

Contents

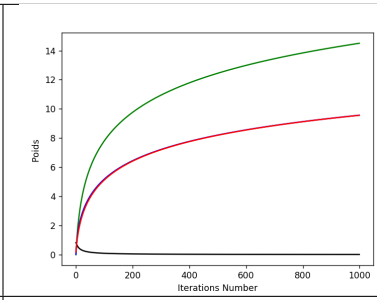
1 TD1

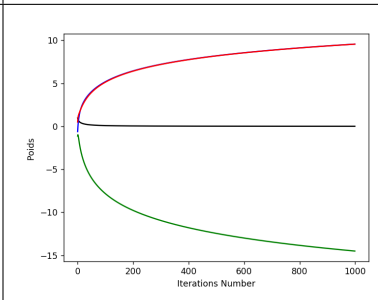
1.1 Introduction

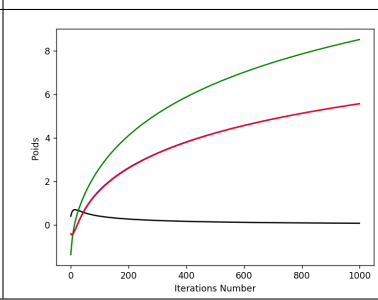
Dans ce TD le but est d'analyser et expliquer le fonctionnement d'un algorithme d'apprentissage aux travers de la modélisation des fonctions logique AND, OR et XOR. Nous allons dans un premier temps exécuter le programme en modifiant les hyper-paramètres, biais et coefficient, pour constater les implications sur les résultats. Puis dans un deuxième temps nous allons analyser les résultats.

1.2 Test des hyperparamètres

Dans les graphes suivant la courbe noire correspond à l'erreur, les autres aux poids synaptiques.

Biais: -1
Coef: 0.7
Erreur finale: 0.01
Poids finaux:
`w0 : 14.50224436178552`
`w1 : 9.565784993015141`
`w2 : 9.559576362470171`
	

Biais: 1
Coef: 0.7
Erreur finale: 0.01
Poids finaux:
`w0 : -14.500647357026958`
`w1 : 9.56472550959522`
`w2 : 9.558513578583344`
	

Biais: -1
Coef: 0.1
Erreur finale: 0.07
Poids finaux:
`w0 : 8.512627685540009`
`w1 : 5.566004777357574`
`w2 : 5.5598045089643255`
	

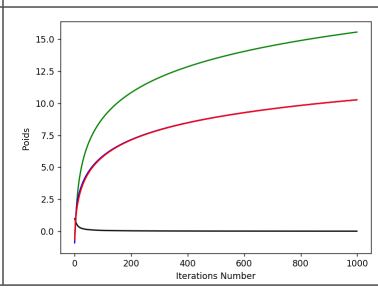
Biais: -1
Coef: 0.99
Erreur finale: 0.007
Poids finaux:
`w0 : -14.500905359803008`
`w1 : 9.564896673685686`
`w2 : 9.55868527537008`
	

Table 1: Test des hyperparamètres

C'est cohérent car comme visible dans le programme:

```
y = bias * weights[0] + inputValue[j][0]
* weights [1] + inputValue[j][1] * weights [2];
```

Avec les deux premiers graphe on comprend que le biais est multiplié seulement avec le premier poids ainsi il n'y a que celui la qui devient négatif pour compenser le biais, cela est cohérent avec le code du perceptron.

On peut aussi remarquer que la valeur du coefficient d'apprentissage influence directement la stabilité de la descente de gradient : un coefficient trop élevé entraîne des oscillations ou une divergence, alors qu'un coefficient trop faible provoque une convergence extrêmement lente. Les résultats observés sont cohérents avec ce comportement.

Dans chaque cas on observe que les poids dans les premières itérations évoluent très vite et l'erreur est forte au fil des itérations les poids se stabilisent et convergent quant à l'erreur elle décroit pour approcher zéro.

1.3 OR et XOR

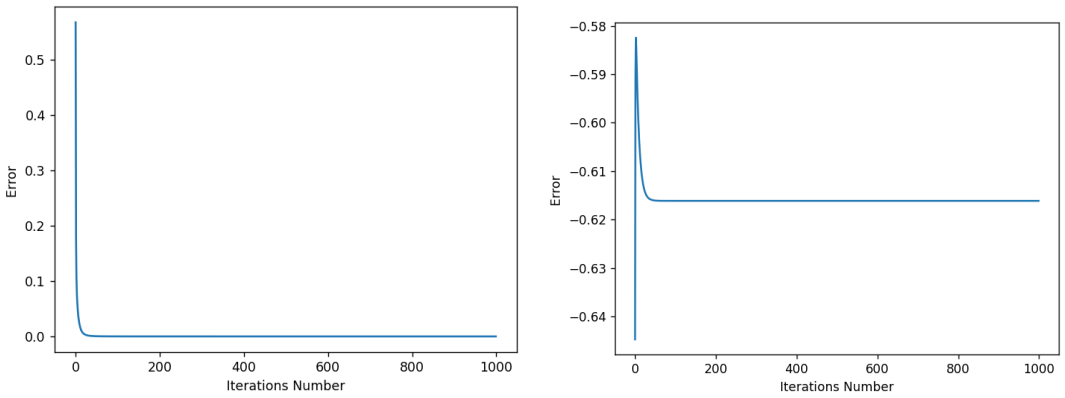
La partie du code responsable de la porte logique se trouve dans la déclaration de l'objectif:

```
desired_out = np. array ([0, 0, 0, 1])
```

Pour modifier le comportement du programme pour une porte OR ou XOR il suffit donc de modifier les valeurs comme tel:

- **OR** - desired_out = np. array ([0, 1, 1, 1])
- **XOR** - desired_out = np. array ([0, 1, 1, 0])

Et voici ce que cela donne.



(a) erreur du OR (b) Erreur du XOR

Figure 1: Erreur par itérations

Cette expérimentation met en évidence une limite fondamentale : le perceptron parvient à modéliser OR mais échoue systématiquement sur XOR. Cela illustre le fait que XOR est un problème non linéaire qui nécessite un réseau à plusieurs couches ou une transformation de l'espace d'entrée.

2 TD2

2.1 Introduction

GAN ou generative adversarial network fonctionne avec une entrée réel (exemple un cercle) et avec deux réseaux, un générateur, qui génère (logique) des sorties à partir d'un bruit aléatoire (exemple un cercle) considéré comme faux et un discriminateur qui essaye distinguer quelle entrée donnée est générée et laquelle est réelle (d'où adversarial network), et ceci en boucle avec le but que le discriminateur s'améliore pour distinguer le faux et le générateur qui essaye de s'améliorer pour le tromper. Dans un cas réel les entrée pour le discriminateur seraient des images de chat issu d'une base de données, et des images générées de chat.

Le but sera de comprendre le fonctionnement des réseaux antagonistes génératifs (GAN), d'expérimenter avec un modèle simple via **GAN Lab**, puis d'instancier et d'utiliser un modèle plus avancé, **StyleGAN3**, pour générer des images réalistes.

2.2 GAN Lab

Hyperparamètre	**Valeur testée**	**Observation**	**Commentaire**
Learning rate	0.001 / 0.01 / 0.1	Trop élevé, le modèle diverge ; Trop faible, convergence lente	Un compromis autour de 0.01 est souvent optimal
Batch size	1 / 10 / 100	Bas, oscillations fortes Grands stabilisent l'apprentissage	Batch moyen (∼32) offre un bon équilibre
Ratio d'entraînement G/D	1:1 / 1:2 / 2:1	Si D est trop entraîné, G n'apprend plus ; si G domine, D devient confus	Ratio équilibré essentiel (1:1 recommandé)
Bruit d'entrée	Uniforme / Gaussien	Les distributions influencent la couverture de l'espace latent	Distribution gaussienne plus stable visuellement
Architecture (nombre de couches)	1 / 2 / 3 couches	Plus de couches, meilleure précision et apprentissage plus long	Trop de couche, risque d'overfitting

Table 2: Analyse des hyperparamètres

L'expérience avec GAN Lab montre la fragilité de l'équilibre entre générateur et discriminateur. De petits changements d'hyper paramètres peuvent complètement changer la dynamique d'apprentissage.

2.3 StyleGan3

Pour StyleGan3 nous avons généré des imageds à l'aide du programme gen_images.py, ci-dessous les différentes générations.

(a) seed 2 (b) seed 42 (c) seed 123 (d) seed 999

Figure 2: Images généré par graines (seed)

(a) truncs 1 (b) truncs 0.5 (c) truncs 0.7 (d) truncs 2

Figure 3: Images généré par truncation (truncs)

```
('--seeds', type=parse_range, help='List of random seeds (e.g., \'0,1,4-6\')', required=True)
('--trunc', 'truncation_psi', type=float, help='Truncation psi', default=1, show_default=True)
('--class', 'class_idx', type=int, help='Class label (unconditional if not specified)')
('--noise-mode', help='Noise mode', type=click.Choice(['const', 'random', 'none'])
('--translate', help='Translate XY-coordinate (e.g. \'0.3,1\')', type=parse_vec2, default='0,0')
('--rotate', help='Rotation angle in degrees', type=float, default=0)
('--outdir', help='Where to save the output images', type=str, required=True)
```

À l'aide du code et des images générés nous comprenons les paramètres du programme; La seed donne un individu différent, et le paramètre truncation psi permet d'ajuster la diversité (valeur haute) et la qualité (valeur basse). Ci-contre un résumé des différents paramètres et leur impact.

Paramètre	**Impact Principal**	**Valeur Recommandée**
–seeds	Quelle(s) image(s)	0-1000 pour explorer
–trunc	Qualité vs Créativité	0.7 (équilibré)
–noise-mode	Détails fins	const (défaut)
–translate	Position	0,0 (centré)
–rotate	Orientation	0 (droit)

Table 3: Paramètres d'inférence et leurs impacts

Network correspond au modèle de génération utilisé, et outdir sert seulement à indiquer ou sauvegarder les images générés.

2.4 Conclusion

- GAN Lab a permis de visualiser les mécanismes fondamentaux des GANs : compétition G/D, sensibilité aux hyperparamètres.

- StyleGAN3 montre l'évolution vers des modèles très performants et stables, capables de générer des images quasi réalistes.

3 TD3

3.1 Introduction

Dans cette partie nous allons au travers d'un notebook entraîner et utiliser deux modèles de langage avec le dataset wikitext-2 de HuggingFace.

Les deux modèles utilisés sont: Causal Language Modeling (CLM) : Prédiction du token suivant Masked Language Modeling (MLM) : Prédiction de tokens masqués On pourra concrètement comparer des approches autoregressives (CLM) et bidirectionnelles (MLM).

3.2 Le dataset

Le dataset WikiText-2-raw-v1 disponible sur HuggingFace a pour caractéristique:

- Train : 36,718 exemples
- Validation : 3,760 exemples
- Test : 4,358 exemples

Il contient des paragraphes complets d'articles des titres de sections ainsi que des lignes vides.

3.3 Comparaison des models

Nous avons exécuté les modèles à l'aide de Google Colab avec un GPU T4 mis à disposition gratuitement pour un temps limité. Les résultats ci-contre:

	CLM	**MLM**
Architecture	GPT-2	BERT
Entraînement(min)	12:19	12:47
Training Loss	6.2267	7.0779
Validation Loss	6.2209	7.0683
Perplexité finale	503.16	1,188.40

Table 4: Paramètres d'inférence et leurs impacts

3.4 Conclusion

Les perplexités obtenues (503 pour CLM, 1188 pour MLM) indiquent que les modèles n'ont pas convergé. Cela est principalement due au nombre d'epochs trop bas et la quantité faible du dataset mais malgrè tout le CLM semble être plus pertinent. Un entraînement from scratch nécessite des données et du temps considérable. Le transfer learning serait plus approprié pour des datasets de cette taille, on pourrait aussi utiliser un dataset plus étoffé et entraîner sur plus d'epochs avec un matériel plus puissant.

4 Conclusion Générale

Ces trois Travaux Dirigés (TD) nous ont permis de manier différentes méthodologies d'apprentissage automatique, du perceptron simple jusqu'aux modèles de langage modernes en passant par des GAN. Ils ont souligné l'impact crucial des hyperparamètres, de la qualité des données, et de l'architecture des modèles sur l'obtention de performances optimales.