Natural gradient-based optimization methods for deep neural networks

Abdoulaye Koroko

Thèse de doctorat

NNT : 2023UPASG068

université PARIS-SACLAY

Méthodes d'optimisation basées sur le gradient naturel pour les réseaux de neurones profonds

Natural gradient-based optimization methods for deep neural networks

Thèse de doctorat de l'université Paris-Saclay

École doctorale n° 580
Sciences et technologies de l'information et de la communication (STIC)
Spécialité de doctorat : Informatique mathématique
Graduate School : Informatique et Sciences du Numérique
Référent : CentraleSupélec

Thèse préparée dans l'unité de recherche
IFPEN-Sciences et Technologies du Numérique,
sous la direction de **Quang Huy TRAN**, ingénieur de recherche HDR,
la co-direction de **Mounir HADDOU**, professeur des universités,
le co-encadrement d'**Ibtihel BEN GHARBIA**, ingénieur de recherche,
et le co-encadrement d'**Ani ANCIAUX-SEDRAKIAN**, ingénieur de recherche
avec la participation de **Valérie GARÈS**, maître de conférences

Thèse soutenue à Rueil-Malmaison, le 16/10/2023, par

Abdoulaye KOROKO

Composition du jury

Membres avec voix délibérative

M. Laurent DUMAS Professeur des universités, Université de Versailles St-Quentin	Président
Mme Hoai An LE THI Professeur des universités, Université de Lorraine	Rapporteur
Mme Aude RONDEPIERRE Professeur des universités, INSA Toulouse	Rapporteur
M. Grégoire ALLAIRE Professeur des universités, École Polytechnique	Examinateur
M. Aziz BELMILOUDI Maître de conférences HDR, Université de Rennes	Examinateur

Remerciements

Je tiens à exprimer ma profonde gratitude envers toutes les personnes qui ont contribué à la réalisation de cette thèse.

En premier lieu, je tiens à adresser mes plus sincères remerciements à mes deux directeurs de thèse, Quang Huy TRAN et Mounir HADDOU, pour leur inestimable contribution à la réalisation de cette thèse. Huy et Mounir, vous avez été des mentors exceptionnels tout au long de ce voyage. Vos expertises, vos passions pour la recherche ainsi que vos capacités à guider mes efforts de manière éclairée ont été des atouts cruciaux. Les conseils que j'ai reçus de vous m'ont ouvert de nouvelles perspectives et m'ont aidé à tenir la cadence dans les moments tumultueux de la thèse. L'opportunité de travailler sous votre direction conjointe a été une expérience enrichissante, et je me considère chanceux d'avoir pu bénéficier de votre mentorat à tous les niveaux de ce travail.

Je tiens également à exprimer ma reconnaissance envers mes trois encadrantes, Ibtihel BEN GHARBIA, Ani ANCIAUX-SEDRAKIAN et Valérie GARÈS, pour leurs précieuses contributions à ce travail. Leurs conseils et suggestions ont grandement enrichi mon parcours. Ibtihel, Ani et Valérie, merci à vous d'avoir été toujours disponibles. Vos engagements dans mon travail de recherche ont été d'une importance capitale. Ce fût très agréable d'avoir travaillé sous votre supervision.

Je suis particulièrement reconnaissant envers Aude RONDEPIERRE et Hoai An LE THI, qui ont accepté la tâche exigeante de rapporter ma thèse. Un grand merci également à Laurent DUMAS, Grégoire ALLAIRE et Aziz BELMILOUDI d'avoir accepté de faire partie de mon jury de thèse.

Je voudrais exprimer ma reconnaissance envers Zakia BENJELLOUN-TOUIMI, Sylvain DESROZIERS et Mathieu FERAILLE qui, en m'acceptant comme apprenti chercheur, m'ont offert l'opportunité d'intégrer le département de recherche de mathémariques appliquées de IFP Energies nouvelles. Cette expérience m'a donné un goût pour la recherche et a conforté ma volonté de faire une thèse de doctorat.

Je souhaite également exprimer ma reconnaissance envers toutes les personnes qui ont collaboré avec moi, en particulier mes camarades doctorants du département, pour leur partage de connaissances et leur amitié. Aussi, voudrais-je remercier chaleureusement mes amis proches pour leur soutien, encouragement et présence tout au long de cette aventure.

Je tiens à exprimer ma profonde gratitude envers mes parrains, Amadou COULIBALY et Eugène Aouélé AKA, pour leur bienveillance et soutien indéfectible tout au long de mon parcours académique.

Enfin, à ma famille, je dédie cette thèse. Je vous remercie du fond du coeur pour votre amour inconditionnel, soutien inébranlable, compréhension et présence constante dans ma vie. Vous êtes ma source de motivation et d'inspiration. Je suis éternellement reconnaissant et honoré de faire partie d'une famille aussi extraordinaire.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Contents

Ce chapitre présente les motivations de la thèse ainsi que ses principales orientations. Il fait aussi office de "résumé en français" requis par l'École Doctorale.

Nous commençons par décrire en §1.1 la situation actuelle en matière de durées d'entraînement des réseaux et de méthodes d'optimisation employées en apprentissage. Nous nous intéressons ensuite dans §1.2 plus particulièrement aux méthodes de type gradient naturel, pour lesquelles nous dressons état des lieux avant de formuler les objectifs de travail. La dernière section §1.3 récapitule nos contributions tout en servant de guide de lecture du mémoire complet.

1.1 Contexte et motivation

1.1.1 Allongement du temps d'entraînement des réseaux

On assiste ces dernières années à une véritable explosion de la quantité de données disponibles pour l'entraînement des modèles d'apprentissage, ainsi qu'à une croissance vertigineuse de la taille des réseaux toujours plus complexes définissant ces modèles. Cette évolution vers le gigantesque, favorisée par les avancées en matière de ressources informatiques, a permis d'accomplir des progrès spectaculaires ayant abouti à moult applications innovantes. Elle n'est toutefois pas sans incidence pratique sur la mise en œuvre même de ces modèles : le temps de calcul nécessaire à l'entraînement d'un réseau est lui aussi en constante augmentation !

À titre d'illustration, rappelons quelques ordres de grandeur[1] :

[1] Les chiffres sont donnés à titre indicatif et peuvent varier énormément selon divers facteurs.

- Un réseau de neurones convolutif (CNN) simple pour la classification d'images avec environ un million de paramètres peut être entraîné en quelques heures à quelques jours sur un ensemble de données de quelques milliers à quelques dizaines de milliers d'images.
- Un réseau de neurones récurrent (RNN) complexe de type LSTM ou GRU avec plusieurs couches et des milliers de paramètres, est plus coûteux en raison de sa nature séquentielle. Un entraînement peut prendre plusieurs jours ou plusieurs semaines.
- Un réseau de neurones profond (DNN) pré-entraîné comme ResNet, VGG ou Inception a des dizaines de millions de paramètres. Son entraînement nécessite plusieurs semaines ou plusieurs mois sur de grandes bases de données contenant des millions d'images.
- Un réseau de neurones auto-attentif (Transformer) pour le traitement du langage naturel tel que GPT-3, qui contient 175 milliards de paramètres, requiert pour son entraînement plusieurs semaines ou même plusieurs mois sur des infrastructures hautement optimisées.
- Les réseaux de neurones génératifs antagonistes pour la génération d'images nécessitent en fonction des spécificités du modèle, des ressources informatiques disponibles et de la taille du jeu un temps d'entraînement considérable, allant de plusieurs semaines à plusieurs mois.

Ces exemples montrent qu'il est vital de chercher à réduire le temps de calcul de la phase d'entraînement d'un réseau. La réussite d'une telle entreprise apporterait les atouts suivants :

1. *Efficacité des ressources* : les utilisateurs pourront libérer plus rapidement la puissance de calcul, la mémoire et l'espace de stockage afin de les allouer à d'autres tâches.
2. *Gain en productivité* : les ingénieurs pourront itérer plus rapidement sur les modèles envisagés, tester de nouvelles idées, ajuster les hyperparamètres, expérimenter différentes architectures de réseau et explorer des ensembles de données plus vastes.
3. *Diminution des coûts* : les factures liées aux ressources informatiques, comme la location de serveurs cloud ou de l'achat d'infrastructures matérielles seront moins élevées.
4. *Rapiditié de déploiement* : une fois entraîné, le modèle est déployé dans un environnement de production pour effectuer des prédictions en temps réel ; plus court est l'entraînement, plus vite peut-on le mettre à jour en le ré-entraînant sur de nouvelles données.

La réduction du temps de calcul peut être réalisée par la "force brute", qui mise sur des accélérations matérielles (comme les GPU dédiés) et les techniques de parallélisation. Conjointement, il convient aussi d'élaborer des algorithmes d'optimisation sinon plus sophistiqués, du moins mieux adaptés au cadre de l'apprentissage. Ce travail s'inscrit dans cette catégorie.

1.1.2 Popularité des méthodes d'optimisation du premier ordre

La technologie actuellement prédominante pour effectuer la phase d'entraînement des réseaux de neurones est celle des méthodes du premier ordre issues du gradient stochastique (SGD, *Stochastic Gradient Descent*). Proposée historiquement dans les années 1950 par Robbins et Monro [141], la méthode SGD a donné naissance à de multiples variantes (cf. §2) dont certaines sont aujourd'hui largement adoptées par la communauté. Pour discuter de leurs avantages et inconvénients, il est utile de revenir sur l'idée même du gradient stochastique.

La méthode SGD exploite la structure spécifique de la fonction objectif à minimiser, qui est le *risque empirique* en apprentissage. Celle-ci prend en effet la forme d'une moyenne sur les données, à savoir

$$h(\theta) = \frac{1}{n}\sum_{b=1}^{n} L(y^{(b)}, f(x^{(b)}, \theta)), \tag{1.1}$$

où n est le nombre de données de la base d'entraînement, $\theta \in \mathbb{R}^p$ regroupe l'ensemble des poids du réseau, et

$$h_b(\theta) := L(y^{(b)}, f(x^{(b)}, \theta)) \tag{1.2}$$

représente une mesure de l'*écart* entre la prédiction $f(x^{(b)}, \theta)$ du modèle à partir de l'entrée $x^{(b)}$ et l'observation en sortie $y^{(b)}$ pour la b-ème donnée. Pour trouver une solution du problème

$$\min_{\theta\in\mathbb{R}^p} h(\theta),$$

la méthode SGD préconise les itérations

$$\theta_{k+1} = \theta_k - \alpha_k \nabla_\theta h_{b(k)}(\theta_k), \tag{1.3}$$

dans lesquelles $\alpha_k > 0$ est le *taux d'apprentissage* et $b(k) \in \{1, \ldots, n\}$ est un indice tiré aléatoirement à chaque itération k. Par rapport à une vraie méthode de descente de gradient

$$\theta_{k+1} = \theta_k - \alpha_k \nabla_\theta h(\theta_k), \tag{1.4}$$

on constate que le vrai gradient

$$\nabla_\theta h(\theta_k) = \frac{1}{n}\sum_{b=1}^{n} \nabla_\theta h_b(\theta_k), \tag{1.5}$$

qui est une moyenne, a été remplacé par un élément arbitraire de la somme. En présence de beaucoup de données, cette approximation audacieuse présente l'avantage de diminuer drastiquement le nombre d'évaluations de gradients élémentaires $\nabla_\theta h_b(\theta_k)$. Par ailleurs la complexité algorithmique de la méthode SGD est linéaire par rapport à n et à p, ce qui est a priori favorable pour s'attaquer aux données massives (n grand) avec les réseaux profonds (p grand).

En contrepartie, le défaut principal de SGD réside dans la mauvaise qualité de l'approximation stochastique du gradient, qui certes est non biaisée mais fortement bruitée Il n'est pas du tout certain que la direction proposée $\nabla_\theta h_{b(k)}(\theta_k)$ soit encore une direction de descente vis-à-vis de la fonction h. Aussi, il est prudent de fixer le taux d'apprentissage α_k très "petit". Même avec cela, les itérés peuvent osciller énormément. Pour réduire la *variance*, il y a essentiellement trois approches [18] : (i) échantillonnage dynamique ; (ii) agrégation de gradients ; (iii) moyennage sur les itérés. La première [24] consiste à calculer un gradient *mini-lot*[2]

$$g(\mathcal{S}_k, \theta_k) = \frac{1}{|\mathcal{S}_k|}\sum_{b\in\mathcal{S}_k} \nabla_\theta h_b(\theta_k), \tag{1.6}$$

obtenu en moyennant les gradients élémentaires sur un petit échantillon de données $\mathcal{S}_k \subset \{1, \ldots, n\}$. Ceci améliore la situation, à condition de bien savoir choisir la taille de $\mathcal{S}_k$ et le taux d'apprentissage α_k au cours des itérations. La deuxième, avec notamment SVRG [84] et

[2] L'idée du mini-lot est en fait indépendante des techniques d'échantillonnage et peut être employée de manière plus statique en guise d'alternative à SGD.

SAGA [34], tente de reconstituer un gradient complet à partir des gradients élémentaires qui ont été déjà évalués pour des arguments différents, stockés en mémoire et réutilisés de manière judicieuse. La troisième [132] lisse sur tous les itérés θ_k obtenus, à l'instar d'une moyenne de Cesàro, en combinaison avec une stratégie de diminution du pas α_k. Nous référons le lecteur à l'article de Bottou et al. [18] pour les résultats théoriques de ces raffinements de SGD.

De façon "orthogonale" à ceux-ci, il existe trois autres familles de méthodes très appréciées chez les praticiens, qui accélèrent plus ou moins SGD sans s'attaquer directement à la variance[3]. Il s'agit de gradient avec : (i) élan ; (ii) accélération de Nesterov ; (ii) adaptativité. La première [161], inspirée de la physique, simule l'effet de frottement dans une équation du mouvement d'ordre 2, ce qui amortit les oscillations. La seconde [118] est une version évoluée de la première, avec un frottement plus "intelligent". La troisième prône un taux d'apprentissage différent pour chaque composante du vecteur inconnu. Elle rassemble un grand nombre de méthodes aux acronymes attrayants comme AdaGrad [45], RMSProp [165], Adam [88], Adadelta [185] et Nadam [44]. Ces familles de méthodes sont décrites plus amplement en §2.2.2.

Une autre critique plus fondamentale qu'on peut adresser à SGD, comme d'ailleurs à toutes les variantes du premier ordre, est que celles-ci font intervenir seulement les dérivées premières de la fonction objectif et non pas les dérivées secondes. Dans ces conditions, les itérations ne sont pas invariantes par rapport à un changement de variables affine. Autrement dit, un changement d'unités en θ produit des itérés qui ne se correspondent pas, à moins qu'on soit capable de le répercuter correctement sur le taux d'apprentissage. À cet égard, signalons l'exception notoire de la méthode de Barzilai-Borwein [11], dans laquelle le taux d'apprentissage est fourni par l'une des deux formules

$$\alpha_k = \frac{\langle \theta_k - \theta_{k-1}, \nabla_\theta h(\theta_k) - \nabla_\theta h(\theta_{k-1}) \rangle}{\|\nabla_\theta h(\theta_k) - \nabla_\theta h(\theta_{k-1}\|^2} \quad \text{ou} \quad \alpha_k = \frac{\|\theta_k - \theta_{k-1}\|^2}{\langle \theta_k - \theta_{k-1}, \nabla_\theta h(\theta_k) - \nabla_\theta h(\theta_{k-1}) \rangle}, \tag{1.7}$$

de sorte à minimiser

$$\|\theta_k - \theta_{k-1} - \alpha(\nabla_\theta h(\theta_k) - \nabla_\theta h(\theta_{k-1}))\|^2 \quad \text{ou} \quad \|\nabla_\theta h(\theta_k) - \nabla_\theta h(\theta_{k-1}) - \alpha^{-1}(\theta_k - \theta_{k-1})\|^2. \tag{1.8}$$

Avec les coefficients (1.7), la formule d'itération (1.4) est homogène du point de vue des unités, ce qui assure l'invariance par rapport à un changement d'échelle. Cette propriété agréable confère à Barzila-Borwein un "goût" de méthode du second ordre, où l'approximation de la matrice hessienne $\nabla^2_{\theta\theta} h(\theta_k)$ par $\alpha_k^{-1} I$ est "optimal" au sens de (1.8). La transposition de Barzilai-Borwein au cadre stochastique de l'apprentissage a été tentée par plusieurs auteurs, en particulier [104, 164]. Nous avons nous-mêmes passé les premiers mois de thèse à étudier cette méthode pour nous familiariser avec ce domaine de l'optimisation et son environnement informatique. Malheureusement, la performance numérique de Barzilai-Borwein n'est pas tout à fait à la hauteur de nos attentes. C'est pourquoi nous ne présenterons pas les résultats correspondants.

Un dernier désavantage des méthodes du premier ordre est qu'elles sont intrinsèquement séquentielles et ne se prêtent pas facilement au parallélisme. Certes, il existe des techniques assez rudimentaires pour exploiter la parallélisme dans les méthodes d'optimisation du premier ordre comme la parallélisation du calcul du gradient (l'ensemble ou le mini-lot des données est réparti en plusieurs sous-ensembles ou sous-lots plus petits, sur chacun desquels l'estimation partielle du gradient est assurée par un processeur) et la parallélisation de la mise à jour des paramètres (les composantes de θ sont divisées en plusieurs sous-ensembles, chacun desquels est actualisé par un

[3] Par "orthogonal" nous entendons que les trois nouvelles familles de méthodes agissent par dessus la mise à jour (1.4) en faisant intervenir une notion de gradient qui peut être le gradient complet, stochastique ou mini-lot.

processeur différent). Ces techniques doivent toutefois surmonter des défis supplémentaires, tels que la synchronisation entre les tâches parallèles et l'optimisation de l'utilisation des ressources matérielles pour éviter les goulots d'étranglement.

Malgré toutes ces faiblesses, les méthodes d'optimisation du premier ordre (SGD et ses variantes) sont couramment utilisées en apprentissage automatique. Elles sont nettement préférées aux méthodes du second ordre que nous allons maintenant évoquer avant de revenir sur les raisons du succès des méthodes du premier ordre.

1.1.3 Potentiel des méthodes d'optimisation du second ordre

En optimisation classique, c'est-à-dire en dehors du cadre de l'apprentissage, il est connu [123] que les méthodes à recommander sont celles du second ordre. À la place de la descente de gradient (1.4), on considère les itérations

$$\theta_{k+1} = \theta_k - \alpha_k [C(\theta_k)]^{-1} \nabla_\theta h(\theta_k), \tag{1.9}$$

où $C(\theta_k)$ est une matrice $p \times p$ contenant des informations de courbure sur h. Par exemple, si

$$C(\theta_k) = \nabla^2_{\theta\theta} h(\theta_k) \tag{1.10}$$

est la matrice hessienne de h, alors on retombe sur la méthode de Newton avec un contrôle du pas de descente. Avec $\alpha_k \equiv 1$ et sous certaines hypothèses convenables sur h et le point initial, on peut établir la convergence quadratique [37] de la méthode de Newton, ce qui surpasse la convergence sous-linéaire d'une descente de gradient du premier ordre. Pour un problème issu de l'apprentissage, la Hessienne peut être stochastique ou estimée par

$$H(\mathcal{S}'_k, \theta_k) = \frac{1}{|\mathcal{S}'_k|} \sum_{b \in \mathcal{S}'_k} \nabla^2_{\theta\theta} h_b(\theta_k) \tag{1.11}$$

sur un mini-lot $\mathcal{S}'_k \in \{1, \dots, n\}$, lequel peut être différent de $\mathcal{S}_k$, celui utilisé pour le gradient.

L'assemblage et le stockage de la matrice hessienne sont onéreux en termes de mémoire et de temps de calcul. Pour y remédier, il a été suggéré [108] des méthodes sans Hessienne (*Hessian-free*), qui exploitent les informations exactes de la matrice hessienne sans jamais avoir à la former et la stocker. Pour cela, on remarque que l'incrément $\theta_{k+1} - \theta_k$ est solution du système linéaire

$$\nabla^2_{\theta\theta} h(\theta_k)(\theta_{k+1} - \theta_k) = -\nabla_\theta h(\theta_k). \tag{1.12}$$

Ce système peut être résolu par des techniques inexactes [35] impliquant le gradient conjugué (CG) [63], lequel n'exige que la connaissance du produit matrice-vecteur. Or, pour un réseau de neurones, chaque produit matrice-vecteur impliquant une hessienne $\nabla^2_{\theta\theta} h_b(\theta_k)$ correspondant à une donnée b peut être effectué à travers un enchaînement de propagation et de rétropropagation [130]. Le nombre d'itérations du gradient conjugué doit être limité pour assurer une certaine rapidité, tout en demeurant suffisamment élevé pour ne pas trop dégrader la vitesse de convergence de ce Newton inexact. Notons enfin que la matrice hessienne peut ne pas être (semi-)positive, ce qui pose problème au CG.

Une alternative à la méthode sans Hessienne consiste à approcher la matrice hessienne par des mises à jour de rang 2 au cours des itérations, ce qui donne lieu à la méthode dite BFGS (*Broyden-Fletcher-Goldfarb-Shanno*) [23, 52, 60, 156]

$$H_{k+1} \approx H_k + \frac{\Delta g_k \Delta g_k^T}{\Delta g_k^T \Delta \theta_k} - \frac{H_k \Delta\theta_k \Delta\theta_k^T H_k^T}{\Delta\theta_k^T H_k \Delta\theta_k} \tag{1.13a}$$

avec

$$\Delta\theta_k = \theta_{k+1} - \theta_k, \qquad \Delta g_k = \nabla_\theta h(\theta_{k+1}) - \nabla_\theta h(\theta_k). \tag{1.13b}$$

Grâce à la formule de Sherman-Morrison-Woodbury, on peut aisément mettre à jour les inverses comme [123]

$$H_{k+1}^{-1} \approx \left(I - \frac{\Delta g_k \Delta\theta_k^T}{\Delta\theta_k^T \Delta g_k}\right)^T H_k^{-1} \left(I - \frac{\Delta g_k \Delta\theta_k^T}{\Delta\theta_k^T \Delta g_k}\right) + \frac{\Delta\theta_k \Delta\theta_k^T}{\Delta\theta_k^T \Delta g_k}. \tag{1.14}$$

L'approximation (1.13) de la Hessienne, qui n'utilise que des informations du premier ordre, est "bonne" en ce sens qu'elle est symétrique semi-positive (si H_0 l'est) et vérifie l'équation de la sécante $H_{k+1}\Delta\theta_k = \Delta g_k$. Cette propriété permet de garantir un taux de convergence locale superlinéaire en déterministe. Malheureusement, les matrices données par (1.13)–(1.14) sont denses, même quand les Hessiennes exactes sont creuses. De surcroît, il faut bien stocker les H_k^{-1} pour pouvoir passer d'une itération à la suivante. Une version plus économique en mémoire existe sous la dénomination L-BFGS (*limited memory BFGS*) [105], dans laquelle on évite le stockage en calculant les produits $H_k^{-1}\nabla_\theta h(\theta_k)$ par une formule approchée ne faisant intervenir que les incréments $(\Delta\theta_k, \Delta g_k)$ les plus récemment sauvegardés. La méthode L-BFGS apparaît comme mieux adéquate au contexte de l'apprentissage, où d'autres variantes ont été développées comme L-BFGS-B (*L-BFGS box constraints*) [25] et O-LBFGS (*online L-BFGS*) [153].

La méthode BFGS et ses variantes appartiennent à famille des méthodes *quasi-Newton*, où l'on renonce à l'idéal de la matrice hessienne en lui substituant des approximations plus simples avec certaines propriétés plus favorables. Une autre classe importante de la famille quasi-Newton est celle des méthodes de Gauss-Newton généralisé (GGN). Appelons z le deuxième argument de l'écart $L(y, z)$, de sorte que

$$h_b(\theta) = L(y^{(b)}, z^{(b)}), \qquad z^{(b)} = f(x^{(b)}, \theta). \tag{1.15}$$

Alors, la matrice de courbre $C(\theta_k)$ est choisie comme la matrice GGN, définie par [152]

$$G(\mathcal{S}'_k, \theta_k) = \frac{1}{|\mathcal{S}'_k|} \sum_{b \in \mathcal{S}'_k} [J_{f_\theta(x^{(b)})}]^T \nabla^2_{zz} L(y^{(b)}, f(x^{(b)}, \theta)) [J_{f_\theta(x^{(b)})}] \tag{1.16}$$

sur un mini-lot $\mathcal{S}'_k$. Il convient de noter que $J_{f_\theta(x^{(b)})}$ désigne la matrice jacobienne de $f(x^{(b)}, \theta)$ par rapport à θ. L'idée de la construction, expliquée en §2.2.3, est de ne garder dans la Hessienne que les informations correspondant aux "carrés" des dérivées premières par rapport à θ, négligeant ainsi celles en provenance des dérivées secondes. Le résultat obtenu (1.16) est une matrice symétrique semi-positive, dont le produit avec un vecteur peut être effectué à l'aide d'un enchaînement de propagation et de rétropropagation à travers le réseau [152] sans qu'on ait à la former explicitement. Pour un écart quadratique $L(y, z) = \frac{1}{2}\|y - z\|^2$, on a $\nabla^2_{zz} L = I$ et retrouve la matrice de Gauss-Newton classique. D'autres propriétés de la matrice GGN sont exposées dans [110, 128], notamment en lien avec celle de Fisher dont il va être question ci-après.

La dernière catégorie de ce bref aperçu des méthodes du second ordre est celle du gradient naturel, initiée par Amari [3]. Sans entrer dans le cadre formel qui sera esquissé en §1.2.1, l'idée de base est de formuler une descente de gradient non pas dans l'espace euclidien des paramètres θ mais sur la variété des distributions de probabilités liées à la prédiction, munie d'une métrique convenable. De par sa signification géométrique intrinsèque, la nouvelle notion de gradient est plus "naturelle". La descente jouit alors en un sens de la propriété d'invariance par rapport à tout changement de variables qui préserve les distributions de probabilité. Au niveau concret, on peut aussi dire très prosaïquement que le gradient naturel est une méthode quasi-Newton

dans laquelle la matrice de courbure $C(\theta_k)$ est prise égale à une matrice symétrique semi-positive dite de Fisher $F(\theta_k)$, dont l'expression sera précisée en §1.2.1. Cette matrice de Fisher étant pleine et de grande taille, on se retrouve face aux mêmes difficultés qu'avec les précédentes méthodes du second ordre. Les premières approximations de la matrice de Fisher par des structures plus simples [95, 96, 124, 133] n'ont pas donné de résultats satisfaisants. L'arrivée de KFAC (*Kronecker-Factored Approximate Curvature*) [112] constitue un progrès majeur et suscite beaucoup d'espoir. Nous y reviendrons en §1.2.2.

Malgré leur supériorité théorique (homogénéité, robustesse et vitesse de convergence), force est de constater que les méthodes d'optimisation du second ordre n'ont pour le moment pas rencontré le succès escompté dans l'usage au quotidien en apprentissage automatique[4]. La principale raison en est que chaque itération du second ordre a un coût exorbitant en mémoire et en temps de calcul par rapport à une itération du premier ordre, et ce même quand on a déployé toutes les astuces pour se dispenser de l'assemblage et du stockage de la matrice de courbure. Lorsque les ressources en mémoire et en calcul sont limitées, les méthodes du premier ordre sont préférées pour leur moindre complexité. Certes, globalement on a besoin de moins d'itérations, mais cela ne suffit pas pour rendre compétitives les méthodes du second ordre. Cette observation confirme l'un des paradoxes [18] de l'apprentissage automatique : "Les bonnes méthodes d'optimisation ne sont pas toujours des bonnes méthodes de *machine learning*".

Un autre facteur explicatif à ce paradoxe est que les méthodes d'optimisation du premier ordre, outre leur simplicité de mise en œuvre, s'adaptent plus facilement à une évolution de l'architecture du réseau. Les méthodes du premier ordre peuvent converger plus rapidement que les méthodes du second ordre au début de l'entraînement. Cela est particulièrement avantageux lorsque le modèle est initialisé avec des poids aléatoires, car elles peuvent rapidement améliorer les performances initiales et trouver des régions de l'espace de recherche qui fournissent des gradients plus riches en information. Cependant, les méthodes du second ordre peuvent être utiles dans certains scénarios, notamment lorsque les données sont rares ou lorsque la forme de la fonction objectif est très courbée. Dans ces cas, les méthodes du second ordre peuvent converger plus rapidement vers des optima locaux de meilleure qualité.

L'objectif de cette thèse est de contribuer à résorber le goulot d'étranglement dû à la matrice de courbure et au système linéaire associé dans les méthodes du second ordre afin d'accroître leur efficacité en grande dimension, notamment sur des réseaux profonds. À cette fin, la piste que nous souhaitons explorer est celle du gradient naturel, qui semble bénéficier d'un fondement mathématique mieux approprié au contexte de l'apprentissage. La conviction qu'une telle entreprise est possible s'appuie sur la performance annoncée pour certaines approximations récentes de la matrice de Fisher, qui sont de type KFAC [68, 111, 112] ou des versions plus évoluées [53, 54, 56, 139]. Avant d'exposer notre démarche et de récapituler nos contributions, il est nécessaire de procéder à une revue plus approfondie des méthodes de type gradient naturel.

1.2 État de l'art sur les méthodes de type gradient naturel

1.2.1 Matrice de Fisher et descente de gradient naturel

La philosophie générale des méthodes de type gradient naturel est d'associer à chaque paramètre $\theta \in \mathbb{R}^p$ une distribution de probabilité représentant l'écart entre la prédiction du modèle et la donnée. Chaque distribution de probabilité est ensuite considérée comme un point d'une

[4] Nous excluons dans ce constat le monde de la recherche où le second ordre fait l'objet d'intenses activités.

structure riemannienne 'naturelle" sur laquelle on peut alors raisonner au lieu de rester dans l'espace euclidien "artificiel" des paramètres. Une telle démarche d'application des techniques de géométrie différentielle à l'étude des objets statistiques relève de la géométrie de l'information [5, 7], discipline aux multiples applications [4, 122].

Pour qu'une telle association soit possible, il faut supposer qu'à une constante additive près, la fonction écart L coïncide avec une log-vraisemblance : autrement dit, il existe une densité de probabilité $\wp$ et une constante ν telle que

$$L(y, z) = -\log \wp(y|z) + \nu. \tag{1.17}$$

En composant avec le modèle de prédiction f, on définit la distribution prédictive conditionnelle $P_{y|x}(\theta)$ par sa densité $p(y|x, \theta)$ vérifiant

$$L(y, f(x, \theta)) = -\log p(y|x, \theta) + \nu, \tag{1.18}$$

ainsi que la distribution prédictive $\mathsf{P}_{x,y}(\theta)$ par sa densité

$$\mathsf{p}(x, y|\theta) = q(x)p(y|x, \theta), \tag{1.19}$$

où q est la densité de la distribution Q_x des entrées x.

En théorie de l'information, la divergence de Kullback-Leibler (ou l'*entropie relative*) est une mesure de dissimilarité entre deux distributions de probabilités. Dans notre cas, elle s'écrit

$$\mathrm{KL}[\mathsf{P} \,\|\, \mathsf{Q}] = \int_{\mathbb{R}^{d_x} \times \mathbb{R}^{d_y}} \mathsf{p}(x, y) \log \frac{\mathsf{p}(x, y)}{\mathsf{q}(x, y)} \,\mathrm{d}x\mathrm{d}y \tag{1.20}$$

pour deux distributions de probabilités portant sur les entrées x et les sorties y, et peut être perçue comme une "distance" bien qu'elle ne soit pas symétrique. Beaucoup plus utile est sa version infinitésimale entre deux distribtions prédictives définies par (1.18)–(1.19). On montre que pour une perturbation paramétrique $\delta \in \mathbb{R}^p$ suffisamment petite, son comportement est en

$$\mathrm{KL}[\mathsf{P}_{x,y}(\theta) \,\|\, \mathsf{P}_{x,y}(\theta + \delta)] = \frac{1}{2}\delta^T F(\theta)\delta + O(\|\delta\|^3), \tag{1.21}$$

où

$$F(\theta) = \mathbb{E}_{(x,y)\sim\mathsf{P}_{x,y}(\theta)}\{\nabla_\theta \log \mathsf{p}(x, y|\theta)[\nabla_\theta \log \mathsf{p}(x, y|\theta)]^T\} \tag{1.22}$$

est la matrice d'information de Fisher (FIM) relative au paramètre θ. L'information de Fisher est une matrice de covariance. Elle est encore égale à

$$F(\theta) = \mathbb{E}_{x\sim Q_x, y\sim P_{y|x}(\theta)}\{\nabla_\theta \log p(y|x, \theta)[\nabla_\theta \log p(y|x, \theta)]^T\} \tag{1.23a}$$

$$= \mathbb{E}_{x\sim Q_x, y\sim P_{y|x}(\theta)}\{\nabla^2_{\theta\theta}[-\log p(y|x, \theta)]\}. \tag{1.23b}$$

La dernière ligne signifie qu'elle est l'espérance de la matrice hessienne par rapport à la distribution prédictive $\mathsf{P}_{x,y}(\theta)$. Ceci est assez remarquable dans la mesure où la matrice de Fisher ne fait intervenir que les dérivées premières en θ, tandis que la matrice hessienne contient aussi les dérivées secondes (lesquelles ont une moyenne nulle selon la distribution considérée). Il convient d'évaluer la matrice de Fisher sur un mini-lot $\mathcal{S}$ par

$$F(\theta) \approx \frac{1}{|\mathcal{S}|} \sum_{x^{(b)}\in\mathcal{S}} \mathbb{E}_{y\sim P_{y|x^{(b)}}(\theta)}\{\nabla_\theta \log p(y|x^{(b)}, \theta)[\nabla_\theta \log p(y|x^{(b)}, \theta)]^T\}, \tag{1.24}$$

où chaque espérance de la somme doit s'effectuer en générant les y selon $P_{x,y}(\theta)$, Il est erroné [94] de prendre les données y de sortie, ce qui conduirait à la matrice de Fisher *empirique*

$$F(\theta) \approx \tilde{F}(\theta) = \frac{1}{|\mathcal{S}|} \sum_{(x^{(b)},y^{(b)})\in\mathcal{S}} \nabla_\theta \log p(y^{(b)}|x^{(b)},\theta)[\nabla_\theta \log p(y^{(b)}|x^{(b)},\theta)]^T. \tag{1.25}$$

Comme pour la matrice hessienne et celle de GGN, le produit de la matrice de Fisher par un vecteur quelconque peut être réalisé par le réseau par propagation et rétropropagation [152], ce qui peut être exploité pour la résolution d'un système linéaire avec la matrice de Fisher.

Un tel système linéaire apparaît lors d'une descente de gradient naturel, qui est de la forme

$$\theta_{k+1} = \theta_k - \alpha_k [F(\theta_k)]^{-1} \nabla_\theta h(\theta_k), \qquad \alpha_k > 0, \tag{1.26}$$

en supposant que les matrices $F(\theta_k)$ sont définies, donc inversibles. La nouvelle direction de descente s'interprète comme la limite des meilleures directions issues des problèmes de région de confiance de tailles décroissantes autour du point courant, à savoir

$$-\frac{[F(\theta)]^{-1}\nabla_\theta h(\theta)}{\|[F(\theta)]^{-1}\nabla_\theta h(\theta)\|_{F(\theta)}} = \lim_{c\downarrow 0} \frac{1}{c} \operatorname*{argmin}_{\|\delta\|_{F(\theta)}=c} h(\theta+\delta), \tag{1.27}$$

où

$$\|\delta\|_{F(\theta)} = [\delta^T F(\theta)\delta]^{1/2} \tag{1.28}$$

est la norme associée à $F(\theta)$. Par conséquent, une descente de gradient naturel est en réalité une descente de plus grande pente — donc, du premier ordre — utilisant une notion plus subtile de gradient, même si au niveau pratique on peut aussi l'appréhender comme une méthode de quasi-Newton — donc, du second ordre — dans l'espace euclidien.

Il est important de noter que l'itération (1.26) n'est stricto sensu *pas* invariante par changement de variable non linéaire. Ce qui est invariant est sa version continue

$$\frac{\mathrm{d}\theta}{\mathrm{d}\tau}(\tau) = -[F(\theta(\tau))]^{-1}\nabla_\theta h(\theta(\tau)), \tag{1.29}$$

où τ représente un temps fictif. La discrétisation de (1.29) par le schéma explicite (1.26), dans lequel α_k joue le rôle d'un pas de temps, a certes provoqué la perte de la propriété d'invariance. Néanmoins, l'existence de celle-ci au niveau continu demeure un avantage théorique de la méthode. Ce phénomène est d'ailleurs fréquent pour d'autres méthodes d'optimisation fondées sur des principes d'invariance [125]. Signalons qu'à l'instar de [158], on peut essayer d'atténuer la perte d'invariance en discrétisant par un schéma d'ordre plus élevé une version modifiée de (1.29) qui incorpore une correction géodésique afin de garantir que la trajectoire reste en permanence sur la variété, rejoignant ainsi les techniques d'optimisation riemannienne [20]. Mais le calcul des symboles de Christoffel intervenant dans la connection de Levi-Civita étant extrêmement lourd pour un réseau, le jeu n'en vaut pas toujours la chandelle.

Les travaux précurseurs d'Amari [3, 127, 183] ont mis en évidence l'intérêt des méthodes de gradient naturel pour l'apprentissage. Cependant, dès que le nombre de paramètres atteint un certain seuil, on se retrouve avec les mêmes difficultés qu'avant : l'inversion de la matrice de Fisher, qui est pleine, devient l'obstruction principale entravant la performance de la méthode.

1.2.2 Approximation KFAC, avant et après

L'approximation de la matrice de Fisher par sa diagonale, suggérée par [95] ou inspirée de [45, 88, 165] pour la matrice de Fisher empirique, s'avère trop grossière et ne retient pas suffisamment d'informations contenues dans la matrice d'origine. Les tentatives suivantes, cherchant à approcher respectivement la matrice de Fisher empirique (méthode TONGA [96]) et la matrice de Fisher exacte [124] par une structure bloc-diagonale avec un bloc par neurone, n'ont pas apporté d'amélioration significative. Pour rendre la matrice de Fisher plus creuse, certains auteurs optent pour des reparamétrages dynamiques du réseau [135, 173, 178] afin que la plupart des quatités scalaires associées aux unités comme l'activité et la dérivée locale soient nulles en moyennes. Une autre idée dans la même veine consiste modifier adaptativement la représentation interne du réseau au cours de la phase d'apprentissage, dans le but de contrôler le conditionnement de la matrice de Fisher (méthode PRONG [38]).

Les premières approximations au moyen d'un produit de Kronecker sont apparues avec [69, 75, 133], où la matrice de Fisher est remplacée une matrice bloc-diagonale. Cette fois, chaque bloc représente une couche du réseau. Si la couche i possède d_{i-1} entrées et d_i sorties, alors il y a $p_i = (d_{i-1}+1)d_i$ poids et le bloc $F_{i,i} \in \mathbb{R}^{p_i \times p_i}$ de la matrice de Fisher est approchée par

$$F_{i,i} \approx \bar{A}_{i-1} \otimes G_i, \tag{1.30}$$

où $\otimes$ désigne le produit de Kronecker (cf. Définition 3.1) et où les matrices

$$\bar{A}_{i-1} \in \mathbb{R}^{(d_{i-1}+1)\times(d_{i-1}+1)} \qquad \text{et} \qquad G_i \in \mathbb{R}^{d_i \times d_i}$$

sont de dimensions bien plus petites que celle de $F_{i,i}$. Ces facteurs sont beaucoup moins chers à stocker et à inverser. Leur produit est aussi facile et peu coûteux à inverser, grâce à la formule

$$(\bar{A}_{i-1} \otimes G_i)^{-1} = \bar{A}_{i-1}^{-1} \otimes G_i^{-1}. \tag{1.31}$$

L'approximation KFAC de Martens et Grosse [112] s'inscrit dans la même lignée (1.30), avec une définition spécifique pour les facteurs $\bar{A}_{i-1}$ et G_i permettant un calcul plus rapide. Ce qui a aussi concouru à son grand succès face à ses prédécesseurs, c'est sa stratégie permettant de préserver la forme factorisée en vue d'une inversion semblable à (1.31) pour l'indispensable régularization de Tikhonov

$$F_\bullet := F + \lambda I_p, \qquad \lambda > 0. \tag{1.32}$$

Plus précisément, chaque bloc diagonal de la matrice de Fisher régularisée est approché par

$$[F_\bullet]_{i,i} \approx \left(\bar{A}_{i-1} + \pi_i \sqrt{\lambda}\, I_{d_{i-1}+1}\right) \otimes \left(G_i + \pi_i^{-1} \sqrt{\lambda}\, I_{d_i}\right) =: [F_{\bullet\mathrm{KFAC}}]_{i,i}, \tag{1.33}$$

avec un coefficient de répartition π_i ajusté "optimalement" en fonction des données, par exemple

$$\pi_i = \sqrt{\frac{\operatorname{tr}(\bar{A}_{i-1})/d_{i-1}}{\operatorname{tr}(\bar{G}_i)/d_i}}. \tag{1.34}$$

De surcroît, le poids λ est adapté au fur et à mesure selon le comportement des itérés. L'ensemble de ces ingrédients fait de KFAC une méthode puissante et efficace qui a marqué un tournant dans le développement des méthodes du gradient naturel. Conçu initialement pour un perceptron multicouche, KFAC a été généralisé à d'autres architectures de réseau comme CNN [68] (la méthode s'appelle alors KFC) et RNN [111]. Pour chaque nouvelle architecture, de nouvelles hypothèses plus ou moins légitimes sont émises pour définir les facteurs $\bar{A}_{i-1}$ et G_i.

De par ses caractéristiques prometteuses, KFAC a suscité beaucoup de travaux ultérieurs. D'une part, il a été déployé avec succès dans le contexte de l'apprentissage profond bayésien [186], de l'apprentissage par renforcement profond [179] et de l'approximation de Laplace [140]. D'autre part, son implantation parallèle dans la perspective des très grands modèles a fait l'objet de nombreuses recherches. Citons d'abord la version distribuée de [8], où le calcul des gradients est réparti sur plusieurs GPU en utilisant le modèle SGD synchrone standard, tandis que les blocs de la matrice de Fisher et leurs inverses sont calculés avec des CPU de manière asynchrone (pendant que le réseau est toujours en cours d'apprentissage). Dans une autre approche [126], tous les calculs sont effectués de manière synchrone avec les GPU en veillant à communiquer les résultats entre les GPU de manière optimale. Ces deux paradigmes distribués revendiquent une réduction du temps de calcul d'un facteur d'au moins 2 par rapport à un SGD séquentiel.

Au niveau algorithmique, plusieurs tentatives de raffinement de KFAC ont vu le jour. La méthode EKFAC [56] remet à l'échelle les facteurs de Kronecker avec une variance diagonale calculée dans une base de vecteurs propres obtenue par une factorisation de Kronecker. La méthode TKFAC [54] préserve l'invariance de la trace entre la matrice de Fisher approchée et la matrice de Fisher exacte. En supposant que chaque bloc de celle-ci correspond à la covariance d'une distribution normale tensorielle dans le modèle, la méthode TNT [139] met en avant une approximation bloc-diagonale qui présente l'avantage de s'affranchir de la structure des couches. Au-delà de la matrice de Fisher, l'idée de la factorisation de Kronecker peut également être appliquée à l'approximation de la matrice hessienne, comme dans KBFGS [61], où la complexité du calcul de l'inverse des facteurs de Kronecker est atténuée par des mises à jour de rang faible. Elle se généralise aussi à l'approximation de la matrice GGN, comme l'atteste [61].

1.2.3 Pistes d'amélioration et objectifs de la thèse

De cette revue, on tire les observations suivantes sur les méthodes développées autour de KFAC :

1. Elles font appel à certaines hypothèses d'indépendance entre diverses quantités pour déduire les facteurs $\bar{A}_{i-1}$ et G_i. Pour KFAC, par exemple, il est postulé que l'espérance d'un produit de Kronecker est égale au produit de Kronecker de leurs espérances (cf. §1.3.1). Or, ces hypothèses sont des actes de foi. Y aurait-il une manière plus rigoureuse de définir ces facteurs, ce qui déboucherait sur d'autres approximations de $F_{i,i}$?

2. Elles reposent sur une approximation bloc-diagonale de la matrice de Fisher, chaque bloc représentant une couche du réseau. L'interaction entre les couches à travers les blocs hors-diagonaux est totalement négligée. Vaudrait-il le coup de prendre en compte de cette interaction, même partiellement et à une échelle grossière avec surcoût marginal (cf. §1.3.2) dans l'espoir de diminuer le nombre d'itérations nécessaires à l'optimiseur ?

3. Jusqu'à présent, la méthode KFAC n'a jamais été déclinée pour les couches de type convolution transposée. Or, une telle architecture émerge dans les réseaux antagonistes génératifs (GAN) qui sont utilisés à IFPEN pour certaines applications, notamment pour la modélisation de structures sédimentaires. Serait-il possible d'appliquer la méthodologie initiale de KFAC à une convolution transposée, indépendamment des perspectives précédentes ?

Les questions soulevées méritent examen au regard des enjeux plus vastes déjà énumérés. Les deux premières touchent à la robustesse et la vitesse de convergence des méthodes. La dernière ne constitue certes pas un défi du même ordre, mais son intérêt réside dans l'aspect applicatif.

1.3 Contributions et plan du mémoire

1.3.1 Approximations alternatives de la matrice de Fisher

Le point de départ de l'approximation KFAC est l'expression d'un bloc diagonal de la matrice de Fisher comme l'espérance d'un produit de Kronecker. Plus précisément, pour chaque couche i d'un perceptron multicouche (voir §2.1.1 pour la description d'un perceptron multicouche), on a

$$F_{i,i} = \mathbb{E}[\bar{a}_{i-1}\bar{a}_{i-1}^T \otimes g_i g_i^T]. \tag{1.35}$$

où $\bar{a}_{i-1} \in \mathbb{R}^{d_{i-1}+1}$ est le vecteur des activations de la couche précédente $i-1$, $g_i \in \mathbb{R}^{d_i}$ est le vecteur des dérivées par rapport aux préactivations de la couche i. À ce stade, il n'est pas capital de connaître les définitions exactes de $\bar{a}_{i-1}$ et de g_i, qui seront données en §3.1. Ce qu'il importe est de voir est la structure abstraite de (1.35), avec le produit de Kronecker $\otimes$ (Définition 3.1) à l'intérieur du symbole $\mathbb{E}$, notation abrégée pour l'espérance $\mathbb{E}_{x\sim Q_x, y\sim P_{y|x}(\theta)}$ introduite en (1.23).

L'approximation KFAC revient à stipuler

$$F_{i,i} \approx \bar{A}_{i-1}^{\text{KFAC}} \otimes G_i^{\text{KFAC}}, \tag{1.36a}$$

avec

$$\bar{A}_{i-1}^{\text{KFAC}} = \mathbb{E}[\bar{a}_{i-1}\bar{a}_{i-1}^T] \in \mathbb{R}^{(d_{i-1}+1)\times(d_{i-1}+1)}, \qquad G_i^{\text{KFAC}} = \mathbb{E}[g_i g_i^T] \in \mathbb{R}^{d_i \times d_i}. \tag{1.36b}$$

Elle se "justifie" par une hypothèse d'indépendance entre les activations $\bar{a}_{i-1}$ et les dérivées par rapport aux préactivations g_i, ce qui se traduit par la distributivité de l'espérance par rapport au produit de Kronecker. Aussi commode soit-elle, cette hypothèse est tout à fait contestable. Il est souhaitable de s'en dispenser. Pour cela, nous nous efforçons dans le chapitre 3 d'asseoir l'approximation des $F_{i,i}$ sur une base plus solide.

Par nouveau produit de Kronecker. Une idée naturelle est de considérer le produit

$$F_{i,i} \approx \bar{A}_{i-1}^{\text{KPSVD}} \otimes G_i^{\text{KPSVD}}, \tag{1.37a}$$

dont les facteurs sont choisis de sorte à réaliser la meilleure approximation (cf. §3.2.2)

$$(\bar{A}_{i-1}^{\text{KPSVD}}, G_i^{\text{KPSVD}}) = \operatorname*{argmin}_{R\in\mathbb{R}^{d_i\times d_i},\, S\in\mathbb{R}^{(d_{i-1}+1)\times(d_{i-1}+1)}} \|F_{i,i} - R\otimes S\|_F \tag{1.37b}$$

$$= \operatorname*{argmin}_{R\in\mathbb{R}^{d_i\times d_i},\, S\in\mathbb{R}^{(d_{i-1}+1)\times(d_{i-1}+1)}} \|\mathbb{E}[\bar{a}_{i-1}\bar{a}_{i-1}^T \otimes g_i g_i^T] - R\otimes S\|_F, \tag{1.37c}$$

dans la norme de Frobenius $\|\cdot\|_F$. Nous appelons cette première méthode KPSVD, car le problème de minimisation (1.37c) peut être résolu au moyen d'une décomposition en valeurs singulière au sens du produit de Kronecker [171]. La clé de cette résolution est de se ramener à la recherche de la meilleure approximation de rang 1 d'une matrice. Concrètement, toute solution de (1.37) est aussi solution de (Théorème 3.14)

$$(\text{vec}(\bar{A}_{i-1}^{\text{KPSVD}}), \text{vec}(G_i^{\text{KPSVD}})) = \operatorname*{argmin}_{R\in\mathbb{R}^{d_i\times d_i},\, S\in\mathbb{R}^{(d_{i-1}+1)\times(d_{i-1}+1)}} \|\mathcal{Z}(F_{i,i}) - \text{vec}(R)\,\text{vec}(S)^T\|_F, \tag{1.38}$$

où

- l'opérateur "vec", défini par (3.14), ordonne les colonnes successvies d'une matrice en un vecteur, si bien que les inconnues deviennent $\mathrm{vec}(R) \in \mathbb{R}^{d_i^2}$ et $\mathrm{vec}(S) \in \mathbb{R}^{(d_{i-1}+1)^2}$;
- l'opérateur $\mathcal{Z}$, défini par (3.23), réarrange une matrice par bloc de $\mathbb{R}^{(d_{i-1}+1)d_i \times (d_{i-1}+1)d_i}$ pour en faire une matrice de $\mathbb{R}^{d_i^2 \times (d_{i-1}+1)^2}$.

Dès lors, il est connu qu'une solution de (1.38) peut être extraite de la SVD ordinaire

$$\mathcal{Z}(F_{i,i}) = U\Sigma V^T, \qquad \Sigma = \mathrm{diag}(\sigma_1, \sigma_2, \ldots), \qquad \sigma_1 \geqslant \sigma_2 \geqslant \ldots \geqslant 0 \tag{1.39}$$

de $\mathcal{Z}(F_{i,i})$ par

$$\mathrm{vec}(\bar{A}_{i-1}^{\mathrm{KPSVD}}) = \sqrt{\sigma_1} u_1, \qquad \mathrm{vec}(G_i^{\mathrm{KPSVD}}) = \sqrt{\sigma_1} v_1, \tag{1.40}$$

où u_1 (première colonne de U) et v_1 (première colonne de V) sont les vecteurs propres à gauche et à droite associés à la plus grande valeur singulière σ_1. Toute la question est de savoir si l'on peut calculer (σ_1, u_1, v_1) efficacement. La réponse est affirmative, grâce à l'algorithme des puissances itérées (cf. §3.B.1 et [146]) dans lequel les produits matrice-vecteur $\mathcal{Z}(F_{i,i})v$ et $\mathcal{Z}(F_{i,i})^T u$ peuvent être effectuées sans former $F_{i,i}$ ou $\mathcal{Z}(F_{i,i})$ par l'intermédiaire des formules (Proposition 3.1)

$$\mathcal{Z}(F_{i,i})v = \mathbb{E}[\, g_i^T \mathrm{MAT}(v) g_i \,\mathrm{vec}(\bar{a}_{i-1}\bar{a}_{i-1}^T)\,], \tag{1.41a}$$

$$\mathcal{Z}(F_{i,i})^T u = \mathbb{E}[\, \bar{a}_{i-1}^T \mathrm{MAT}(u) \bar{a}_{i-1} \,\mathrm{vec}(g_i g_i^T)\,], \tag{1.41b}$$

où l'opérateur "MAT" fait l'inverse de "vec" en convertissant un (long) vecteur en une matrice.

En appliquant cet opérateur "MAT" au couple de vecteurs de type (1.40), on obtient les facteurs $(\bar{A}_{i-1}^{\mathrm{KPSVD}}, G_i^{\mathrm{KPSVD}})$ désirés. Ici surgit un doute concernant la symétrie et le caractère semi-défini positif de ces matrices. Nous montrons (Proposition 3.2) que la symétrie découle automatiquement de la procédure indiquée, tandis qu'il est possible de modifier le minimiseur obtenu pour passer à un autre minimiseur de (1.37c) pour lequel les matrices $(\bar{A}_{i-1}, G_i)$ sont semi-définies positives. En présence d'une régularisation de Tikhonov, l'approximation de la matrice de Fisher régularisée se fait de manière analogue à (1.33)–(1.34), à savoir

$$[F_\bullet]_{i,i} \approx \big(\bar{A}_{i-1}^{\mathrm{KPSVD}} + \pi_i \sqrt{\lambda}\, I_{d_{i-1}+1}\big) \otimes \big(G_i^{\mathrm{KPSVD}} + \pi_i^{-1} \sqrt{\lambda}\, I_{d_i}\big) =: [F_{\bullet \mathrm{KPSVD}}]_{i,i}, \tag{1.42}$$

avec par exemple

$$\pi_i = \sqrt{\frac{\mathrm{tr}(\bar{A}_{i-1}^{\mathrm{KPSVD}})/d_{i-1}}{\mathrm{tr}(\bar{G}_i^{\mathrm{KPSVD}})/d_i}}. \tag{1.43}$$

Par une somme de deux produits de Kronecker. Dans le prolongement de KPSVD, on peut envisager de chercher la meilleure approximation de $F_{i,i}$ par une somme de deux produits de Kronecker, c'est-à-dire

$$F_{i,i} \approx \bar{A}_{i-1}^{(1)} \otimes G_i^{(1)} + \bar{A}_{i-1}^{(2)} \otimes G_i^{(2)}, \tag{1.44a}$$

avec (cf. §3.2.3)

$$(\bar{A}_{i-1}^{(1)}, G_i^{(1)}, \bar{A}_{i-1}^{(2)}, G_i^{(2)}) = \underset{R,S,P,Q}{\mathrm{argmin}} \, \|F_{i,i} - (R \otimes S + P \otimes Q)\|_F. \tag{1.44b}$$

L'idée est de disposer non seulement du terme "d'ordre 1", représenté par $\bar{A}_{i-1}^{(1)} \otimes G_i^{(1)}$, mais aussi d'un terme supplémentaire "d'ordre 2", incarné par $\bar{A}_{i-1}^{(2)} \otimes G_i^{(2)}$, afin d'affiner l'approximation.

Là encore, les opérateurs de vectorialisation "vec" et de réarrangement $\mathcal{Z}$ permettent de ramener le problème (1.44b) à celui de l'approximation de rang 2

$$(\mathrm{vec}(\bar{A}_{i-1}^{(1)}), \mathrm{vec}(G_i^{(1)}), \mathrm{vec}(\bar{A}_{i-1}^{(2)}), \mathrm{vec}(G_i^{(2)}) \\ = \operatorname*{argmin}_{R,S,P,Q} \|\mathcal{Z}(F_{i,i}) - (\mathrm{vec}(R)\,\mathrm{vec}(S)^T + \mathrm{vec}(P)\,\mathrm{vec}(Q)^T)\|_F. \quad (1.45)$$

À nouveau, la décomposition en valeurs singulières de $\mathcal{Z}(F_{i,i})$ fournit un minimiseur, cette fois en ne gardant que les deux premiers modes singuliers. Autrement dit,

$$\mathrm{vec}(\bar{A}_{i-1}^{(1)}) = \sqrt{\sigma_1}u_1, \qquad \mathrm{vec}(G_i^{(1)}) = \sqrt{\sigma_1}v_1, \quad (1.46a)$$
$$\mathrm{vec}(\bar{A}_{i-1}^{(2)}) = \sqrt{\sigma_2}u_2, \qquad \mathrm{vec}(G_i^{(2)}) = \sqrt{\sigma_2}v_2. \quad (1.46b)$$

Quant à la détermination effective des deux premiers modes singuliers, elle peut se faire par deux algorithmes :

1. *Déflation.* Puisque le premier mode singulier est le même que dans KPSVD, on a

$$(\mathrm{vec}(\bar{A}_{i-1}^{(1)}), \mathrm{vec}(G_i^{(1)})) = (\mathrm{vec}(\bar{A}_{i-1}^{\mathrm{KPSVD}}), \mathrm{vec}(G_i^{\mathrm{KPSVD}})), \quad (1.47)$$

   qu'on calcule par les puissances itérées comme décrite précédemment. Une fois cette étape accomplie, on applique les puissances itérées pour trouver la meilleure approximation $P \otimes Q$ de la différence $F_{i,i} - \bar{A}_{i-1}^{(1)} \otimes G_i^{(1)}$. Dans cette seconde étape, les produits matrice-vecteur

$$\mathcal{Z}(F_{i,i} - \bar{A}_{i-1}^{(1)} \otimes G_i^{(1)})v \quad \text{et} \quad \mathcal{Z}(F_{i,i} - \bar{A}_{i-1}^{(1)} \otimes G_i^{(1)})^T u$$

   peuvent encore être évalués efficacement en exploitant les relations

$$\mathcal{Z}(R \otimes S)v = \langle \mathrm{vec}(S), v \rangle \,\mathrm{vec}(R), \qquad \mathcal{Z}(R \otimes S)^T u = \langle \mathrm{vec}(R), u \rangle \,\mathrm{vec}(S). \quad (1.48)$$

2. *Bidiagonalisation.* On applique l'algorithme de bidiagonalisation de Golub-Kahan-Lanczos [62] ou une version avec redémarrage [146] pour calculer les deux premiers modes singuliers simultanément.

Enfin, une variante de (1.44) consiste à figer le terme d'ordre 1 à la valeur de l'approximation KFAC et de calculer le terme d'ordre 2 comme étant la meilleure correction ad hoc possible. En d'autres termes,

$$F_{i,i} \approx \bar{A}_{i-1}^{\mathrm{KFAC}} \otimes G_i^{\mathrm{KFAC}} + \bar{A}_{i-1}^{\mathrm{corr}} \otimes G_i^{\mathrm{corr}}, \quad (1.49a)$$

avec (cf. §3.2.4)

$$(\bar{A}_{i-1}^{\mathrm{corr}}, G_i^{\mathrm{corr}}) = \operatorname*{argmin}_{P,Q} \|F_{i,i} - \bar{A}_{i-1}^{\mathrm{KFAC}} \otimes G_i^{\mathrm{KFAC}} - P \otimes Q\|_F. \quad (1.49b)$$

Comme avant, un minimiseur de (1.49b) peut être obtenu en appliquant l'algorithme des puissances itérées à la matrice $\mathcal{Z}(F_{i,i} - \bar{A}_{i-1}^{\mathrm{KFAC}} \otimes G_i^{\mathrm{KFAC}})$. Nous appelons cette dernière méthode KFAC-Corrigé. Pour celle-ci comme pour les autres variantes impliquant une somme de deux produits de Kronecker, il faut noter que :

▷ La prise en compte d'une régularisation de Tikhonov se fait uniquement à travers le premier produit de Kronecker, c'est-à-dire

$$[F_\bullet]_{i,i} \approx \big(\bar{A}_{i-1}^{(0)} + \pi_i\sqrt{\lambda}\,I\big) \otimes \big(G_i^{(0)} + \pi_i^{-1}\sqrt{\lambda}\,I\big) + \bar{A}_{i-1}^{(1)} \otimes G_i^{(1)} =: [F_{\bullet\text{variante}}]_{i,i}, \tag{1.50}$$

avec π_i donné par une formule analogue à (1.34).

▷ L'inversion de $[F_{\bullet\text{variante}}]_{i,i}$ doit suivre un procédé différent de (1.31). Bien qu'il y ait plusieurs possibilités, celle que nous adoptons en §3.2.5 semble être la mieux adaptée au problème, dans la mesure où elle tire profit de la symétrie et du caractère défini positif des facteurs du premier produit de Kronecker.

Les méthodes KPVSD, Déflation, Bidiagonalisation et KFAC-Corrigé ont été étudiées sur trois types d'autoencodeurs utilisant respectivement les données CURVES, MNIST et FACE (cf. §3.D). Les résultats, présentés et commentés en §3.3, ont fait l'objet d'un article [90] à paraître dans *ESAIM: Proceedings and Surveys*. Le chapitre 3 reproduit cet article en ajoutant une section (cf. §3.4) consacrée au cas des réseaux de neurones convolutifs (CNN).

Par des hypothèses minimales pour les CNN. Lorsque la couche i est une convolution avec plusieurs canaux d'entrée et de sortie, les paramètres inconnus les coefficients des filtres, sont partagés et donc en petit nombre. Depuis [27], il est coutume de dupliquer les activations et les dérivées par rapport aux préactivations autour de chaque position de l'image afin de bénéficier d'une plus grande localité. On se retrouve pour chaque emplacement $t \in \mathcal{T}_i$ avec des vecteurs $\bar{a}_{i-1,t}$ et $g_{i,t}$ regroupant les quantités correspondantes dans un voisinage de t et concaténant les canaux. Le i-ème bloc diagonal de la matrice de Fisher est alors égal à

$$F_{i,i} = \sum_{t\in\mathcal{T}_i}\sum_{t'\in\mathcal{T}_i} \mathbb{E}\big[\bar{a}_{i-1,t}\bar{a}_{i-1,t'}^T \otimes g_{i,t}g_{i,t'}^T\big]. \tag{1.51}$$

L'approximation KFC [68], qui généralise KFAC aux CNN, fait appel à trois hypothèses. Outre l'indépendance entre les activations et les dérivées par rapport aux préactivations (IAD) qui a déjà servi dans KFAC, les deux hypothèses supplémentaires portent sur l'homogénéité spatiale (SH) et des dérivées spatialement décorrélées (SUD). Sans s'attarder sur leurs contenus, il suffit de savoir que leur but ultime est de simplifier le calcul du second-membre de (1.51) en réduisant la double somme à une seule somme, puis à un unique produit de Kronecker.

Pour nous émanciper de ces hypothèses elles aussi discutables, nous commençons par remarquer dans les approximations alternatives proposées, l'essentiel de la lourdeur des calculs se trouve dans l'évaluation des produits matrice-vecteur $\mathcal{Z}(F_{i,i})v$ et $\mathcal{Z}(F_{i,i})^T u$ pour l'algorithme des puissances itérées. En l'occurrence, ceux-ci valent exactement

$$\mathcal{Z}(F_{i,i})v = \sum_{t\in\mathcal{T}_i}\sum_{t'\in\mathcal{T}_i} \mathbb{E}\big[\big(g_{i,t}^T\mathrm{MAT}(v)g_{i,t'}\big)\bar{a}_{i-1,t}\bar{a}_{i-1,t'}^T\big], \tag{1.52a}$$

$$\mathcal{Z}(F_{i,i})^T u = \sum_{t\in\mathcal{T}_i}\sum_{t'\in\mathcal{T}_i} \mathbb{E}\big[\big(\bar{a}_{i-1,t}^T\mathrm{MAT}(u)\bar{a}_{i-1,t'}\big)g_{i,t}g_{i,t'}^T\big], \tag{1.52b}$$

et c'est la double somme sur les positions $(t,t') \in \mathcal{T}_i^2$ qui coûte cher. De cette observation naît l'espoir qu'une approximation raisonnable de la double somme peut émerger en sommant seulement sur une bande

$$\mathcal{T}_i^2(r) = \big\{(t,t') \in \mathcal{T}_i \times \mathcal{T}_i \;:\; \|t-t'\|_\infty \leqslant r\big\} \tag{1.53}$$

contenant les couples (t, t') dont les éléments ne sont pas trop éloignés l'un de l'autre. Physiquement, cela correspond à l'intuition que plus deux pixels sont proches, plus ils sont fortement corrélés. Au-delà d'une distance critique r, on peut négliger leur corrélation. Cette hypothèse, qui paraît plus facile à accepter que (SH) et (SUD), mène finalement aux approximations

$$\mathcal{Z}(F_{i,i})v \approx \sum_{(t,t')\in\mathcal{T}_i^2(r)} \mathbb{E}\big[\big(g_{i,t}^T \mathrm{MAT}(v) g_{i,t'}\big)\bar{a}_{i-1,t}\bar{a}_{i-1,t'}^T\big], \tag{1.54a}$$

$$\mathcal{Z}(F_{i,i})^T u \approx \sum_{(t,t')\in\mathcal{T}_i^2(r)} \mathbb{E}\big[\big(\bar{a}_{i-1,t}^T \mathrm{MAT}(u) \bar{a}_{i-1,t'}\big) g_{i,t} g_{i,t'}^T\big]. \tag{1.54b}$$

Les résultats de cet essai sont présentés et commentés en §3.4.3.

1.3.2 Réintégration partielle de l'interaction entre les couches

Repartons de l'approximation

$$F_{\bullet\,\mathrm{KFAC}} = \mathrm{diag}([F_{\bullet\,\mathrm{KFAC}}]_{1,1}, [F_{\bullet\,\mathrm{KFAC}}]_{2,2}, \ldots, [F_{\bullet\,\mathrm{KFAC}}]_{\ell,\ell}). \tag{1.55}$$

de la matrice de Fisher régularisée $F_\bullet$, où ℓ représente le nombre de couches, et essayons de l'affiner dans une autre direction. Puisque la nature bloc-diagonale de (1.55) induit la perte totale des informations concernant la corrélation entre les couches, on peut se demander s'il serait opportun de récupérer une partie des informations perdues dans l'espoir de faire évoluer la matrice approchée vers la vraie matrice $F_\bullet$.

La voie que nous empruntons est celle d'une correction "grossière", qui consiste à ajouter dans la matrice inverse $F_{\bullet\mathrm{KFAC}}^{-1}$ un terme représentant la corrélation entre les couches à une échelle "macroscopique", par opposition à l'échelle "microscopique" des interactions entre les neurones au sein d'une couche. Elle procède par analogie avec les stratégies de préconditionnement à deux niveaux en décomposition de domaines [42], avec toutefois des différences notoires sur lesquelles nous reviendrons. À notre connaissance, la première et unique application d'une telle technique au contexte de l'apprentissage automatique est due à Tselepidis et al. [167], qui cherchaient justement à améliorer l'approximation KFAC. Nous souligerons également les différences entre notre contribution et celle de [167].

Auparavant, il est nécessaire d'introduire quelques notations. Pour chaque $i \in \{1, \ldots, \ell\}$, soit $R_i \in \mathbb{R}^{p_i \times p}$ l'opérateur de restriction

$$(R_i)_{\xi\eta} = \begin{cases} 1 & \text{si } \eta = p_1 + \ldots + p_{i-1} + \xi, \\ 0 & \text{sinon.} \end{cases} \tag{1.56}$$

Cette notion permet d'exprimer $F_{\bullet\,\mathrm{KFAC}}^{-1}$ de manière globale par

$$F_{\bullet\,\mathrm{KFAC}}^{-1} = \sum_{i=1}^{\ell} R_i^T [F_{\bullet\,\mathrm{KFAC}}]_{i,i}^{-1} R_i. \tag{1.57}$$

Cette écriture peut sembler artificielle, mais est courante en décomposition de domaines où chaque indice i correspond à un sous-domaine[5]. En poursuivant l'analogie[6], on peut voir chaque $[F_{\bullet\,\mathrm{KFAC}}]_{i,i}^{-1}$ comme un solveur local.

[5] Les sous-domaines peuvent avoir des recouvrements et les matrices R_i peuvent être moins triviales que (1.56).

[6] L'analogie n'est pas parfaite car ici on ne peut pas raisonner sur un problème physique continu, modélisé par une EDP dont la discrétisation donne lieu à la matrice.

Un problème bien connu avec les préconditionneurs de la forme (1.57), dits *à un niveau*, est leur scalabilité parallèle [42] : le facteur d'accélération ne croît pas proportionnellement par rapport au nombre de sous-domaines (donc, de processeurs). La raison en est que, quand le nombre de sous-domaines augmente, il faut davantage d'itérations pour qu'une information locale à un sous-domaine puisse être propagée aux autres sous-domaines. Le remède habituel à cette difficulté est d'ajouter une correction globale afin que les sous-domaines puissent communiquer entre eux plus rapidement. Les nouveaux préconditionneurs sont dits *à deux niveaux*.

Par une correction grossière consistante. À la place de $F_{\bullet\,\mathrm{KFAC}}^{-1}$, nous proposons de considérer (cf. §4.2.2)

$$F_{\bullet\,\mathrm{KFAC\text{-}2L}}^{-1} = F_{\bullet\,\mathrm{KFAC}}^{-1} + R_0^T F_{\mathrm{coarse}}^{-1} R_0 (I - F_\bullet F_{\bullet\,\mathrm{KFAC}}^{-1}), \tag{1.58}$$

où

- La matrice $R_0 \in \mathbb{R}^{N\times p}$ est à choisir convenablement, avec $N \geqslant \ell$. Le sous-espace de $\mathbb{R}^p$ engendré par les N colonnes de R_0^T est appelé *espace grossier*. Plusieurs possibilités pour R_0 sont résumées ci-dessous (cf. §4.2.3).
- Le solveur grossier
$$F_{\mathrm{coarse}} = R_0 F_\bullet R_0^T \tag{1.59}$$
correspond à la "projection" sur R_0 de la matrice $F_\bullet$.

La valeur du dernier terme au second-membre de (1.58) provient de la minimisation d'une norme de résidu associé à la résolution du système linéaire

$$F_\bullet \zeta = \nabla_\theta h \tag{1.60}$$

dans lequel ζ représente $\theta_k - \theta_{k+1}$, l'opposé de l'incrément d'une itération de l'optimiseur. En partant de la solution résultant de l'approximation KFAC

$$\zeta_{\mathrm{KFAC}} = F_{\bullet\,\mathrm{KFAC}}^{-1} \nabla_\theta h, \tag{1.61}$$

nous cherchons la meilleure correction possible par un vecteur de l'espace grossier

$$\zeta_{\mathrm{KFAC\text{-}2L}} = \zeta_{\mathrm{KFAC}} + R_0^T \beta^*, \tag{1.62}$$

de sorte à réaliser

$$\beta^* = \operatorname*{argmin}_{\beta\in\mathbb{R}^\ell} \|(\zeta_{\mathrm{KFAC}} + R_0^T\beta) - \zeta\|_{F_\bullet}^2 = \operatorname*{argmin}_{\beta\in\mathbb{R}^\ell} \|F_\bullet(\zeta_{\mathrm{KFAC}} + R_0^T\beta) - \nabla_\theta h\|_{F_\bullet^{-1}}^2. \tag{1.63}$$

La solution du problème de moindres carrées (1.63) est donnée par

$$\beta^* = (R_0 F_\bullet R_0^T)^{-1} R_0 (\nabla_\theta h - F_\bullet \zeta_{\mathrm{KFAC}}). \tag{1.64}$$

En combinant avec (1.61), on obtient la correction de l'incrément

$$R_0^T\beta^* = R_0^T F_{\mathrm{coarse}}^{-1} R_0^T (I - F_\bullet F_{\bullet\,\mathrm{KFAC}}^{-1}) \nabla_\theta h, \tag{1.65}$$

d'où la matrice inverse corrigée (1.58).

Par rapport à la corection proposée par Tselepidis, Kohler et Orvieto [167], qui s'écrit

$$F_{\bullet\mathrm{TKO}}^{-1} = F_{\bullet\mathrm{KFAC}}^{-1} + R_0^T \bar{F}_{\mathrm{coarse}}^{-1} R_0, \tag{1.66}$$

il y a deux différences. La première, mineure, concerne la définition de l'opérateur grossier. Les auteurs de [167] utilisent $\bar{F}_{\mathrm{coarse}} = R_0 \bar{F}_\bullet R_0^T$ avec

$$[\bar{F}_\bullet]_{i,j} = \begin{cases} \mathbb{E}[\bar{a}_{i-1}\bar{a}_{j-1}^T] \otimes \mathbb{E}[g_i g_j^T] & \text{si } \; i \neq j, \\ (\bar{A}_{i-1} + \pi_i \lambda^{1/2} I) \otimes (G_i + \pi_i^{-1} \lambda^{1/2} I) & \text{si } \; i = j. \end{cases} \tag{1.67}$$

La seconde, majeure, est l'absence d'un facteur multiplicatif analogue à $I - F_\bullet F_{\bullet\mathrm{KFAC}}^{-1}$ dans le correcteur. Cela rend la correction (1.66) *inconsistante*, alors que celle en (1.58) est *consistante*. Par ce vocable, nous entendons ceci : si $F_{\bullet\mathrm{KFAC}}$ constitue déjà une approximation de $F_\bullet^{-1}$, alors $I - F_\bullet F_{\bullet\mathrm{KFAC}}^{-1}$ est une approximation de la matrice nulle. Il s'ensuit que dans (1.58), le dernier terme est homogène à zéro et la matrice au second-membre est homogène à $F_\bullet^{-1}$. En revanche, ce n'est pas le cas pour (1.66). Les deux termes au second-membre sont tous homogènes à $F_\bullet^{-1}$, si bien que leur somme se comporte en $2F_\bullet^{-1}$!

En décomposition de domaines, un correcteur de type (1.66) est appelé *additif*, tandis qu'un correcteur de type (1.58) est appelé *multiplicatif*. Le fait qu'un correcteur additif n'est pas consistant avec l'inverse de la matrice considérée n'est pas gênant, tant qu'on utilise la matrice inverse corrigée seulement comme *préconditionneur* : lors de la résolution du système, on fait encore appel à la matrice exacte (via la multiplication avec un vecteur) et la solution obtenue demeure exacte. Dans notre cas, les règles du jeu sont différentes : il s'agit d'oublier la matrice exacte en remplaçant d'emblée son inverse par une approximation avant la résolution du système linéaire. Cette résolution n'étant pas exacte, toute inconsistance dans l'approximation de l'inverse peut avoir de graves conséquences sur la validité de la solution approchée.

Par des espaces grossiers plus variés. Quel que soit le choix de l'espace grossier R_0^T, la construction précédente diminue l'erreur par rapport à la solution exacte, à savoir

$$\|\zeta_{\text{KFAC-2L}} - \zeta\|_{F_\bullet}^2 \leqslant \|\zeta_{\text{KFAC}} - \zeta\|_{F_\bullet}^2. \tag{1.68}$$

Le choix effectif de R_0^T est un compromis entre deux injonctions contraires : réduire significativement l'erreur et garder une faible dimension N. Nous préconisons la forme a priori

$$R_0^T = \begin{bmatrix} V_1 & 0 & \dots & \dots & 0 \\ 0 & V_2 & \dots & \dots & 0 \\ \vdots & \vdots & \ddots & & \vdots \\ 0 & 0 & \dots & \dots & V_\ell \end{bmatrix} \in \mathbb{R}^{p \times N}, \tag{1.69}$$

où chaque $V_i \in \mathbb{R}^{p_i \times N_i}$ possède N_i colonnes avec $N_i \ll p_i$. Autrement dit, chaque bloc i suggère un petit nombre N_i de directions à explorer, le long desquelles sont condensées les interactions entre les couches. Ci-dessous nous décrivons les 4 choix d'espace grossier, fondés sur des justifications plus ou moins heuristiques (cf. §4.2.3).

▷ *Nicolaides.* Historiquement, c'est le premier espace grossier introduit en décomposition de domaines [121]. Il correspond à $N_1 = \ldots = N_\ell = 1$ et $V_i = [1, \ldots, 1]^T \in \mathbb{R}^{p_i}$. Ce vecteur est la discrétisation d'un état propre correspondant à la valeur propre 0 de l'opérateur

continu $-\nabla \cdot (\kappa \nabla)$. L'incorporer dans R_0^T permet de prendre en compte le mode de plus basse fréquence. Dans notre problème, cependant, il n'y aucune raison ni pour que 0 soit valeur propre, ni pour que $\mathbf{1}$ soit vecteur propre. Le choix de l'espace grossier de Nicolaides par Tselepidis et al. [167] est donc simplement motivé par le fait qu'il est commode.

▷ *Spectral.* Le principe est toujours de capter le mode de plus basse fréquence [117], mais comme la plus petite valeur propre et son vecteur propre ne sont pas connus à l'avance, il faut les calculer. On prend donc $N_1 = \ldots = N_\ell = 1$ et $V_i =$ vecteur propre associé à la plus petite valeur propre de $[F_{\bullet \mathrm{KFAC}}]_{i,i}$. Dans notre cas, il est possible d'exploiter la structure du produit de Kronecker de $[F_{\bullet \mathrm{KFAC}}]_{i,i}$ pour faciliter le calcul du vecteur propre.

▷ *Krylov.* Une variante de l'espace grossier spectral est envisageable si l'on souhaite éviter le calcul du vecteur propre associé à la plus petite valeur propre de $[F_{\bullet \mathrm{KFAC}}]_{i,i}$. En effet, on sait que ce vecteur propre peut être obtenu par la méthode des puissances inverses. L'idée est alors d'effectuer quelques itérations de cette méthode, voire une ou deux, et de mettre les itérés dans la base grossière. Ainsi, si $N_i - 1 \geqslant 1$ est le nombre d'itérations de la méthode des puissances inverses appliquées à $[F_{\bullet \mathrm{KFAC}}]_{i,i}$, alors

$$V_i = [v_i, \ [F_{\bullet \mathrm{KFAC}}]_{i,i}^{-1} v_i, \ \ldots, \ [F_{\bullet \mathrm{KFAC}}]_{i,i}^{-(N_i-1)} v_i]^T \in \mathbb{R}^{p_i \times N_i} \tag{1.70}$$

où $v_i \in \mathbb{R}^{p_i}$ est un vecteur arbitraire, supposé ne pas être colinéaire à un vecteur propre de $[F_{\bullet \mathrm{KFAC}}]_{i,i}$ pour que les colonnes de V_i soient indépendantes.

▷ *Résiduel.* Une philosophie radicalement différente que nous souhaitons promouvoir est la suivante. Au lieu de nous focaliser sur le vecteur propre associé à la plus petite propre, nous portons l'attention sur la qualité de l'approximation de $(\zeta_{\mathrm{KFAC}} + R_0^T \beta) - \zeta$. Nous remarquons que si $\zeta - \zeta_{\mathrm{KFAC}}$ appartenait au sous-espace engendré par les colonnes de R_0^T, alors le minimiseur de (1.63) serait idéal. Malheureusement, nous ne pouvons pas ajouter directement $\zeta - \zeta_{\mathrm{KFAC}}$ dans R_0^T car ζ n'est pas connu. Toutefois, nous savons que

$$\zeta - \zeta_{\mathrm{KFAC}} = F_\bullet^{-1} r_{\mathrm{KFAC}}, \tag{1.71}$$

où le résidu $r_{\mathrm{KFAC}} = \nabla_\theta h - F_\bullet \zeta_{\mathrm{KFAC}}$ n'est pas trop cher à calculer puisqu'il n'implique que le produit matrice-vecteur direct. C'est l'inversion par $F_\bullet^{-1}$ qui constitue l'obstacle au calcul de $\zeta - \zeta_{\mathrm{KFAC}}$. Mais nous pouvons approcher cette erreur en inversant avec $F_{\bullet \mathrm{KFAC}}^{-1}$ au lieu de $F_\bullet^{-1}$. Par conséquent, nous proposons de construire R_0^T de sorte que ses colonnes puissent engendrer le vecteur $F_{\bullet \mathrm{KFAC}}^{-1} r_{\mathrm{KFAC}}$. Pour cela, nous découpons ce dernier en ℓ morceaux, chacun correspondant à une couche. Cela revient à prescrire

$$V_i = [F_{\bullet \mathrm{KFAC}}]_{i,i}^{-1} r_{\mathrm{KFAC}}[i] \in \mathbb{R}^{p_i}, \qquad r_{\mathrm{KFAC}}[i] = \nabla_\theta h[i] - (F_\bullet \zeta_{\mathrm{KFAC}})[i], \tag{1.72}$$

où $\xi[i] = \xi(p_{i-1} + 1 : p_i)$ désigne la portion du vecteur $\xi \in \mathbb{R}^p$ relative à la couche i.

Les résultats de cette étude sont présentés et commentés en §4.3.

1.3.3 Application de KFAC aux réseaux antagonistes génératifs

En moins d'une décennie, les réseaux antagonistes génératifs (GANs) [64] se sont imposés comme un paradigme élégant et puissant pour générer des données synthétiques atteignant un très haut degré de "ressemblance" avec des échantillons réels. Ils sont utilisés dans de nombreuses applications à IFPEN.

L'originalité de leur fonctionnement réside dans la compétition entre deux réseaux neuronaux : un générateur G qui crée de "faux" échantillons, et un discriminateur D dont le rôle est de les détecter. En jouant l'un contre l'autre, les deux s'améliorent jusqu'à ce que le discriminateur ne soit plus en mesure de distinguer "le bon grain de l'ivraie". L'entraînement des GANs peut ainsi être formulé comme un problème d'optimisation minimax de la forme

$$\min_{\theta_G} \max_{\theta_D} \mathbb{E}_{x\sim Q_x}\big[\log(D(x;\theta_D))\big] + \mathbb{E}_{z\sim P_z}\big[\log(1 - D(G(z;\theta_G);\theta_D))\big], \tag{1.73}$$

où θ_G et θ_D désignent respectivement les poids du générateur et discriminateur. Q_x représente la distribution des données réelles. Quant à P_z, il désigne une distribution définie a priori régissant les bruits utilisés par le générateur.

Malgré leur succès remarquable, les GANs sont malheureusement réputés difficiles et instables à entraîner. Cela s'explique par le fait que la résolution du problème (1.73) dans le cadre des GANs équivaut à trouver un équilibre de Nash [137] dans un problème d'optimisation mini-max non convexe, non concave et en grande dimension. Plusieurs évolutions se sont succédé dans le but de stabiliser l'entraînement des GANs, notamment fGAN (cf. §5.1.1) et WGAN (cf. 5.1.3), mais peu s'intéressent aux méthodes d'optimisation utilisées pour résoudre le problème (1.73).

À cause la présence de nombreuses singularités (point-selle, optimum local, plateau) dans le paysage de la fonction objectif, les méthodes du premier ordre couramment utilisées pour les problèmes de type minimax comme le *gradient descendant ascendant* (GDA) ou sa version stochastique (SGDA) présentent de sérieuses limites dans le cas des GANs. En accédant aux informations de courbure, les méthodes du second ordre peuvent échapper à ces singularités et avoir une meilleure chance de converger, constituant ainsi un remède aux difficultés rencontrées.

Dans le contexte des GANs, les méthodes de gradient adaptatif comme Adam ont démontré leur efficacité par rapport à SGDA et restent donc des méthodes privilégiées par les praticiens. Plusieurs auteurs [82,106] attribuent les performances supérieures de Adam par rapport à SGDA à sa vitesse de convergence plus élevée. Cependant, les performances des méthodes du second ordre dans le cadre de l'optimisation des GANs n'ont pas encore été étudiées de manière approfondie. Des travaux récents [15,48,50] ont étudié le comportement de l'entraînement des GANs en utilisant uniquement les valeurs propres de matrice hessience et sont parvenus à la conclusion que les informations de courbure peuvent aider à améliorer l'entraînement des GANs.

Nous proposons d'examiner au chapitre 5 la performance de la méthode de type gradient naturel pour l'entraînement des GANs, ce qui semble n'avoir jamais entrepris dans la littérature. À cette fin, nous choisissons la version de base de KFAC plutôt que les variantes développées en §3–§4, car nous avons vu que les premières ont généralement des performances inférieures à KFAC sur les réseaux neuronaux convolutionnels, tandis que les secondes ont des performances comparables à KFAC malgré des coûts de calcul supplémentaires.

Extension de KFAC à une convolution transposée. Étant donné que les convolutions transposées constituent un élément essentiel dans l'architecture des GANs, nous commençons par étendre la méthode KFAC aux couches de convolution transposée. À notre connaissance, une telle extension n'a jamais été envisagée jusqu'à présent.

Pour cela, nous procédons par analogie avec les convolutions traditionnelles. En effet, une convolution transposée peut être implémentée en utilisant une approche similaire à celle d'une convolution traditionnelle, dont nous allons rappeler brièvement le principe. Lors d'une convolution, un noyau (ou filtre) est appliqué à une entrée (par exemple, une image) pour produire une sortie de dimensions spatiales inférieures à celles de l'entrée. Le noyau glisse sur l'entrée

en multipliant les valeurs d'entrée par les poids correspondants du noyau, puis en sommant les résultats pour produire une valeur de sortie dans le résultat convolué. Dans une convolution transposée, le processus est essentiellement inversé. Au lieu de réduire la dimension de l'entrée, une convolution transposée l'augmente. En partant d'une entrée de taille réduite (par exemple, une image avec une dimension faible), elle applique un filtre à cette entrée de manière similaire à une convolution traditionnelle. Cependant, au lieu de sommer les résultats, elle effectue une opération de dilatation pour produire une sortie de taille supérieure à l'entrée.

Au regard de ce qui précède et à quelques adaptations près, ces deux opérations peuvent être implémentées de manière analogue. Concrètement, on dilate l'entrée en insérant des zéros entre ses éléments et en reformattant le tenseur des filtres. Une fois ces modifications faites, on implémente la couche de convolution transposée en utilisant le même procédé qu'une convolution traditionnelle. En un langage plus formel, si nous avons une couche de convolution transposée avec une entrée $\mathcal{A}$, un tenseur de filtre $\mathcal{F}$, un padding τ et un pas e, alors

$$\text{convolution transposée}(\mathcal{A}, \mathcal{F}, \tau, e) = \text{convolution}(\mathcal{A}', \mathcal{F}', \tau', e', \bar{e}),$$

où $\mathcal{A}'$, $\mathcal{F}'$, τ' et $(e', \bar{e})$ correspondent à des modifications de $\mathcal{A}$, $\mathcal{F}$, τ et e respectivement (cf. §5.2.2 pour plus de détails).

Pour mettre en œuvre KFAC, rappelons d'abord que pour une couche de convolution traditionnelle ayant pour poids W' (qui représente la version matricielle du tenseur des filres $\mathcal{F}'$), l'itération KFAC associée à la couche (cf. §3.4.1) est donnée par

$$W'_{k+1} = W'_k - \alpha_k (G'_k)^{-1} \nabla_{W'_k} h(\theta'_k) (\bar{A}'_k)^{-1}, \tag{1.74}$$

avec

$$\bar{A}'_k = \mathbb{E}\Big[\sum_{t \in \mathcal{T}'} (\bar{a}'_t)_k (\bar{a}'_t)_k^T \Big], \qquad G'_k = \frac{1}{|\mathcal{T}'|} \mathbb{E}\Big[\sum_{t \in \mathcal{T}'} (g'_t)_k (g'_t)_k^T \Big]. \tag{1.75}$$

Considérons à présent une couche de convolution transposée avec les entrées $\mathcal{A}$, $\mathcal{F}$, τ et e. Pour obtenir une itération KFAC à sa matrice de poids W, nous procédons en deux étapes :

1. Nous construisons d'abord les nouvelles entités $\mathcal{A}'$, $\mathcal{F}'$, τ' et $(e', \bar{e})$ de $\mathcal{A}$, $\mathcal{F}$, τ et e comme définies par la Relation 1 dans §5.2.2.

2. Nous appliquons ensuite l'itération KFAC donnée par (1.74). Nous obtenons enfin W_{k+1} en appliquant à W'_{k+1} l'inverse de l'opération de reformattage qui avait été utilisée pour passer de $\mathcal{F}$ à $\mathcal{F}'$.

Cette extension a été numériquement évaluée sur des réseaux auto-encodeurs convolutionels. Les résultats sont présentés et commentés en §5.3.1.

Trois scénarios d'utilisation de KFAC. Pour évaluer les performances de l'optimiseur obtenu sur des tâches d'entraînement des GANs, nous envisageons trois scénarios, selon que KFAC est déployé pour le générateur seul, le discriminateur seul ou les deux. Nous choisissons d'étudier ces trois scénarios car nous souhaitons analyser l'impact de l'optimiseur KFAC sur chaque joueur (générateur ou discriminateur). Ce choix des scénarios est également motivé par [51, 82, 83] selon lesquels il faut relativement bien entraîner le discriminateur par rapport au générateur. Nous espérons à travers l'étude de ces trois scénarios corroborer cette recommandation. Les résultats de cette étude sont présentés et commentés en §5.3.2.

Chapter 2

A brief overview of neural networks and optimization algorithms

Contents

This chapter introduces some elementary notions that will be used throughout the manuscript. Most of the materials can be skipped by readers who are familiar with the domain of machine learning.

We first provide, in §2.1, a short beginner's guide to deep learning and to the popular architectures associated with it. Then, in §2.2, we review the state of the art regarding the optimization methods prominently used in the context of deep learning. We take this opportunity to discuss about the advantages and drawbacks of first-order methods (§2.2.2) in comparison with second-order methods (§2.2.3). In the latter category, we place particular emphasis on natural gradient methods (§2.2.4) and most notably the KFAC approximation of the Fisher matrix as well as its variants.

2.1 Network architectures for deep learning

Deep learning (DL) is a subfield of *machine learning* (ML) that involves the use of multiple layers of artificial neurons to learn and create nonlinear hierarchical representations from input data. As the backbone of DL, *artificial neural networks* (ANN) are computational models inspired by the structure and the low-level functioning of biological neurons in the human brain [144]. They constitute flexible and very powerful function approximators [33]. Thanks to recent advances in the design of ANN models and also to the availability of larger datasets and powerful computing resources, deep learning has shown tremendous success in solving many complex problems (in various domains) which were difficult or even impossible to solve with conventional

ML techniques. This includes tasks from computer vision [74, 93], speech recognition [147, 155], natural language processing [9, 55], self driving cars, and robotics, among others, where DL models have produced results comparable to human performance.

The main types of artificial neural network architectures are *multi-layer perceptrons* (MLP), *convolutional neural networks* (CNN), *recurrent neural networks* (RNN) and *graph neural networks* (GNN). In the following subsections, we will provide an introduction and overview of the first three types. For a detailed explanation of GNNs, we recommend referring to [149].

2.1.1 Multi-layer perceptrons

A *perceptron* or *neuron* [143] is the first and simplest model proposed in DL. It acts as a binary classifier: it takes as input a vector $x \in \mathbb{R}^d$ and computes a binary output a (so called activation) of it. The perceptron is said to be *activated* if the activation value is equal to 1 otherwise (activation value equal to 0), it is said to be non-activated. Figure 2.1 depicts the architecture of a perceptron. The vector $w = (w_1, \dots, w_d)$ represents the weights, b denotes the bias and σ is a nonlinear monotonous and differentiable function called *activation function*.

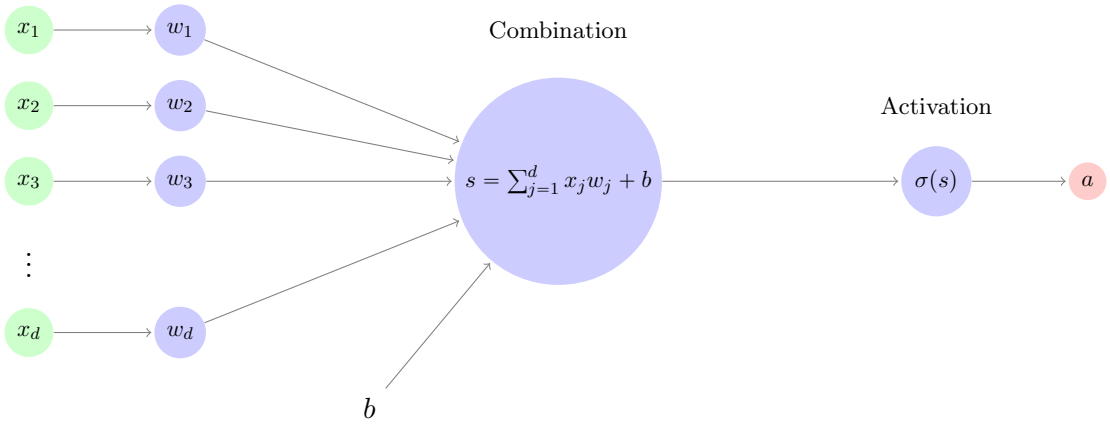


Figure 2.1: Architecture of a perceptron or neuron.

Activation functions are a critical component of neural networks and remain one of the most important elements which enable the great successes of DNNs [41]. They help to introduce nonlinearity into the model and make it possible for the network to learn complex patterns and relationships in the data. Without activation functions, a neural network would simply be a series of linear transformations, which would severely limit its learning capacities. Each activation function has its own characteristics, and some are better suited for certain types of problems than others. Thus, the choice of an activation function depends on the specific problem being solved. For example, the sigmoid function [77] is a common choice for binary classification problems because it maps outputs of neurons to probabilities between 0 and 1, which can be interpreted as the likelihoods of the inputs belonging to certain classes. The hyperbolic tangent functions (tanh) [59] is similar to the *sigmoid* function but maps the output to a range between -1 and 1. On the other hand, the *rectified linear unit* (ReLU) [59, 73] has become increasingly fashionable in recent years because of its simplicity and effectiveness. It simply returns the input if it is positive, and 0 if it is negative. This helps to prevent the vanishing gradient problem, which can occur when using sigmoid or tanh activations in deep networks.

A *feed-forward neural network* (FNN) is a type of artificial neural network organized in several layers in which information flows in one direction, from the input layer to the output layer only, i.e., there are no cycles or loops in the network. A *multi-layer perceptron* (MLP) or *fully connected network* (FCN) belongs to the class of FNN and is a stack of several layers of neurons connected to each other (Figure 2.2). Except for the neurons of input layer, each neuron in the network uses an activation function. Neurons of a hidden layer receive as inputs the outputs of neurons of the layer which precedes this layer and send their outputs to the neurons of the next layer through the synaptic weights.

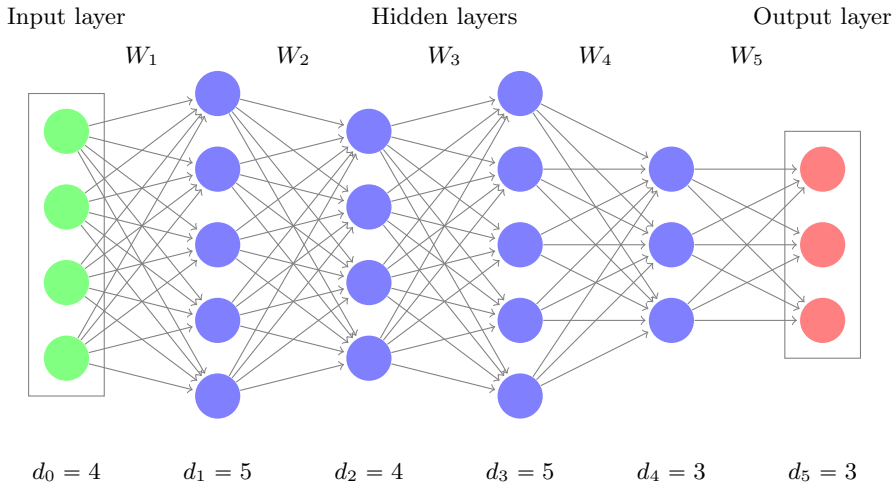


Figure 2.2: An example of MLP architecture with $\ell = 5$ layers. It contains an input layer, 4 hidden layers and an output layer.

Similarly, neurons of the output layer receive as inputs the outputs of the last hidden layer. Formally, for an ℓ-layer MLP f_θ parametrized by θ, the output

$$z := a_\ell = f_\theta(x) \in \mathbb{R}^{d_\ell} \tag{2.1}$$

from input $x \in \mathbb{R}^{d_0}$ is computed as

$$a_0 := x, \qquad s_i = W_i \bar{a}_{i-1}, \qquad a_i = \sigma_i(s_i), \qquad \text{for } i \text{ from 1 to } \ell, \tag{2.2}$$

where $W_i \in \mathbb{R}^{d_i \times (d_{i-1}+1)}$ is the weights matrix associated to layer i. s_i is referred to as pre-activation of the layer. $\bar{a}_{i-1} = (1, a_i^T)^T$ is the augmented activation vector (value 1 is used for the bias) and σ_i the activation function at layer i. The number of neurons at layer i is d_i and the total number of parameters is $p = \sum_{i=1}^{\ell} d_i(d_{i-1}+1)$. The MLP is parameterised by θ containing all its weights,

$$\theta = [\mathrm{vec}(W_1)^T, \mathrm{vec}(W_2)^T, \ldots, \mathrm{vec}(W_\ell)^T]^T \in \mathbb{R}^p,$$

where "vec" the operator that transforms a matrix into a vector according to Definition 3.2.

In spite of their simple architectures, MLPs have proven to be excellent in modeling with tabular data. However, when considering unstructured data such as images or texts, MLPs turn to be inefficient, and it is therefore necessary to consider different architectures.

2.1.2 Convolutional neural networks

As mentioned in the previous subsection, MLPs have very limited capabilities when it comes to processing unstructured and high dimensional input data such as images. One reason for this is that the number of parameters of MLPs is intrinsically dependent on the dimension of the input data and increases dramatically as the input dimension becomes larger. Furthermore, MLPs do not take into account the spatial structure of images i.e. relationships between pixels, which is essential for image processing tasks. On the other side, *convolutional neural networks* (CNN) [98–100] are convenient for image data because not only their number of parameters is not related to the dimension of the input data, but they also have a property of weights sharing between units [163] (this means that the same set of weights is applied to different parts of the input images, which significantly reduces the number of parameters). The most critical feature that explains the great success of CNNs for object recognition tasks is the spatial invariance property [99, 163]. This property gives them the ability to recognize patterns that are shifted, tilted or warped within the input images.

A convolution layer is a fundamental building block of CNNs and performs a convolution operation on the input data. The convolution operation involves sliding a set of small matrices, called filters or kernels (which represent the weights of the layer), with a defined stride e, over the input data, computing the dot product between the filters and the input at each position, and producing output feature maps. For instance, the convolution operation between a two-dimensional input $\mathcal{A}_{i-1} \in \mathbb{R}^{h_{i-1} \times w_{i-1}}$ (h_{i-1} and w_{i-1} denote the height and width respectively) and a filter $\mathcal{F}_i$ of size $m_i \times m_i$, with a stride e gives a two dimensional output $S_i \in \mathbb{R}^{h_i \times w_i}$. For $(t_1, t_2) \in \{1, \dots, h_i\} \times \{1, \dots, w_i\}$,

$$[S_i]_{t_1,t_2} = (\mathcal{A}_{i-1} * \mathcal{F}_i)_{t_1,t_2} = \sum_{\varsigma_1} \sum_{\varsigma_2} [\mathcal{F}_i]_{\varsigma_1,\varsigma_2} [\mathcal{A}_{i-1}]_{t_1-\varsigma_1,t_2-\varsigma_2}, \tag{2.3a}$$

$$h_i = \left\lfloor \frac{h_{i-1} - m_i}{e} + 1 \right\rfloor, \tag{2.3b}$$

$$w_i = \left\lfloor \frac{w_{i-1} - m_i}{e} + 1 \right\rfloor, \tag{2.3c}$$

where $\lfloor \cdot \rfloor$ designates the integer part.

Often some number τ of rows and columns containing zero values are added around the boundaries of the input before performing convolution operations. This technique under the name *padding* is used to address the problem of information loss due to the fact that convolution operations considerably shrink the spatial dimensions of the input and therefore consider less the features contained in the borders of the input. When padding τ is applied to both of spatial dimensions, the output is of shape

$$h_i = \left\lfloor \frac{h_{i-1} + 2\tau - m_i}{e} + 1 \right\rfloor, \qquad w_i = \left\lfloor \frac{w_{i-1} + 2\tau - m_i}{e} + 1 \right\rfloor. \tag{2.4}$$

When the input is three dimensional, e.g., red, green and blue (RGB) images, the filters must also be three dimensional (this third dimension is referred as channel number). In this case, the convolution operation in equation (2.3) remains valid only that one must also sum over the channel number. Note that the final output of a convolution layer is the concatenation of different results obtained by convolving the input with each of the set of filters. As with an MLP layer, after the convolutional filters are applied to the input, a bias term is added, and a nonlinear activation function is applied.

Implementing a convolution operation using the traditional formula as described in equation (2.3) can be computational expensive, especially for high-resolution input images. So in practice, in order to speed up computations, traditional convolution operations are turned into matrix-matrix or matrix-vector multiplications using the unrolling approach [27], whereby the input/output data are copied and rearranged into new matrices (see Figure 2.3).

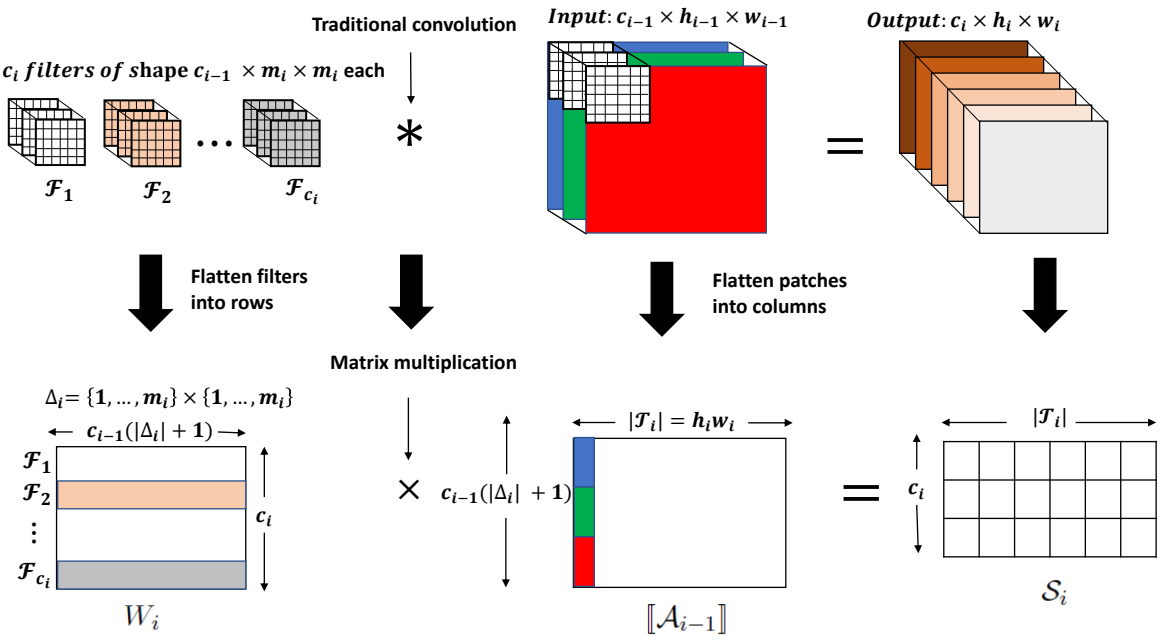


Figure 2.3: Traditional convolution turned into matrix-matrix multiplication with the *unrolling* approach.

Assume a convolution layer which receives an input $\mathcal{A}_{i-1} \in \mathbb{R}^{c_{i-1} \times h_{i-1} \times w_{i-1}}$ (c_{i-1} denotes the number of input channels). Let consider c_i filters, each of shape $c_{i-1} \times m_i \times m_i$. Let denote by $\mathcal{T}_{i-1} = \{1, \dots, h_{i-1}\} \times \{1, \dots, w_{i-1}\}$ and $\Delta_i = \{1, \dots, m_i\} \times \{1, \dots, m_i\}$ the spacial positions of the input and the filters respectively. We form a weight matrix W_i of shape $c_i \times (c_{i-1}|\Delta_i| + 1)$, where each row corresponds to a single filter flattened into a vector. Note that the additional 1 in the column dimension of W_i is required for the bias parameter. Around each position $t \in \mathcal{T}_{i-1}$, we define the local column vector $a_{i-1,t} \in \mathbb{R}^{c_{i-1}|\Delta_i|}$ by extracting the patch data from $\mathcal{A}_{i-1}$ (cf. [68] for explicit formulas). The output $\mathcal{A}_i \in \mathbb{R}^{c_i \times |\mathcal{T}_i|}$ is computed as follows: for $t \in \mathcal{T}_i$, the column $\tilde{a}_{i,t}$ of $\mathcal{A}_i$ associated to position t is given by

$$s_{i,t} = W_i \bar{a}_{i-1,t}, \qquad \tilde{a}_{i,t} = \sigma_i(s_{i,t}), \tag{2.5}$$

where $\bar{a}_{i-1,t} \in \mathbb{R}^{c_{i-1}\Delta_i + 1}$ is a_{i-1} concatenated with 1 in order to capture the bias. In matrix form, let $[\![\mathcal{A}_{i-1}]\!] \in \mathbb{R}^{(c_{i-1}\Delta_i + 1) \times |\mathcal{T}_i|}$ be the matrix whose columns are $\bar{a}_{i-1,t}$'s. Then we have

$$S_i = W_i [\![\mathcal{A}_{i-1}]\!], \qquad \mathcal{A}_i = \sigma_i(S_i). \tag{2.6}$$

A CNN belongs to the family of FNNs and is a successive stack of several convolution layers that progressively detect high level features from the input image. A toy FCN located at the

head of the CNN then uses the detected features to compute the prediction (see Figure 2.4). Sometimes, pooling layers [150] are used just after convolution layers. Their role is to shrink the spatial size of the output, to speed up computation, and to make features detection more robust. Pooling operators consist of a fixed-shape window that is slided over all regions in the input according to a certain stride and for each position, one takes either the maximum value (Max pooling) or the average (Average pooling). Note that a pooling layer has no learnable parameters but instead has hyper-parameters (size of the sliding window, the stride number, Max or Average pooling) which have to be carefully chosen.

When considering very deep CNN architectures, the gradient of the loss function w.r.t to the parameters of earlier layers becomes very small. This problem, known as vanishing gradient, is prohibitive for training such architectures [58]. Thankfully, the vanishing gradient problem has been tackled with the introduction of residual blocks [74,80]. Architectures using residual blocks back-propagate the gradient through shortcut connections and overcome the gradient vanishing problem. Over the last decade, CNNs have achieved state-of-the-art results for different computer vision tasks including image classification [74,93,157], object detection [57,138], semantic and instance segmentation [72,142].

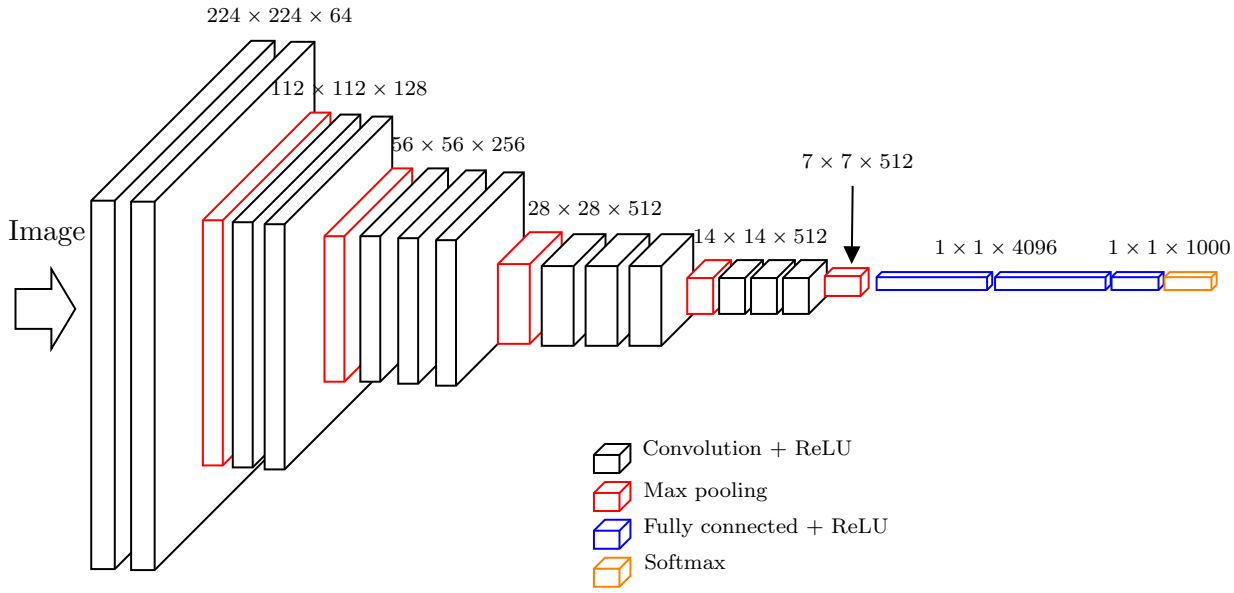


Figure 2.4: An example of a CNN, the standard VGG16 [157] network architecture.

2.1.3 Recurrent neural networks

So far, we have exposed two types of data (tabular data and images) and presented the appropriate network architectures for each. However, when it comes to processing sequential or temporal data like text or audio signal, it is necessary to consider another architecture. There are two main problems that make standard feed-forward architectures (MLPs and CNNs) not work for modelling sequential data. The first is that the inputs and outputs can be different lengths in different training samples and it is very difficult to handle such type of data with network architectures with fixed-length inputs/outputs. And then the second and the most serious problem is that these types of architectures do not share features learned across different temporal positions.

Recurrent neural networks (RNN) were introduced as an extension of FNNs for sequential data. In contrast to FNNs, they use recurrent connections between hidden units and map their input sequences into output sequences. As represented in Figure 2.5, the output at a time step is computed using the inputs at this time step and hidden state at the previous time step. The hidden state is assumed to store all necessary information from previous time steps to the current time step. Formally, a standard RNN transforms its input $x = (x^{(1)}, x^{(2)}, \ldots, x^{(T)})$ to an output $z = (z^{(1)}, z^{(2)}, \ldots, z^{(T)})$ through the sequential computations

$$a^{(0)} := 0, \qquad a^{(t)} = \sigma(W_x x^{(t)} + W_a a^{(t-1)} + b_a), \qquad z^{(t)} = \sigma(W_z a^{(t)} + b_z), \tag{2.7}$$

for t from 1 to T, where W_x is a matrix of weights connecting the inputs to the hidden layer, W_a is the weights matrix of recurrent connections, W_z the weights matrix used to compute the predictions, b_a, b_z denote bias vectors and σ a nonlinear activation function. An important property of an RNN is that parameters are shared accross time steps, i.e., W_x, W_a and W_z are the same for every time step t. RNNs can be classified into several types of architecture depending on the time step lengths T_x of the inputs and T_z of the outputs [85] (see Figure 2.6).

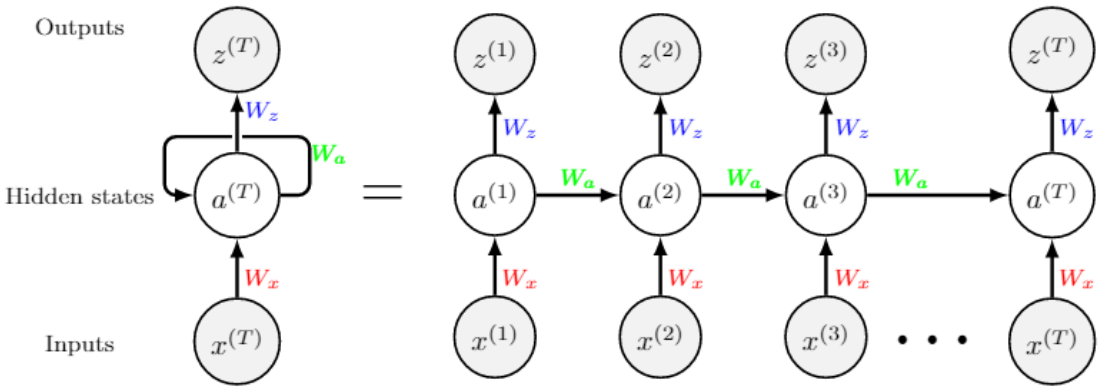
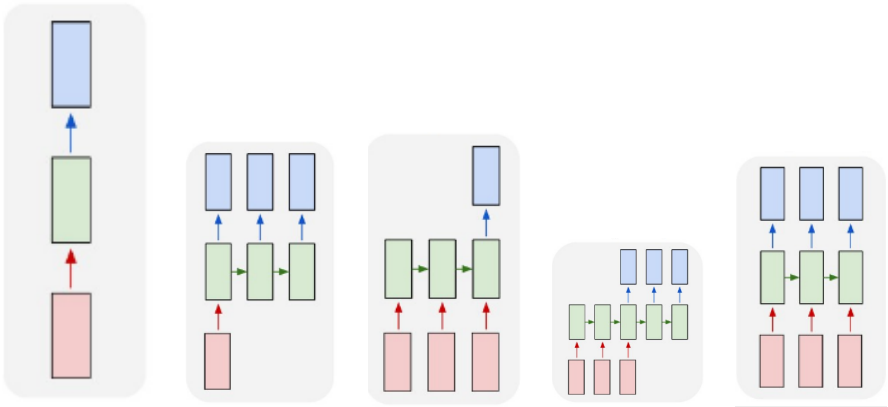


Figure 2.5: A standard RNN architecture.

The RNN architectures presented so far are unidirectional or forward directional only. To compute the prediction at a time step t, they only use information from previous states and current state. For certain classes of problem such as Named-entity recognition, information from future states is also needed to calculate the prediction at the current state. To address this issue, an extension to RNNs called *bidirectional recurrent neural networks* (BRNNs) has been proposed in [154]. BRNNs are trained using all available input information in the past and future of a specific time step and are proven to be efficient for problems requiring information to be processed in the two directions [154].

Despite their undisputed effectiveness in learning with sequential data, it turns out that one of the main problems of standard RNNs is that they run into vanishing gradient problems. Some data such as texts can have long sequences with very long-term dependencies i.e. information from earlier states can significantly affect the prediction in much later states. But it has been empirically observed that the basic RNN architectures we have seen so far are not very good at capturing very long-term dependencies (memorizing information for a very long time). To address this issue, sophisticated RNN architectures using units that implement a gating mechanism to control the flow of information have been proposed, namely, *long-short term memory* (LSTM) [79] and *gated recurrent unit* (GRU) [29]. They have been proven to be capable to

learn very long range connections in a sequence and they outperform standard RNNs on various tricky tasks such as music modeling and speech signal modeling [30].



(a) One to one: $T_x = T_z = 1$ (b) One to many: $T_x = 1, T_z > 1$ (c) Many to one: $T_x > 1, T_z = 1$ (d) Many to many: $T_x > 1, T_z > 1$ (e) Many to many: $T_x > 1, T_z > 1$

Figure 2.6: Different types of RNNs as presented in [85].

Very sophisticated sequential data processing models that do not use any of the standard RNN, GRU or LSTM units have been developed in recent years. These models called *transformers* are based on a self-attention mechanism [172] and have been proven to outperform LSTM and GRU based-architectures in many natural language processing tasks [22,40]. Transformer models have also been successfully applied to computer vision tasks such as image recognition [43,166], object detection [26,28,49,177,193], image segmentation [159,181,191], etc.

2.2 Optimization algorithms

2.2.1 Relationship between deep learning and optimization

Supervised learning is a branch of statistical learning where data composed of inputs $x \in \mathcal{X}$ and targets $y \in \mathcal{Y}$ (labelled data) is given. The inputs are supposed to have some influence on the targets and the goal is to learn correlations between the inputs and targets so that targets can be predicted for future inputs that the model has never seen [65]. In contrast, in an *unsupervised learning* task, we are given a data of inputs without any labels, and the goal is to extract classes or groups of individuals with common characteristics. Although there are many unsupervised deep learning problems, in this work we only consider those that are supervised. For simplicity and without any ambiguity, we will simply say *deep learning* instead of *supervised deep learning*.

A deep learning problem can be decomposed into three steps: *modeling*, *optimization* and *generalization* [160]. Modeling consists of choosing or defining a family of parameterized models e.g. neural networks that best describe the problem. Formally, we consider the family of prediction functions

$$\mathcal{M} = \{f_\theta : \mathcal{X} \to \mathcal{Y}, \theta \in \mathbb{R}^p\}.$$

Then comes the optimization step, the aim of which is to find the prediction function in this family that best fits the problem. This is equivalent to finding the best value of parameter θ

that minimizes the expected error induced by a certain defined loss function

$$L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R},$$

that given an input-output pair (x, y), yields the quantity $L(z, y)$, which corresponds to the disagreement between the model prediction $z := f_\theta(x)$ and the actual target y. The generalization step consists in using the function found at the end of the first two steps to make predictions on unseen data. Each step has a corresponding error, i.e. modelling or representation error for step 1, optimization or training error for step 2 and generalization error for step 3. In DL and more generally in ML, these three steps are studied independently [14, 187, 190].

The second step which is the main focus of this work is where optimization and deep learning intersect. From now on, $\mathcal{X}$ and $\mathcal{Y}$ are supposed to be real vector spaces i.e. $\mathcal{X} = \mathbb{R}^{d_x}$ and $\mathcal{Y} = \mathbb{R}^{d_y}$ with $(d_x, d_y) \in \mathbb{N}^* \times \mathbb{N}^*$.

Expected risk. The input-target space $\mathbb{R}^{d_x} \times \mathbb{R}^{d_y}$ is supposed to be equipped with a probability distribution $\mathsf{Q}_{x,y}$ representing the true relationship between inputs and targets. Optimally, we seek to find the parameter θ that minimizes the excepted error yielded by any input-target pair $(x, y) \sim Q_{x,y}$. The error to be minimized is thus given by

$$R(\theta) = \mathbb{E}_{(x,y)\sim \mathsf{Q}_{x,y}}[L(y, f_\theta(x))] = \int_{\mathbb{R}^{d_x}\times\mathbb{R}^{d_y}} L(y, f_\theta(x))\, \mathrm{d}\mathsf{q}(x, y), \tag{2.8}$$

where $\mathsf{q}(x, y)$ is the density function of $\mathsf{Q}_{x,y}$. $R(\theta)$ is referred as the *expected risk*.

Empirical risk. In general, the expected risk is inaccessible due to the fact that the distribution $Q_{x,y}$ is unknown. In practice, we are given $n \in \mathbb{N}^*$ observations of input-target pairs supposed to be independently drawn from $\mathsf{Q}_{x,y}$. These observations define the training data

$$\mathcal{D} = \left\{(x^{(1)}, y^{(1)}),\ (x^{(2)}, y^{(2)}),\ \ldots,\ (x^{(n)}, y^{(n)}) \mid (x^{(b)}, y^{(b)}) \in \mathbb{R}^{d_x \times d_y},\ 1 \leqslant b \leqslant n\right\},$$

from which we compute an estimate of the expected risk

$$h(\theta) := \frac{1}{n}\sum_{b=1}^{n} L(y^{(b)}, f_\theta(x^{(b)})), \tag{2.9}$$

also referred to as the *empirical risk*. This is the objective function of the optimization problem.

In deep learning, the activation functions and the loss function are generally chosen such that the objective function is differentiable with respect to the network parameter θ. Most of optimization problems involved in deep learning are complex and do not have analytical solutions. So numerical optimization algorithms are used to compute approximate solutions. There are two main classes of numerical optimization algorithms used in deep learning: *first-order* and *second-order* methods.

2.2.2 First-order optimization methods

First-order or *gradient descent* (GD) based algorithms are the most commonly used methods for neural network optimization. As their name suggests, these algorithms only use first order information i.e. the first derivatives or the gradient. Starting from an initial point $\theta_0 \in \mathbb{R}^p$, the standard or batch GD method iterately computes updates through the equation

$$\theta_{k+1} = \theta_k - \alpha_k \nabla_\theta h(\theta_k), \tag{2.10}$$

where α_k is the step-size (the so-called learning rate) and

$$\nabla_\theta h(\theta_k) = \frac{1}{n}\sum_{b=1}^{n} \nabla_\theta L(y^{(b)}, f_{\theta_k}(x^{(b)})) \tag{2.11}$$

the full gradient of the objective function calculated at the iteration k. The GD algorithm is motivated by the fact that by definition, the gradient gives the direction of steepest ascent of the objective function, and therefore, to minimize the function, it seems natural to move in the opposite direction of the gradient. When the objective function is strongly convex and if the learning rate is appropriately chosen (lower than $1/L$, where L denotes the constant of Lipschitz continuousness of the gradient), then GD converges linearly [18]. While GD is convenient for problems with small or medium-sized data, it is computationally prohibitive for problems involving large datasets, because at each iteration, it requires that the model be evaluated on each example in the dataset.

Stochastic gradient descent. A more practical version of GD is the *stochastic gradient descent* (SGD) method [141]. The idea of SGD is to estimate at each iteration the full gradient by the gradient calculated with an example randomly drawn from the dataset. Each iteration of the optimization process becomes therefore very cheap, involving only the calculation of the quantity $\nabla h_{b_k}(\theta_k), b_k \sim \mathcal{U}([\![1;n]\!])$[1], corresponding to one sample. This makes SGD the prevailing method for large scale machine learning. When the objective function is strongly convex, the SGD algorithm converges to the global optimum with a sublinear rate of convergence. And for general nonconvex objective functions, when the sequence of learning rates $\{\alpha_k\}_{k\in\mathbb{N}}$ satisfies Robbins and Monro conditions [141]

$$\sum_{k=0}^{\infty} \alpha_k = \infty \qquad \text{and} \qquad \sum_{k=0}^{\infty} \alpha_k^2 < \infty, \tag{2.12}$$

the SGD method is guaranteed to converge to a local minimum [18].

One of the remarkable properties of SGD that could explain its success in many complex problems is that, unlike GD, the sequence of iterates $\{\theta_k\}_{k\in\mathbb{N}}$ is not deterministic, i.e. it is not uniquely determined by the initial parameter θ_0 and the sequence of learning rates $\{\alpha_k\}_{k\in\mathbb{N}}$, but rather is a stochastic process driven by the sequence of random variables $\{b_k\}_{k\in\mathbb{N}}$ [18].

Notwithstanding its advantages, one of the main shortcomings of SGD lies in the poor approximation quality of the full gradient. In fact, although the stochastic gradient $\nabla h_{b_k}(\theta_k)$ is an unbiased estimator of the full gradient thanks to

$$\mathbb{E}_{b_k\sim\mathcal{U}([\![1;n]\!])}\left[\nabla h_{b_k}(\theta_k)\right] = \frac{1}{n}\sum_{b=1}^{n} \nabla_\theta L(y^{(b)}, f_{\theta_k}(x^{(b)})) = \nabla_\theta h(\theta_k), \tag{2.13}$$

it introduces a high variance. To address this issue, many noise reduction methods have been introduced in the literature. The proposed methods reduce errors in the gradient estimates by either aggregating gradients, averaging iterates or dynamically increasing the size of examples used to estimate the gradient. The aim of such methods is to improve convergence rate from sublinear to linear [18]. They have been proven to be effective in practice and have interesting theoretical properties [18,34,84,132,151]. Another reduction technique which, due to its simplicity, is the most used in the community is the *mini-batch SGD* approach. Instead of estimating

[1] $[\![1;n]\!] = [1;n] \cap \mathbb{N}$

the gradient with a single sample as SGD does, this approach which is a trade-off between standard GD and SGD, uses a random subset $\mathcal{S}_k$ drawn from the training dataset to compute an estimate

$$\nabla_\theta h(\mathcal{S}_k, \theta_k) = \frac{1}{|\mathcal{S}_k|} \sum_{(x^{(b)}, y^{(b)}) \in \mathcal{S}_k} \nabla_\theta L(y^{(b)}, f_{\theta_k}(x^{(b)})) \tag{2.14}$$

of the full gradient. In addition to variance reduction, mini-batch SGD has the additional advantage of resolving SGD's intrinsic problem of non-parallelization. In the following, we will use SGD to refer to both SGD and mini-batch SGD noting that SGD is a special case of mini-batch SGD with $|\mathcal{S}_k| = 1$.

Momentum and Nesterov accelerated SGD. Other variants of GD that are widely used in the deep learning community are the stochastic versions of the *heavy-ball* method [131] (SGD with momentum) and *Nesterov Accelerated Gradient* method [118] (NAG SGD). Both methods were introduced in order to smooth fluctuations encountered with SGD and therefore achieve a faster convergence rate [18, 161, 190]. Momentum SGD updates iterates with both the current gradient as well as the past gradients via the equation

$$m_{k+1} = \beta m_k + (1-\beta)\nabla_\theta h(\mathcal{S}_k, \theta_k), \tag{2.15a}$$
$$\theta_{k+1} = \theta_k - \alpha_k m_{k+1}. \tag{2.15b}$$

As for NAG SGD, the parameter is updated by

$$m_{k+1} = \beta m_k + (1-\beta)\nabla_\theta h(\mathcal{S}_k, \theta_k - \beta\alpha_k m_k), \tag{2.16a}$$
$$\theta_{k+1} = \theta_k - \alpha_k m_{k+1}. \tag{2.16b}$$

In both systems, $\beta \in [0, 1]$ and denotes the momentum coefficient. With the right choice of learning rate and momentum parameter, such methods have proven to be very efficient and achieve state-of-the-art results in many deep learning optimization problems [161].

Adaptive gradient methods. In all versions of GD presented so far, the learning rate α_k at iteration k is identical for all components of the parameter. However, due to the variation of the curvature of the objective function in different directions, various components of the parameter may require dinstinct learning rates. A high fixed learning rate will lead to a faster progress in directions with low curvature, while it will lead to fluctuations in directions with high curvature. In contrast, a small learning rate will lead to a faster progress in regions of high curvature, but will cause small steps in regions of low curvature [109, 190]. This has motivated many adaptive gradient methods[2] which compute component-wise learning rates in the updating process.

AdaGrad [45] computes the learning rate for each component of the parameter by scaling the initial constant learning rate with the inverse of the square root of the sum of all of the historical squared values of the gradient in the direction of that component. Mathematically,

$$D_{k+1} = \operatorname{diag}\Big(\sum_{j=0}^{k} \nabla_\theta h(\mathcal{S}_j, \theta_j) \nabla_\theta h(\mathcal{S}_j, \theta_j)^T \Big) + \varepsilon I, \tag{2.17a}$$
$$\theta_{k+1} = \theta_k - \alpha_0 D_k^{-1/2} \nabla_\theta h(\mathcal{S}_k, \theta_k), \tag{2.17b}$$

[2] Adaptive gradient methods are commonly presented as first-order methods, but can also be interpreted as stochastic versions of the natural gradient descent with a diagonal approximation of the empirical Fisher matrix.

where α_0 is the initial constant learning rate, "diag" is the operator that convert a matrix into a diagonal matrix by conversing only the diagonal, I identity matrix and ε a small value used to avoid divisions by zero. Despite its effectiveness for solving convex optimization problems [65], it has been empirically observed that AdaGrad struggles in neural network optimization. This is due by the fact that gradient accumulation since the beginning of the optimization process can lead to D_{k+1} with exponentially large values resulting in vanishing learning rates. RMSProp [165] was introduced to address the problem of monotonically decreasing learning rates encountered in AdaGrad. The core idea of RMSProp is to decrease the weight of the past accumulated gradients in D_{k+1}. With RMSProp algorithm, the parameter is updated via the equation

$$G_{k+1} = \beta G_k + (1-\beta)\nabla_\theta h(\mathcal{S}_k, \theta_k)\nabla_\theta h(\mathcal{S}_k, \theta_k)^T, \tag{2.18a}$$
$$D_{k+1} = \operatorname{diag}(G_{k+1}) + \varepsilon I, \tag{2.18b}$$
$$\theta_{k+1} = \theta_k - \alpha_0 D_k^{-1/2}\nabla_\theta h(\mathcal{S}_k, \theta_k). \tag{2.18c}$$

Adaptive Moment Estimation (Adam) [88] uses both the ideas behind momentum and adaptive learning rate methods. Specifically, it combines the RMSProp algorithm and the Momentum algorithm through the iterations

$$m_{k+1} = \beta_1 m_k + (1-\beta_1)\nabla_\theta h(\mathcal{S}_k, \theta_k), \tag{2.19a}$$
$$v_{k+1} = \beta_2 v_k + (1-\beta_2)\nabla_\theta h(\mathcal{S}_k, \theta_k) \odot \nabla_\theta h(\mathcal{S}_k, \theta_k), \tag{2.19b}$$
$$\hat{m}_{k+1} = \frac{m_{k+1}}{1-\beta_1^k}, \tag{2.19c}$$
$$\hat{v}_{k+1} = \frac{v_{k+1}}{1-\beta_2^k}, \tag{2.19d}$$
$$\theta_{k+1} = \theta_k - \frac{\alpha_0}{\sqrt{\hat{v}_{k+1}} + \varepsilon} \odot \hat{m}_{k+1}. \tag{2.19e}$$

In the above equations, $\odot$ denotes Hadamard product (element-wise product) operation of two vectors, and $(\beta_1, \beta_2) \in [0,1]^2$ is a pair of hyperparameters.

There are a few other adaptive methods such as Adadelta [185] and Nadam [44]. Interested readers are referred to [190] for more details. In practice, adaptive gradient methods perform very well, but it has been observed that in some complex neural network optimization problems, such methods perform poorly compared to a well-tuned SGD with momentum.

Limitations of first-order methods. Although first-order methods are straightforward to implement, they have several limitations when used to train DNNs. To begin with, almost all optimization problems arising in DL are nonconvex and highly nonlinear. The objective function usually has many local optima and saddle points. A first-order gradient descent may be more easily trapped into a local minimum or a saddle point. The optimization process then gets stuck at such a location, even though it is not a global minimum.

Moreover, the landscape of the objective function may contain huge variations in curvature along different directions. Although the stochastic nature of SGD and a careful random initialization of the initial parameter such as LeCun initialization [101] or Xavier initialization [58] can help with local minimum and saddle points problems [161], a first-order method remains very sensitive to the problem of large curvature variations along the surface of the objective function [109].

Most embarrassing of all, first-order methods are inconsistent with the homogeneity of the units of the parameters [190]. Indeed, if θ has a specific unit, then the increment $\theta_{k+1} - \theta_k$ should have the same unit as well. However, from equation (2.10) it can be seen that

$$\text{unit of } \alpha_k \nabla_\theta h(\theta_k) \;\propto\; \frac{\text{unit of } L}{\text{unit of } \theta},$$

where $\propto$ stands for "proportional" and α_k is regarded as dimensionless. As a consequence, a change of unit in θ will produce iterates that no longer match, unless we are able to reflect this correctly on the learning rate. A more sophisticated formulation of this is to state that first-order methods are not invariant to affine transformations of the variable [18]. Indeed, if we apply the reparametrization

$$\theta = A\widehat{\theta}, \tag{2.20}$$

where A is a $p \times p$ invertible matrix, then the gradient descent in the new parameter reads

$$\widehat{\theta}_{k+1} = \widehat{\theta}_k - \alpha_k \nabla_{\widehat{\theta}} h(A\widehat{\theta}_k) = \widehat{\theta}_k - \alpha_k A^T \nabla_\theta h(A\widehat{\theta}_k). \tag{2.21}$$

Multiplying the last equality by A, we obtain

$$\theta_{k+1} = \theta_k - \alpha_k A A^T \nabla_\theta h(\theta_k). \tag{2.22}$$

It is obvious that algorithm (2.22) will behave differently from algorithm (2.10).

Another downside of first-order methods is that they are very sensitive to the learning rate or step size. They require careful tuning of that hyper-parameter to ensure convergence. If the step size is too large, the algorithm may diverge, and if it is too small, the algorithm may converge slowly. Last but not least, first-order methods are inherently sequential and do not lend themselves easily to parallelism, as pointed out earlier for SGD.

2.2.3 Second-order optimization methods

Second-order methods attempt to circumvent the above-mentioned difficulties through the use of second-order information. At each iteration k, they seek the best update δ_k to be added to θ_k by solving the sub-problem involving a local quadratic approximation to the objective function h at θ_k, namely,

$$\delta_k = \underset{\delta \in \mathbb{R}^p}{\operatorname{argmin}}\, h(\theta_k) + \nabla_\theta h(\theta_k)^T \delta + \frac{1}{2} \delta^T C(\theta_k) \delta, \tag{2.23}$$

where $C(\theta_k)$ denotes a positive semi-definite matrix, referred to as *curvature matrix* and supposed to capture local curvature information of h at θ_k. The solution to problem (2.23) is

$$\delta_k = -[C(\theta_k)]^{-1} \nabla_\theta h(\theta_k). \tag{2.24}$$

Thus, a second-order iterate takes the form

$$\theta_{k+1} = \theta_k - \alpha_k [C(\theta_k)]^{-1} \nabla_\theta h(\theta_k), \tag{2.25}$$

where the step-size $\alpha_k > 0$ has been reintroduced to enforce safety whenever necessary.

Unlike first-order methods, second-order methods have the great virtue of being invariant to any linear or affine reparametrization of the parameter [18, 39]. They offer a powerful solution to the problem of variations in curvature along different directions by rescaling the gradient components individually along directions of eigenvectors of the curvature matrix [109].

Newton, Hessian-free and BFGS. Newton-Raphson's method [37] is the most widely used second-order method and serves as the basis for all other second-order methods. It amounts to setting $C(\theta_k)$ as the Hessian matrix

$$H(\theta_k) = \nabla^2_{\theta\theta} h(\theta_k) \tag{2.26}$$

in (2.25). In this case, the quadratic model defined in equation (2.23) coincides with the second-order Taylor approximation of h at θ_k. For a problem arising from machine learning, the Hessian matrix can be estimated by

$$H(\mathcal{S}'_k, \theta_k) = \frac{1}{|\mathcal{S}'_k|} \sum_{b \in \mathcal{S}'_k} \nabla^2_{\theta\theta} h_b(\theta_k) \tag{2.27}$$

over a mini-batch $\mathcal{S}'_k \in \{1, \dots, n\}$ which can differ from $\mathcal{S}_k$, the one used for the gradient.

In the spirit of its design, Newton's method computes updates based on minimization of an exact second-order Taylor model of the objective function at each iteration, while gradient descent based algorithms rely on a model that is only first-order accurate. Thus, Newton's method usually achieves a higher rate of convergence. For instance, with $\alpha_k \equiv 1$ and under certain suitable assumptions about h and the starting point, we can establish quadratic convergence [37] of Newton's method, which outperforms the sublinear convergence of a first-order gradient descent. The extreme example is when the function h is a convex quadratic function, e.g., $h(\theta) = \frac{1}{2}\|A\theta - b\|^2$. In this case, Newton's method converges in just one iteration.

As an exercise, let us check that Newton's method is invariant with respect to the linear reparametrization (2.20). In the new variable $\widehat{\theta} = A^{-1}\theta$ and the new function $\widehat{h}(\widehat{\theta}) = h(\theta)$, a Newton iteration reads

$$\begin{aligned}
\widehat{\theta}_{k+1} &= \widehat{\theta}_k - \alpha_k \big[\nabla^2_{\widehat{\theta}\widehat{\theta}} \widehat{h}(\widehat{\theta}_k)\big]^{-1} \nabla_{\widehat{\theta}} h(\widehat{\theta}_k) \\
&= \widehat{\theta}_k - \alpha_k \big[A^T \nabla^2_{\theta\theta} h(A\theta_k) A\big]^{-1} \big[A^T \nabla_\theta h(A\widehat{\theta}_k)\big] \\
&= \widehat{\theta}_k - \alpha_k A^{-1} \big[\nabla^2_{\theta\theta} h(A\widehat{\theta}_k)\big]^{-1} \nabla_\theta h(A\widehat{\theta}_k).
\end{aligned}$$

Multiplying the last equality by A, we end up with

$$\theta_{k+1} = \theta_k - \alpha_k [\nabla^2_{\theta\theta} h(\theta_k)]^{-1} \nabla_\theta h(\theta_k). \tag{2.28}$$

Put another way, the sequences $\{\theta_k\}_{k \geqslant 0}$ and $\{\widehat{\theta}_k\}_{k \geqslant 0}$ produced by the algorithm match with each other through the transformation.

However, the main concern with Newton's method is that in the non-convex configuration as in deep learning, the Hessian matrix is not necessarily positive semi-definite, which can give rise to an update in the wrong direction, i.e., direction of increase in the objective function [123]. This issue can be addressed with regularization techniques such as trust region [31] or Levenberg–Marquardt damping [103, 107]. Besides, assembling and storing the Hessian matrix is costly in terms of memory and computing time. Simple-minded recipes such as direction inversion of a diagonal approximation [12] are very rough.

To address this cost issue, Hessian-free methods [108, 113] were suggested, which rely on the information of the Hessian matrix without ever having to form and store it. To this end, we note that the increment $\theta_{k+1} - \theta_k$ is a solution of the linear system

$$\nabla^2_{\theta\theta} h(\theta_k)(\theta_{k+1} - \theta_k) = -\nabla_\theta h(\theta_k). \tag{2.29}$$

This system can be solved by means of inexact techniques [35] using a Krylov subspace-based conjugate gradient (CG) [63], which only requires knowledge of the matrix-vector product. However, for a neural network, each matrix-vector product involving an elementary Hessian of a single datum can be performed through a sequence of propagation and backpropagation [130]. The number of CG iterations must be limited to ensure some speed, while remaining sufficiently high not to degrade too much the convergence speed of this inexact Newton.

An alternative to the Hessian-free method is to approximate the Hessian matrix with rank-1 updates during iterations, which gives rise to the BFGS method (*Broyden-Fletcher-Goldfarb-Shanno*) [23, 52, 60, 156]

$$H_{k+1} \approx H_k + \frac{\Delta g_k \Delta g_k^T}{\Delta g_k^T \Delta \theta_k^T} - \frac{H_k \Delta \theta_k \Delta \theta_k^T H_k^T}{\Delta \theta_k^T H_k \Delta \theta_k} \tag{2.30a}$$

with

$$\Delta \theta_k = \theta_{k+1} - \theta_k, \qquad \Delta g_k = \nabla_\theta h(\theta_{k+1}) - \nabla_\theta h(\theta_k). \tag{2.30b}$$

Thanks to Sherman-Morrison-Woodbury's formula, the inverse can be easily updated as [123]

$$H_{k+1}^{-1} \approx \left(I - \frac{\Delta g_k \Delta \theta_k^T}{\Delta \theta_k^T \Delta g_k} \right)^T H_k^{-1} \left(I - \frac{\Delta g_k \Delta \theta_k^T}{\Delta \theta_k^T \Delta g_k} \right) + \frac{\Delta \theta_k \Delta \theta_k^T}{\Delta \theta_k^T \Delta g_k}. \tag{2.31}$$

The Hessian approximation (2.30), which uses only first-order information, is "good" on the grounds that it is semi-positive symmetric (if H_0 is) and verifies the secant equation

$$H_{k+1} \Delta \theta_k = \Delta g_k.$$

This property guarantees a superlinear local convergence rate in deterministic mode. Unfortunately, matrices given by (2.30)–(2.31) are dense, even when the exact Hessian is sparse. Additionally, we need to store the H_k^{-1} to be able to move from one iteration to the next. A more memory-efficient version exists under the name L-BFGS (*limited memory BFGS*) [105], in which storage is avoided by calculating the products $H_k^{-1} \nabla_\theta h(\theta_k)$ by an approximate formula involving only the most recently saved increments $(\Delta \theta_k, \Delta g_k)$. The L-BFGS method appears to be better suited to the learning context, where other variants have been developed such as L-BFGS-B (*L-BFGS box constraints*) [25] and O-LBFGS (*online L-BFGS*) [153].

Gauss-Newton and generalized Gauss-Newton. The BFGS method and its variants belong to the family of *quasi-Newton* methods, where the ideal Hessian matrix is abandoned in favor of simpler approximations with more favorable properties. Another important class in the quasi-Newton family are the Gauss-Newton methods [152]. Originally, it was introduced in classical optimization to approximate the Hessian matrix in nonlinear least squares problems.

Assuming a least-squares loss $L(y, z) = \frac{1}{2}\|y - z\|^2$, the contribution to the objective function of a single training sample (x, y) is

$$h(\theta) = L(y, f_\theta(x)) = \frac{1}{2}\|f_\theta(x) - y\|_2^2. \tag{2.32}$$

By virtue of the chain rule, the gradient is given by

$$\nabla_\theta h(\theta) = [J_{f_\theta(x)}]^T (f_\theta(x) - y), \tag{2.33}$$

where $J_{f_\theta(x)} = \nabla_\theta f_\theta(x)$ is the Jacobian of $f_\theta(x)$ with respect to θ. The Hessian is also obtained by chain rule

$$\nabla^2_{\theta\theta} h(\theta) = [J_{f_\theta(x)}]^T J_{f_\theta(x)} + \sum_{\nu=1}^{d_y} (f_\theta(x) - y)^{[\nu]} \nabla^2_{\theta\theta} f_\theta^{[\nu]}(x), \tag{2.34}$$

where $v^{[\nu]}$ designates the ν^{th} entry of vector v. The Gauss-Newton method is tantamount to ignoring the second term in (2.34) and therefore to approximating the Hessian by

$$G(\theta) = [J_{f_\theta(x)}]^T J_{f_\theta(x)}, \tag{2.35}$$

called *Gauss-Newton matrix*. When the residual $r(\theta) = f_\theta(x) - y$ is small, that is, the model fits well with data, the Gauss-Newton matrix is very close to the true Hessian. Another way to derive the Gauss-Newton matrix is to plug the first-order expansion

$$f_\theta(x) = f_{\theta_k}(x) + J_{f_{\theta_k}(x)}(\theta - \theta_k) \tag{2.36}$$

of the predictive function (2.32) into the least-squares function at each iteration k. This yields the linear least-squares problem

$$\min_{\theta \in \mathbb{R}^p} \frac{1}{2} \| J_{f_{\theta_k}(x)}(\theta - \theta_k) - (y - f_{\theta_k}(x)) \|_2^2 = \min_{\delta\theta \in \mathbb{R}^p} \frac{1}{2} \| J_{f_{\theta_k}(x)} \delta\theta - (y - f_{\theta_k}(x)) \|_2^2, \tag{2.37}$$

the normal equation of which involves matrix (2.35).

The advantage of the Gauss-Newton matrix over the Hessian is that it is computationally less expensive, since it only requires first derivatives of the predictive function. On top of that, the Gauss-Newton matrix is always positive semi-definite even if the problem is non-convex and the related Hessian is indefinite. When the Gauss-Newton matrix is singular, it can be regularized by $G(\theta) + \lambda I$, $\lambda > 0$ to become positive definite. A shortcoming of the Gauss-Newton approach is that the quality of the approximation is strongly related to the residual $r(\theta)$. A large residue will result in an approximation, which is far from the Hessian. In addition, the approximation disregards all second derivatives of the predictive function and this seems contradictory for a matrix that is supposed to provide information on the curvature of the objective function.

The Gauss-Newton approximation can be extended to any type of loss function $L(y, z)$ [152]. Starting from the objective function of a single training pair

$$h(\theta) = L(y, f_\theta(x)) = L(y, z)_{|z=f_\theta(x)}, \tag{2.38}$$

we have

$$\nabla_\theta h = [J_{f_\theta(x)}]^T \nabla_z L(y, z)_{|z=f_\theta(x)} \tag{2.39}$$

for the gradient and

$$\nabla^2_{\theta\theta} h(\theta) = [J_{f_\theta(x)}]^T H_L(z) J_{f_\theta(x)} + \sum_{\nu=1}^{d_y} \nabla_z L^{[\nu]}(y, z)_{|z=f_\theta(x)} \nabla^2_{\theta\theta} f_\theta^{[\nu]}(x) \tag{2.40}$$

for the Hessian, with $H_L(z) = \nabla^2_{zz} L(z, y)_{|z=f_\theta(x)}$. The *generalized Gauss-Newton* (GGN) matrix is then defined as

$$G(\theta) = [J_{f_\theta(x)}]^T H_L(z) J_{f_\theta(x)}. \tag{2.41}$$

We recover (2.35) when $H_L(z) = I$. If the loss function $L(z, y)$ is convex with respect to z, the GGN matrix is always positive semi-definite, making it the ideal choice as a curvature matrix in neural network optimisation. Using a mini-batch $\mathcal{S}'$, expression (2.41) can be estimated by

$$G(\mathcal{S}', \theta) = \frac{1}{|\mathcal{S}'|} \sum_{(x^{(b)}, y^{(b)}) \in \mathcal{S}'} [J_{f_\theta(x^{(b)})}]^T H_L(z_b) J_{f_\theta(x^{(b)})}. \tag{2.42}$$

Limitations of second-order methods. Despite their theoretical superiority (homogeneity, robustness and speed of convergence), we must admit that second-order optimization methods have not yet met with the expected success in everyday use in machine learning[3]. The main reason for this is that each second-order iteration is prohibitively more expensive in terms of memory and computing time than a first-order iteration, even when all the tricks have been deployed to dispense with the assembly and storage of the curvature matrix. When memory and computing resources are limited, first-order methods are preferred for their lower complexity. While fewer iterations are needed overall, this is not enough to make second-order methods competitive. This observation corroborates one of the paradoxes of machine learning: "Good optimization methods are not always good machine learning methods".

Another explanation for this paradox is that first-order optimization methods, in addition to their simplicity of implementation, are more easily adaptable to changes in network architecture. First-order methods can converge faster than second-order methods at the beginning of training. This is particularly valuable when the model is initialized with random weights, as they can quickly improve initial performance and find regions of the search space that provide more informative gradients. However, second-order methods can be useful in certain scenarios, notably when data is sparse or when the shape of the objective function is highly curved. In these cases, second-order methods can converge more quickly to local optima of higher quality.

In this thesis, we wish to investigate some original ideas that could alleviate the bottleneck caused by the curvature matrix and the associated linear system. The path we wish to explore is that of the natural gradient, the mathematical foundation of which seems to be better suited to the context of learning. The feeling that such an undertaking should be possible is based on the performance claimed for more recent approximations of the Fisher matrix, which are of the KFAC type [68, 111, 112]. To understand what this is about, a thorough review of natural gradient methods is in order.

2.2.4 Focus on natural gradient descent methods

Very prosaically, the natural gradient descent can be thought of as a quasi-Newton method in which the curvature matrix $C(\theta_k)$ is taken to be a so-called Fisher information matrix $F(\theta_k)$. At a deeper level, the philosophy is to formulate a gradient descent not in the Euclidean space of parameters, but on the manifold of probability distributions related to the prediction, equipped with a suitable metric. Thanks to its intrinsic geometric meaning, the new notion of gradient is more "natural". In the following, we are going to outline the corresponding formal framework.

To each parameter $\theta \in \mathbb{R}^p$ we associate a probability distribution $\mathsf{P}_{x,y}(\theta)$ representing the discrepancy between the model's prediction $f_\theta(x)$ and the data y. Each probability distribution is then considered as a point of a "natural" Riemannian structure, the metric tensor of which is the Fisher matrix $F(\theta)$. The use of differential geometry techniques to the study of statistical objects is at the heart of information geometry [5,7], a discipline with a wide range of applications [4,122].

Hypothesis on the loss function. For such an association to be possible, it must be assumed that, up to an additive constant, the loss function L coincides with a log-likelihood: in other words, there is a probability density $\wp$ and a real constant ν such that

$$L(y, z) = -\log \wp(y|z) + \nu. \tag{2.43}$$

[3] This does not apply to the field of research, where second-order methods are the subject of intense activities.

For instance, if the elementary loss corresponds to the least-squares function

$$L(y,z) = \frac{1}{2}\|y-z\|_2^2, \tag{2.44a}$$

then we can take the normal density

$$\wp(y|z) = (2\pi)^{-d_y/2}\exp(-\tfrac{1}{2}\|y-z\|_2^2), \tag{2.44b}$$

so that

$$L(y,z) = -\log\wp(y|z) - \frac{d_y}{2}\log(2\pi). \tag{2.44c}$$

Introduce the notation $p(y|x,\theta) = \wp(y|f_\theta(x))$. Then, the composite loss function

$$L(y,z) = L(y,f_\theta(x)) = -\log p(y|x,\theta) \tag{2.45}$$

derives from the density function $p(y|x,\theta)$ of the model's conditional predictive distribution $P_{y|x}(\theta)$. In this case, the objective function coincides with the negative Log-likelihood. As shown above, $P_{y|x}(\theta)$ is multivariate normal for the standard square loss function. It can also be proved that $P_{y|x}(\theta)$ is multinomial for the cross-entropy one. The learned distribution is therefore $\mathsf{P}_{x,y}(\theta)$ with density

$$\mathsf{p}(x,y|\theta) = q(x)p(y|x,\theta), \tag{2.46}$$

where $q(x)$ is the density of data distribution Q_x over inputs $x \in \mathbb{R}^{d_x}$.

Kullback-Leibler divergence. Given two probability distributions P and Q over $\mathbb{R}^{d_x}\times\mathbb{R}^{d_y}$, there is a standard mathematical tool to measure how much the former is dissimilar from the latter. The *Kullback-Leibler* (KL) *divergence* between two distributions P and Q of continuous random variables over $\mathbb{R}^{d_x}\times\mathbb{R}^{d_y}$ is defined to be

$$\mathrm{KL}[\mathsf{P}\,\|\,\mathsf{Q}] = \int_{\mathbb{R}^{d_x}\times\mathbb{R}^{d_y}} \mathsf{p}(x,y)\log\frac{\mathsf{p}(x,y)}{\mathsf{q}(x,y)}\,\mathrm{d}x\,\mathrm{d}y, \tag{2.47}$$

where $\mathsf{p}(x,y)$ and $\mathsf{q}(x,y)$ denote the probability densities of P and Q.

This notion, also known as *relative entropy*, quantifies the expected excess surprise from using Q as a model when the actual distribution is P. It is not a distance because $\mathrm{KL}[\mathsf{P}\,\|\,\mathsf{Q}] \neq \mathrm{KL}[\mathsf{Q}\,\|\,\mathsf{P}]$, but it satisfies the axioms of a *divergence*. But what will be most helpful to us is its infinitesimal version when setting $\mathsf{P} = \mathsf{P}_{x,y}(\theta)$ and $\mathsf{Q} = \mathsf{P}_{x,y}(\theta+\delta)$ and letting $\delta \to 0$. Before deriving a local quadratic approximation for $\mathrm{KL}[\mathsf{P}_{x,y}(\theta)\,\|\,\mathsf{P}_{x,y}(\theta+\delta)]$ (cf. Lemma 2.1), we need to introduce a new object.

Fisher information matrix. For all $\theta \in \mathbb{R}^p$, the *Fisher information matrix* (FIM) is defined to be

$$F(\theta) = \mathbb{E}_{(x,y)\sim\mathsf{P}_{x,y}(\theta)}\{\nabla_\theta \log\mathsf{p}(x,y|\theta)[\nabla_\theta\log\mathsf{p}(x,y|\theta)]^T\}. \tag{2.48}$$

This notion, also known as *Fisher information* or simply *information*, is a way of measuring the amount of information that an observable random variable carries about an unknown parameter of a distribution that models the random variable. Formally, the FIM is a covariance matrix. The following statement gives two other formulas for the FIM, one of which reveals that it can also be seen as the expectation of elementary Hessian matrices.

Proposition 2.1. *The FIM (2.48) can also be expressed as*

1. *the expectation with respect to $\mathsf{P}_{x,y}$ of the Hessian matrices of $-\log \mathsf{p}$, that is,*

$$F(\theta) = \mathbb{E}_{(x,y)\sim \mathsf{P}_{x,y}(\theta)}\{\nabla^2_{\theta\theta}[-\log \mathsf{p}(x,y|\theta)]\}; \tag{2.49}$$

2. *the expectation with respect to Q_x of the conditional expectation with respect to $P_{y|x}$ of the Hessian matrices of $L = -\log p$, that is,*

$$F(\theta) = \mathbb{E}_{x\sim Q_x, y\sim P_{y|x}(\theta)}\{\nabla_\theta \log p(y|x,\theta)[\nabla_\theta \log p(y|x,\theta)]^T\} \tag{2.50a}$$

$$= \mathbb{E}_{x\sim Q_x, y\sim P_{y|x}(\theta)}\{\nabla^2_{\theta\theta}[-\log p(y|x,\theta)]\}. \tag{2.50b}$$

Proof. We first observe that

$$\begin{aligned}\mathbb{E}_{(x,y)\sim \mathsf{P}_{x,y}(\theta)}\{\nabla_\theta \log \mathsf{p}(x,y|\theta)\} &= \int_{\mathbb{R}^{d_x}\times\mathbb{R}^{d_y}} \mathsf{p}(x,y|\theta)\nabla_\theta \log \mathsf{p}(x,y|\theta)\,\mathrm{d}x\mathrm{d}y \\ &= \int_{\mathbb{R}^{d_x}\times\mathbb{R}^{d_y}} \mathsf{p}(x,y|\theta)\frac{\nabla_\theta \mathsf{p}(x,y|\theta)}{\mathsf{p}(x,y|\theta)}\,\mathrm{d}x\mathrm{d}y \\ &= \nabla_\theta \int_{\mathbb{R}^{d_x}\times\mathbb{R}^{d_y}} \mathsf{p}(x,y|\theta)\,\mathrm{d}x\,\mathrm{d}y = \nabla_\theta 1 = 0.\end{aligned} \tag{2.51}$$

Transposing and taking the gradient with respect to θ, we have

$$\nabla_\theta \mathbb{E}_{(x,y)\sim \mathsf{P}_{x,y}(\theta)}\{[\nabla_\theta \log \mathsf{p}(x,y|\theta)]^T\} = 0. \tag{2.52}$$

However,

$$\begin{aligned}\nabla_\theta \mathbb{E}_{(x,y)\sim \mathsf{P}_{x,y}(\theta)}\{[\nabla_\theta \log \mathsf{p}(x,y|\theta)]^T\} &= \nabla_\theta \int_{\mathbb{R}^{d_x}\times\mathbb{R}^{d_y}} \mathsf{p}(x,y|\theta)[\nabla_\theta \log \mathsf{p}(x,y|\theta)]^T\,\mathrm{d}x\mathrm{d}y \\ &= \int_{\mathbb{R}^{d_x}\times\mathbb{R}^{d_y}} \nabla_\theta \mathsf{p}(x,y|\theta)[\nabla_\theta \log \mathsf{p}(x,y|\theta)]^T\,\mathrm{d}x\mathrm{d}y \\ &+ \int_{\mathbb{R}^{d_x}\times\mathbb{R}^{d_y}} \mathsf{p}(x,y|\theta)\nabla^2_{\theta\theta} \log \mathsf{p}(x,y|\theta)\,\mathrm{d}x\mathrm{d}y.\end{aligned} \tag{2.53}$$

Because $\nabla_\theta \mathsf{p} = \mathsf{p}\nabla_\theta \log \mathsf{p}$, the last equality can be reformulated as

$$\begin{aligned}\nabla_\theta \mathbb{E}_{(x,y)\sim \mathsf{P}_{x,y}(\theta)}\{[\nabla_\theta \log \mathsf{p}(x,y|\theta)]^T\} &= \mathbb{E}_{(x,y)\sim \mathsf{P}_{x,y}(\theta)}\{\nabla_\theta \log \mathsf{p}(x,y|\theta)[\nabla_\theta \log \mathsf{p}(x,y|\theta)]^T\} \\ &+ \mathbb{E}_{(x,y)\sim \mathsf{P}_{x,y}(\theta)}\{\nabla^2_{\theta\theta} \log \mathsf{p}(x,y|\theta)]\}.\end{aligned} \tag{2.54}$$

From (2.52), it follows that

$$\mathbb{E}_{(x,y)\sim \mathsf{P}_{x,y}(\theta)}\{\nabla_\theta \log \mathsf{p}(x,y|\theta)[\nabla_\theta \log \mathsf{p}(x,y|\theta)]^T\} = -\mathbb{E}_{(x,y)\sim \mathsf{P}_{x,y}(\theta)}\{\nabla^2_{\theta\theta} \log \mathsf{p}(x,y|\theta)\}. \tag{2.55}$$

This proves (2.49). To derive (2.50), we notice that

$$\begin{aligned}\nabla_\theta \log \mathsf{p}(x,y|\theta) &= \nabla_\theta \log[q(x)p(y|x,\theta)] \\ &= \nabla_\theta \log q(x) + \nabla_\theta \log p(y|x,\theta) \\ &= \nabla_\theta \log p(y|x,\theta),\end{aligned} \tag{2.56}$$

the last equality being due to the fact that $q(x)$ does not depend on θ. □

Equation (2.50a) shows that the FIM is always positive semi-definite, since it is defined as the expectation of positive semi-definite matrices. From equation (2.50b), the FIM can be interpreted as being the expectation over the model distribution of the Hessian of an elementary loss function. This makes the FIM "homogeneous" to a Hessian matrix of the loss function.

We are now in a position to connect the KL divergence to the FIM and to highlight the role of the latter in the local behavior of the former.

Lemma 2.1. *The FIM* (2.48) *defines the local quadratic approximation of the KL divergence. In other words, for all* $\delta \in \mathbb{R}^p$,

$$\mathrm{KL}[\mathsf{P}_{x,y}(\theta) \,\|\, \mathsf{P}_{x,y}(\theta+\delta)] = \frac{1}{2}\delta^T F(\theta)\delta + O(\|\delta\|^3). \tag{2.57}$$

Proof. For convenience, let us put $\phi(\theta) = \log \mathsf{p}(x,y|\theta)$ and $\phi(\theta+\delta) = \log \mathsf{p}(x,y|\theta+\delta)$. From definition (2.47), we infer that

$$\begin{aligned} \mathrm{KL}[\mathsf{P}_{x,y}(\theta) \,\|\, \mathsf{P}_{x,y}(\theta+\delta)] &= \int \mathsf{p}(x,y|\theta) \log \frac{\mathsf{p}(x,y|\theta)}{\mathsf{p}(x,y|\theta+\delta)} \,\mathrm{d}x\mathrm{d}y \\ &= \int \mathsf{p}(x,y|\theta)\phi(\theta)\,\mathrm{d}x\mathrm{d}y - \int \mathsf{p}(x,y|\theta)\phi(\theta+\delta)\,\mathrm{d}x\mathrm{d}y, \end{aligned} \tag{2.58}$$

where all integrals are implicitly taken over $\mathbb{R}^{d_x} \times \mathbb{R}^{d_y}$. Using the Taylor expansion

$$\phi(\theta+\delta) = \phi(\theta) + \delta^T \nabla_\theta \phi(\theta) + \frac{1}{2}\delta^T \nabla^2_{\theta\theta}\phi(\theta)\delta + O(\|\delta\|^3). \tag{2.59}$$

we have

$$\begin{aligned} \int \mathsf{p}(x,y|\theta)\phi(\theta+\delta)\,\mathrm{d}x\mathrm{d}y = \int \mathsf{p}(x,y|\theta)\phi(\theta)\,\mathrm{d}x\mathrm{d}y + \delta^T \int \mathsf{p}(x,y|\theta)\nabla_\theta\phi(\theta)\,\mathrm{d}x\mathrm{d}y \\ + \frac{1}{2}\delta^T \left[\int \mathsf{p}(x,y|\theta)\nabla^2_{\theta\theta}\phi(\theta)\,\mathrm{d}x\mathrm{d}y \right] \delta + O(\|\delta\|^3). \end{aligned} \tag{2.60}$$

The second summand in the right-hand side of (2.60) vanishes, for

$$\int \mathsf{p}(x,y|\theta)\nabla_\theta\phi(\theta)\,\mathrm{d}x\mathrm{d}y = \int \mathsf{p}(x,y|\theta)\frac{\nabla_\theta \mathsf{p}(x,y|\theta)}{\mathsf{p}(x,y|\theta)}\mathrm{d}x\mathrm{d}y = \nabla_\theta \int \mathsf{p}(x,y|\theta)\mathrm{d}x\mathrm{d}y = \nabla_\theta 1. \tag{2.61}$$

The third summand in the right-hand side of (2.60) coincides with $-\frac{1}{2}\delta^T F(\theta)\delta$ by virtue of (2.49). Therefore, inserting

$$\int \mathsf{p}(x,y|\theta)\phi(\theta+\delta)\,\mathrm{d}x\mathrm{d}y = \int \mathsf{p}(x,y|\theta)\phi(\theta)\,\mathrm{d}x\mathrm{d}y - \frac{1}{2}\delta^T F(\theta)\delta + O(\|\delta\|^3) \tag{2.62}$$

into equation (2.58), we end up with the desired result (2.57). □

Natural gradient descent. Let us go back to the problem of minimizing the empirical risk $h(\theta)$. Assuming that $F(\theta)$ is invertible for all θ, the natural gradient descent (NGD) method [3] is defined as

$$\theta_{k+1} = \theta_k - \alpha_k [F(\theta_k)]^{-1} \nabla_\theta h(\theta_k), \tag{2.63}$$

where $\alpha_k > 0$ is a dimensionless learning rate. In practice, we will of course need to consider a regularized FIM

$$F_\bullet(\theta_k) = F(\theta_k) + \lambda_k I, \qquad \lambda_k > 0, \tag{2.64}$$

and to perform instead

$$\theta_{k+1} = \theta_k - \alpha_k [F_\bullet(\theta_k)]^{-1} \nabla_\theta h(\theta_k). \tag{2.65}$$

But for the time being, we shall stay with the unregularized version (2.63) in order to better understand the essence of NGD.

The NGD (2.63) can be trivially seen as a second-order method in which the curvature matrix $C(\theta_k)$ is set to be the Fisher matrix $F(\theta_k)$. According to the next Proposition, NGD can also be regarded as a first-order method using the natural descent direction $-[F(\theta)]^{-1}\nabla_\theta h(\theta)$, which is the steepest descent direction in the sense of the KL divergence.

Proposition 2.2. *Let $\|\cdot\|_{F(\theta)}$ be the norm defined by the FIM, that is,*

$$\|\delta\|_{F(\theta)} = [\delta^T F(\theta)\delta]^{1/2}, \qquad \delta \in \mathbb{R}^p. \tag{2.66}$$

Then,

$$-\frac{[F(\theta)]^{-1}\nabla_\theta h(\theta)}{\|[F(\theta)]^{-1}\nabla_\theta h(\theta)\|_{F(\theta)}} = \lim_{c \downarrow 0} \frac{1}{c} \operatorname*{argmin}_{\|\delta\|_{F(\theta)}=c} h(\theta + \delta). \tag{2.67}$$

Proof. For a fixed $c > 0$, consider the Lagrangian

$$\mathcal{L}(\delta, \lambda) = h(\theta + \delta) + \lambda(\delta^T F(\theta)\delta - c^2) \tag{2.68}$$

and let us look for its saddle-points $(\delta(c), \lambda(c))$, which solve

$$\nabla_\theta h(\theta + \delta(c)) + 2\lambda(c) F(\theta)\delta(c) = 0, \tag{2.69a}$$

$$[\delta(c)]^T F(\theta)\delta(c) - c^2 = 0. \tag{2.69b}$$

From the first equation, we deduce that

$$\delta(c) = -\frac{1}{2\lambda(c)} [F(\theta)]^{-1}\nabla_\theta h(\theta + \delta(c)). \tag{2.70}$$

Hence, by taking the $F(\theta)$-norm,

$$\|\delta(c)\|_{F(\theta)} = \frac{1}{2|\lambda(c)|} \|[F(\theta)]^{-1}\nabla_\theta h(\theta + \delta(c))\|_{F(\theta)}. \tag{2.71}$$

Since $\|\delta(c)\|_{F(\theta)} = c$, we must have

$$2\lambda(c) = \pm \frac{1}{c} \|[F(\theta)]^{-1}\nabla_\theta h(\theta + \delta(c))\|_{F(\theta)}, \tag{2.72}$$

which entails

$$\delta(c) = \mp c \frac{[F(\theta)]^{-1}\nabla_\theta h(\theta + \delta(c))}{\|[F(\theta)]^{-1}\nabla_\theta h(\theta + \delta(c))\|_{F(\theta)}}. \tag{2.73}$$

It can be verified that the $+$ sign corresponds to an ascent direction, while the $-$ sign provides a descent direction. Thus,

$$\frac{\delta(c)}{c} = -\frac{[F(\theta)]^{-1}\nabla_\theta h(\theta + \delta(c))}{\|[F(\theta)]^{-1}\nabla_\theta h(\theta + \delta(c))\|_{F(\theta)}}. \tag{2.74}$$

When $c \downarrow 0$, it is obvious that $\delta(c) \to 0$ because $\|\delta(c)\|_{F(\theta)} = c$. As a result,

$$\lim_{c\downarrow 0} \frac{\delta(c)}{c} = -\frac{[F(\theta)]^{-1}\nabla_\theta h(\theta)}{\|[F(\theta)]^{-1}\nabla_\theta h(\theta)\|_{F(\theta)}}, \tag{2.75}$$

as claimed. □

Formula (2.67), which clarifies the meaning to be given to the natural gradient descent direction $-[F(\theta)]^{-1}\nabla_\theta h(\theta)$, is to be compared with

$$-\frac{\nabla h(\theta_k)}{\|\nabla h(\theta_k)\|_2} = \lim_{c\downarrow 0} \frac{1}{c} \operatorname*{argmin}_{\|\delta\|_2 = c} h(\theta_k + \delta), \tag{2.76}$$

which shows that the ordinary descent gradient $-\nabla_\theta h(\theta)$ is the steepest descent direction in the sense of the Euclidean norm. In short, we have merely switched from the Euclidean metric to the local metric associated with the FIM.

Continuous NGD and invariance. The natural gradient descent (2.63) can be envisioned as the discretization by Euler's explicit scheme of the differential equation

$$\frac{\mathrm{d}\theta}{\mathrm{d}\tau}(\tau) = -[F(\theta(\tau))]^{-1}\nabla_\theta h(\theta(\tau)), \tag{2.77}$$

in which τ plays the role of a fictitious time and α_k can be thought of as a time-step. The ODE (2.77), said to be the *continuous NGD*, has the remarkable following feature.

Proposition 2.3. *The continuous NGD (2.77) is invariant to all differentiable and invertible transformations. In other words, for any differentiable and invertible change of variable and functions*

$$\theta = \Psi(\widehat{\theta}), \qquad \widehat{h}(\widehat{\theta}) = h(\theta), \qquad \widehat{F}(\widehat{\theta}) = F(\theta), \tag{2.78}$$

the trajectory of

$$\frac{\mathrm{d}\widehat{\theta}}{\mathrm{d}\tau}(\tau) = -[\widehat{F}(\widehat{\theta}(\tau))]^{-1}\nabla_{\widehat{\theta}}\widehat{h}(\widehat{\theta}(\tau)) \tag{2.79}$$

matches pointwise that of (2.63) by $\theta(\tau) = \Psi(\widehat{\theta}(\tau))$.

Proof. Owing to the chain rule, we have

$$\frac{\mathrm{d}\widehat{\theta}}{\mathrm{d}\tau} = [\nabla_{\widehat{\theta}}\Psi]^{-1}\frac{\mathrm{d}\theta}{\mathrm{d}\tau}, \qquad \nabla_{\widehat{\theta}}\widehat{h}(\widehat{\theta}) = [\nabla_{\widehat{\theta}}\Psi]^T\nabla_\theta h(\theta). \tag{2.80}$$

To compute $\widehat{F}(\widehat{\theta})$, let us write

$$\widehat{F}(\widehat{\theta}) = \mathbb{E}(\nabla_{\widehat{\theta}}\widehat{L}(\widehat{\theta})\,[\nabla_{\widehat{\theta}}\widehat{L}(\widehat{\theta}))]^T), \tag{2.81}$$

where $\widehat{L}(\widehat{\theta}) = L(\theta)$. Plugging $\nabla_{\widehat{\theta}}\widehat{L}(\widehat{\theta}) = [\nabla_{\widehat{\theta}}\Psi]^T\nabla_\theta L(\theta)$ into (2.81) yields

$$\widehat{F}(\widehat{\theta}) = [\nabla_{\widehat{\theta}}\Psi]^T F(\theta)\nabla_{\widehat{\theta}}\Psi. \tag{2.82}$$

Inserting (2.80) and (2.82) into (2.79), we get

$$[\nabla_{\widehat{\theta}}\Psi]^{-1}\frac{\mathrm{d}\theta}{\mathrm{d}\tau} = -\{[\nabla_{\widehat{\theta}}\Psi]^T F(\theta)\nabla_{\widehat{\theta}}\Psi\}^{-1}[\nabla_{\widehat{\theta}}\Psi]^T\nabla_\theta h(\theta) = -[\nabla_{\widehat{\theta}}\Psi]^{-1}[F(\theta)]^{-1}\nabla_\theta h(\theta). \tag{2.83}$$

Multiplying by $\nabla_{\widehat{\theta}}\Psi$, we recover (2.77). □

Undeniably, invariance with respect to any nonlinear reparametrization is a strongly favorable property that singles out the continous NGD method from its competitors. The Euclidean gradient is highly sensitive to the choice of θ, since it is unable to reflect the underlying structure of the objects represented by the parameters. By complying with the intrinsic geometry of probability distributions, the natural gradient benefits from a more solid mathematical foundation and offers a better chance of tackling high dimensional problems.

It is however worth noting that as soon as a discretization such as (2.63) is applied to the continuous NGD, invariance with respect to nonlinear reparametrization is definitely lost, although invariance with respect to linear or affine changes of variables still holds true. Consequently, it cannot be rigourously claimed that the NGD (2.63) is invariant with respect to nonlinear reparametrization. Nevertheless, the existence of invariance at the continuous level remains a theoretical advantage that supports the method. This phenomenon is also frequently encountered in other optimization methods built on invariance principles [125].

Following [158] and resorting to Riemannian optimization techniques [20], we can try to mitigate the loss of invariance by discretizing with a higher-order scheme a modified version of (1.29) that incorporates a geodesic correction to guarantee that the trajectory remains on the manifold. But since the calculation of the Christoffel symbols involved in the Levi-Civita connection is extremely heavy for a network, it is not always worth the effort.

Fisher efficiency. Another attractive feature of NGD that is often mentioned in the literature is its asymptotic *Fisher efficiency* property [3, 109, 110]. In other words, the solution produced by (2.63) will be an unbiased estimator of the global optimum θ^{opt} and its asymptotic variance will reach the Cramer-Rao bound [32, 136]. Owing to this property, NGD seems to be the best possible method for optimizing the model parameter θ. Unfortunately, the "Fisher efficiency" property is based on several assumptions (convergence of iterates to the global optimum, computation of the FIM with full training data, "realizability" of the model) that are very unlikely to be hold in neural network optimization [109]. Still, the NGD method remains very attractive and many authors [19, 109, 116] have proposed results similar to the Fisher efficiency property and which are not or less restrictive.

True versus empirical FIM. In general, the true distribution Q_x of the inputs x is unknown. The FIM is therefore approximated using the empirical distribution $\hat{Q}_x$ through a mini-batch $\mathcal{S}'$ by

$$F(\theta) \approx \frac{1}{|\mathcal{S}'|} \sum_{x^{(b)} \in \mathcal{S}'} \mathbb{E}_{y \sim P_{y|x^{(b)}}(\theta)} \{\nabla_\theta \log p(y|x^{(b)}, \theta) [\nabla_\theta \log p(y|x^{(b)}, \theta)]^T\}. \tag{2.84}$$

This expression can be viewed as a Monte-Carlo approximation of the FIM according to the distribution Q_x. Sometimes, $F(\theta)$ is approximated using both the empirical distribution of inputs and targets from training data. This yields the expression

$$F(\theta) \approx \frac{1}{|\mathcal{S}'|} \sum_{(x^{(b)}, y^{(b)}) \in \mathcal{S}'} \nabla_\theta \log p(y^{(b)}|x^{(b)}, \theta) [\nabla_\theta \log p(y^{(b)}|x^{(b)}, \theta)]^T =: \widetilde{F}(\theta). \tag{2.85}$$

This matrix $\widetilde{F}(\theta)$, referred to as the *empirical FIM*, has been used in many works [56, 96, 133]. However, this approximation is not a Monte-Carlo estimate of the FIM insofar as the targets y are not sampled according to the model conditional distribution $P_{y|x}(\theta)$, but rather according to the data conditional distribution $Q_{y|x}$.

Two conditions must be met in order for the empirical Fisher to converge to the FIM [94]. Firstly, the model must be "realizable", that is, there exists an optimum parameter θ^{opt} such that $P_{y|x}(\theta^{\text{opt}}) = Q_{y|x}$. Secondly, the number of data points used to estimate $\widetilde{F}(\theta)$ must be sufficiently large to allow $\widetilde{F}(\theta)$ to converge to the FIM. These unlikely conditions undermine the quality of empirical Fisher and its use is therefore not recommended [94, 110]. A more relevant approximation to the FIM advocated by various authors [68, 112] is to keep the same expression as $\widetilde{F}(\theta)$, but instead use targets y sampled according to the conditional distribution of the model $P_{y|x}(\theta)$. However, the main drawback of this approach is that it requires to explicitly define $P_{y|x}(\theta)$ and sample targets y's according to it. While the definition of $P_{y|x}(\theta)$ is straightforward for most use cases, sampling according to it can be very costly, especially when inputs and targets are high-dimensional vectors.

Relationship with GGN matrix. As shown by (2.49), the FIM is an average of elementary Hessian matrices. The density with respect to which the expectation is carried out acts in such a way that the result no longer contains any second derivative. This reminds us of the GGN matrix (2.41) and makes us wonder about the relationship between them. The statement below says that for some distributions $P_{y|x}(\theta)$, these two matrices are identical to each other.

Theorem 2.1 (Martens [110])**.** *If $p(y|z)$ is an exponential family distribution with natural parameters z, namely,*

$$\log p(y|z) = z^T T(y) - \Xi(z) + \log \Upsilon(y), \tag{2.86}$$

where $T(y)$ is a sufficient statistic, $\Xi(z)$ is the cumulant generating function and $\Upsilon(y)$ is the normalizing constant, then the FIM coincides with the GGN matrix.

Proof. We are going to derive the equality for a single sample (x, y) but the proof remains valid in the case of a mini-batch. By the chain rule, we have

$$\nabla_\theta \log \left(p(y|z)_{z=f_\theta(x)}\right. = [J_{f_\theta(x)}]^T \nabla_z \log \left(p(y|z)_{|z=f_\theta(x)}\right. \tag{2.87}$$

and so,

$$\begin{aligned} F(\theta) &= \mathbb{E}_{y\sim P_{y|x}(\theta)}\{\nabla_\theta \log p(y|x,\theta)[\nabla_\theta \log p(y|x,\theta)]^T\} \\ &= \mathbb{E}_{y\sim P_{y|x}(\theta)}\{[J_{f_\theta(x)}]^T \nabla_z \log p(y|z)_{|z=f_\theta(x)}[\nabla_z \log p(y|z)]^T_{|z=f_\theta(x)} J_{f_\theta(x)}\} \\ &= [J_{f_\theta(x)}]^T \mathbb{E}_{y\sim P_{y|x}(\theta)}\{\nabla_z \log p(y|z)_{|z=f_\theta(x)} \nabla_z \log p(y|z)]^T_{|z=f_\theta(x)}\} J_{f_\theta(x)} \\ &= [J_{f_\theta(x)}]^T \mathbb{E}_{y\sim P_{y|x}(\theta)}\{-\nabla^2_{zz} \log p(y|z)_{|z=f_\theta(x)}\} J_{f_\theta(x)}. \end{aligned} \tag{2.88}$$

The last equality results from

$$\mathbb{E}_{y\sim P_{y|x}(\theta)}\{\nabla_z \log p(y|z)_{|z=f_\theta(x)}[\nabla_z \log p(y|z)]^T_{|z=f_\theta(x)}\} = \mathbb{E}_{y\sim P_{y|x}(\theta)}\{-\nabla^2_{zz} \log p(y|z)_{|z=f_\theta(x)}\},$$

which can be proven by following the same process as in the proof Proposition 2.1.

Let us now consider the GGN expression (2.41) with $L(z, y) = -\log p(y|z)_{|z=f_\theta(x)}$. We have

$$G(\theta) = [J_{f_\theta(x)}]^T H_L(z) J_{f_\theta(x)}, \tag{2.89}$$

with

$$H_L(z) = \nabla^2_{zz} L(z, y)_{|z=f_\theta(x)} = -\nabla^2_{zz} \log p(y|z) = \nabla^2_{zz} \Xi(z) \tag{2.90}$$

which does not depend on y. Hence,

$$\mathbb{E}_{y\sim P_{y|x}(\theta)}\left[H_L(z)\right] = H_L(z). \tag{2.91}$$

The combination of equations (2.88), (2.89) and (2.91) yields $F(\theta) = G(\theta)$. □

Approximation of the Fisher matrix. The pioneering works of Amari [3, 127, 183] highlighted the value of natural gradient methods for learning. However, as soon as the number of parameters reaches a certain threshold, we are faced with the same difficulties as for other second-order methods: the inversion of the FIM, which is full, becomes the main obstruction hampering the method's performance.

The approximation of the FIM by its diagonal, suggested by [95] or inspired from [45, 88, 165] for the empirical FIM, turns out to be too coarse and does not retain enough of the information contained in the original matrix. The next attempts, to approximate respectively the empirical Fisher matrix (TONGA method [96]) and the exact Fisher matrix [124] by a block-diagonal structure with one block per neuron, brought no significant improvement. To make the FIM sparser, some authors opt for dynamic reparametrizations of the network [135, 173, 178] so that most of the scalar quantities associated with units such as activity and local derivative are zero on average. Another idea in the same vein is to adaptively modify the internal representation of the network during the learning phase, with the aim of controlling the FIM's condition number (PRONG method [38]).

The first approximations using a Kronecker product appeared with [69, 75, 133], where the Fisher matrix is replaced by a block-diagonal matrix. This time, each block represents a network layer. If layer i has d_{i-1} inputs and d_i outputs, then there are $p_i = (d_{i-1}+1)d_i$ weights and block $F_{i,i} \in \mathbb{R}^{p_i \times p_i}$ of the Fisher matrix is approximated by

$$F_{i,i} \approx \bar{A}_{i-1} \otimes G_i, \tag{2.92}$$

where $\otimes$ denotes the Kronecker product (see Definition 3.1) and the matrices

$$\bar{A}_{i-1} \in \mathbb{R}^{(d_{i-1}+1)\times(d_{i-1}+1)} \qquad \text{et} \qquad G_i \in \mathbb{R}^{d_i \times d_i}$$

are much smaller than $F_{i,i}$. These factors are much cheaper to store and reverse. Their product is also easy and inexpensive to invert, thanks to the formula

$$(\bar{A}_{i-1} \otimes G_i)^{-1} = \bar{A}_{i-1}^{-1} \otimes G_i^{-1}. \tag{2.93}$$

The KFAC approximation introduced by Martens and Grosse [112] follows the same line as (2.92), with a specific definition for the factors $\bar{A}_{i-1}$ and G_i enabling faster computation. What also contributed to its great success over its predecessors was its strategy of preserving the factorized form for the much-needed Tikhonov regularization

$$F_\bullet := F + \lambda I_p, \qquad \lambda > 0. \tag{2.94}$$

More specifically, each diagonal block of the regularized FIM is approximated by

$$[F_\bullet]_{i,i} \approx \left(\bar{A}_{i-1} + \pi_i \sqrt{\lambda}\, I_{d_{i-1}+1}\right) \otimes \left(G_i + \pi_i^{-1} \sqrt{\lambda}\, I_{d_i}\right) =: [F_{\bullet \mathrm{KFAC}}]_{i,i}, \tag{2.95}$$

where the coefficient π_i is "optimally" adjusted according to the data, for instance

$$\pi_i = \sqrt{\frac{\mathrm{tr}(\bar{A}_{i-1})/d_{i-1}}{\mathrm{tr}(\bar{G}_i)/d_i}}. \tag{2.96}$$

Furthermore, the *lambda* weight is gradually adjusted according to the behavior of the iterates. All these ingredients make KFAC a powerful and efficient method that marked a turning point in the development of natural gradient methods. Initially designed for a multilayer perceptron,

KFAC has been generalized to other network architectures such as CNN [68] (the method is then called KFC) and RNN [111]. For each new architecture, new more or less legitimate assumptions are made to define the factors $\bar{A}_{i-1}$ and G_i.

KFAC's promising features have prompted a great many subsequent work. On the one hand, it has been successfully deployed in the context of Bayesian deep learning [186], deep reinforcement learning [179] and Laplace approximation [140]. On the other hand, its parallel implementation in the perspective of very large models has been the subject of much research. One example is the distributed version of [8], where the gradient computation is distributed over several GPUs using the standard synchronous SGD model, while the FIM blocks and their inverses are computed with CPUs asynchronously (while the network is still learning). In another approach [126], all computations are carried out synchronously with GPUs, with results communicated optimally between GPUs. Both these distributed paradigms claim a reduction in computation time by a factor of at least 2 compared with a sequential SGD.

At the algorithmic level, several attempts have been made to refine KFAC. The EKFAC [56] method rescales Kronecker factors with a diagonal variance computed in a Kronecker-factored eigenbasis. The TKFAC method [54] preserves trace invariance relationship between the approximate and the exact FIM. By assuming that each block of the latter corresponds to the covariance of a tensor normal distribution in the model, the TNT method [139] puts forward a Kronecker block-diagonal approximation that has the advantage of being free of the layer structure. Beyond the Fisher matrix, the idea of Kronecker factorization can also be applied to the approximation of the Hessian matrix, as in KBFGS [61], where the complexity of computing the inverse of the Kronecker factors is mitigated by low-rank updates. It also generalizes to the approximation of the GGN matrix of MLPs, as shown in [61].

2.A Experimental evaluation of first-order methods

Here, we evaluate and compare the performance of different first-order optimization methods described in subsection 2.2.2. To do so, we consider the optimization of two CNN architectures in a classification task; Resnet-18 [74] and Resnet-50 [74], with two datasets, one of which, CIFAR-10 [92] is of medium size and the other, ImageNet [36] is of very large size. All experiments were performed with PyTorch framework [129] on Topaze supercomputer with Nividia Ampere A100 GPU and AMD Milan@2.45GHz CPU. In both experiments, the loss function used is the cross-entropy and the metric used is the **top-1** accuracy. For Imagenet, due to large number of classes (1000), we also measured the **top-5** accuracy. When considering a classification task with C classes, the predicted output is a vector of size C containing probabilities of different classes. In the case of **top-1** accuracy, a prediction is correct when the highest probability in the predicted vector corresponds to the target class. As for the **top-5** accuracy, a prediction is said to be correct when one at least of the 5 highest probabilities in the predicted vector corresponds to the target class. In both cases, the accuracy is defined as the number of correct predictions divided by the number of data points evaluated.

2.A.1 CIFAR10

CIFAR-10 (Canadian Institute For Advanced Research) [92] is a dataset of images commonly used to benchmark deep learning algorithms. It consists of 60000 32×32 colour images in 10 classes, with 6000 images per class. It is divided into two parts: 50000 training images and 10000 validation images. For this dataset, we used Resnet-18 [74] which is a CNN with 18 layers. For data augmentation, we used the same techniques as in [67] i.e. the input image is 224×224 randomly resized-crop and also randomly flipped horizontally. For each optimizer, we selected the best hyper-parameters that give the best results on the validation set (see Table 2.1 for such hyper-parameters). We found that the learning rate has a large impact on the performance of each optimizer. Finally for each optimizer, we trained for 90 epochs and used a batch size of 256.

Figure 2.7 shows the performance of the optimizers on the considered problem. As we can see on the figure, for each measured quantity (loss and accuracy), either on training or validation data, SGD shows the best performance, then comes Adam, followed by AdaGrad and finally comes RMSProp.

	Learning rate	Momentum parameter β	(β_1, β_2)	Weight decay
SGD	10^{-1}	0.9	✗	10^{-4}
ADAGRAD	10^{-3}	✗	✗	10^{-4}
ADAM	10^{-3}	✗	$(0.9, 0.999)$	10^{-4}
RMSPROP	10^{-3}	0.9	✗	10^{-4}

Table 2.1: Hyper-parameters of each optimizer of experiments on CIFAR-10.

2.A.2 ImageNet

The ImageNet dataset [145] consists of 1.2 million training images and 50000 validation images and contains 1000 classes. We used the Resnet-50 architecture and the same data augmentation techniques as for CIFAR-10. For the choice of learning rates, we used a learning rate schedule

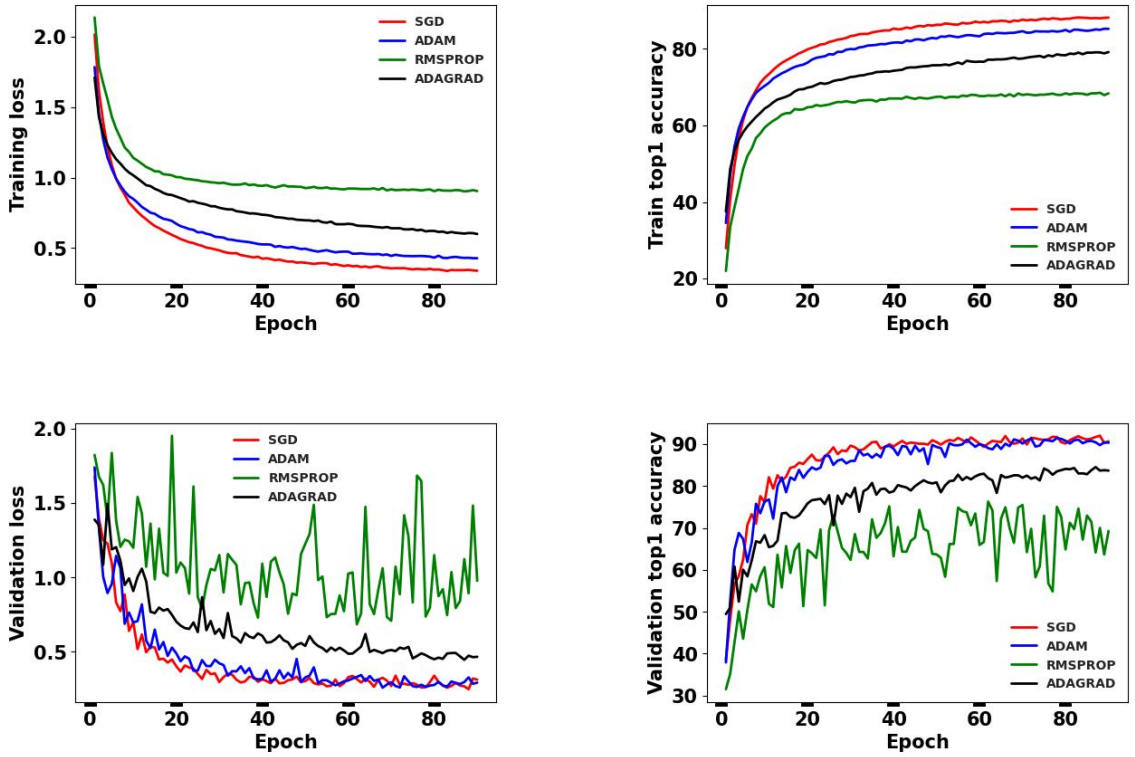
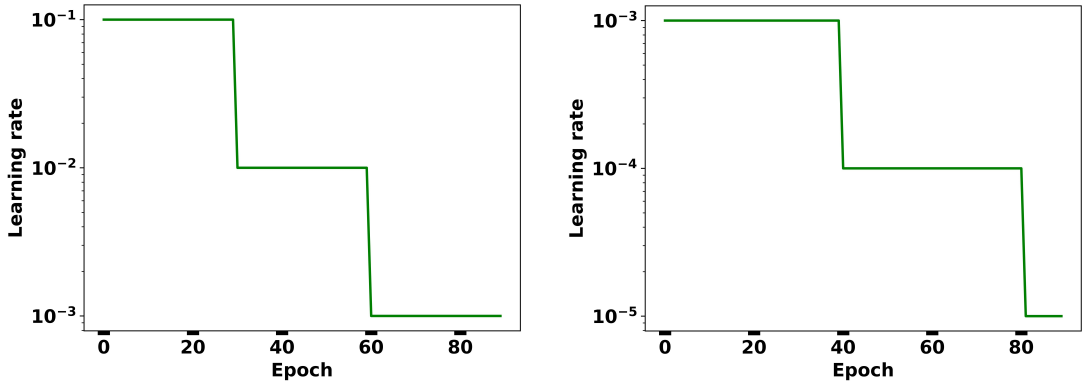


Figure 2.7: Error and accuracy curves of different first-order optimizers in CIFAR10 experiments. In the **first** row, **first** figure displays training loss *vs* epoch while the **second** one represents training top-1 accuracy *vs* epoch. In the **second** row, the first **figure** depicts validation loss *vs* epoch and the **last** displays validation top-1 accuracy *vs* epoch.

as suggested in [67]. To this end, we adopted for each optimizer, the learning schedule that we found to work best for it (see Figure 2.8). The other hyper-parameters are the same as in the CIFAR-10 experiments. As the size of the training set is very large, we adopted the *distributed synchronous data-parallel* approach [67], where data is distributed across different GPU workers and the model is replicated across them. We used 64 GPU workers and a total batch size of 2048, which corresponds to a batch size of 32 per GPU worker. Similarly to CIFAR10 experiments, we trained the model for 90 epochs with each optimizer. Note that with this distributed approach and the 64 GPUs, training of the network with each optimizer lasted for about 3 hours. With such a large amount of data, the training could last several days or even weeks without the distributed approach.

As we can see in Figure 2.9, like in the case of CIFAR10, SGD shows the best performance, followed by Adam, AdaGrad and RMSProp in decreasing order of performance. The two performed experiments support a well known fact in the literature: a well-tuned SGD generally outperforms adaptive gradient optimizers.



(a) SGD & ADAGRAD (b) RMSPROP & ADAM

Figure 2.8: Learning schedule used for each optimizer in Imagenet experiments.

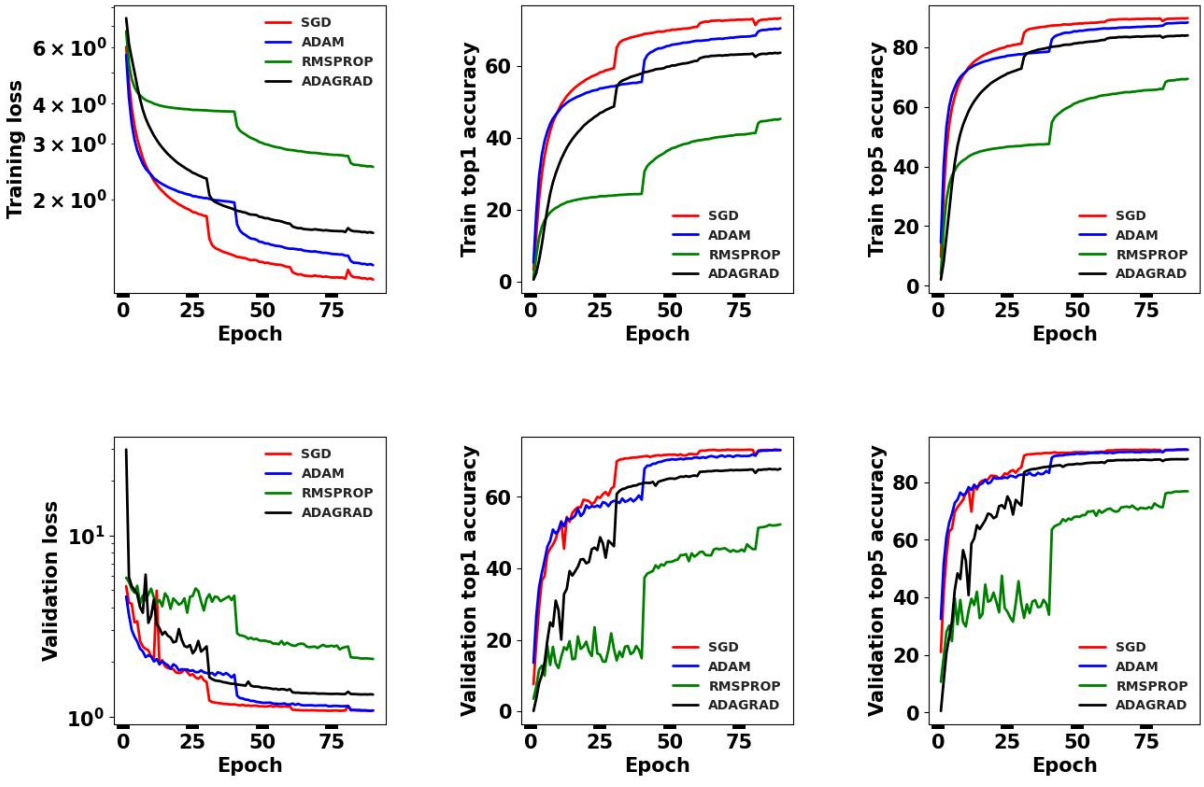


Figure 2.9: Error and accuracy curves of different first-order optimizers in Imagenet experiments. In the **first** row, from right to left, **first** figure shows training loss *vs* epoch, **second** one displays training top-1 accuracy *vs* epoch and the **third** depicts training top-5 accuracy *vs* epoch. As for the **second** row, from left to right, **first**, **second** and **last** figures display respectively validation loss*vs* epoch, validation top-1 accuracy *vs* epoch and validation top-5 accuracy *vs* epoch.

Chapter 3

Efficient approximations to the Fisher matrix of deep neural networks using Kronecker-product singular value decomposition

Contents

In this chapter, we present our first contribution to the subject of natural gradient methods for neural networks. More specifically, we propose a few alternative approximations of the Fisher matrix that do not rely on any objectionable statistical assumptions, while keeping the spirit of KFAC [112] and KFC [68].

The preliminary backgrounds regarding the Kronecker product and the KFAC approximation are first recalled in §3.1. Next, in §3.2, we design four novel methods all of which are based solely on the principle of minimizing some error between two matrices in the Frobenius norm. The calculation of these approximations involves the Kronecker-product SVD and their computational efficiency is carefully inspected. Several numerical experiments for autoencoders are described and commented on in §3.3. Finally, in §3.4, we investigate a new extension of our methods to convolutional layers.

Except for the last section §3.4 on CNNs, the text below is a ***replica of the article [90]****, which was* ***accepted for publication*** *in* ESAIM: Proceedings and Surveys.

3.1 Preliminary backgrounds

In addition to key concepts presented in the previous chapter, we introduce in this section new prerequisites necessary to elaborate the four novels approximations.

We consider a feed forward neural network (MLP) that maps its input x to an output z according to equation (2.2). For a given input-target pair (x, y), the gradient of the loss $L(y, f_\theta(x))$ w.r.t to the weights is computed by the back-propagation algorithm [97]. For convenience, we adopt the shorthand notation $\mathcal{D}v = \nabla_v L$ for the derivative of L w.r.t any variable v, as well as the special symbol $g_i = \mathcal{D}s_i$ for the preactivation derivative. Starting from $\mathcal{D}a_\ell = \partial_z L(y, z = a_\ell)$, we perform

$$g_i = \mathcal{D}a_i \odot \sigma_i'(s_i), \qquad \mathcal{D}W_i = g_i \bar{a}_{i-1}^T, \qquad \mathcal{D}a_{i-1} = W_i^T g_i, \tag{3.1}$$

for i from ℓ to 1, where $\odot$ denotes the component-wise product. Finally, the gradient $\nabla_\theta L$ is retrieved as

$$\mathcal{D}\theta = [\mathrm{vec}(\mathcal{D}W_1)^T, \mathrm{vec}(\mathcal{D}W_2)^T, \dots, \mathrm{vec}(\mathcal{D}W_\ell)^T]^T. \tag{3.2}$$

The FIM associated to the network parameter θ is therefore defined as

$$F(\theta) = \mathbb{E}_{x\sim Q_x, y\sim P_{y|x}(\theta)}[\mathcal{D}\theta(\mathcal{D}\theta)^T] = \begin{bmatrix} F_{1,1} & \dots & F_{1,\ell} \\ \vdots & & \vdots \\ F_{\ell,1} & \dots & F_{\ell,\ell} \end{bmatrix}, \tag{3.3}$$

in which the block

$$\begin{aligned} F_{i,j} &= \mathbb{E}_{x\sim Q_x, y\sim P_{y|x}(\theta)}[\mathrm{vec}(\mathcal{D}W_i)\mathrm{vec}(\mathcal{D}W_j)^T] \\ &= \mathbb{E}_{x\sim Q_x, y\sim P_{y|x}(\theta)}[\mathrm{vec}(g_i \bar{a}_{i-1}^T)\mathrm{vec}(g_j \bar{a}_{j-1}^T)^T] \end{aligned} \tag{3.4}$$

is a $d_i(d_{i-1}+1) \times d_j(d_{j-1}+1)$ matrix with d_i being the number of neurons at layer i. For brevity and without any risk of ambiguity, we will omit the subscripts for the expectation and write $\mathbb{E}$ instead of $\mathbb{E}_{x\sim Q_x, y\sim P_{y|x}}(\theta)$.

The blocks of F can be given the following meaning: $F_{i,i}$ contains second-order statistics of weight derivatives on layer i, while $F_{i,j,i\neq j}$ represents correlation between weight derivatives of layers i and j.

3.1.1 Kronecker product

The *Kronecker product* and its related properties are an essential ingredient for different types of FIM approximation we will be presenting throughout this manuscript. It is therefore necessary to become familiar with this notion and some of its properties.

Definition 3.1. Let $A \in \mathbb{R}^{m_A \times n_A}$ and $B \in \mathbb{R}^{m_B \times n_B}$. Then the *Kronecker Product* of A and B is defined as the matrix

$$A \otimes B = \begin{bmatrix} A_{1,1}B & \dots & A_{1,n_A}B \\ \vdots & & \vdots \\ A_{m_A,1}B & \dots & A_{m_A,n_A}B \end{bmatrix} \in \mathbb{R}^{m_A m_B \times n_A n_B}. \tag{3.5}$$

The very first properties to be mentioned are concerned with associativity and distributivity of $\otimes$ with respect to addition. Note, however, that the Kronecker product is not commutative, i.e., $A \otimes B \neq B \otimes A$ in general.

Theorem 3.1. *For matrices A, B, C, D, and a scalar value γ, the following associativity properties hold*

1. $(A + B) \otimes C = A \otimes C + B \otimes C$,
2. $C \otimes (A + B) = C \otimes A + C \otimes B$,
3. $(\gamma A) \otimes B = A \otimes (\gamma B) = \gamma(A \otimes B)$,
4. $(A \otimes B) \otimes C = A \otimes (B \otimes C)$.

Proof. All identities directly follow from Definition 3.1. □

The next two statements clarify the Kronecker product in two special cases, when the matrices involved are vectors and when they are diagonal.

Theorem 3.2. *Let $x \in \mathbb{R}^{m_x}$ and $y \in \mathbb{R}^{m_y}$. Then*

$$x \otimes y^T = y^T \otimes x = xy^T. \tag{3.6}$$

Proof. On the one hand,

$$x \otimes y^T = \left[x^{[1]}y, \dots, x^{[m_x]}y\right]^T = \begin{bmatrix} x^{[1]}y^{[1]} & \dots & x^{[1]}y^{[m_y]} \\ \vdots & & \vdots \\ x^{[m_x]}y^{[1]} & \dots & x^{[m_x]}y^{[m_y]} \end{bmatrix} = xy^T.$$

On the other hand,

$$y^T \otimes x = \left[y^{[1]}x, \dots, y^{[m_y]}x\right] = \begin{bmatrix} x^{[1]}y^{[1]} & \dots & x^{[1]}y^{[m_y]} \\ \vdots & & \vdots \\ x^{[m_x]}y^{[1]} & \dots & x^{[m_x]}y^{[m_y]} \end{bmatrix} = xy^T.$$

This completes the proof. □

Theorem 3.3. *The Kronecker product of two diagonal matrices yields a diagonal matrix.*

Proof. Directly follows from Definition 3.1. □

The upcoming assertion, known as *mixed-product property*, is a handy result that expresses the ordinary matrix product of two Kronecker products.

Theorem 3.4. *Let $A \in \mathbb{R}^{m_A \times n_A}$, $B \in \mathbb{R}^{m_B \times n_B}$, $C \in \mathbb{R}^{m_C \times n_C}$, $D \in \mathbb{R}^{m_D \times n_D}$ with $n_A = m_C$ and $n_B = m_D$. Then,*

$$(A \otimes B)(C \otimes D) = AC \otimes BD \in \mathbb{R}^{m_A m_B \times n_C n_D}. \tag{3.7}$$

Proof. We have

$$\begin{aligned}(A \otimes B)(C \otimes D) &= \begin{bmatrix} A_{1,1}B & \dots & A_{1,n_A}B \\ \vdots & & \vdots \\ A_{m_A,1}B & \dots & A_{m_A,n_A}B \end{bmatrix} \begin{bmatrix} C_{1,1}D & \dots & C_{1,n_C}D \\ \vdots & & \vdots \\ C_{m_C,1}D & \dots & C_{m_C,n_C}D \end{bmatrix} \\ &= \begin{bmatrix} \sum_{k=1}^{n_A} A_{1,k}C_{k,1}BD & \dots & \sum_{k=1}^{n_A} A_{1,k}C_{k,n_C}BD \\ \vdots & & \vdots \\ \sum_{k=1}^{n_A} A_{m_A,k}C_{k,1}BD & \dots & \sum_{k=1}^{n_A} A_{m_A,k}C_{k,n_C}BD \end{bmatrix} \\ &= (AC) \otimes (BD)\end{aligned}$$

which concludes the proof. □

As a consequence of the previous theorem, the power operator is distributive with respect to the Kronecker product, as stated by the the following corollary.

Corollary 3.1. *For two square matrices A and B and an integer n, the following equality holds*

$$(A \otimes B)^n = A^n \otimes B^n.$$

Proof. By induction.
For $n = 2$, using Theorem 3.4, we have

$$(A \otimes B)^2 = (A \otimes B)(A \otimes B) = AA \otimes BB = A^2 \otimes B^2.$$

Let us assume the statement is true for $n \geqslant 2$ (induction hypothesis). Using the induction hypothesis and again Theorem 3.4, we have

$$(A \otimes B)^{n+1} = (A \otimes B)^n (A \otimes B) = (A^n \otimes B^n)(A \otimes B) = (A^n A) \otimes (B^n B) = A^{n+1} \otimes B^{n+1}.$$

□

Also very useful is the transpose of a Kronecker product, with two consequences regarding the preservation of symmetry and orthogonality by the Kronecker product.

Theorem 3.5. *For all matrices A and B, we have*

$$(A \otimes B)^T = A^T \otimes B^T. \tag{3.8}$$

Proof. Directly follows from definition of transpose and Definition 3.1. □

Corollary 3.2. *Let A and B be two matrices. If A and B are symmetric, then $A \otimes B$ is symmetric.*

Proof. $(A \otimes B)^T = A^T \otimes B^T = A \otimes B$ □

Corollary 3.3. *Let A and B be two square matrices. If A and B are both orthogonal, then $A \otimes B$ is also orthogonal.*

Proof. $(A \otimes B)^T (A \otimes B) = (A^T \otimes B^T)(A \otimes B) = (A^T A) \otimes (B^T B) = I \otimes I = I.$ □

Similarly, the inverse of a Kronecker product between two matrices is the Kronecker product of the inverse of each matrix in the same order. To put it another way, the inverse of a Kronecker product can be easily deduced from those of the (smaller) matrices involved.

Theorem 3.6. *Let A and B be two square matrices. If A and B are both invertible, then $A \otimes B$ is also invertible and its inverse is given as*

$$(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}. \tag{3.9}$$

Proof. By Theorem 3.4, we have

$$(A \otimes B)(A^{-1} \otimes B^{-1}) = (AA^{-1}) \otimes (BB^{-1}) = I \otimes I = I.$$

From this, we infer that $A^{-1} \otimes B^{-1}$ is the inverse of $A \otimes B$. □

Likewise, the singular value decomposition (SVD) of a Kronecker product can be deduced from those of the (smaller) matrices involved.

Theorem 3.7. *Let $A \in \mathbb{R}^{m_A \times n_A}$, $B \in \mathbb{R}^{m_B \times n_B}$ be two matrices whose singular value decompositions are*

$$A = U_A \Sigma_A V_A^T, \qquad B = U_B \Sigma_B V_B^T, \tag{3.10}$$

where $U_A \in \mathbb{R}^{m_A} \times \mathbb{R}^{m_A}$, $U_B \in \mathbb{R}^{m_B} \times \mathbb{R}^{m_B}$ are orthogonal, $V_A \in \mathbb{R}^{n_A} \times \mathbb{R}^{n_A}$, $V_B \in \mathbb{R}^{n_B} \times \mathbb{R}^{n_B}$ are orthogonal, $\Sigma_A \in \mathbb{R}^{m_A} \times \mathbb{R}^{n_A}$, $\Sigma_B \in \mathbb{R}^{m_B} \times \mathbb{R}^{n_B}$ diagonal. Then,

$$A \otimes B = (U_A \otimes U_B)(\Sigma_A \otimes \Sigma_B)(V_A \otimes V_B)^T \tag{3.11}$$

is the SVD of $A \otimes B$.

Proof. We have

$$A \otimes B = (U_A \Sigma_A V_A^T) \otimes (U_B \Sigma_B V_B^T).$$

Thanks to Theorem 3.4, we can write

$$A \otimes B = (U_A \otimes U_B)(\Sigma_A \otimes \Sigma_B)(V_A \otimes V_B)^T.$$

Since $U_A \otimes U_B$ and $V_A \otimes V_B$ are orthogonal matrices (Corollary 3.3), and $\Sigma_A \otimes \Sigma_B$ is diagonal (Theorem 3.3), then $(U_A \otimes U_B)(\Sigma_A \otimes \Sigma_B)(V_A \otimes V_B)^T$ is the SVD of $A \otimes B$. □

A direct consequence of the above theorem is the distributivity of rank with respect to $\otimes$; as is clarified by the following corollary.

Corollary 3.4. *For two square matrices A and B and an integer n, the following identity holds*

$$rank(A \otimes B) = rank(B \otimes A) = rank(A) rank(B).$$

Proof. Simply use Theorem 3.7 with the fact that the rank of a matrix is the number of its non-zero singular values. □

Similar to the SVD, if it exists, the eigenvalue decomposition of a Kronecker product can be derived from those the two smaller matrices. A direct and interesting consequence regards the preservation of positive semi-definiteness with respect to the Kronecker product.

Theorem 3.8. *Let A and B be two square matrices. Let (μ, x) and (λ, y) be any pairs of eigenvalue-eigenvector of A and B respectively. Then $\mu\lambda$ is an eigenvalue of $A \otimes B$ and $x \otimes y$ is its corresponding eigenvector.*

Proof. $(A \otimes B)(x \otimes y) = (Ax) \otimes (By) = (\mu x) \otimes (\lambda y) = \mu\lambda(x \otimes y)$. □

Corollary 3.5. *If two square matrices A and B are positive (semi) definite, then $A \otimes B$ is a positive (semi) definite matrix.*

The following two statements indicate how to obtain the determinant/trace of a Kronecker product from the traces/determinants of the matrices involved.

Theorem 3.9. *Let $A \in \mathbb{R}^{m_A \times m_A}$ and $B \in \mathbb{R}^{m_B \times m_B}$ be two square matrices. Then,*

$$\det(A \otimes B) = (\det A)^{m_B} (\det B)^{m_A}. \tag{3.12}$$

Proof. From Theorem 3.7, it follows that

$$\det(A \otimes B) = \det\left(\Sigma_A \otimes \Sigma_B\right),$$

with

$$\Sigma_A \otimes \Sigma_B = \begin{bmatrix} [\Sigma_A]_{1,1}\Sigma_B & & \\ & \ddots & \\ & & [\Sigma_A]_{m_A,m_A}\Sigma_B \end{bmatrix}$$

a block-diagonal matrix. Thus, we have

$$\begin{aligned} \det(A \otimes B) &= \prod_{k=1}^{m_A} \det([\Sigma_A]_{k,k}\Sigma_B) \\ &= \prod_{k=1}^{m_A} [\Sigma_A]_{k,k}^{m_B} \det(\Sigma_B) \\ &= (\det\Sigma_B)^{m_A} \left(\prod_{k=1}^{m_A} [\Sigma_A]_{k,k} \right)^{m_B} \\ &= (\det\Sigma_B)^{m_A} (\det\Sigma_A)^{m_B} \\ &= (\det B)^{m_A} (\det A)^{m_B} \end{aligned}$$

as claimed. □

Theorem 3.10. *Let $A \in \mathbb{R}^{m_A \times m_A}$ and $B \in \mathbb{R}^{m_B \times m_B}$ be two square matrices. Then,*

$$\mathrm{tr}(A \otimes B) = (\mathrm{tr}A)(\mathrm{tr}B). \tag{3.13}$$

Proof. The trace can be computed as

$$\begin{aligned}\mathrm{tr}(A\otimes B) &= \mathrm{tr}\left(\begin{bmatrix} A_{1,1}B & \dots & A_{1,m_A}B \\ \vdots & & \vdots \\ A_{m_A,1}B & \dots & A_{m_A,m_A}B \end{bmatrix}\right)\\ &= \sum_{k=1}^{m_A}\mathrm{tr}(A_{k,k}B)\\ &= \sum_{k=1}^{m_A}A_{k,k}(\mathrm{tr}B)\\ &= (\mathrm{tr}B)\sum_{k=1}^{m_A}A_{k,k}\\ &= (\mathrm{tr}B)(\mathrm{tr}A)\end{aligned}$$

which is the desired result. □

It is sometimes more convenient to deal with vectors rather than matrices. For this purpose, we wil consider the vectorization operator introduced in the upcoming definition.

Definition 3.2. For a matrix

$$A = \begin{bmatrix} A_{1,1} & A_{1,2} & \cdots & A_{1,n_A} \\ A_{2,1} & A_{2,2} & \cdots & A_{2,n_A} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m_A,1} & A_{m_A,2} & \cdots & A_{m_A,n_A} \end{bmatrix} \in \mathbb{R}^{m_A\times n_A},$$

$\mathrm{vec}(A)$ is the vector obtained by stacking the columns of A together, i.e.,

$$\mathrm{vec}(A) = [A_{1,1}, A_{2,1}, \dots, A_{m_A,1}, A_{1,2}, \dots, A_{m_A,2}, \dots, A_{1,n_A}, \dots A_{m_A,n_A}]^T \in \mathbb{R}^{m_An_A}. \tag{3.14}$$

The forthcoming theorems highlight interesting properties of "vec" operator that will serve us throughout the thesis.

Theorem 3.11. *Let $A_1, A_2, \dots, A_k$ be k matrices of same size. Then, for any scalars $\lambda_1, \dots, \lambda_k$, we have*

$$\mathrm{vec}\left(\sum_{j=1}^{k}\lambda_jA_j\right) = \sum_{j=1}^{k}\lambda_j\mathrm{vec}\left(A_j\right). \tag{3.15}$$

Proof. Directly follows from Definition 3.2. □

Theorem 3.12. *Let $x \in \mathbb{R}^{m_x}$ and $y \in \mathbb{R}^{m_y}$ be two vectors. Then,*

$$\mathrm{vec}(xy^T) = y\otimes x. \tag{3.16}$$

Proof. Directly follows from Definition 3.2. □

Theorem 3.13. *Let $A \in \mathbb{R}^{m_A\times n_A}$, $B \in \mathbb{R}^{m_B\times n_B}$ and $C \in \mathbb{R}^{m_C\times n_C}$ with $m_C = n_B$. Then,*

$$(C^T\otimes A)\mathrm{vec}(B) = \mathrm{vec}(ABC) \tag{3.17}$$

Proof. Let $B = [b_1, \dots, b_{n_B}], b_j \in \mathbb{R}^{m_B}$ denotes the j-th column of B. We have

$$B = \sum_{j=1}^{n_B} b_j e_j^T,$$

where e_j is the j-th column of the identity matrix I_{m_B}. Then, by applying Theorems 3.11, 3.12, 3.4, we have

$$\begin{aligned}
\text{vec}(ABC) &= \text{vec}\left(A\left(\sum_{j=1}^{n_B} b_j e_j^T\right)C\right) \\
&= \sum_{j=1}^{n_B} \text{vec}(Ab_j e_j^T C) \\
&= \sum_{j=1}^{n_B} (C^T e_j \otimes Ab_j) \\
&= \sum_{j=1}^{n_B} (C^T \otimes A)(e_j \otimes b_j) \\
&= (C^T \otimes A) \sum_{j=1}^{n_B} \text{vec}(b_j e_j^T) \\
&= (C^T \otimes A)\text{vec}(\sum_{j=1}^{n_B} b_j e_j^T) \\
&= (C^T \otimes A)\text{vec}(B).
\end{aligned}$$

This completes the proof. □

3.1.2 KFAC method

Let's go back to the FIM F expression defined by the equation (3.3). Using the properties of the Kronecker product, each block can be written as

$$\begin{aligned}
F_{i,j} &= \mathbb{E}[\text{vec}(\mathcal{D}W_i)\text{vec}(\mathcal{D}W_j)^T] \\
&= \mathbb{E}[\text{vec}(g_i \bar{a}_{i-1}^T)\text{vec}(g_j \bar{a}_{j-1}^T)^T] \\
&= \mathbb{E}[(\bar{a}_{i-1} \otimes g_i)(\bar{a}_{j-1} \otimes g_j)^T] \\
&= \mathbb{E}[(\bar{a}_{i-1} \otimes g_i)(\bar{a}_{j-1}^T \otimes g_j^T)] \\
&= \mathbb{E}[\bar{a}_{i-1}\bar{a}_{j-1}^T \otimes g_i g_j^T].
\end{aligned} \tag{3.18}$$

The Kronecker-factored approximate curvature (KFAC) method introduced in [112] is grounded on two assumptions that provide a computationally efficient approximation of F.

The first assumption is that $F_{i,j} = 0$ for $i \neq j$. In other words, weight derivatives in two different layers are uncorrelated. This results in block-diagonal approximation

$$F \approx \text{diag}(F_{1,1}, F_{2,2}, \dots F_{\ell,\ell}),$$

with $\forall i \in [\![1; \ell]\!]$, $F_{i,i} \in \mathbb{R}^{(d_{i-1}+1)d_i \times (d_{i-1}+1)d_i}$. This first approximation is insufficient, insofar as the blocks of $F_{i,i}$ are very large for neural networks with high number of units in layers. A further approximation is in order.

The second assumption is that of independent activations and derivatives (IAD): activations and pre-activation derivatives are independent. i.e. $\forall i$, $a_{i-1} \perp\!\!\!\perp g_i$. This allows each block $F_{i,i}$ to be factorized into a Kronecker product of two smaller matrices, i.e.,

$$\begin{aligned} F_{i,i} &= \mathbb{E}[\bar{a}_{i-1}\bar{a}_{i-1}^T \otimes g_i g_i^T] \\ &\approx \mathbb{E}[\bar{a}_{i-1}\bar{a}_{i-1}^T] \otimes \mathbb{E}[g_i g_i^T] \\ &=: \bar{A}_{i-1}^{\mathrm{KFAC}} \otimes G_i^{\mathrm{KFAC}}, \end{aligned} \tag{3.19}$$

with $\bar{A}_{i-1}^{\mathrm{KFAC}} = \mathbb{E}[\bar{a}_{i-1}\bar{a}_{i-1}^T] \in \mathbb{R}^{(d_{i-1}+1)\times(d_{i-1}+1)}$ and $G_i^{\mathrm{KFAC}} = \mathbb{E}[g_i g_i^T] \in \mathbb{R}^{d_i \times d_i}$.

These two assumptions yield the KFAC approximation

$$F \approx F_{\mathrm{KFAC}} = \mathrm{diag}([F_{\mathrm{KFAC}}]_{1,1}, \dots, [F_{\mathrm{KFAC}}]_{\ell,\ell}), \tag{3.20}$$

with $\forall i \in [\![1;\ell]\!]$, $[F_{\mathrm{KFAC}}]_{i,i} = \bar{A}_{i-1}^{\mathrm{KFAC}} \otimes G_i^{\mathrm{KFAC}}$. The decisive advantage of F_{KFAC} is that it can be inverted in a very economical way. Indeed, owing to the properties 3.6 and 3.13 of the Kronecker product, the approximate natural gradient $F_{\mathrm{KFAC}}^{-1}\nabla h$ can be evaluated as

$$F_{\mathrm{KFAC}}^{-1}\nabla h = \begin{bmatrix} \mathrm{vec}(G_1^{-1}(\nabla_{W_1} h)\bar{A}_0^{-1}) \\ \vdots \\ \mathrm{vec}(G_\ell^{-1}(\nabla_{W_\ell} h)\bar{A}_{\ell-1}^{-1}) \end{bmatrix}, \tag{3.21}$$

where the KFAC superscripts are dropped from now on to alleviate notations. This drastically reduces computations and memory requirements, since we only need to store, invert and multiply the smaller matrices $\bar{A}_{i-1}$'s and G_i's.

In practice, because the curvature changes relatively slowly [112], the factors $(\bar{A}_{i-1}, G_i)$ are computed at every T_1 iterations and their inverses at every T_2 iterations. Moreover, $(\bar{A}_{i-1}, G_i)$ are estimated using exponentially decaying moving average. At iteration k, let $(\bar{A}_{i-1}^{\mathrm{old}}, G_i^{\mathrm{old}})$ be the factors previously computed at iteration $k - T_1$ and $(\bar{A}_{i-1}^{\mathrm{new}}, G_i^{\mathrm{new}})$ be those computed with the current mini-batch. Then, setting $\rho = \min(1 - 1/k, \gamma)$ with $\gamma \in [0, 1]$, we have

$$\begin{aligned} \bar{A}_{i-1} &= \rho\bar{A}_{i-1}^{\mathrm{old}} + (1-\rho)\bar{A}_{i-1}^{\mathrm{new}}, \\ G_i &= \rho G_i^{\mathrm{old}} + (1-\rho)G_i^{\mathrm{new}}. \end{aligned}$$

Another crucial ingredient of KFAC is the Tikhonov regularization to enforce invertibility of F_{KFAC}. The straightforward damping $F_{\mathrm{KFAC}} + \lambda I$ deprives us of the possibility of applying the formula of Theorem 3.6.

To overcome this issue, Martens & Grosse [112] advocated the more judicious Kronecker product regularization

$$[F_{\bullet\,\mathrm{KFAC}}]_{i,i} = (\bar{A}_{i-1} + \pi_i \lambda^{1/2} I_{\bar{A}_{i-1}}) \otimes (G_i + \pi_i^{-1}\lambda^{1/2} I_{G_i}),$$

where $I_{\bar{A}_{i-1}}$ and I_{G_i} denote identity matrices of same size as $\bar{A}_{i-1}$ and G_i respectively, $\lambda > 0$ denotes regularization or damping parameter and

$$\pi_i = \sqrt{\frac{\mathrm{tr}(\bar{A}_{i-1})/(d_{i-1}+1)}{\mathrm{tr}(G_i)/d_i}}.$$

3.2 Four novel methods

3.2.1 Motivation

The fundamental assumption on which KFAC hinges is the independence between activations and pre-activation derivatives. We believe that this premise, which has no theoretical foundation, is at the root of a poor quality of the FIM approximation. This is why, we wish to put forward four Kronecker-factored block-diagonal approximations that aim at more accurately representing the FIM by removing this assumption. To this end, we minimize the Frobenius norm of the difference between the original matrix and a prescribed form for the approximation, which is achievable through the Kronecker product singular value decomposition.

While staying within the framework of the first assumption (block-diagonal approximation), we now design four new methods that break free from the second hypothesis (IAD) in order to achieve a better accuracy: KPSVD, Deflation, Lanczos-bidiagonalization and KFAC-corrected.

3.2.2 KPSVD

In our first method, called KPSVD, the factors $(\bar{A}_{i-1}, G_i)$ are specified as the arguments of the best possible approximation of $F_{i,i}$ by a single Kronecker product. Thus,

$$(\bar{A}_{i-1}, G_i) = \underset{(R,S)}{\operatorname{argmin}} \|F_{i,i} - R \otimes S\|_F = \underset{(R,S)}{\operatorname{argmin}} \|\mathbb{E}[\bar{a}_{i-1}\bar{a}_{i-1}^T \otimes g_i g_i^T] - R \otimes S\|_F, \tag{3.22}$$

where $\|\cdot\|_F$ denotes Frobenius norm. Although the minimization problem (3.22) has already been introduced in abstract linear algebra by van Loan [170,171], it has never been considered in the context of neural networks, at least to the best of our knowledge. Anyhow, it can be solved at a low cost by means of the Kronecker product singular value decomposition technique [170]. To write down the solution, we need the following notion.

Definition 3.3. Let

$$M = \begin{bmatrix} M_{1,1} & \dots & M_{1,d} \\ M_{2,1} & \dots & M_{2,d} \\ \vdots & & \vdots \\ M_{d,1} & \dots & M_{d,d} \end{bmatrix} \in \mathbb{R}^{d'd \times d'd}$$

be a uniform block matrix, that is, $M_{\mu,\nu} \in \mathbb{R}^{d' \times d'}$ for all $(\mu, \nu) \in \{1, \dots d\}^2$. The *zigzag rearrangement* operator $\mathcal{Z}$ converts M into the matrix

$$\mathcal{Z}(M) = \begin{bmatrix} \operatorname{vec}(M_{1,1})^T \\ \vdots \\ \operatorname{vec}(M_{d,1})^T \\ \vdots \\ \operatorname{vec}(M_{1,d})^T \\ \vdots \\ \operatorname{vec}(M_{d,d})^T \end{bmatrix} \in \mathbb{R}^{d^2 \times (d')^2}, \tag{3.23}$$

by flattening out each block in a column-wise order and by transposing the resulting vector. This operator is to be applied to each $M = F_{i,i}$ with $d = d_{i-1} + 1$ and $d' = d_i$.

The following theorem testifies that problem (3.22) is equivalent to a classical low rank approximation problem.

Theorem 3.14. *Any solution of* (3.22) *is also a solution of the ordinary rank-1 matrix approximation problem*

$$(\mathrm{vec}(\bar{A}_{i-1}^{\mathrm{KPSVD}}), \mathrm{vec}(G_i^{\mathrm{KPSVD}})) = \underset{(R,S)}{\mathrm{argmin}} \|\mathcal{Z}(F_{i,i}) - \mathrm{vec}(R)\,\mathrm{vec}(S)^T\|_F. \tag{3.24}$$

Proof. See appendix 3.A.1. □

Problem (3.24) is solved as follows. Let $U^T\mathcal{Z}(F_{i,i})V = \Sigma$ be the singular value decomposition (SVD) of $\mathcal{Z}(F_{i,i})$. Let σ_1 be the greatest singular value of $\mathcal{Z}(F_{i,i})$ and (u_1, v_1) be the associated left and right singular vectors. A solution to (3.24) is

$$\bar{A}_{i-1}^{\mathrm{KPSVD}} = \sqrt{\sigma_1}\,\mathrm{MAT}(u_1), \qquad G_i^{\mathrm{KPSVD}} = \sqrt{\sigma_1}\,\mathrm{MAT}(v_1),$$

where "MAT," the converse of "vec," turns a vector into a matrix. The question to be addressed now is how to efficiently compute u_1, v_1 and σ_1. We recommend the power SVD algorithm (see appendix 3.B.1), which only requires the matrix-vector multiplications $\mathcal{Z}(F_{i,i})v$ and $\mathcal{Z}(F_{i,i})^T u$. These operations can be performed without explicitly forming $F_{i,i}$ nor $\mathcal{Z}(F_{i,i})$, as elaborated on in the upcoming Proposition.

Proposition 3.1. *For all* $u \in \mathbb{R}^{(d_{i-1}+1)^2}$ *and* $v \in \mathbb{R}^{d_i^2}$,

$$\mathcal{Z}(F_{i,i})v = \mathbb{E}[\, g_i^T V g_i\, \mathrm{vec}(\bar{a}_{i-1}\bar{a}_{i-1}^T)\,], \qquad \mathcal{Z}(F_{i,i})^T u = \mathbb{E}[\, \bar{a}_{i-1}^T U \bar{a}_{i-1}\, \mathrm{vec}(g_i g_i^T)\,],$$

with $U = \mathrm{MAT}(u)$ *and* $V = \mathrm{MAT}(v)$.

Proof. See appendix 3.A.2. □

Estimating $\mathcal{Z}(F_{i,i})v$ and $\mathcal{Z}(F_{i,i})^T u$

Let us consider a batch $\mathcal{S} = \{(x^{(1)}, y^{(1)}), \dots (x^{(m)}, y^{(m)})\}$ drawn from the training data $\mathcal{D}$. We recall that the expectation is taken with respect to both Q_x (data distribution over inputs x) and $P_{y|x}(\theta)$ (predictive distribution of the network). To estimate $\mathcal{Z}\left(F_{i,i}\right)v$ and $\mathcal{Z}\left(F_{i,i}\right)^T u$, we use the Monte-Carlo method as suggested by Martens & Grosse [112]: we first compute the $\bar{a}_{i-1}$'s and g_i's during an additional back-propagation performed using targets y's sampled from $P_{y|x}(\theta)$ and then set

$$\mathcal{Z}(F_{i,i})v \approx \frac{1}{m}\sum_{b=1}^{m} (g_i^{(b)})^T V g_i^{(b)}\, \mathrm{vec}(\bar{a}_{i-1}^{(b)}(\bar{a}_{i-1}^{(b)})^T)$$

and

$$\mathcal{Z}(F_{i,i})^T u \approx \frac{1}{m}\sum_{b=1}^{m} \bar{a}_{i-1}^{(b)})^T U (\bar{a}_{i-1}^{(b)}\, \mathrm{vec}(g_i^{(b)}(g_i^{(b)})^T),$$

where the subscript b loops over the number m of data points in the batch $\mathcal{S}$. In practice, it is more economical (in terms of implementation cost) to compute $\mathcal{Z}(F_{i,i})v$ and $\mathcal{Z}(F_{i,i})^T u$ using matrix forms rather than previous expressions. To derive these matrix expressions, let us define the matrices

$$\hat{\mathcal{A}}_{i-1} = (\bar{a}_{i-1}^{(1)}, \dots, \bar{a}_{i-1}^{(m)}) \in \mathbb{R}^{d_{i-1} \times m} \text{ and } \hat{\mathcal{G}}_i = (g_i^{(1)}, \dots, g_i^{(m)}) \in \mathbb{R}^{d_i \times m},$$

where the b^{th} column of $\hat{\mathcal{A}}_{i-1}$ and that of $\hat{\mathcal{G}}_i$ correspond respectively to the activation and pre-activation derivative computed with the b^{th} sample in $\mathcal{S}$. Then, using basic matrix operations, it is straightforward to show that

$$\text{MAT}\big(\mathcal{Z}(F_{i,i})v\big) = \frac{1}{m}\left(\text{diag}\left(\hat{\mathcal{G}}_i^T V \hat{\mathcal{G}}_i\right)^T * \hat{\mathcal{A}}_{i-1}\right)\hat{\mathcal{A}}_{i-1}^T$$

and

$$\text{MAT}\big(\mathcal{Z}(F_{i,i})^T u\big) = \frac{1}{m}\left(\text{diag}\left(\hat{\mathcal{A}}_{i-1}^T U \hat{\mathcal{A}}_{i-1}\right)^T * \hat{\mathcal{G}}_i\right)\hat{\mathcal{G}}_i^T,$$

where $*$ defines the operator that, given a row vector $z = (z^{[1]}, \dots, z^{[m]}) \in \mathbb{R}^{1 \times m}$ and a matrix $A = (a_1, \dots, a_m) \in \mathbb{R}^{d_A \times m}$ multiplies the j^{th} column of A with the j^{th} component of z:

$$z * A = (z^{[1]}a_1, \dots, z^{[m]}a_m).$$

So far, we have not paid attention to the symmetry and positive semi-definiteness of the matrices $(\bar{A}_{i-1}^{\text{KPSVD}}, G_i^{\text{KPSVD}})$ in problem (3.22). It turns out that symmetry is automatic, while positive semi-definiteness occurs for some solutions to be selected.

Proposition 3.2. *All solutions $(\bar{A}_{i-1}^{\text{KPSVD}}, G_i^{\text{KPSVD}})$ of problem (3.22) are symmetric. Besides, we can select solutions for which these matrices are positive semi-definite.*

Proof. See appendix 3.A.3. □

3.2.3 Kronecker rank-2 approximation to $F_{i,i}$

Since the KPSVD method of §3.2.2 is merely a Kronecker rank-1 approximation of $F_{i,i}$, it is most natural to look for higher order approximations. The two methods presented in this section are based on seeking a Kronecker rank-2 approximation $R \otimes S + P \otimes Q$ of $F_{i,i}$ that achieves

$$\min_{(R,S,P,Q)} \|F_{i,i} - (R \otimes S + P \otimes Q)\|_F. \tag{3.25}$$

Again, the zigzag rearrangement operator $\mathcal{Z}$ enables us to reformulate (3.25) as an ordinary rank-2 matrix approximation problem. To determine a solution of the latter, there are two techniques in practice: *deflation* [146] and *Lanczos bi-diagonalization* [62].

3.2.3.1 Deflation

The rank-1 factors (R, S) and the rank-2 factors (P, Q) are computed successively, one after another:

1. Apply the power SVD algorithm to $\mathcal{Z}(F_{i,i})$ to compute (R, S) so as to minimize $\|F_{i,i} - R \otimes S\|_F$. The solution is known to be $(R, S) = (\bar{A}_{i-1}^{\text{KPSVD}}, G_i^{\text{KPSVD}})$.

2. Let $\widehat{F}_{i,i} = F_{i,i} - R \otimes S$. Apply the power SVD algorithm to $\mathcal{Z}(\widehat{F}_{i,i})$ to compute (P, Q) so as to minimize $\|\widehat{F}_{i,i} - P \otimes Q\|_F$.

3. Set $F_{i,i} \approx R \otimes S + P \otimes Q$.

In step 2, we need to calculate the matrix-vector products $\mathcal{Z}(\widehat{F}_{i,i})v$ and $\mathcal{Z}(\widehat{F}_{i,i})^T u$. These operations can be done efficiently without explicitly forming $\widehat{F}_{i,i}$ or $\mathcal{Z}(\widehat{F}_{i,i})$. Indeed,

$$\mathcal{Z}(\widehat{F}_{i,i})v = \mathcal{Z}(F_{i,i})v - \mathcal{Z}(R \otimes S)v, \qquad \mathcal{Z}(\widehat{F}_{i,i})^T u = \mathcal{Z}(F_{i,i})^T u - \mathcal{Z}(R \otimes S)^T u.$$

On one hand, we know how compute $\mathcal{Z}(F_{i,i})v$ and $\mathcal{Z}(F_{i,i})^T u$ from Proposition 3.1. On the other hand, it is not difficult to show that

$$\mathcal{Z}(R \otimes S)v = \langle \mathrm{vec}(S), v \rangle \, \mathrm{vec}(R), \qquad \mathcal{Z}(R \otimes S)^T u = \langle \mathrm{vec}(R), u \rangle \, \mathrm{vec}(S),$$

where $\langle \cdot, \cdot \rangle$ stands for the dot product.

3.2.3.2 Lanczos bi-diagonalization

In contrast to deflation, the Lanczos bi-diagonalization algorithm (see appendix 3.B.2) computes (R, S) and (P, Q) at the same time. It does so by simultaneously computing the two largest singular values $\sigma_1 \geqslant \sigma_2$ of $\mathcal{Z}(F_{i,i})$ with the associated singular vectors (u_1, v_1) and (u_2, v_2). Once these singular elements are determined, it remains to set

$$\begin{aligned} R &= \sqrt{\sigma_1}\,\mathrm{MAT}(u_1), \qquad & S &= \sqrt{\sigma_1}\,\mathrm{MAT}(v_1), \\ P &= \sqrt{\sigma_2}\,\mathrm{MAT}(u_2), \qquad & Q &= \sqrt{\sigma_2}\,\mathrm{MAT}(v_2). \end{aligned}$$

Similarly to KPSVD, we only have to perform the matrix-vector multiplications $\mathcal{Z}(F_{i,i})v$ and $\mathcal{Z}(F_{i,i})^T u$ without forming and storing $F_{i,i}$ or $\mathcal{Z}(F_{i,i})$.

In pratice, it is advisable to implement the restarted version of the algorithm [146], which consists of three steps:

1. **Start**: Choose an initial vector $q^{(0)}$ and a dimension K for the Krylov subspace.
2. **Iterate**: Perform Lanczos bidiagonalization algorithm (appendix 3.B.2).
3. **Restart**: Compute the desired singular vectors. If stopping criterion satisfied, stop. Else set $q^{(0)}$equal to linear combination of singular vectors and go to 2.

3.2.4 KFAC-CORRECTED

Another idea is to simply add an *ad hoc* correction to the KFAC approximation. Put another way, we consider

$$F_{i,i} \approx \bar{A}_{i-1}^{\mathrm{KFAC}} \otimes G_i^{\mathrm{KFAC}} + \bar{A}_{i-1}^{\mathrm{corr.}} \otimes G_i^{\mathrm{corr.}},$$

using the best possible correctors, that is,

$$(\bar{A}_{i-1}^{\mathrm{corr.}}, G_i^{\mathrm{corr.}}) = \underset{(P,Q)}{\mathrm{argmin}} \, \|F_{i,i} - \bar{A}_{i-1}^{\mathrm{KFAC}} \otimes G_i^{\mathrm{KFAC}} - P \otimes Q\|_F. \tag{3.26}$$

Again, the solution of (3.26) can be computed by applying the power SVD algorithm to the matrix $\mathcal{Z}(F_{i,i} - \bar{A}_{i-1}^{\mathrm{KFAC}} \otimes G_i^{\mathrm{KFAC}})$. The matrix-vector multiplications required can be done in the same way as in the *deflation* method without explicitly forming and storing the matrices.

3.2.5 Efficient inversion of $A \otimes B + C \otimes D$

For each of the last three methods, we need to solve a linear system of the form $(A \otimes B + C \otimes D)u = v$ in an efficient way. This is far from obvious, since due to the sum, the well-known and powerful identities $(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}$ and $(A \otimes B)^{-1}\text{vec}(X) = \text{vec}(B^{-1}XA^{-T})$ can no longer be applied. There are many good methods to compute u, but the most appropriate for our problem is that of Martens & Grosse [112], since it takes advantage of symmetry and definiteness of the matrices. Below is a summary of the algorithm, the full details of which are in [112].

1. Compute $A^{-1/2}$, $B^{-1/2}$ and the symmetric eigen/SVD-decompositions

$$A^{-1/2}CA^{-1/2} = E_1 S_1 E_1^T, \qquad B^{-1/2}DB^{-1/2} = E_2 S_2 E_2^T,$$

   where $S_{j:j\in[\![1;2]\!]}$ are diagonal and $E_{j:j\in[\![1;2]\!]}$ are orthogonal.

2. Set $K_1 = A^{-1/2}E_1$, $K_2 = B^{-1/2}E_2$. Then,

$$u = \text{vec}(K_2[(K_2^T V K_1) \oslash (\mathbf{1}\mathbf{1}^T + s_2 s_1^T)]K_1^T),$$

   where $E \oslash F$ denotes the Hadamard or element-wise division of E by F, $s_{j:j\in[\![1;2]\!]} = \text{diag}(S_{j:j\in[\![1;2]\!]})$, $\mathbf{1}$ vector of ones and $V = \text{MAT}(v)$. Note that $K_{j:j\in[\![1;2]\!]}$, $s_{j:j\in[\![1;2]\!]}$ can be stored and reused for different choices of v.

Despite the numerous steps involved, the inversion of $A \otimes B + C \otimes D$ is much cheaper than that of $F_{i,i}$. Indeed, if d denotes the number of neurons in the current layer, then the matrices A, B, C, D are of size $d \times d$ each, and therefore the inversion of $A \otimes B + C \otimes D$ has $O(d^2)$ memory requirement and $O(d^3)$ computational cost. Meanwhile, since $F_{i,i}$ is a matrix size $d^2 \times d^2$, its inversion requires $O(d^4)$ memory and $O(d^6)$ flops.

3.3 Experiments

We have evaluated our proposed methods as well as KFAC, SGD and ADAM on the three standard deep-auto-encoder problems used for benchmarking neural network optimization methods in deep learning community [17, 108, 112, 161]. The benchmarks consist of training three different auto-encoder architectures with CURVES, MNIST and FACES datasets respectively. See appendix 3.D for a complete description of the network architectures and datasets. In our experiments, all our proposed methods as well as KFAC use approximations of the true FIM F i.e. the FIM computed with targets sampled from the model's conditional distribution. Experiments were performed with PyTorch framework [129] on supercomputer with Nividia Ampere A100 GPU and AMD Milan@2.45GHz CPU.

The precision value ϵ for power SVD and Lanczos bi-diagonalization algorithm was set to 10^{-6}. Also for these two algorithms, we used a warm-start technique which means that the final results of the previous iteration are used as a starting point (instead of a random point) for the current iteration. This has resulted in a faster convergence. In all experiments, the batch sizes used are 256, 512 and 1024 for CURVES, MNIST and FACES datasets respectively.

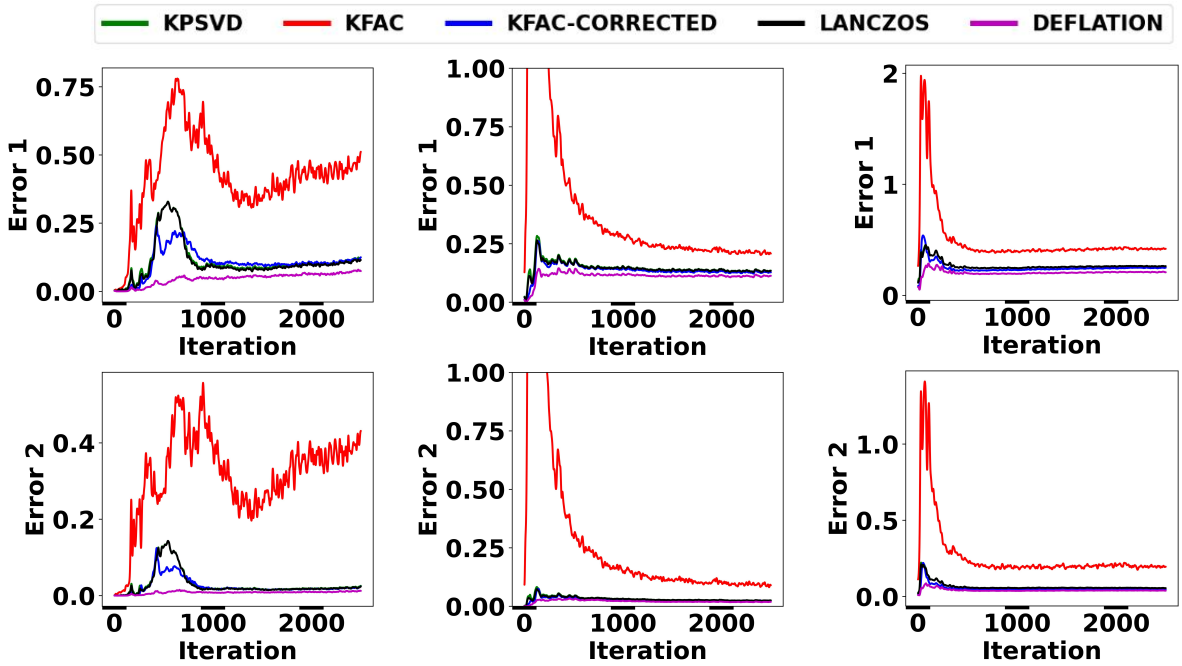
We first evaluate the approximation qualities of the FIM and then report the results on performance of the optimization objective.

3.3.1 Approximation qualities of the FIM

We investigated how well our proposed methods and KFAC approximate blocks of the exact FIM. To do so, we computed for each of the problems the exact FIM and its different approximations of the 5th layer of the network. For a fair comparison, the exact FIM as well as its different approximations were computed during the same optimization process with an independent optimizer (SGD or ADAM). We ran two independent tests with SGD and ADAM optimizers respectively and ended up with the same results. We therefore decided to report only the results obtained with ADAM. Let F be the exact FIM of the 5th layer of the network and $\hat{F}$ be any approximation to F ($\hat{F}$ is in the form $A \otimes B$ for KFAC and KPSVD, and $R \otimes S + P \otimes Q$ for KFAC corrected, Deflation and Lanczos). We measured the following two types of error:

- **Error 1**: Frobenius norm error between F and $\hat{F}$: $\|F - \hat{F}\|_F / \|F\|_F$;
- **Error 2**: ℓ_2 norm error between the spectra of F and $\hat{F}$: $\|\text{spec}(F) - \text{spec}(\hat{F})\|_2 / \|\text{spec}(F)\|_2$ where $\text{spec}(M)$ denotes the spectrum of M and $\|\cdot\|_2$ is the ℓ_2 norm.

Note that here the Fisher matrices were estimated without the exponentially decaying averaging scheme which means that only the mini-batch at iteration k is used to compute the Fisher matrices at this iteration. As we can see in Figure 3.1, for each of the problems, the Deflation



(a) CURVES (b) MNIST (c) FACES

Figure 3.1: Comparison between FIM approximation qualities of our methods and KFAC. For each problem, at each training iteration of the network with ADAM optimizer, the exact FIM and its different approximations are computed for layer 5 of the network. **Error 1** and **Error 2** described in §3.3.1 are measured. For the sake of visual comparison between different methods, the display scale in the axis of ordinates was deliberately restricted to $[0, 1]$ for the MNIST problem. It thus seems that the error curves for KFAC, whose peak amplitudes are about 6.5, are truncated.

method gives the best approximation, followed by the other methods. Although Deflation and Lanczos bi-diagonalization may appear as two implementations of the same idea (i.e., computing the two largest singular vectors), the former turns out to be more robust than the latter, in the sense that it converges much faster to the two dominant pairs and also produces a much smaller error. This accounts for the difference in performance between the two methods. The **Error 1** and **Error 2** made by our different methods remain lower than those caused by KFAC throughout the optimization process. This suggests that our methods give a better approximation to the Fisher than KFAC, and that increasing the rank does improve the quality of approximation. One can go further in this direction if there is no prohibitive extra cost.

3.3.2 Optimization performance

We now consider the network optimization in each of the three problems. We have evaluated our methods against KFAC and the baselines (SGD and ADAM). Here the different approximations to the FIM were computed using the exponentially decaying technique as described in §3.1.2. The decay factor γ was set to 0.95 as in [112]. Since the goal of KFAC as well as our methods is optimization performance rather than generalization, we performed Grid Search for each method and selected hyperparameters that gave a better reduction to the training loss. The learning rate α and the damping parameter λ are in range $\{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 3\cdot10^{-1}, 3\cdot10^{-2}, 3\cdot10^{-3}, 3\cdot10^{-4}\}$, and the clipping parameter c belongs to $\{10^{-2}, 10^{-3}\}$ (see appendix 3.E for definition of c). Note that damping and clipping are used only in KFAC and our proposed methods. Update frequencies T_1 and T_2 were set to 100. The momentum parameters were $\beta = 0.9$ for SGD and $(\beta_1, \beta_2) = (0.9, 0.999)$ for ADAM.

Figure 3.2 shows the performance of the different optimizers on the three studied problems. The first observation is that in each problem, KFAC as well as our methods optimize the training loss function faster than SGD and ADAM both with respect to epoch and time. Although our methods may seem much more computationally expensive than KFAC since at each iteration we perform the power SVD or Lanczos bi-diagonalization to estimate the Fisher matrix, they actually have the same order of magnitude in computational cost as KFAC. See appendix 3.C for a comparison of the computational costs. For each of the three problems, we observe that KFAC and KPSVD perform about the same while the DEFLATION, LANCZOS and KFAC-CORRECTED methods have the ability to optimize the objective function much faster both with respect to epoch and time.

Although this is not our object of study, we observe that for each of the three problems, our proposed methods also maintain a good generalization (see Figure 3.3).

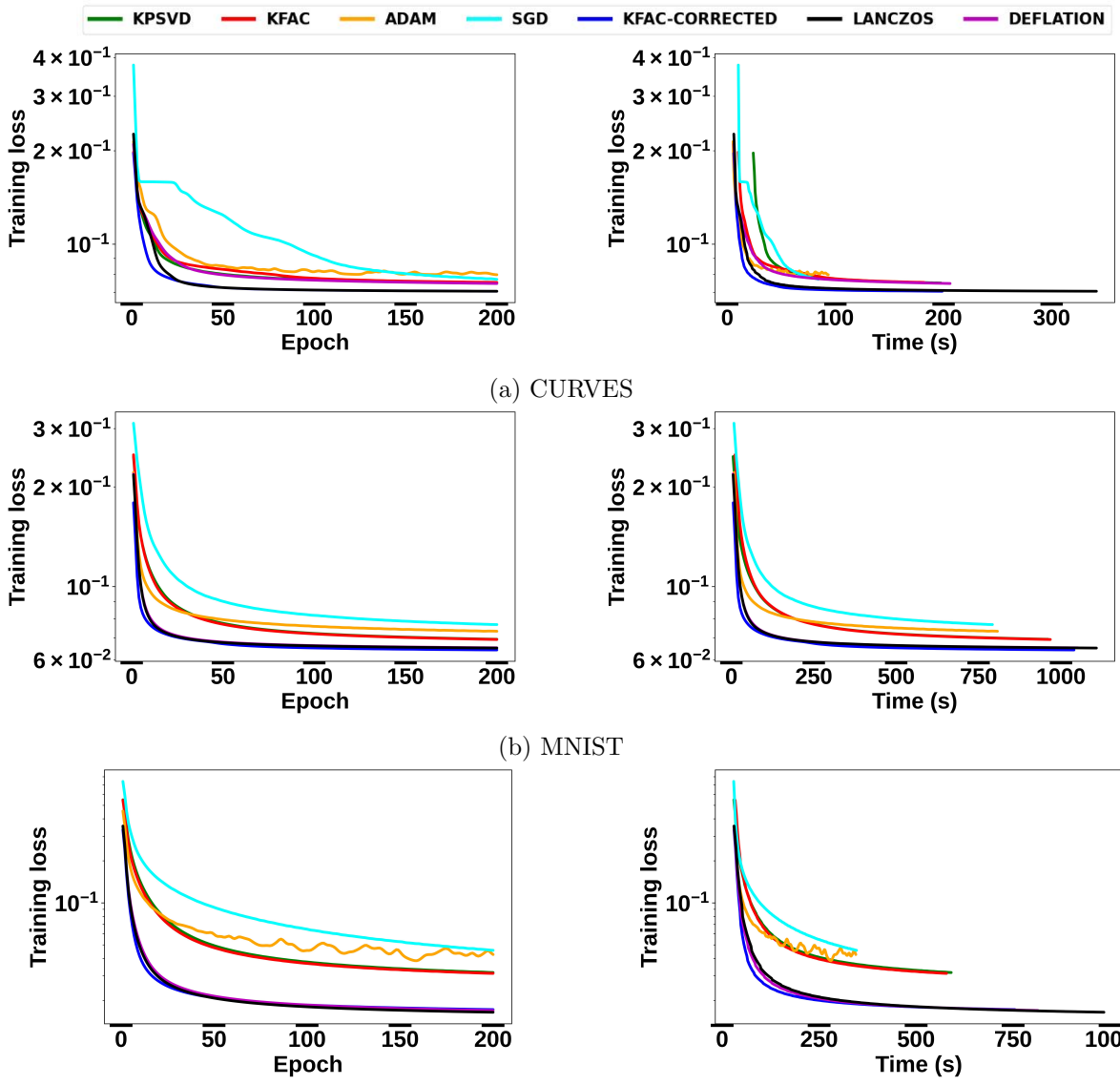
3.4 Case study of convolutional neural networks

3.4.1 KFAC for convolution layers

Grosse and Martens [68] extended KFAC to CNNs, where it was renamed KFC. However, due to weight sharing in convolutional layers, it was necessary to add two extra assumptions regarding spatial homogeneity and spatially uncorrelated derivatives.

Using the back propagation algorithm combined with equation (2.5), the gradient of the loss function with respect to the parameters of a convolutional layer i is computed as

$$g_{i,t} = \mathcal{D}\widetilde{a}_{i,t} \odot \sigma_i'(s_{i,t}), \quad \mathcal{D}W_i = \sum_{t\in\mathcal{T}_i} g_{i,t}\bar{a}_{i-1,t}^T, \quad \mathcal{D}a_{i-1,t} = W_i^T g_{i,t}, \tag{3.27}$$

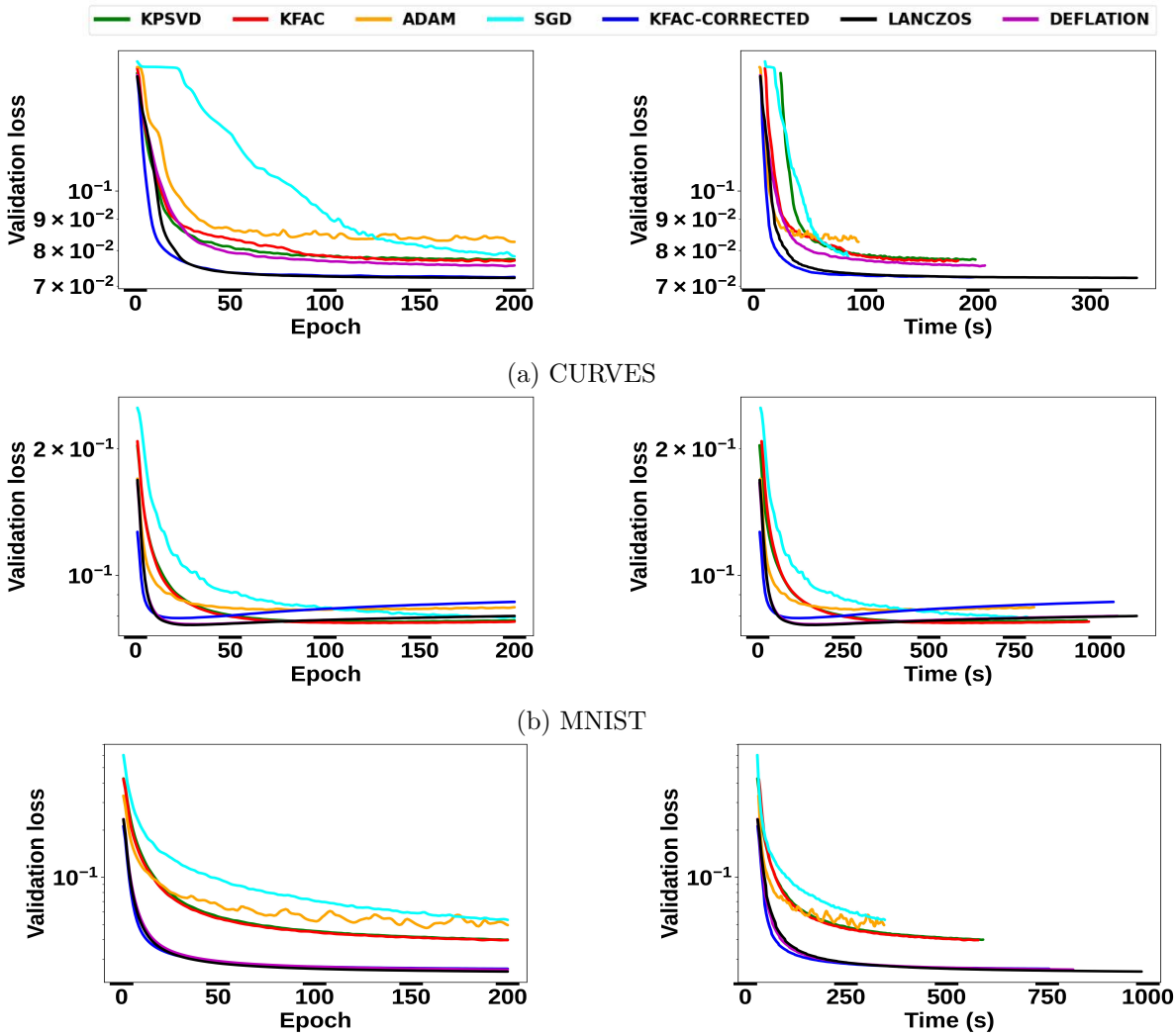


(a) CURVES

(b) MNIST

(c) FACES

Figure 3.2: Comparison of optimization performance of different algorithms on each of the 3 problems (CURVES **top** row, MNIST **middle** row and FACES **last** row). For each problem, **first** figure displays training loss *vs* epoch and **second** one represents training loss *vs* time.



(a) CURVES

(b) MNIST

(c) FACES

Figure 3.3: Validation losses of different algorithms on each of the 3 problems (CURVES **first** row, MNIST **second** row and FACES **third** row). For each problem, **first** figure depicts validation loss *vs* epoch while the **second** one displays validation loss *vs* time.

for $t \in \mathcal{T}_i$, where the special symbol $g_{i,t} := \mathcal{D}s_{i,t}$ stands for the pre-activation derivative. The diagonal block of the FIM associated to layer i can be thus formulated as

$$\begin{aligned} F_{i,i} &= \mathbb{E}\Big[\mathrm{vec}(\mathcal{D}W_i)\mathrm{vec}(\mathcal{D}W_i)^T\Big] \\ &= \mathbb{E}\Big[\mathrm{vec}\Big(\sum_{t\in\mathcal{T}_i} g_{i,t}\bar{a}_{i-1,t}^T\Big)\mathrm{vec}\Big(\sum_{t\in\mathcal{T}_i} g_{i,t}\bar{a}_{i-1,t}^T\Big)^T\Big] \\ &= \mathbb{E}\Big[\sum_{t\in\mathcal{T}_i}\sum_{t'\in\mathcal{T}_i} (\bar{a}_{i-1,t}\otimes g_{i,t})(\bar{a}_{i-1,t'}\otimes g_{i,t'})^T\Big] \\ &= \mathbb{E}\Big[\sum_{t\in\mathcal{T}_i}\sum_{t'\in\mathcal{T}_i} \bar{a}_{i-1,t}\bar{a}_{i-1,t'}^T\otimes g_{i,t}g_{i,t'}^T\Big] \\ &= \mathbb{E}\Big[\sum_{t\in\mathcal{T}_i}\sum_{t'\in\mathcal{T}_i} \Omega_i(t,t')\otimes\Gamma_i(t,t')\Big], \end{aligned} \tag{3.28}$$

with $\Omega_i(t,t') = \bar{a}_{i-1,t}\bar{a}_{i-1,t'}^T$ and $\Gamma_i(t,t') = g_{i,t}g_{i,t'}^T$. Note that $F_{i,i} \in \mathbb{R}^{c_i(c_{i-1}|\Delta_i|+1)\times c_i(c_{i-1}|\Delta_i|+1)}$ (we recall that c_{i-1} and c_i denote input and output channel respectively and Δ_i corresponds to spatial locations of filters). In order to factorize $F_{i,i}$ into a Kronecker product of two smaller matrices, Grosse & Martens [68] resort to three hypotheses. First, similarly to MLP layers, activations and pre-activation derivatives are assumed to be independent (**IAD**). Secondly, postulating spatial homogeneity (**SH**), the second-order statistics of the activations and pre-activation derivatives at any two spatial locations t and t' depend only on the difference $t-t'$. Finally, the pre-activation derivatives at any two distinct spatial locations are declared to be uncorrelated (**SUD**), i.e., $\Gamma_i(t,t') = 0\,\text{for}\,t \neq t'$. Combining these three assumptions yields the approximation

$$F_{i,i} \approx [F_{\mathrm{KFAC}}]_{i,i} = \mathbb{E}\Big[\sum_{t\in\mathcal{T}_i}\Omega_i(t,t)\Big]\otimes\frac{1}{|\mathcal{T}_i|}\mathbb{E}\Big[\sum_{t\in\mathcal{T}_i}\Gamma_i(t,t)\Big] =: \bar{A}_{i-1}^{\mathrm{KFAC}}\otimes G_i^{\mathrm{KFAC}}, \tag{3.29}$$

with

$$\bar{A}_{i-1}^{\mathrm{KFAC}} = \mathbb{E}\Big[\sum_{t\in\in\mathcal{T}_i}\Omega_i(t,t)\Big] \in \mathbb{R}^{(c_{i-1}|\Delta_i|+1)\times(c_{i-1}|\Delta_i|+1)}, \tag{3.30a}$$

$$G_i^{\mathrm{KFAC}} = \frac{1}{|\mathcal{T}_i|}\mathbb{E}\Big[\sum_{t\in\mathcal{T}_i}\Gamma_i(t,t)\Big] \in \mathbb{R}^{c_i\times c_i}. \tag{3.30b}$$

Note that in above equations, $|\mathcal{T}_i|$ denotes the cardinality of $\mathcal{T}_i$ i.e. the number of spatial locations in layer i.

Remark 3.1. It should be mentioned that, in the same spirit, a KFAC-type approximation has been developed for the RNN (Recurrent Neural Network), but with much more assumptions. In this work, we do not consider recurrent layers. The readers interested in KFAC for RNN are referred to [111].

3.4.2 Extension of the new methods to convolution layers

While one can believe that **SH** hypothesis is a fairly reasonable approximation because of the spatial invariance property of convolutional layers, one can not say the same for **SUD** and **IAD** assumptions, which do not have any theoretical or heuristic justification.

Similarly to the case of MLP layers, with the aim of getting a better approximation of $F_{i,i}$, we remove the three assumptions and minimize the Frobenius norm of the difference between

$F_{i,i}$ and a Kronecker-factored form. To this end, we use the same methods derived in §3.2, which only require to perform matrix-vector multiplications $\mathcal{Z}(F_{i,i})v$ and $\mathcal{Z}(F_{i,i})^T u$ with the new expression (3.28) of $F_{i,i}$. Let $v \in \mathbb{R}^{c_i^2}$, then $\mathcal{Z}(F_{i,i})v$ is given by

$$\mathcal{Z}(F_{i,i})v = \mathbb{E}\Big[\sum_{t\in\mathcal{T}_i}\sum_{t'\in\mathcal{T}_i}\mathcal{Z}\big(\bar{a}_{i-1,t}\bar{a}_{i-1,t'}^T \otimes g_{i,t}g_{i,t'}^T\big)v\Big] \tag{3.31}$$

$$= \sum_{t\in\mathcal{T}_i}\sum_{t'\in\mathcal{T}_i}\mathbb{E}\Big[\big(g_{i,t}^T V g_{i,t'}\big)\bar{a}_{i-1,t}\bar{a}_{i-1,t'}^T\Big], \tag{3.32}$$

with $V = \mathrm{MAT}(v)$. Similarly, for $u \in \mathbb{R}^{(c_{i-1}|\Delta_i|+1)^2}$,

$$\mathcal{Z}(F_{i,i})^T u = \mathbb{E}\Big[\sum_{t\in\mathcal{T}_i}\sum_{t'\in\mathcal{T}_i}\mathcal{Z}\big(\bar{a}_{i-1,t}\bar{a}_{i-1,t'}^T \otimes g_{i,t}g_{i,t'}^T\big)^T u\Big] \tag{3.33}$$

$$= \sum_{t\in\mathcal{T}_i}\sum_{t'\in\mathcal{T}_i}\mathbb{E}\Big[\big(\bar{a}_{i-1,t}^T U \bar{a}_{i-1,t'}\big)g_{i,t}g_{i,t'}^T\Big], \tag{3.34}$$

with $U = \mathrm{MAT}(u)$. From these expressions, in order to compute $\mathcal{Z}(F_{i,i})v$ and $\mathcal{Z}(F_{i,i})^T u$, we must compute a double sum containing $|\mathcal{T}_i|^2$ terms where each term is an expectation of matrix multiplication. This is computationally prohibitive, even for moderate size layers. We therefore propose to use approximations of $\mathcal{Z}(F_{i,i})v$ and $\mathcal{Z}(F_{i,i})^T u$. For this purpose, for $r > 0$, we define

$$\mathcal{T}_i^2(r) = \big\{(t,t') \in \mathcal{T}_i \times \mathcal{T}_i \ : \|t - t'\|_\infty \leqslant r\big\}, \tag{3.35}$$

where the infinity norm of the difference between t and t' is given by

$$\|t - t'\|_\infty = \max(|t_x - t'_x|, |t_y - t'_y|).$$

Using this notion, we derive approximations to $\mathcal{Z}(F_{i,i})v$ and $\mathcal{Z}(F_{i,i})^T u$ as follows:

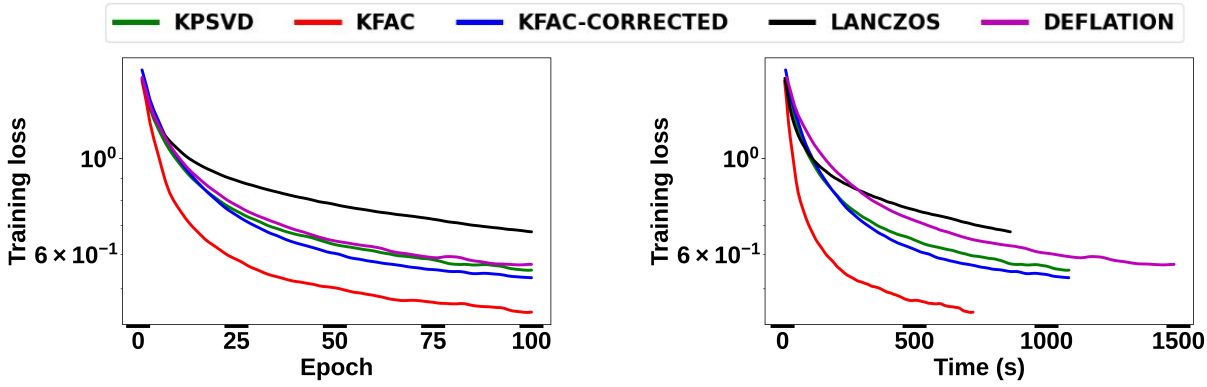
$$\mathcal{Z}(F_{i,i})v \approx \sum_{(t,t')\in\mathcal{T}_i^2(r)}\mathbb{E}\Big[\big(g_{i,t}^T V g_{i,t'}\big)\bar{a}_{i-1,t}\bar{a}_{i-1,t'}^T\Big], \tag{3.36a}$$

$$\mathcal{Z}(F_{i,i})^T u \approx \sum_{(t,t')\in\mathcal{T}_i^2(r)}\mathbb{E}\Big[\big(\bar{a}_{i-1,t}^T U \bar{a}_{i-1,t'}\big)g_{i,t}g_{i,t'}^T\Big]. \tag{3.36b}$$

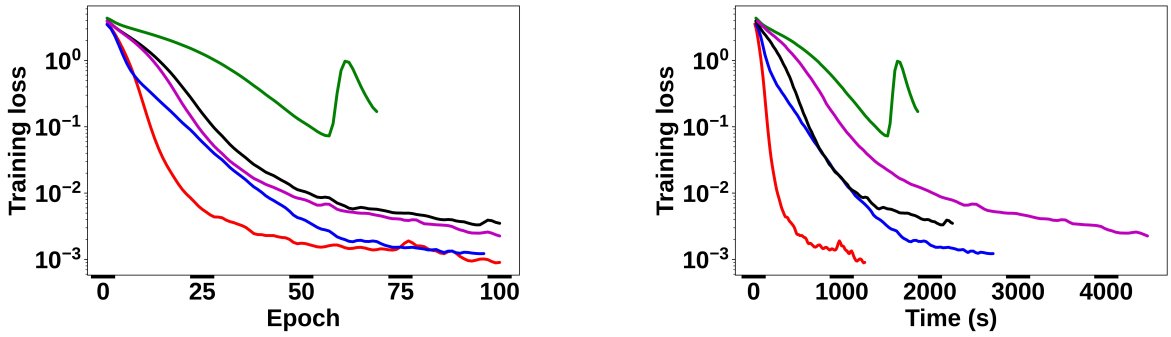
The idea behind these approximations is that there is no a significant interaction between two pixels that are far from each other, and therefore we can neglect the term corresponding to that interaction. This seems reasonable since intrinsically, convolution operations act locally (in a neighborhood defined by the size of the filter).

3.4.3 Numerical results

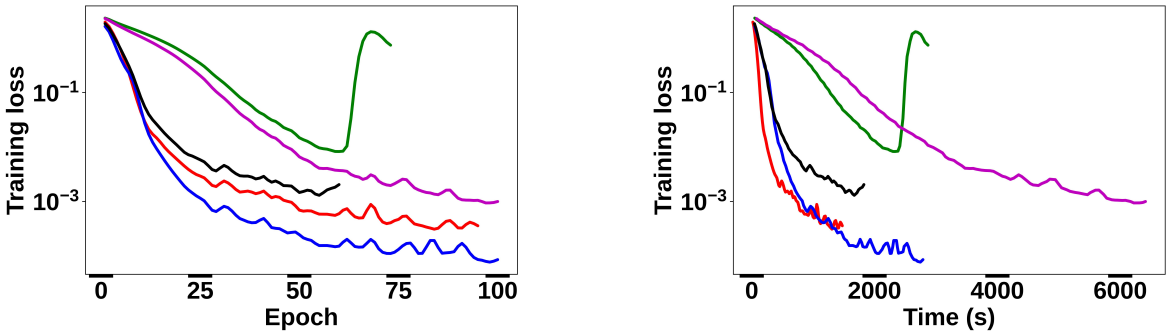
To test how well our proposed methods perform on convolutional architectures against KFAC, we consider the optimization of three different CNNs namely ResNet 18 [74], Cuda-convnet and ResNet 34 [74]. Note that Cuda-convnet is the architecture used to evaluate the original KFAC method for convolutional layers in [68]. It must be mentioned that it contains 3 convolution layers and one MLP layer.
We train Cuda-convnet on CIFAR10 dataset [92] with a batch size equal to 256, and ResNet



(a) CUDA-CONVNET



(b) RESNET 18



(c) RESNET 34

Figure 3.4: Optimization performance of our different algorithms against KFAC on three convolutional neural networks (CUDA-CONVNET **top** row, RESNET 18 **middle** row and RESNET 34 **last** row). In each row, **first** figure displays training loss *vs* epoch and **second** one represents training loss *vs* time.

18 on CIFAR100 [92] with a batch size equal to 128. Finally, we train ResNet 34 on the SVHN dataset [119] with a batch size equal to 512.

In each layer i, in order to define $\mathcal{T}_i^2(r)$ (see equation (3.35)), we set r equal to the size of the filter used in that layer. This choice is driven by the assumption that pixels that do not belong to the same region defined by the filter do not interact significantly.

Figure 3.4 shows the results obtained for the three considered architectures. Except the case of ResNet 34 where one of our methods (KFAC-CORRECTED) outperforms KFAC, in the other cases, KFAC converges faster than our methods both with respect to number of iterations (epoch) and time. This is unfortunate because the assumptions used in our methods seem less severe than those used in KFAC, and we therefore expected to obtain better optimization performance than KFAC. For a thorough study, we have extended the interaction area defined by $\mathcal{T}_i^2(r)$ by choosing a radius r larger than the size of the filter. We even considered the case where we take all the terms of the double sums (thus no approximation). The results we obtained remain unsatisfactory insofar as in most cases, despite the additional costs, our methods do not outperform KFAC.This suggests that the **IAD**, **SH** and **SUD** approximations used in KFAC for convolutional neural networks, despite their lack of theoretical foundations, do not hurt the optimization performance.

3.5 Conclusion

In this chapter, we proposed a series of novel Kronecker factorizations to the diagonal blocks of the Fisher matrix of muli-layer perceptrons using the Kronecker product Singular Value Decomposition technique. Tests realized on the three standard deep auto-encoder problems showed that 3 out of 4 of our proposed methods (DEFLATION, LANCZOS, KFAC-CORRECTED) outperform KFAC both in terms of Fisher approximation quality and in terms of optimization speed of the objective function. This ranking, which goes from the most efficient one to the least efficient one, testifies to the fact that higher-rank approximations yield better results than lower-rank ones.

We also extended our proposed methods to convolutional layers. However, we did not observe any gain in performance compared to KFAC. Our methods are more theoretically sound than KFAC and should therefore outperform it. The fact that we don't do better, or often even worse than KFAC on convolutional layers, is unexpected and deserves clarification. We leave it for future work.

3.A Proofs of section 3.2

3.A.1 Proof of Theorem 3.14

Proof. We are going to derive the identity

$$\|F_{i,i} - R \otimes S\|_F = \|\mathcal{Z}(F_{i,i}) - \mathrm{vec}(R)\mathrm{vec}(S)^T\|_F \tag{3.37}$$

for all $R \in \mathbb{R}^{(d_{i-1}+1)\times(d_{i-1}+1)}$ and $S \in \mathbb{R}^{d_i \times d_i}$, from which Theorem 3.14 will follow. For notational convenience, let

$$M = F_{i,i}, \qquad d = d_{i-1} + 1, \qquad d' = d_i.$$

We recall that M has the block structure

$$M = \begin{bmatrix} M_{1,1} & \dots & M_{1,d} \\ M_{2,1} & \dots & M_{2,d} \\ \vdots & & \vdots \\ M_{d,1} & \dots & M_{d,d} \end{bmatrix} \in \mathbb{R}^{d'd \times d'd},$$

where each block $M_{\mu,\nu}$, $(\mu,\nu) \in \{1,\dots,d\}^2$, is of size $d' \times d'$. By definition of the Frobenius norm,

$$\begin{aligned} \|M - R \otimes S\|_F^2 &= \sum_{\mu=1}^{d}\sum_{\nu=1}^{d} \|M_{\mu,\nu} - R_{\mu,\nu}S\|_F^2 \\ &= \sum_{\mu=1}^{d}\sum_{\nu=1}^{d} \|\mathrm{vec}(M_{\mu,\nu}) - R_{\mu,\nu}\mathrm{vec}(S)\|_2^2 = \sum_{\mu=1}^{d}\sum_{\nu=1}^{d} \|\mathrm{vec}(M_{\mu,\nu})^T - R_{\mu,\nu}\mathrm{vec}(S)^T\|_2^2, \end{aligned} \tag{3.38}$$

where $R_{\mu,\nu}$ is the (μ,ν)-scalar entry of R and $\|\cdot\|_2$ denotes the Euclidean norm. By virtue of

$$\mathcal{Z}(M) = \begin{bmatrix} \mathrm{vec}(M_{1,1})^T \\ \vdots \\ \mathrm{vec}(M_{d,1})^T \\ \vdots \\ \mathrm{vec}(M_{1,d})^T \\ \vdots \\ \mathrm{vec}(M_{d,d})^T \end{bmatrix}, \qquad \mathrm{vec}(R)\mathrm{vec}(S)^T = \begin{bmatrix} R_{1,1}\mathrm{vec}(S)^T \\ \vdots \\ R_{d,1}\mathrm{vec}(S)^T \\ \vdots \\ R_{1,d}\mathrm{vec}(S)^T \\ \vdots \\ R_{d,d}\mathrm{vec}(S)^T \end{bmatrix},$$

the last equality of (3.38) also reads $\|M - R\otimes S\|_F^2 = \|\mathcal{Z}(M) - \mathrm{vec}(R)\mathrm{vec}(S)^T\|_F^2$, which proves (3.37).

□

3.A.2 Proof of Proposition 3.1

Proof. Using the shorthand notations

$$\mathtt{A} = \bar{a}_{i-1}\bar{a}_{i-1}^T, \qquad \mathtt{G} = g_i g_i^T, \qquad d = d_{i-1} + 1, \qquad d' = d_i$$

we have

$$F_{i,i} = \mathbb{E}[\mathtt{A} \otimes \mathtt{G}] = \mathbb{E}\left(\begin{bmatrix} \mathtt{A}_{1,1}\mathtt{G} & \dots & \mathtt{A}_{1,d}\mathtt{G} \\ \vdots & & \vdots \\ \mathtt{A}_{d,1}\mathtt{G} & \dots & \mathtt{A}_{d,d}\mathtt{G} \end{bmatrix}\right) \in \mathbb{R}^{d'd \times d'd}.$$

Hence,

$$\mathcal{Z}(F_{i,i}) = \mathbb{E}\left(\begin{bmatrix} \mathrm{vec}(\mathtt{A}_{1,1}\mathtt{G})^T \\ \vdots \\ \mathrm{vec}(\mathtt{A}_{d,1}\mathtt{G})^T \\ \vdots \\ \mathrm{vec}(\mathtt{A}_{1,d}\mathtt{G})^T \\ \vdots \\ \mathrm{vec}(\mathtt{A}_{d,d}\mathtt{G})^T \end{bmatrix}\right) \in \mathbb{R}^{d^2 \times (d')^2}$$

For all $v \in \mathbb{R}^{(d')^2}$,

$$\mathcal{Z}(F_{i,i})v = \mathbb{E}\left(\begin{bmatrix} \mathrm{vec}(\mathtt{A}_{1,1}\mathtt{G})^T \\ \vdots \\ \mathrm{vec}(\mathtt{A}_{d,1}\mathtt{G})^T \\ \vdots \\ \mathrm{vec}(\mathtt{A}_{1,d}\mathtt{G})^T \\ \vdots \\ \mathrm{vec}(\mathtt{A}_{d,d}\mathtt{G})^T \end{bmatrix}\right) v = \mathbb{E}\left(\begin{bmatrix} \mathtt{A}_{1,1}\mathrm{vec}(\mathtt{G})^T v \\ \vdots \\ \mathtt{A}_{d,1}\mathrm{vec}(\mathtt{G})^T v \\ \vdots \\ \mathtt{A}_{1,d}\mathrm{vec}(\mathtt{G})^T v \\ \vdots \\ \mathtt{A}_{d,d}\mathrm{vec}(\mathtt{G})^T v \end{bmatrix}\right) = \mathbb{E}[\,(\mathrm{vec}(\mathtt{G})^T v)\,\mathrm{vec}(\mathtt{A})].$$

The scalar quantity $\mathrm{vec}(\mathtt{G})^T v$ can be further detailed as

$$\mathrm{vec}(\mathtt{G})^T v = (\mathrm{vec}(g_i g_i^T))^T v = (g_i \otimes g_i)^T v = (g_i^T \otimes g_i^T)\,\mathrm{vec}(\mathrm{MAT}(v)),$$

owing to the identities $\mathrm{vec}(xy^T) = y \otimes x$ and $(A \otimes B)^T = A^T \otimes B^T$. Invoking now $(A \otimes B)\,\mathrm{vec}(X) = \mathrm{vec}(BXA^T)$, we end up with

$$\mathrm{vec}(\mathtt{G})^T v = \mathrm{vec}(g_i^T\,\mathrm{MAT}(v)\,g_i) = \mathrm{vec}(g_i^T V g_i).$$

Therefore, $\mathcal{Z}(F_{i,i})v = \mathbb{E}[(g_i^T V g_i)\,\mathrm{vec}(\mathtt{A})]$. The proof of $\mathcal{Z}(F_{i,i})^T u = \mathbb{E}[(\bar{a}_{i-1}^T U \bar{a}_{i-1})\,\mathrm{vec}(\mathcal{G}_i)]$ for all $u \in \mathbb{R}^{d^2}$ goes along the same lines. □

3.A.3 Proof of Proposition 3.2

Proof.
▷ *Symmetry.* By construction and up to a choice of sign,

$$\mathrm{vec}(\bar{A}_{i-1}^{\mathrm{KPSVD}}) = \sqrt{\sigma_1}u_1, \qquad \mathrm{vec}(G_i^{\mathrm{KPSVD}}) = \sqrt{\sigma_1}v_1,$$

where σ_1 is the largest singular value of $\mathcal{Z}(F_{i,i})$ associated with left and right singular vectors (u_1, v_1). From the standard SVD properties

$$\mathcal{Z}(F_{i,i})v_1 = \sigma_1 u_1, \qquad \mathcal{Z}(F_{i,i})^T u_1 = \sigma_1 v_1,$$

we infer that

$$\sqrt{\sigma_1}\mathrm{vec}(\bar{A}_{i-1}^{\mathrm{KPSVD}}) = \mathcal{Z}(F_{i,i})v_1 = \mathbb{E}[\,(g_i^T\mathrm{MAT}(v_1)g_i)\,\mathrm{vec}(\bar{a}_{i-1}\bar{a}_{i-1}^T)\,],$$

the last equality being a consequence of Proposition 3.1. The scalar quantity $g_i^T\mathrm{MAT}(v_1)g_i$ can be moved into the argument of the "vec" operator, after which we can permute $\mathbb{E}$ and "vec" to obtain

$$\sqrt{\sigma_1}\mathrm{vec}(\bar{A}_{i-1}^{\mathrm{KPSVD}}) = \mathbb{E}[\,\mathrm{vec}((g_i^T\mathrm{MAT}(v_1)g_i)\,\bar{a}_{i-1}\bar{a}_{i-1}^T)\,] = \mathrm{vec}(\mathbb{E}[\,(g_i^T\mathrm{MAT}(v_1)g_i)\,\bar{a}_{i-1}\bar{a}_{i-1}^T\,]).$$

Hence, upon taking the "MAT" operator,

$$\sqrt{\sigma_1}\bar{A}_{i-1}^{\mathrm{KPSVD}} = \mathbb{E}[\,(g_i^T\mathrm{MAT}(v_1)g_i)\,\bar{a}_{i-1}\bar{a}_{i-1}^T\,].$$

Since each $(g_i^T\mathrm{MAT}(v_1)g_i)\,\bar{a}_{i-1}\bar{a}_{i-1}^T$ is a symmetric matrix, their expectation is also symmetric. The symmetry of G_i^{KPSVD} is proven in a similar fashion.

▷ *Positive and semi-definiteness.* The proof of this part is inspired from [171, Theorem 5.8]. Since $\bar{A}_{i-1}^{\mathrm{KPSVD}}$ and G_i^{KPSVD} are symmetric, they can be diagonalized as

$$\begin{aligned}\bar{A}_{i-1}^{\mathrm{KPSVD}} &= U^TDU, \qquad D = \mathrm{diag}(\alpha_1, \alpha_2, \ldots, \alpha_{d_{i-1}+1}),\\ G_i^{\mathrm{KPSVD}} &= V^TEV, \qquad E = \mathrm{diag}(\beta_1, \beta_2, \ldots, \beta_{d_i}),\end{aligned}$$

with orthogonal matrices U and V. We are going to show that it is possible to modify the matrices, while preserving minimality of the Frobenius norm, so that the α's and the β's all have the same sign. To this end, we first observe that

$$\bar{A}_{i-1}^{\mathrm{KPSVD}} \otimes G_i^{\mathrm{KPSVD}} = (U^TDU) \otimes (V^TEV) = (U \otimes V)^T(D \otimes E)(U \otimes V),$$

which leads us to introduce

$$C = (U \otimes V)F_{i,i}\,(U \otimes V)^T.$$

By unitary invariance of the Frobenius norm, we have

$$\|F_{i,i} - \bar{A}_{i-1}^{\mathrm{KPSVD}} \otimes G_i^{\mathrm{KPSVD}}\|_F^2 = \|(U \otimes V)^T(C - D \otimes E)(U \otimes V)\|_F^2 = \|C - D \otimes E\|_F^2.$$

The last quantity can be expressed as

$$\|C - D \otimes E\|_F^2 = \sum_{\omega=1}^{d_i(d_{i-1}+1)} (C_{\omega,\omega} - (D \otimes E)_\omega)^2 + \sum_{\xi \neq \eta} C_{\xi,\eta}^2 = \sum_{\omega=1}^{d_i(d_{i-1}+1)} (C_{\omega,\omega} - \alpha_{\mu(\omega)}\beta_{\tau(\omega)})^2 + \sum_{\xi \neq \eta} C_{\xi,\eta}^2,$$

where $\mu(\omega) \in \{1, \ldots, d_{i-1}+1\}$ and $\tau(\omega) \in \{1, \ldots, d_i\}$ can be uniquely determined[1] from $\omega \in \{1, \ldots, d_i(d_{i-1}+1)\}$ in such a way that $\omega = (\mu(\omega)-1)d_i + \tau(\omega)$. Because $F_{i,i}$ is positive semi-definite, C is also positive semi-definite, which implies that $C_{\omega,\omega} \geqslant 0$. Thus, for all ω,

$$(C_{\omega,\omega} - \alpha_{\mu(\omega)}\beta_{\tau(\omega)})^2 - (C_{\omega,\omega} - |\alpha_{\mu(\omega)}||\beta_{\tau(\omega)}|)^2 = 2C_{\omega,\omega}(|\alpha_{\mu(\omega)}\beta_{\tau(\omega)}| - \alpha_{\mu(\omega)}\beta_{\tau(\omega)}) \geqslant 0.$$

This means that if we set, for instance,

$$R = U^T|D|U, \qquad S = V^T|E|V,$$

[1] The solution is given by $\mu(\omega) = 1 + \lfloor(\omega-1)/d_i\rfloor$ and $\tau(\omega) = d_i\lfloor(\omega-1)/d_i\rfloor$, where $\lfloor\cdot\rfloor$ is the integer part, but this does not matter here.

with $|D| = \text{diag}(|\alpha_1|, \alpha_2|, \ldots, |\alpha_{d_{i-1}+1}|)$ and $|E| = \text{diag}(|\beta_1|, |\beta_2|, \ldots, |\beta_{d_i}|)$, then

$$\|F_{i,i} - R \otimes S\|_F^2 \leqslant \|F_{i,i} - \bar{A}_{i-1}^{\text{KPSVD}} \otimes G_i^{\text{KPSVD}}\|_F^2.$$

If the inequality were strict, minimality of $(\bar{A}_{i-1}^{\text{KPSVD}}, G_i^{\text{KPSVD}})$ would be contradicted. Therefore, we must have equality. This entails that (R, S) is another minimizer for which the eigenvalues of R, as well as those of S, are all non-negative. In such a case, we select this pair (R, S) for the factors $(\bar{A}_{i-1}^{\text{KPSVD}}, G_i^{\text{KPSVD}})$. □

3.B Algorithms

3.B.1 Power SVD algorithm

The *Power SVD* algorithm is an extension of the well-known *Power Method* [146] and is used to approximate the dominant singular value $\sigma_1 = \sigma_{\max}$ of a real rectangular matrix and associated right and left singular vectors. We provide here an overview of the algorithm. Readers interested in a thorough description such as convergence analysis of the method are referred to [146].

Algorithm 1: SVD Power algorithm

Input: $A \in \mathbb{R}^{m_A \times n_A}$, $v^{(0)} \in \mathbb{R}^{m_A}$, ϵ (precision), $k_{\max}$ (maximum iteration).
Output: σ_1, u_1 and v_1 ($Av_1 = \sigma_1 u_1$, $A^T u_1 = \sigma_1 v_1$

for $k = 1, 2, \ldots, k_{\max}$ **do**
- $w^{(k)} = Av^{(k-1)}$; $u^{(k)} = w^{(k)}/\|w^{(k)}\|_2$;
- $z^{(k)} = A^T u^{(k)}$; $v^{(k)} = z^{(k)}/\|z^{(k)}\|_2$;
- $\sigma^{(k)} = \|z^{(k)}\|_2$;
- `error` $= \|Av^{(k)} - \sigma^{(k)} u^{(k)}\|_2$;
- **if** `error` $\leqslant \epsilon$ **then**
  - Break;
- **end**

end

3.B.2 Lanczos bi-diagonalization algorithm

The *Lanczos bi-diagonalization* algorithm [62] is an iterative method used to decompose an input matrix $A \in \mathbb{R}^{m_A \times n_A}$ into the form

$$A = UBV^T, \tag{3.39}$$

where $U \in \mathbb{R}^{m_A \times m_A}$ and $V \in \mathbb{R}^{n_A \times n_A}$ are orthogonal matrices and $B \in \mathbb{R}^{m_A \times n_A}$ is an upper bi-diagonal matrix. The algorithm can be used to find the k most highest singular values with associated singular vectors. Formally, to build a rank-k approximation

$$A \approx A_K = U_K \Sigma_K V_K^T \tag{3.40}$$

of A, where $\Sigma_K \in \mathbb{R}^{K \times K}$ is a diagonal matrix and $U_K \in \mathbb{R}^{n_A \times K}$, $V_K \in \mathbb{R}^{m_A \times K}$ are rectangular passage matrices, we first apply Algorithm 2 to obtain the outputs $P \in \mathbb{R}^{n_A \times K}$, $Q \in \mathbb{R}^{m_A \times K}$, $H \in \mathbb{R}^{K \times K}$. The matrix H represents a truncated version of B and P, Q represent truncated

versions of U and V from equation (3.39). The key observation here is that H is an upper bi-diagonal matrix of small size, therefore its SVD

$$H = X_K \Sigma_K Y_K^T = \sum_{i=1}^{K} \sigma_i x_i y_i^T,$$

with

$$\sigma_1 \geqslant \sigma_2 \geqslant \ldots \geqslant \sigma_K, \qquad X_K \in \mathbb{R}^{K\times K}, \qquad Y_K \in \mathbb{R}^{K\times K},$$

is not expensive to compute. Going back to the initial basis by the left and right multiplications

$$A_K := PHQ^T = PX_K \Sigma_K Y_K^T Q^T,$$

we end up with the desired approximation (3.40) by noticing that

$$U_K = PX_K, \qquad V_K = QY_K.$$

Algorithm 2: Lanczos bi-diagonalization algorithm

Input: $A \in \mathbb{R}^{m_A \times n_A}$, $q^{(0)} \in \mathbb{R}^{m_A}$, $\|q^{(0)}\| = 1$, K (dimension of Krylov subspace), ϵ (precision)
Output: Matrices $P \in \mathbb{R}^{n_A \times K}$, $Q \in \mathbb{R}^{m_A \times K}$, $H \in \mathbb{R}^{K\times K}$
Start:
$w^{(0)} = Aq^{(0)}$
$\alpha^{(0)} = \|w^{(0)}\|$
$p^{(0)} = w^{(0)}/\alpha^{(0)}$
$H[0,0] = \alpha_0$
$P[:,0] = p^{(0)}$
$Q[:,0] = q^{(0)}$
for $k = 0, 1, \ldots, K-1$ **do**
  $z^{(k)} = A^T p^{(k)} - \alpha^{(k)} q^{(k)}$
  $\beta^{(k)} = \|z^{(k)}\|$
  if $\beta^{(k)} \leqslant \epsilon$ **then**
    Break
  else
    $q^{(k+1)} = z^{(k)}/\beta^{(k)}$;
    $w^{(k+1)} = Aq^{(k+1)} - \beta^{(k)} p^{(k)}$;
    $\alpha^{(k+1)} = \|w^{(k+1)}\|$;
    $p^{(k+1)} = w^{(k+1)}/\alpha^{(k+1)}$;
    $H[k+1,k+1] = \alpha^{(k+1)}$;
    $H[k,k+1] = \beta^{(k)}$;
    $P[:,k+1] = p^{(k+1)}$;
    $Q[:,k+1] = q^{(k+1)}$;
  end
end

3.C Computational costs

Here we estimate the computation costs required to compute $\hat{F}$ (estimate of F), $\hat{F}^{-1}$ and $\hat{F}^{-1}\nabla h$ of our proposed methods compared to KFAC. We recall that here d denotes the number of

neurons in each layer, ℓ denotes the number of network layers and m the mini-batch size. Table 3.1 summarizes orders of computational costs required by each method. We did not include forwards and backwards/additional backwards costs as they are the same for all methods. K is the dimension of Krylov subspace in Lanczos bi-diagonalization algorithm (see §3.B.2). k_1 and k_2 represent the number of iterations at which the corresponding algorithm has converged (power SVD or Lanczos bi-diagonalization algorithm). In our experiments, we found that they are of the order of tens. As for c_1 and c_2 they denote implementation constants.

As we can see in Table 3.1, our proposed methods are of the same order of magnitude as KFAC in terms of computation costs.

Table 3.1: Range of the computational costs per update.

	$\hat{F}$	$\hat{F}^{-1}$	$\hat{F}^{-1}\nabla h$
KFAC	$2\ell md^2$	$2\ell d^3$	$2\ell d^3$
KPSVD	$4k_1\ell md^2$	$2\ell d^3$	$2\ell d^3$
DEFLATION	$4k_1\ell md^2 + 4k_1\ell md^2$	$c_1\ell d^3$	$c_2\ell d^3$
LANCZOS	$4k_2\ell md^2 + \ell K^3 + 2\ell Kd^2$	$c_1\ell d^3$	$c_2\ell d^3$
KFAC-CORRECTED	$2\ell md^2 + 4k_1\ell md^2$	$c_1\ell d^3$	$c_2\ell d^3$

Explanation of the entries of Table 3.1

- **KFAC**: To compute $\hat{F}$, we need to compute 2ℓ terms $\bar{A}_{i-1} = \mathbb{E}[\bar{a}_{i-1}\bar{a}_{i-1}^T]$ and $G_i = \mathbb{E}[g_i g_i^T]$ of computational costs $O(md^2)$ each. For $\hat{F}^{-1}$, the inverses of the ℓ pairs $\bar{A}_{i-1}$ and G_i are required. The computational cost of each $\bar{A}_{i-1}^{-1}$ or G_i^{-1} is $O(d^3)$. As for $\hat{F}^{-1}\nabla h$, we need to perform ℓ matrix-matrix multiplications $G_i^{-1}\nabla_W h A_{i-1}^{-1}$ (see equation (3.21)).

- **KPSVD**: The computation of $\hat{F}$ requires to apply the power SVD algorithm. If k_1 is the iteration number of convergence, then for each layer i, we need to perform k_1 matrix-vector multiplications

$$\mathcal{Z}(F_{i,i})v = \mathbb{E}[(g_i^T V g_i)\text{vec}(\bar{a}_{i-1}\bar{a}_{i-1}^T)] \quad \text{and} \quad \mathcal{Z}(F_{i,i})^T u = \mathbb{E}[(\bar{a}_{i-1}^T U \bar{a}_{i-1})\text{vec}(g_i g_i^T)].$$

  The computational cost of $\mathcal{Z}(F_{i,i})v$ or $\mathcal{Z}(F_{i,i})^T u$ is $O(md^2)$. The computational costs required for $\hat{F}^{-1}$ and $\hat{F}^{-1}\nabla h$ are the same as in KFAC.

- **KFAC-CORRECTED**: The computation of $\hat{F}$ is a combination of the computation of $\hat{F}$ in KFAC and in KPSVD so the complexity is the sum of the complexity in KFAC and KPSVD. As for $\hat{F}^{-1}$ and $\hat{F}^{-1}\nabla h$ the technique described in §3.2.5 is used and the complexities are $O(c_1\ell d^3)$ for $\hat{F}^{-1}$ (SVD and matrix-matrix multiplications) and $O(c_2\ell d^3)$ for $\hat{F}^{-1}\nabla h$ (matrix-matrix multiplications).

- **DEFLATION**: To compute $\hat{F}$ for a single layer, we have applied twice the power SVD algorithm and each application has the same cost as in KPSVD. So the total computational cost of computing $\hat{F}$ in DEFLATION is twice the total computational cost of computing $\hat{F}$ in KPSVD. The computational costs required for $\hat{F}^{-1}$ and $\hat{F}^{-1}\nabla h$ are the same as in KFAC-CORRECTED.

- **LANCZOS**: To compute $\hat{F}$, the Lanczos bi-diagonalization algorithm is applied for each layer. Like in KPSVD, if k_2 is the iteration number of convergence then k_2 $\mathcal{Z}(F_{i,i})v$ and $\mathcal{Z}(F_{i,i})^T u$ (in $O(md^2)$ each) were necessary for each layer. At the end of the Lanczos bi-diagonalization algorithm, we need to perform for each layer, the SVD of matrix $H \in \mathbb{R}^{K\times K}$ (in $O(K^3)$) and matrix-matrix operations PX_k (in $O(Kd^2)$) and QY_k (in $O(Kd^2)$).The computational costs required for $\hat{F}^{-1}$ and $\hat{F}^{-1}\nabla h$ are the same as in DEFLATION or KFAC-CORRECTED.

3.D Activation, loss functions, network architectures and datasets

An auto-encoder is a particular type of feed forward neural network used to learn data encoding in an unsupervised way. The aim of such a network is to build a low-dimensional representation of a high-dimensional input data while capturing the most important part of information contained in the input data. An auto-encoder is composed of two modules: the encoder and decoder networks (Figure 3.5). The encoder compresses the input data from the initial space to the encoded or latent space. As for the decoder, it decompresses the encoded data back to initial space. Formally if e_{θ_1} and d_{θ_2} denote the encoder and decoder respectively, then the auto-encoder is defined as

$$f_\theta(x) = d_{\theta_1} \circ e_{\theta_1}(x)$$

where $\circ$ denotes the composition operator and θ is the auto-encoder's parameter and is a concatenation of θ_1 and θ_2. The training process of an auto-encoder consists of minimizing the gap between the input vector x and output $\hat{x} = z = f_\theta(x)$ of the decoder. The loss function is thus defined as

$$L(z, y) = L(z, x) = L(\hat{x}, x) = L\left(f_\theta(x), x\right) = L\left(d_{\theta_1} \circ e_{\theta_1}(x), x\right).$$

Auto-encoders have been extended to variational auto-encoders (VAEs) [13]. Unlike classical auto-encoders which encode the input data as a vector, VAEs learn the latent distribution of the training data. This feature of VAEs allows them to generate new data of the same type as training data.

The two activation functions considered in our work are

- the ReLU function

$$\begin{aligned}\sigma\colon \mathbb{R} &\to \mathbb{R}_+\\ z &\mapsto \max(0, z).\end{aligned}$$

- the Sigmoid function

$$\begin{aligned}\sigma\colon \mathbb{R} &\to [0,1]\\ z &\mapsto \frac{1}{1+\exp(-z)}.\end{aligned}$$

The two loss functions considered in our work are

- The binary cross entropy

$$\begin{aligned}L\colon [0,1]\times[0,1] &\to \mathbb{R}_+\\ (z, y) &\mapsto -\left(y\log(z) + (1-y)\log(1-z)\right).\end{aligned}$$

- The mean square error (MSE)

$$L\colon \mathbb{R}\times\mathbb{R}\to\mathbb{R}_+$$
$$(z,y)\mapsto \|z-y\|_2^2=\sum_{j=1}^{d_y}(z^{[j]}-y^{[j]})^2.$$

The datasets and network architectures [78] used in our tests are described below.

- **Auto-encoder problem 1**
  - Network architecture: $784-1000-500-250-30-250-500-1000-784$
  - Activations functions: ReLU−ReLU−ReLU−ReLU−ReLU−ReLU−ReLU−Sigmoid
  - Data : MNIST (images of shape 28×28 of handwritten digits. 50000 training images and 10000 validation images).
  - Loss function: binary cross entropy

- **Auto-encoder problem 2**
  - Network architecture: $625-2000-1000-500-30-500-1000-2000-625$
  - Activation functions: ReLU−ReLU−ReLU−ReLU−ReLU−ReLU−ReLU−Linear
  - Data : FACES (images of shape 25×25 people. 82800 training images and 20700 validation images).
  - Loss function: mean square error.

- **Auto-encoder problem 3**
  - Network architecture: 784−400−200−100−50−25−6−25−50−100−200−400−784
  - Activations functions: ReLU − ReLU − ReLU − ReLU − ReLU − ReLU − ReLU − ReLU − ReLU − ReLU − ReLU − Sigmoid
  - Data : CURVES (images of shape 28×28 of simulated handdrawn curves. 16000 training images and 4000 validation images).
  - Loss function: binary cross entropy.

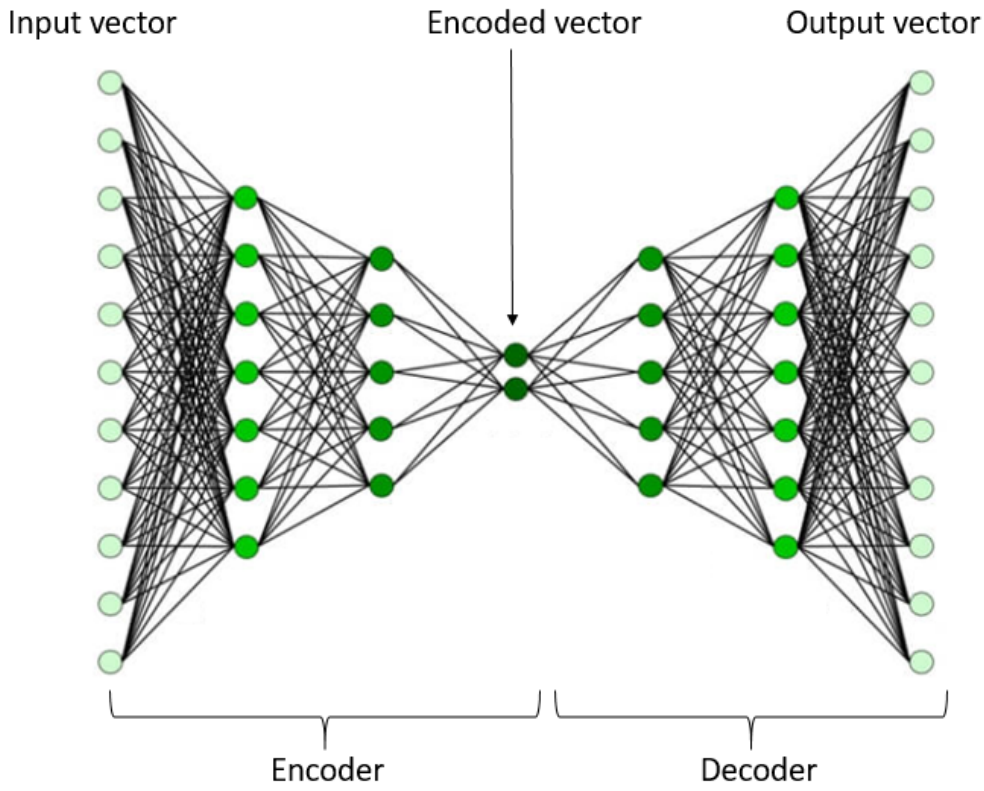


Figure 3.5: A standard auto-encoder architecture.

3.E Gradient clipping

The KL-clipping technique [8] is a method used for step size selection and is an alternative to learning rate schedules. It uses local curvature information to control the amount by which the predictive distribution is allowed to change after each update. In particular, after preconditioning the gradients, one scales them by a factor ν given by

$$\nu = \min\left(1, \sqrt{\frac{c}{\sum_{i=1}^{\ell} |\mathcal{G}_i^T \nabla h(W_i)|}}\right),$$

where $\mathcal{G}_i$ denotes the preconditioned gradient and c is a constant that represents the clipping parameter (the clipping parameter can be interpreted as the maximum value allowed for the squared Fisher norm). Using this normalization, the learning rate α is selected as

$$\alpha = \min\left(\alpha_{\max}, \sqrt{\frac{c}{\sum_{i=1}^{\ell} |\mathcal{G}_i^T \nabla h(W_i)|}}\right),$$

where $\alpha_{\max} > 0$ is a constant number and denotes the maximum step size allowed.

Chapter 4

Analysis and comparison of several two-level KFAC methods for training deep neural networks

Contents

This chapter is devoted to our second contribution to natural gradient methods for neural networks. Calling into question the validity of the block approximation in KFAC, we seek to restore some low-frequency interactions between the layers into the approximation of the Fisher matrix.

We elaborate on the motivations and discuss about the previous works in this direction in §4.1. In §4.2, we work out a series of novel two-level KFAC methods. Inspired from domain decomposition, these are based on several choices of coarse space. In §4.3, we present and comment on several experimental results, which include various test cases and analysis in order to assess the new correctors as fairly as possible. The results obtained in §4.3 lead us to compare KFAC against exact natural gradient in §4.4.

Except for the last section §4.4, the text below is ***a replica of the paper [91]****, which was* ***submitted*** *to* Optimization Methods and Software.

4.1 Towards more accuracy by rough layer interactions?

For computation and memory purposes, KFAC as well as all related variants use only a block-diagonal approximation of the curvature matrix, where each block corresponds to a layer. This results in a loss of information about the correlations between different layers. The question then naturally arises as to whether it is worth trying to recover some of the lost information in hope of making the approximate FIM closer to the true one, thus improving the convergence speed of the optimizer without paying an excessive price.

To this question, Tselepidis et al. [167] provided an element of answer by considering a "coarse" correction to the inverse of the approximate FIM. This additional term is meant to represent the interaction between layers at a "macroscopic" scale, in contrast with the "microscopic" scale of the interaction between neurons inside each layer. Their approach proceeds by formal analogy with the two-level preconditioning technique in domain decomposition [42], substituting the notion of layer for that of subdomain. The difference with domain decomposition, however, lies in the fact that the matrix at hand does not stem from the discretization of any PDE system, and this prevents the construction of coarse spaces from being correctly guided by any physical sense. Notwithstanding this concern, some ready-made recipes can be blindly borrowed from two-level domain decomposition. In this way, Tselepidis et al. [167] reached a positive conclusion regarding the advisability of enriching the approximate FIM with some reduced information about interactions between layers. Nevertheless, their coarse correction is objectionable in some respects, most notably because of inconsistency in the formula for the new matrix (see §4.2 for a full discussion), while for the single favorable case on which is based their conclusion, the network architecture selected is a little too simplistic (see §4.5 for details). Therefore, their claim should not be taken at face value.

Although he did not initially intend to look at the question as formulated above, Benzing [14] recently brought another element of answer that runs counter to the former. By carefully comparing KFAC and the exact natural gradient (as well as FOOF, a method of his own), he came to the astonishingly counterintuitive conclusion that KFAC outperforms the exact natural gradient in terms of optimization performance. In other words, there is no benefit whatsoever in trying to embed any kind of information about the interaction between layers into the curvature matrix, since even the full FIM seems to worsen the situation. While one may not be convinced by his heuristical explanation (whereby KFAC is argued to be a first-order method), his numerical results eloquently speak for themselves. Because Benzing explored a wide variety of networks, it is more difficult to mitigate his findings.

In light of these two contradictory sets of results, we undertook this work in an effort to clarify the matter. To this end, our objective is first to design a family of coarse corrections to KFAC that do not suffer from the mathematical flaws of Tselepidis et al.'s one. This gives rise to a theoretically sound family of approximate FIMs that will next be compared to the original KFAC.

4.2 Two-level KFAC methods

Let us recall the natural gradient descent (NGD) iteration

$$\theta_{k+1} = \theta_k - \alpha_k F_\bullet^{-1} \nabla_\theta h(\mathcal{S}_k, \theta_k), \tag{4.1}$$

where

$$F_\bullet = F + \lambda I_p, \qquad \lambda > 0. \tag{4.2}$$

is a regularized version of F. Likewise, the regularized version of KFAC iteration is given by

$$\theta_{k+1} = \theta_k - \alpha_k F_{\bullet\,\mathrm{KFAC}}^{-1} \nabla_\theta h(\mathcal{S}_k, \theta_k), \tag{4.3a}$$

with

$$F_{\bullet\,\mathrm{KFAC}} = \mathrm{diag}([F_{\bullet\,\mathrm{KFAC}}]_{1,1}, [F_{\bullet\,\mathrm{KFAC}}]_{2,2}, \ldots, [F_{\bullet\,\mathrm{KFAC}}]_{\ell,\ell}). \tag{4.3b}$$

Henceforth, for simplicity and without any loss of generality, the learning rate is assumed to be $\alpha_k = 1$. Let

$$\zeta_k = \theta_k - \theta_{k+1} = [C(\theta_k)]^{-1} \nabla_\theta h(\mathcal{S}_k, \theta_k) \tag{4.4}$$

be the negative increment of θ at iteration k of the generic descent algorithm (2.25). To further alleviate notations, we shall drop the subscript k and omit the dependence on θ_k. For the regularized NGD (4.1), we have

$$\zeta = F_\bullet^{-1} \nabla_\theta h, \tag{4.5}$$

while for the regularized KFAC method (4.3a), we have

$$\zeta_{\mathrm{KFAC}} = F_{\bullet\,\mathrm{KFAC}}^{-1} \nabla_\theta h, \tag{4.6}$$

being understood that the matrices are regularized whenever necessary.

We want to build a new matrix $F_{\bullet\,\mathrm{KFAC\text{-}2L}}^{-1}$, an augmented version of $F_{\bullet\,\mathrm{KFAC}}^{-1}$, such that the solution

$$\zeta_{\mathrm{KFAC\text{-}2L}} = F_{\bullet\,\mathrm{KFAC\text{-}2L}}^{-1} \nabla_\theta h, \tag{4.7}$$

is a better approximation to ζ than ζ_{KFAC}, namely,

$$\|\zeta_{\mathrm{KFAC\text{-}2L}} - \zeta\|_F \ll \|\zeta_{\mathrm{KFAC}} - \zeta\|_F. \tag{4.8}$$

By "augmented" we mean that, at least partially and at some rough scale, $F_{\mathrm{KFAC\text{-}2L}}^{-1}$ takes into account the information about layer interactions that was discarded by the block-diagonal approximation KFAC. The basic tenet underlying this initiative is the belief that a more accurate approximation to the NGD solution ζ at each descent iteration will help the global optimization process to converge faster.

4.2.1 Analogy and dissimilarity with domain decomposition

The construction philosophy of $F_{\bullet\,\mathrm{KFAC\text{-}2L}}^{-1}$ proceeds by analogy with insights from domain decomposition. To properly explain the analogy, we first need to cast the matrix $F_{\bullet\,\mathrm{KFAC}}^{-1}$ under a slightly different form.

For each $i \in \{1, \ldots, \ell\}$, let $R_i \in \mathbb{R}^{p_i \times p}$ be the matrix of the restriction operator from $\mathbb{R}^p$, the total space of all parameters, to the subspace of parameters pertaining to layer i, whose dimension is p_i. In other words, for $(\xi, \eta) \in \{1, \ldots, p_i\} \times \{1, \ldots, p\}$,

$$(R_i)_{\xi\eta} = \begin{cases} 1 & \text{if } \eta = p_1 + \ldots + p_{i-1} + \xi, \\ 0 & \text{otherwise.} \end{cases} \tag{4.9}$$

The transpose $R_i^T \in \mathbb{R}^{p \times p_i}$ then represents the prolongation operator from the subspace of parameters in layer i to the total space of all parameters. Obviously, the i-th diagonal block of the regularized FIM can be expressed as

$$[F_\bullet]_{i,i} = R_i F_\bullet R_i^T.$$

If there were no approximation of each diagonal block by a Kronecker product, then the block-diagonal approximation of F would give rise to the inverse matrix

$$F_{\bullet\,\text{block-diag}}^{-1} = \sum_{i=1}^{\ell} R_i^T [F_\bullet]_{i,i}^{-1} R_i = \sum_{i=1}^{\ell} R_i^T (R_i F_\bullet R_i^T)^{-1} R_i. \tag{4.10}$$

In the case of KFAC, it follows from (3.20)–(3.21) that

$$F_{\bullet\,\text{KFAC}}^{-1} = \sum_{i=1}^{\ell} R_i^T [F_{\bullet\,\text{KFAC}}]_{i,i}^{-1} R_i = \sum_{i=1}^{\ell} R_i^T (\bar{A}_{i-1} + \pi_i \lambda^{1/2} I)^{-1} \otimes (G_i + \pi_i^{-1} \lambda^{1/2} I)^{-1} R_i.$$

In the context of the domain decomposition methods to solve linear systems arising from the discretization of PDEs, the spatial domain of the initial problem is divided into several subdomains. The system is then projected onto the subdomains and the local subproblems are solved independently of each other as smaller systems. In this stage, parallelism can be fully taken advantage of by assigning a processor to each subdomain. This produces a local solution on each subdomain. These local solutions are next combined to create an approximate global solution on the overall domain. Algebraically, the whole process is tantamount to using an inverse matrix of a form similar to (4.10)–(4.11) either within a Schwarz-like iterative procedure or as a preconditioner [42]. The counterparts of $[F_\bullet]_{i,i}^{-1}$ or $[F_{\bullet\,\text{KFAC}}]_{i,i}^{-1}$ are referred to as *local solvers*.

Remark 4.1. The above analogy is not a perfect one. In domain decomposition, the subdomains are allowed (and even recommended!) to overlap each other, so that an unknown can belong to two or more subdomains. In this case, the restriction operators R_i can be much more intricate than the one trivially defined in (4.9).

A well-known issue with domain decomposition methods of the form (4.10)–(4.11) is the disappointingly slow rate of convergence, which results in a lack of *scalability* [42]: the speed-up factor does not grow proportionally with the number of subdomains (and therefore of processors). The reason is that, as the number of subdomains increases, it takes more iterations for an information local to one subdomain to be propagated and taken into account by the others. The common remedy to this problem is to append a "coarse" correction that enables subdomains to communicate with each other in a faster way. The information exchanged in this way is certainly not complete, but only concerns the low frequencies.

Remark 4.2. In domain decomposition, there is a physical problem (represented by the PDE at the continuous level) that serves as a support for the mathematical and numerical reasoning. This is not the case here, where we have to think in a purely algebraic way.

4.2.2 Multiplicative vs. additive coarse correction

We are going to present the idea of two-level KFAC in a very elementary fashion. Let $N \geqslant \ell$ be an integer and $R_0 \in \mathbb{R}^{N \times p}$ be a given matrix. The subspace of $\mathbb{R}^p$ spanned by the columns of $R_0^T \in \mathbb{R}^{p \times N}$ is called the *coarse space*. The choice of the coarse space will be discussed later on. For the moment, we can assume that it is known.

The idea is to add to ζ_{KFAC} a correction term that lives in the coarse space, in such a way that the new vector minimizes the error in the $F_\bullet$-norm with respect to the FIM solution $\zeta = F_\bullet^{-1} \nabla_\theta h$. More concretely, this means that for the negative increment, we consider

$$\zeta_{\text{KFAC-2L}} = \zeta_{\text{KFAC}} + R_0^T \beta^*, \tag{4.11}$$

where

$$\beta^* = \underset{\beta \in \mathbb{R}^\ell}{\operatorname{argmin}} \, \|(\zeta_{\text{KFAC}} + R_0^T \beta) - \zeta\|_{F_\bullet}^2 = \underset{\beta \in \mathbb{R}^\ell}{\operatorname{argmin}} \, \|(\zeta_{\text{KFAC}} + R_0^T \beta) - F_\bullet^{-1} \nabla_\theta h\|_{F_\bullet}^2. \tag{4.12}$$

The solution of the quadratic minimization problem (4.12) is given by

$$\beta^* = (R_0 F_\bullet R_0^T)^{-1} R_0 (\nabla_\theta h - F_\bullet \zeta_{\text{KFAC}}), \tag{4.13}$$

provided that the matrix

$$F_{\text{coarse}} := R_0 F_\bullet R_0^T \in \mathbb{R}^{N \times N}, \tag{4.14}$$

representing the *coarse operator*, be invertible. This is a small size matrix, insofar as N will be in practice taken to be equal to ℓ or 2ℓ, and will in any case remain much smaller than p. This is in agreement with domain decomposition where the size of the coarse system is usually equal to the number of subdomains.

As for the vector

$$r_{\text{KFAC}} := \nabla_\theta h - F_\bullet \zeta_{\text{KFAC}}, \tag{4.15}$$

it is referred to as the *residual* associated to the approximate solution ζ_{KFAC}. Plugging (4.13) into (4.11) and recalling that $\zeta_{\text{KFAC}} = F_{\bullet\,\text{KFAC}}^{-1} \nabla_\theta h$, we end up with

$$\zeta_{\text{KFAC-2L}} = F_{\bullet\,\text{KFAC-2L}}^{-1} \nabla_\theta h, \tag{4.16}$$

with

$$F_{\bullet\,\text{KFAC-2L}}^{-1} = F_{\bullet\,\text{KFAC}}^{-1} + R_0^T F_{\text{coarse}}^{-1} R_0 (I - F_\bullet F_{\bullet\,\text{KFAC}}^{-1}). \tag{4.17}$$

The matrix (4.17) that we propose can be checked to be consistent: if $F_{\bullet\,\text{KFAC}}^{-1}$ and $R_0^T F_{\text{coarse}}^{-1} R_0$ are both homogeneous to $F_\bullet^{-1}$, then $F_{\bullet\,\text{KFAC-2L}}^{-1}$ is homogenous to

$$F_\bullet^{-1} + F_\bullet^{-1} - F_\bullet^{-1} F_\bullet F_\bullet^{-1} = F_\bullet^{-1}$$

too. In the language of domain decomposition, the coarse corrector of (4.17) is said to act *multiplicatively*, to the extent that

$$I - F_{\bullet\,\text{KFAC-2L}}^{-1} F_\bullet = [I - (R_0^T F_{\text{coarse}}^{-1} R_0) F_\bullet][I - F_{\bullet\,\text{KFAC}}^{-1} F_\bullet]. \tag{4.18}$$

as can be straightforwarldy verified. If G is an approximation of $F_\bullet^{-1}$, the matrix $I - GF_\bullet$ measures the quality of this approximation. Equality (4.18) shows that the approximation quality of $F_{\bullet\,\text{KFAC-2L}}^{-1}$ is the product of those of $R_0^T F_{\text{coarse}}^{-1} R_0$ and $F_{\bullet\,\text{KFAC}}^{-1}$.

A common practice in domain decomposition is to drop the factor $I - F_\bullet F_{\bullet\,\text{KFAC}}^{-1}$ (which is equivalent to replacing the residual $r_{\text{KFAC}} = \nabla_\theta h - F_\bullet \theta_{\text{KFAC}}$ by $\nabla_\theta h$). This amounts to approximating $F_{\bullet\,\text{KFAC-2L}}^{-1}$ as

$$F_{\bullet\,\text{KFAC-2L}}^{-1} \approx F_{\bullet\,\text{KFAC}}^{-1} + R_0^T F_{\text{coarse}}^{-1} R_0. \tag{4.19}$$

The coarse corrector of (4.19) is said to act *additively* in domain decomposition. Clearly, the resulting matrix is inconsistent with $F_\bullet^{-1}$: in fact, it is consistent with $2F_\bullet^{-1}$! No matter how crude it is, this coarse corrector is actually valid as long as $F_{\bullet\,\text{KFAC-2L}}^{-1}$ is used only as a preconditioner in the resolution of the system $F_\bullet \zeta = \nabla_\theta h$, which means that we solve instead

$F_{\bullet\,\text{KFAC-2L}}^{-1} F_\bullet \zeta = F_{\bullet\,\text{KFAC-2L}}^{-1} \nabla_\theta h$ to benefit from a more favorable conditioning but the solution we seek remains the same.

Here, in our problem, $F_\bullet^{-1}$ is directly approximated by $F_{\bullet\,\text{KFAC-2L}}^{-1}$ and therefore the inconsistent additive coarse corrector (4.19) is not acceptable. Note that Tselepidis et al. [167] adopted this additive coarse correction, in which F_{coarse} is approximated as

$$F_{\text{coarse}} \approx R_0 \bar{F}_\bullet R_0^T, \tag{4.20a}$$

where $\bar{F}_\bullet$ is the block-diagonal matrix whose blocks $[\bar{F}_\bullet]_{i,j}$ are given by

$$[\bar{F}_\bullet]_{i,j} = \begin{cases} \mathbb{E}[\bar{a}_{i-1}\bar{a}_{j-1}^T] \otimes \mathbb{E}[g_i g_j^T] & \text{if } i \neq j, \\ (\bar{A}_{i-1} + \pi_i \lambda^{1/2} I) \otimes (G_i + \pi_i^{-1} \lambda^{1/2} I) & \text{if } i = j. \end{cases} \tag{4.20b}$$

In this work, we focus to the consistent multiplicative coarse corrector (4.17) and also consider the exact value (4.14) for F_{coarse}.

4.2.3 Choice of the coarse space R_0^T

By the construction (4.11)–(4.12), we are guaranteed that

$$\|\zeta_{\text{KFAC-2L}} - \zeta\|_{F_\bullet}^2 \leqslant \|\zeta_{\text{KFAC}} - \zeta\|_{F_\bullet}^2 \tag{4.21}$$

for any coarse space R_0^T, since the right-hand side corresponds to $\beta = 0$. The choice of R_0^T is a compromise between having a small dimension $N \ll p$ and lowering the new error

$$\|\zeta_{\text{KFAC-2L}} - \zeta\|_{F_\bullet}^2 = \| -[I - R_0^T (R_0 F_\bullet R_0^T)^{-1} R_0 F_\bullet][I - F_{\bullet\,\text{KFAC}}^{-1} F_\bullet]\zeta \|_{F_\bullet}^2 \tag{4.22}$$

as much as possible. But it seems out of reach to carry out the minimization of the latter with respect to the entries of R_0^T.

In the context of the preconditioner, the idea behind a two-level method is to remove first the influence of very large eigenvalues which correspond to high-frequency modes, then remove the smallest eigenvalues thanks to the second level, which affect greatly the convergence. To do so, we need a suitable coarse space to efficiently deal with this second level [117]. Ideally, we would like to choose the deflation subspace which consists of the eigenvectors associated with the small eigenvalues of the preconditioned operator. However, this computation is more costly than solving a linear system itself.

This leads us to choose the coarse space in an a priori way. We consider the a priori form

$$R_0^T = \begin{bmatrix} V_1 & 0 & \dots & \dots & 0 \\ 0 & V_2 & \dots & \dots & 0 \\ \vdots & \vdots & \ddots & & \vdots \\ 0 & 0 & \dots & \dots & V_\ell \end{bmatrix} \in \mathbb{R}^{p \times N}, \tag{4.23}$$

where each block $V_i \in \mathbb{R}^{p_i \times N_i}$ has N_i columns with $N_i \ll p_i$, and

$$N_1 + N_2 + \ldots + N_\ell = N. \tag{4.24}$$

To provide a comparative study, we propose to evaluate several coarse space choices of the form (4.23) that are discussed below.

Nicolaides coarse space. Historically, this is the first [121] coarse space ever proposed in domain decomposition. Transposed to our case, it corresponds to

$$N_1 = \ldots = N_\ell = 1, \qquad N = \ell, \tag{4.25}$$

and for all $i \in \{1, \ldots, \ell\}$,

$$V_i = \begin{bmatrix} 1, \ldots, 1 \end{bmatrix}^T \in \mathbb{R}^{p_i}. \tag{4.26}$$

Originally, the motivation for selecting the vector whose all components are equal to 1 is that it is the discrete version of a continuous constant field, which is the eigenvector associated with the eigenvalue 0 of the operator $-\nabla \cdot (\kappa \nabla)$ (boundary conditions being set aside). Inserting it into the coarse space helps the solver take care of the lowest frequency mode. In our problem, however, there is no reason for 0 to be an eigenvalue of F, nor for 1 to be an eigenvector if this is the case. Hence, there is no justification for the Nicolaides coarse space. Still, this choice remains convenient and practical. This is probably the reason why Tselepidis et al. [167] have opted for it.

Spectral coarse space. This is a slightly refined version of the Nicolaides coarse space. The idea is always to capture the lowest mode [117], but since the lowest eigenvalue and eigenvector are not known in advance, we have to compute them. More specifically, we keep the values (4.25) for the column sizes within R_0^T, while prescribing

$$V_i = \text{eigenvector associated to the smallest eigenvalue of } [F_{\bullet\,\mathrm{KFAC}}]_{i,i} \tag{4.27}$$

for all $i \in \{1, \ldots, \ell\}$. In our case, an advantageous feature of this definition is that the cost of computing the eigenvectors is "amortized" by that of the inverses of $[F_{\mathrm{KFAC}}]_{i,i}$, in the sense that these two calculations can be carried out simultaneously. Indeed, let

$$\bar{A}_{i-1} + \pi_i \lambda^{1/2} I = U_{\bar{A}_{i-1}} \Sigma_{\bar{A}_{i-1}} V_{\bar{A}_{i-1}}^T, \qquad G_i + \pi_i^{-1} \lambda^{1/2} I = U_{G_i} \Sigma_{G_i} V_{G_i}^T \tag{4.28}$$

be the singular value decompositions of $\bar{A}_{i-1} + \pi_i \lambda^{1/2} I$ and $G_i + \pi_i^{-1} \lambda^{1/2} I$ respectively. Then, we have

$$\begin{aligned} [F_{\bullet\,\mathrm{KFAC}}]_{i,i}^{-1} &= (\bar{A}_{i-1} + \pi_i \lambda^{1/2} I)^{-1} \otimes (G_i + \pi_i^{-1} \lambda^{1/2} I)^{-1} \\ &= (U_{\bar{A}_{i-1}} \Sigma_{\bar{A}_{i-1}} V_{\bar{A}_{i-1}}^T)^{-1} \otimes (U_{G_i} \Sigma_{G_i} V_{G_i}^T)^{-1} \\ &= (U_{\bar{A}_{i-1}} \Sigma_{\bar{A}_{i-1}}^{-1} V_{\bar{A}_{i-1}}^T) \otimes (U_{G_i} \Sigma_{G_i}^{-1} V_{G_i}^T). \end{aligned} \tag{4.29}$$

Since $\Sigma_{\bar{A}_{i-1}}$ and Σ_{G_i} are diagonal matrices, their inverses are easy to compute. Now, if $V_{\bar{A}_{i-1}}$ and V_{G_i} are the eigenvectors associated to the smallest eigenvalues of $\bar{A}_{i-1}$ and G_i respectively, then the eigenvector associated to the smallest eigenvalue of $[F_{\bullet\,\mathrm{KFAC}}]_{i,i}$ is given by

$$V_i = V_{\bar{A}_{i-1}} \otimes V_{G_i}. \tag{4.30}$$

Krylov coarse space. If we do not wish to compute the eigenvector associated to the smallest eigenvalue of $[F_{\bullet\,\mathrm{KFAC}}]_{i,i}$, then a variant of the spectral coarse space could be the following. We know that this eigenvector can be obtained by the inverse power method. The idea is then to perform a few iterations of this method, even barely one or two, and to include the iterates into

the the coarse subspace. If $N_i - 1 \geqslant 1$ is the number of inverse power iterations performed for $[F_{\bullet\,\mathrm{KFAC}}]_{i,i}$, then we take

$$V_i = [v_i,\ \ [F_{\bullet\,\mathrm{KFAC}}]_{i,i}^{-1} v_i,\ \ \ldots,\ \ [F_{\bullet\,\mathrm{KFAC}}]_{i,i}^{-(N_i-1)} v_i] \in \mathbb{R}^{p_i \times N_i} \tag{4.31}$$

where $v_i \in \mathbb{R}^{p_i}$ is an arbitrary vector, assumed to not be an eigenvector of $[F_{\bullet\,\mathrm{KFAC}}]_{i,i}$ to ensure that the columns of V_i are not collinear. By appropriately selecting v_i, we are in a position to use this approach to enrich the Nicolaides coarse space and the residuals coarse space (cf. next construction).

The increase in the number of columns for V_i is not the price to be paid to avoid the eigenvector calculation: we could have put only the last iterate $[F_{\bullet\,\mathrm{KFAC}}]_{i,i}^{-(N_i-1)} v_i$ into V_i. But since we have computed the previous ones, it seems more cost-effective to use them all to enlarge the coarse space. The larger the latter is, the lower is the minimum value of the objective function. In this work, we consider the simplest case

$$N_1 = \ldots = N_\ell = 2, \qquad N = 2\ell. \tag{4.32}$$

Residuals coarse space. We now introduce a very different philosophy of coarse space, which to our knowledge has never been envisioned before. From the construction (4.11)–(4.12), it is obvious that if the error $\zeta - \zeta_{\mathrm{KFAC}}$ belongs to the coarse space R_0^T, that is, if it can be written as a linear combination $R_0^T \beta^\sharp$ of the coarse matrix columns, then the vector $\zeta_{\mathrm{KFAC}} + R_0^T \beta^\sharp$ coincides with the exact solution ζ and the correction would be ideally optimal. Although this error $\zeta - \zeta_{\mathrm{KFAC}}$ is unknown, it is connected to the residual (4.15) by

$$\zeta - \zeta_{\mathrm{KFAC}} = F_\bullet^{-1} r_{\mathrm{KFAC}}. \tag{4.33}$$

The residual r_{KFAC} is not too expensive to compute. as it consists of a direct matrix-product $F\zeta_{\mathrm{KFAC}}$. Unfortunately, solving a linear system involving F as required by (4.33) is what we want to avoid.

But we can just approximate this error by inverting with $F_{\bullet\,\mathrm{KFAC}}^{-1}$ instead of $F_\bullet^{-1}$. Therefore, we propose to build a coarse space that contains $F_{\bullet\,\mathrm{KFAC}}^{-1} r_{\mathrm{KFAC}}$ instead of $F_\bullet^{-1} r_{\mathrm{KFAC}}$. To this end, we split $F_{\bullet\,\mathrm{KFAC}}^{-1} r_{\mathrm{KFAC}}$ into ℓ segments, each corresponding to a layer. This amounts to choosing the values (4.25) for the column sizes and set the columns of R_0^T as

$$V_i = [F_{\bullet\,\mathrm{KFAC}}]_{i,i}^{-1} r_{\mathrm{KFAC}}[i] \in \mathbb{R}^{p_i}, \qquad r_{\mathrm{KFAC}}[i] = \mathrm{vec}(\mathcal{D}W_i) - (F_\bullet \zeta_{\mathrm{KFAC}})[i] \tag{4.34}$$

for $i \in \{1, \ldots, \ell\}$, where for a vector $\xi \in \mathbb{R}^p$ the notation $\xi[i] = \xi(p_{i-1} + 1 : p_i)$ designates the portion related to layer i. Formulas (4.34) ensure that $F_{\bullet\,\mathrm{KFAC}}^{-1} r_{\mathrm{KFAC}}$ belongs to the coarse space. Indeed, taking $\beta = [1, \ldots, 1]^T \in \mathbb{R}^\ell$, we find $R_0^T \beta = F_{\bullet\,\mathrm{KFAC}}^{-1} r_{\mathrm{KFAC}}$.

Taylor coarse space. The previous coarse space is the zeroth-order representative of a family of more sophisticated constructions based on a formal Taylor expansion of $F_\bullet^{-1}$, which we now present but which will not be implemented. Setting

$$E = I - F_{\bullet\,\mathrm{KFAC}}^{-1} F_\bullet \tag{4.35}$$

and observing that $F_\bullet = F_{\bullet\,\mathrm{KFAC}}(I - E)$, we have

$$F_\bullet^{-1} = (I - E)^{-1} F_{\bullet\,\mathrm{KFAC}}^{-1} = (I + E + \ldots + E^{q-1} + \ldots) F_{\bullet\,\mathrm{KFAC}}^{-1}. \tag{4.36}$$

The formal series expansion in the last equality rests upon the intuition that E measures the approximation quality of $F_\bullet^{-1}$ by $F_{\bullet\,\mathrm{KFAC}}^{-1}$ and therefore can be assumed to be small. Multiplying both sides by the residual r_{KFAC} and stopping the expansion at order $q-1 \geqslant 0$, we obtain the approximation

$$(I + E + \ldots + E^{q-1}) F_{\bullet\,\mathrm{KFAC}}^{-1} r_{\mathrm{KFAC}} \tag{4.37}$$

for the error $F_\bullet^{-1} r_{\mathrm{KFAC}} = \zeta - \zeta_{\mathrm{FAC}}$, which is also the ideal correction term. As earlier, we impose that this approximate correction vector (4.37) must be contained in the coarse space R_0^T. This suggests to extract the components in layer i of the vectors

$$\{F_{\bullet\,\mathrm{KFAC}}^{-1} r_{\mathrm{KFAC}},\ E F_{\bullet\,\mathrm{KFAC}}^{-1} r_{\mathrm{KFAC}},\ \ldots,\ E^{q-1} F_{\bullet\,\mathrm{KFAC}}^{-1} r_{\mathrm{KFAC}}\}$$

and assign them to the columns of V_i. In view of (4.35), the space spanned by the above vectors is the same as the one spanned by

$$\{F_{\bullet\,\mathrm{KFAC}}^{-1} r_{\mathrm{KFAC}},\ (F_{\bullet\,\mathrm{KFAC}}^{-1} F_\bullet) F_{\bullet\,\mathrm{KFAC}}^{-1} r_{\mathrm{KFAC}},\ \ldots,\ (F_{\bullet\,\mathrm{KFAC}}^{-1} F_\bullet)^{q-1} F_{\bullet\,\mathrm{KFAC}}^{-1} r_{\mathrm{KFAC}}\}.$$

Consequently, we can take

$$N_1 = \ldots = N_\ell = q, \qquad N = q\ell, \tag{4.38}$$

and

$$V_i = [w_1[i],\ w_2[i],\ \ldots,\ w_q[i]] \in \mathbb{R}^{p_i \times N_i} \tag{4.39}$$

where

$$w_1 = F_{\bullet\,\mathrm{KFAC}}^{-1} r_{\mathrm{KFAC}} \in \mathbb{R}^p, \qquad w_{j+1} = F_{\bullet\,\mathrm{KFAC}}^{-1} F_\bullet\, w_j \in \mathbb{R}^p, \tag{4.40}$$

for $1 \leqslant j \leqslant q-1$. The case $q = 1$ degenerates to the residuals coarse space. From (4.40), we see that upgrading to the next order is done by multiplying by $F_\bullet$, an operation that mixes the layers.

For the practical implementation of these coarse spaces, we need efficient computational methods for two essential building blocks, namely, the matrix-vector product $F_\bullet u$ and the coarse operator F_{coarse}. These will be described in appendix §4.A.

4.2.4 Pseudo-code for two-level KFAC methods

Algorithm 3 summarizes the steps for setting up a two-level KFAC method.

Algorithm 3: High-level pseudo-code for a two-level KFAC method

Input: θ_0 (Initial point), $k_{\max}$ (maximum number of iterations), and α (learning rate)
Output: $\theta_{k_{\max}}$
for $k = 0, 1, \ldots, k_{\max} - 1$ **do**
- Compute an estimate $\nabla_\theta h(\mathcal{S}_k, \theta_k)$ of the gradient on a mini-batch $\mathcal{S}_k$ randomly sampled from the training data;
- Compute $\theta_{\mathrm{KFAC}} = F_{\bullet\,\mathrm{KFAC}}^{-1} \nabla_\theta h(\mathcal{S}_k, \theta_k)$;
- Choose a coarse space R_0^T and compute the associated coarse correction $R_0\beta^* = R_0^T (F_{\mathrm{coarse}})^{-1} R_0 r_{\mathrm{KFAC}}$;
- Compute $\theta_{\text{KFAC-2L}} = \theta_{\mathrm{KFAC}} + R_0\beta^*$;
- Update $\theta_{k+1} = \theta_k - \alpha\theta_{\text{KFAC-2L}}$;

end

4.3 Numerical results

In this section, we compare the new two-level KFAC methods designed in §4.2 with the standard KFAC from the standpoint of convergence speed. For a thorough analysis, we also include the two-level KFAC version of Tselepidis et al. [167] and baseline optimizers (ADAM and SGD).

We run a series of experiments to investigate the optimization performance of deep auto-encoders, CNNs, and deep linear networks. Since our primary focus is on convergence speed rather than generalization, we shall only be concerned with the ability of optimizers to minimize the objective function. In particular, we report only training losses for each optimizer. To equally treat all methods, we adopt the following rules. We perform a Grid Search and select hyper-parameters that give the best reduction to the training loss. Learning rates for all methods and damping parameters for KFAC and two-level KFAC methods are searched in the range

$$\{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 10^{0}, 10^{1}, 10^{2}, 10^{3}, 10^{4}\}.$$

For each optimizer, we apply the Early Stopping technique with patience of 10 epochs i.e. we stop training the network when there is no decrease in the training loss during 10 consecutive epochs). We also include weight decay with a coefficient of 10^{-3} for all optimizers.

All experiments presented in this work are performed with PyTorch framework [129] on a supercomputer with Nvidia Ampere A100 GPU and AMD Milan@2.45GHz CPU. For ease of reading, the following table explains all abbreviations of two-level KFAC methods that we will use in the figure legends.

Table 4.1: Name abbreviations of two-level KFAC optimizers.

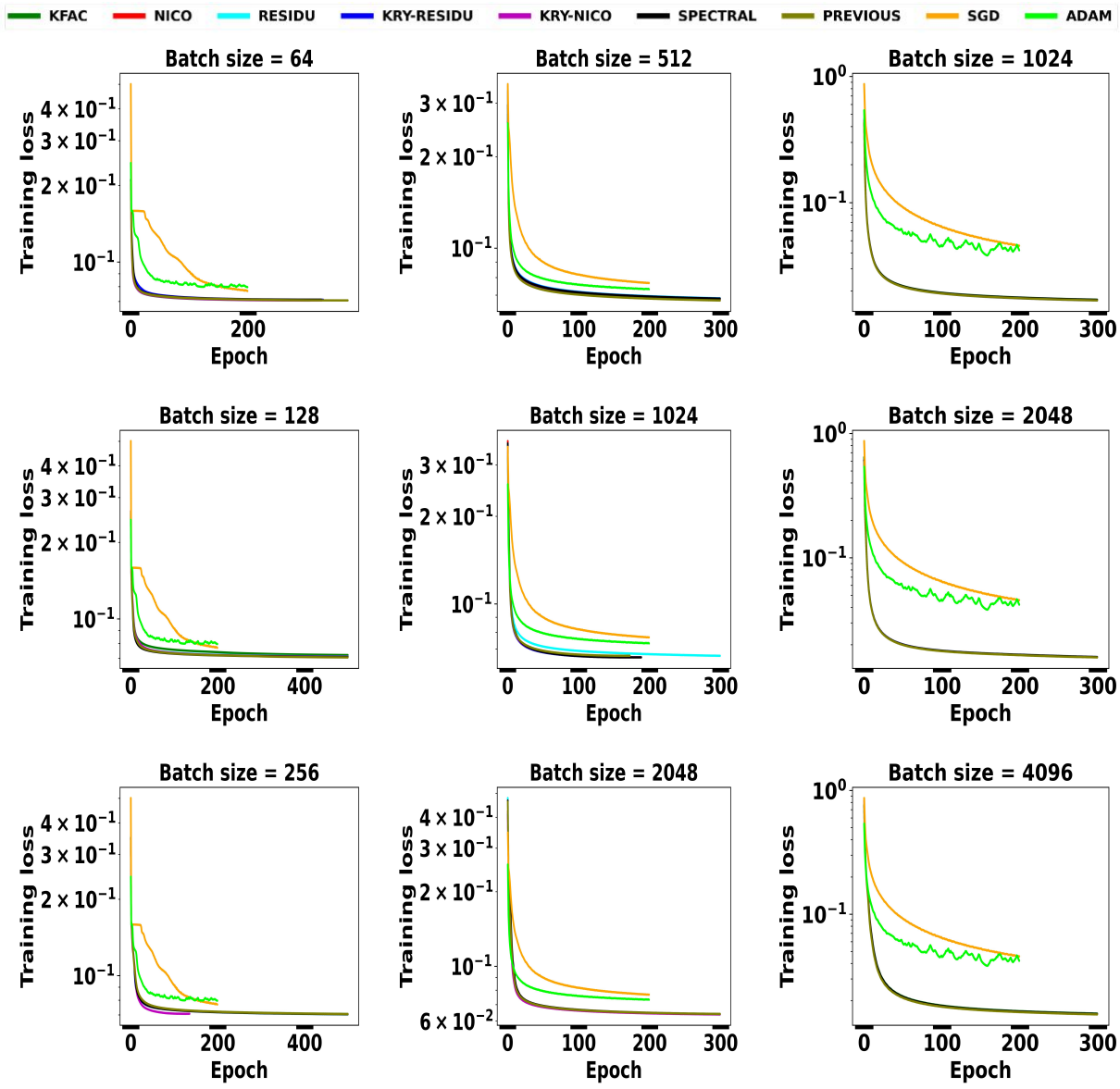
Optimizer	Name abbreviation
Two-level KFAC with Nicolaides coarse space	NICO
Two-level KFAC with spectral coarse space	SPECTRAL
Two-level KFAC with residuals coarse space	RESIDU
Two-level KFAC with Krylov Nicolaides coarse space	KRY-NICO
Two-level KFAC with Krylov residuals coarse space	KRY-RESIDU
Two-level KFAC of Tselepidis et al. [167]	PREVIOUS

4.3.1 Deep auto-encoder problems

The first set of experimental tests consists in optimizing the three deep auto-encoder problems (see appendix 3.D, chapter 3 for the description of the problems i.e. network architectures and datasets). For each problem, we train the network with three different batch sizes.

Figure 4.1 shows the obtained results. The first observation is that, as expected, natural gradient-based methods (KFAC and two-level KFAC methods) outperform baseline optimizers (ADAM and SGD). The second and most important observation is that, for each of the three problems, regardless of the batch size, the training curve of KFAC and those of all two-level KFAC methods (the one of Tselepidis et al. [167] and those proposed in this work) are overlaid, which means that taking into account the extra-diagonal terms of the Fisher matrix through two-level decomposition methods does not improve the convergence speed of KFAC method.

This second observation is quite puzzling, since theoretically two-level methods are supposed to offer a better approximation to the exact natural gradient than KFAC does and therefore



(a) CURVES (b) MNIST (c) FACES

Figure 4.1: Comparison of KFAC against two-level KFAC methods on the three deep auto-encoder problems (CURVES **first** column, MNIST **middle** column and FACES **last** column). Three different batch sizes are considered for each problem (each row corresponds to a different batch size).

should at least slightly outperform KFAC in terms of optimization performance. Note that we repeated these experiments on three different random seeds and obtained very similar results.

These surprising results are in line with the findings of Benzing [14], according to which KFAC outperforms the exact natural gradient in terms of optimization performance (see §4.4 for an in-depth discussion of this statement). This suggests that extra-diagonal blocks of the FIM do not contribute to improving the optimization performance, and sometimes even affect it negatively.

4.3.2 Convolution neural networks

The second set of experiments concerns the optimization of three different CNNs namely Resnet 18 [74], Cuda-convnet and Resnet 34 [74]. We consider in particular Cuda-convnet which is the architecture used to evaluate the original KFAC method in [68]. We train Cuda-convnet on CIFAR10 dataset [92] with a batch size equal to 256, and Resnet 18 on CIFAR100 [92] with a batch size equal to 128. Finally, we train Resnet 34 on the SVHN dataset [119] with a batch size equal to 512.

For these CNNs (see Figure 4.2), we arrive at quite similar observations and conclusions to those we mention for deep auto-encoder problems. In particular, like in [167], when considering CNNs, we do not observe any significant gain in the convergence speed of KFAC when we enrich it with cross-layer information through two-level decomposition methods. Once again, these results corroborate the claims of Benzing [14] and suggest that we do not need to take into account the extra diagonal blocks of the FIM.

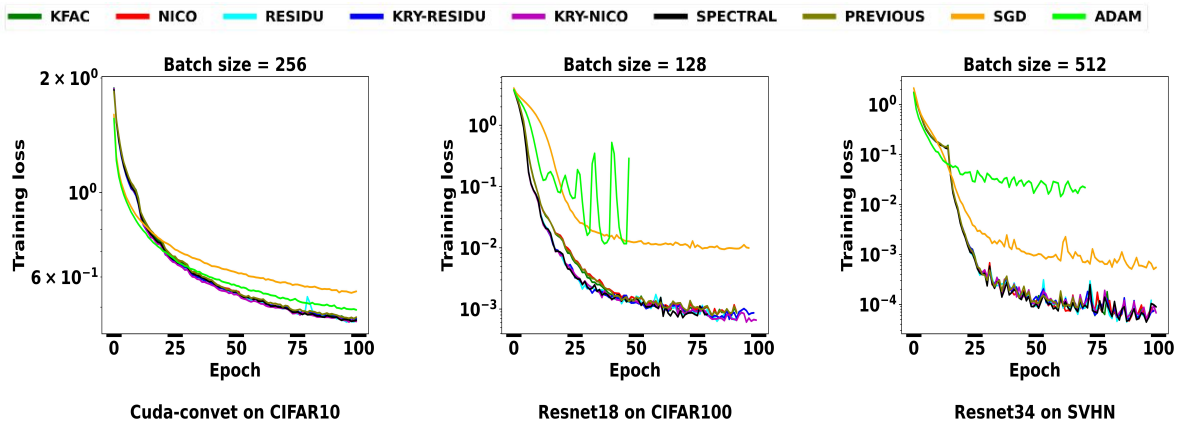


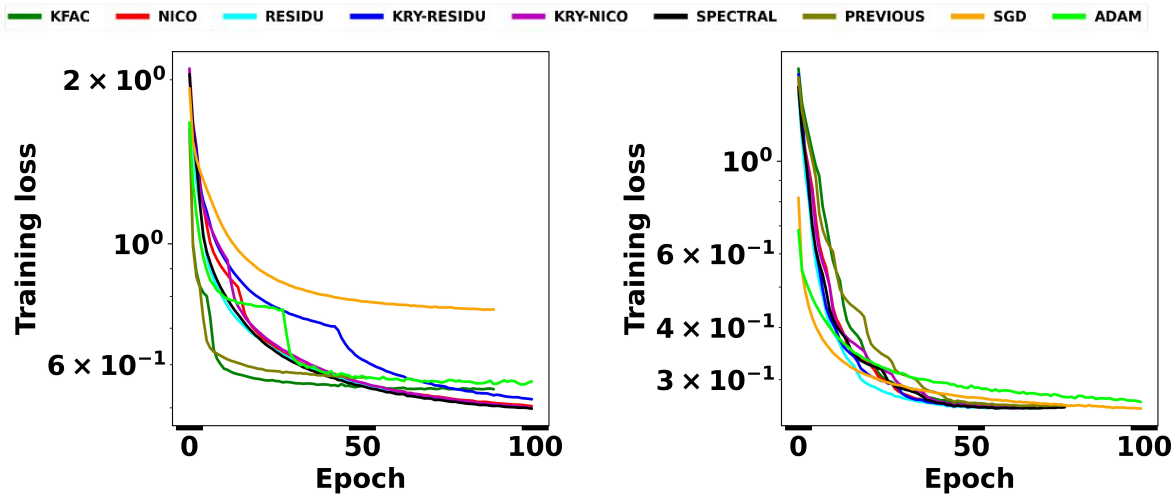
Figure 4.2: Optimization performance evaluation of KFAC and two-level KFAC methods on three different CNNs.

4.3.3 Deep linear networks

The last experiments concern relatively simple optimization problems: linear networks optimization. A deep linear network refers to a neural network architecture that consists of multiple layers of linear units without any non-linear activation functions. In other words, each layer in a deep linear network performs a linear transformation on its inputs, and the output of one layer becomes the input for the next layer without introducing non-linearities. We consider two deep linear networks. These tests are motivated by the results obtained by Tselepidis et al. [167] for their two-level method. Indeed, for an extremely simple linear network with 64 layers (each layer

contains 10 neurons and a batch normalization layer) trained with randomly generated ten-size input vectors, they outperform KFAC in terms of optimization performance. Here, we first consider the same architecture but train the network on the Fashion MNIST dataset [180] (since we could not use the same dataset). Then, we consider another linear network that contains 14 layers with batch normalization, with this time much larger layers. More precisely we consider the following architecture: $784-1000-900-800-700-600-500-400-300-200-100-50-20-10$. We train this second network on the MNIST dataset. As in [167], both networks are trained with a batch size of 512.

Figure 4.3 shows the training curves obtained in both cases. Here, we observe like in [167] an improvement in the optimization performance of two-level optimizers over KFAC. However, this gain remains too small and only concerns simple linear networks that are not used for practical applications. We therefore do not encourage enriching KFAC with two-level methods that require additional computational costs.



(a) Deep linear network of 64 layers, each layer containing 10 neurons, trained on Fashion MNIST.

(b) Deep linear network of 14 larger layers, trained on MNIST.

Figure 4.3: Optimization performance evaluation of KFAC and Tow-level KFAC optimizers on two different deep linear networks.

4.3.4 Verification of error reduction for linear systems

In the above experiments, two-level methods do not seem to outperform KFAC in terms of optimization performance. We, therefore, wish to verify that at each descent iteration, the negative increment of KFAC-2L, $\zeta_{\text{KFAC-2L}}$, which is obtained by the coarse correction, is closer to the regularized natural gradient one ζ, than the negative increment ζ_{KFAC} which corresponds to the original KFAC. In other words, in this section, we verify the inequality (4.21) by evaluating first the error reduction at each descent iteration and validating the expected behaviour.

For $\beta \in \mathbb{R}^m$, let

$$\mathfrak{E}(\beta) = \|\zeta_{\text{KFAC}} + R_0^T \beta - \zeta\|_F^2, \tag{4.41}$$

be the function to be minimized at fixed R_0^T in the construction (4.11)–(4.12), where it is recalled

that

$$\zeta = F_\bullet^{-1}\nabla_\theta h, \qquad \zeta_{\text{KFAC}} = F_{\bullet\,\text{KFAC}}^{-1}\nabla_\theta h.$$

Note that

$$\mathfrak{E}(0) = \|\zeta_{\text{KFAC}} - \zeta\|_{F_\bullet}^2 \tag{4.42}$$

is the squared $F_\bullet$-distance between the KFAC increment and that natural gradient one, regardless of R_0^T. Meanwhile, if β^* is taken to be the optimal value (4.13), then

$$\mathfrak{E}(\beta^*) = \|\zeta_{\text{KFAC-2L}} - \zeta\|_{F_\bullet}^2. \tag{4.43}$$

To see whether (4.21) is satisfied, the idea is to compute the difference $\mathfrak{E}(\beta^*) - \mathfrak{E}(0)$ and check that it is negative. The goal of the game, however, is to avoid using the unknown natural gradient solution ζ. Owing to the identity $\|a\|^2 - \|b\|^2 = (a-b, a+b)$ for the $F_\bullet$-dot product, this difference can be transformed into

$$\begin{aligned}
\mathfrak{E}(\beta^*) - \mathfrak{E}(0) &= \|\zeta_{\text{KFAC-2L}} - \zeta\|_{F_\bullet}^2 - \|\zeta_{\text{KFAC}} - \zeta\|_{F_\bullet}^2 \\
&= (\zeta_{\text{KFAC-2L}} - \zeta_{\text{KFAC}},\, \zeta_{\text{KFAC-2L}} + \zeta_{\text{KFAC}} - 2\zeta)_{F_\bullet} \\
&= \|\zeta_{\text{KFAC-2L}} - \zeta_{\text{KFAC}}\|_{F_\bullet}^2 + 2(\zeta_{\text{KFAC-2L}} - \zeta_{\text{KFAC}},\, \zeta_{\text{KFAC}} - \zeta)_{F_\bullet} \\
&= \|R_0^T\beta^*\|_{F_\bullet}^2 + 2(R_0^T\beta^*,\, \zeta_{\text{KFAC}} - \zeta)_{F_\bullet} \\
&= \big\langle F_\bullet R_0^T\beta^*, R_0^T\beta^*\big\rangle + 2\big\langle R_0^T\beta^*,\, F_\bullet(\zeta_{\text{KFAC}} - \zeta)\big\rangle,
\end{aligned} \tag{4.44}$$

where $\langle\cdot,\cdot\rangle$ denotes the Euclidean dot product. But

$$F_\bullet(\zeta_{\text{KFAC}} - \zeta) = F_\bullet\zeta_{\text{KFAC}} - \nabla_\theta h = -r_{\text{KFAC}} \tag{4.45}$$

is the opposite of the residual (4.15), which can be computed without knowing ζ. Finally, the desired difference can also be computed as

$$\mathfrak{E}(\beta^*) - \mathfrak{E}(0) = \big\langle R_0 F_\bullet R_0^T\beta^*,\, \beta^*\big\rangle - 2\big\langle R_0^T\beta^*,\, r_{\text{KFAC}}\big\rangle. \tag{4.46}$$

For the two-level method of Tselepidis-Kohler-Orvieto [167], the correction reads

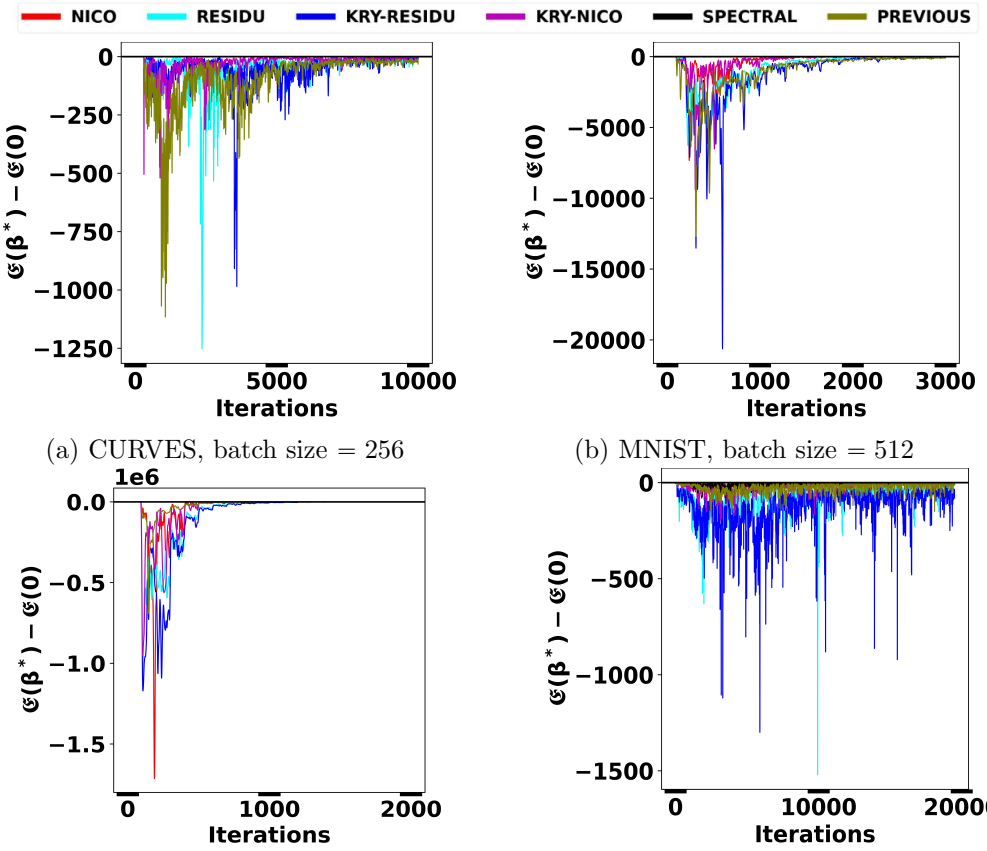
$$\zeta_{\text{TKO}} = \zeta_{\text{KFAC}} + R_0^T\beta^*_{\text{TKO}} \tag{4.47a}$$

with

$$\beta^*_{\text{TKO}} = (R_0 F_\bullet R_0^T)^{-1} R_0\nabla_\theta h \tag{4.47b}$$

instead of β^*, the KFAC-2L value (4.13). The difference $\mathfrak{E}(\beta^*_{TKO}) - \mathfrak{E}(0)$ is then given by a formula similar to (4.46) in which β^* is simply replaced by β^*_{TKO}.

We compute the error $\mathfrak{E}(\beta^*) - \mathfrak{E}(0)$ associated to various two-level methods in the experiments conducted above. More specifically, we do it for the three deep auto-encoder problems and also for a CNN (cuda-convnet). The results obtained are shown in Figure 4.4. The observation is that all two-level methods proposed in this work as well as the TKO two-level method [167] have negative gaps $\mathfrak{E}(\beta^*) - \mathfrak{E}(0)$ throughout the optimization process. This implies that two-level methods solve the linear system (4.5) more accurately than KFAC does. It also means that the approximate natural gradients obtained with Two-level methods are closer to the exact natural gradient than the one obtained with KFAC. These results reveal that closeness to the exact natural gradient instigated with the inclusion of extra-diagonal blocks of the FIM does not necessarily results in a more efficient algorithm.



(a) CURVES, batch size = 256

(b) MNIST, batch size = 512

(c) FACES, batch size=1024

(d) Cuda-convnet, batch size=256

Figure 4.4: Evolution of $\mathfrak{E}(\beta^*) - \mathfrak{E}(0)$ during training for each of the two-level methods considered. All methods proposed in this work as well as the TKO two-level method [167] have the gap $\mathfrak{E}(\beta^*) - \mathfrak{E}(0)$ negative throughout the training process.

4.4 Comparison of KFAC against exact natural gradient

From §4.3.4, we come to the conclusion that despite the fact that two-level methods do not outperform KFAC in terms of optimization performance, they offer an increment much closer to the exact natural gradient (NG) than that obtained with KFAC. In this section, we propose to investigate more this study. This leads us to wonder about the effectiveness of exact NG compared to KFAC.

To our concerns, Benzig states in [14] that KFAC outperforms exact NG in terms of optimization performance. Here we wish to corroborate this statement which a priori seems contradictory, since KFAC is an approximation of the natural gradient. To do so, we implement the exact NG with the whole FIM. We also include the exact block natural gradient (block-diagonal NG) using the diagonal blocks of the FIM, without Kronecker-factored approximation.

As in [14], to design the exact NG and block-diagonal NG, we make use of the Sherman-Morrison-Woodbury inversion formula [47].

4.4.1 Design of exact NG and block-diagonal NG via matrix inversion lemma

Exact NG. The decisive step for setting up the exact NG algorithm (4.1) is the computation of the negative increment

$$\zeta = F_\bullet^{-1}\,\nabla_\theta h = (F + \lambda I_p)^{-1}\nabla_\theta h,$$

where F is computed with a batch of samples $\mathcal{S} \subset \mathcal{D}$, using Monte-Carlo estimation i.e.

$$F = \mathbb{E}[\mathcal{D}\theta(\mathcal{D}\theta)^T] \approx \frac{1}{m}\sum_{b=1}^{m}(\mathcal{D}\theta)^{(b)}((\mathcal{D}\theta)^{(b)})^T = \frac{1}{m}JJ^T, \tag{4.48}$$

where $J \in \mathbb{R}^{p\times m}$ is the matrix whose b-th column is $(\mathcal{D}\theta)^{(b)}$. We will derive the formulas for any vector u. One just needs to take u equal to $\nabla_\theta h$ in the case of the exact NG. Let us consider $u \in \mathbb{R}^p$, and

$$\zeta_u = (F + \lambda I_p)^{-1}u = (\lambda I_p + \frac{1}{m}JJ^T)^{-1}u. \tag{4.49}$$

Applying the Sherman-Morrison-Woodbury inversion formula to right hand side of (4.49), one obtains

$$\zeta_u = \lambda^{-1}u - \lambda^{-2}m^{-1}J(I_m + \frac{1}{\lambda m}J^TJ)^{-1}J^Tu. \tag{4.50}$$

The decisive advantage of (4.50) is that unlike in (4.49) where it was necessary to solve a linear system of size p (the number of parameters of the model), now we just need solve a system of size m (batch size). This new formulation makes the implementation of the exact NG possible, even for very deep networks, since the batch size m is independent of the model size and is usually small (up to a few thousand).

From (4.50), the computation of ζ_u can be performed in three main steps:

1. Computation of matrix-vector product $J^Tu \in \mathbb{R}^m$.
2. Computation of the matrix $J^TJ \in \mathbb{R}^{m\times m}$.
3. computation of matrix-vector product $Jv \in \mathbb{R}^p$, for a vector $v \in \mathbb{R}^m$.

Steps $1-3$ can be performed in an economical way, without explicitly forming J or J^T. The reader is referred to appendix 4.B for details about these steps.

Block-diagonal NG. In the case of block-diagonal NG, we independently apply the natural gradient algorithm to each layer. Concretely, for any layer $i \in [\![1;\ell]\!]$, we update the weights associated to that layer through the equation

$$(\theta_{[i]})_{k+1} = (\theta_{[i]})_k - \alpha_k[F_\bullet((\theta_{[i]})_k)]^{-1}\nabla_\theta h_{[i]}, \tag{4.51}$$

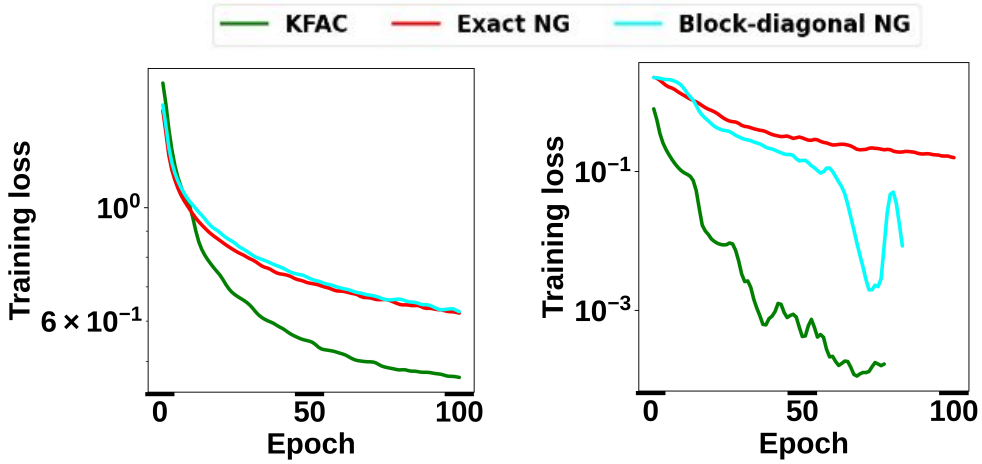
with $\theta_{[i]} = \mathrm{vec}(W_i)$, $F_\bullet((\theta_{[i]})_k) = F_{i,i} + \lambda I_{p_i}$ and $\nabla_\theta h_{[i]} = \nabla_{(\theta_{[i]})_k}h$ is the part of the gradient corresponding to layer i. As in the case of the exact NG, we need to estimate the negative increment with a batch $\mathcal{S}$ of samples, namely

$$\zeta_i = (F_{i,i} + \lambda I_{p_i})^{-1}\nabla_\theta h_{[i]} = (\lambda I_{p_i} + \frac{1}{m}J_iJ_i^T)^{-1}\nabla_\theta h_{[i]}, \tag{4.52}$$

where

$$J_i = \big(\mathrm{vec}(\mathcal{D}W_i)^{(1)}, \mathrm{vec}(\mathcal{D}W_i)^{(2)}, \dots, \mathrm{vec}(\mathcal{D}W_i)^{(m)}\big) \in \mathbb{R}^{p_i\times m}$$

is the matrix containing the batch of gradients related to layer i. Comparing (4.52) to (4.49), it is obvious that the rest of computations are performed exactly in the same way as in the case of exact NG.



(a) Cuda-Convnet trained on CIFAR10 with a batch size equal to 256.

(b) VGG 11 trained on SVHN with a batch size equal to 512.

Figure 4.5: Optimization performance evaluation Exact NG, block-diagonal NG and KFAC on two different CNNs.

Experiments. To empirically compare the optimization performances of exact NG and block-diagonal NG against KFAC, we consider the three deep auto-encoder problems. We also consider two different CNNs optimization problems, namely Cuda-convnet trained with CIFAR10 and VGG 11 trained with SVHN dataset.

Figures 4.6 and 4.5 show the obtained results for the three deep auto-encoder problems and CNNs respectively. The first observation is that in all cases, as stated by Benzig [14], KFAC performs better than exact NG and block-diagonal NG. The second observation is that, in general, block-diagonal NG performs slightly better than exact NG. The fact that KFAC and block-diagonal NG outperform exact NG in terms of optimization performance suggests that the layers of the network interact negatively with each other and this hurts the performance of the resulting optimizer. This could in fact be an empirical explanation for the fact that two-level methods designed in §4.2 do not improve KFAC.

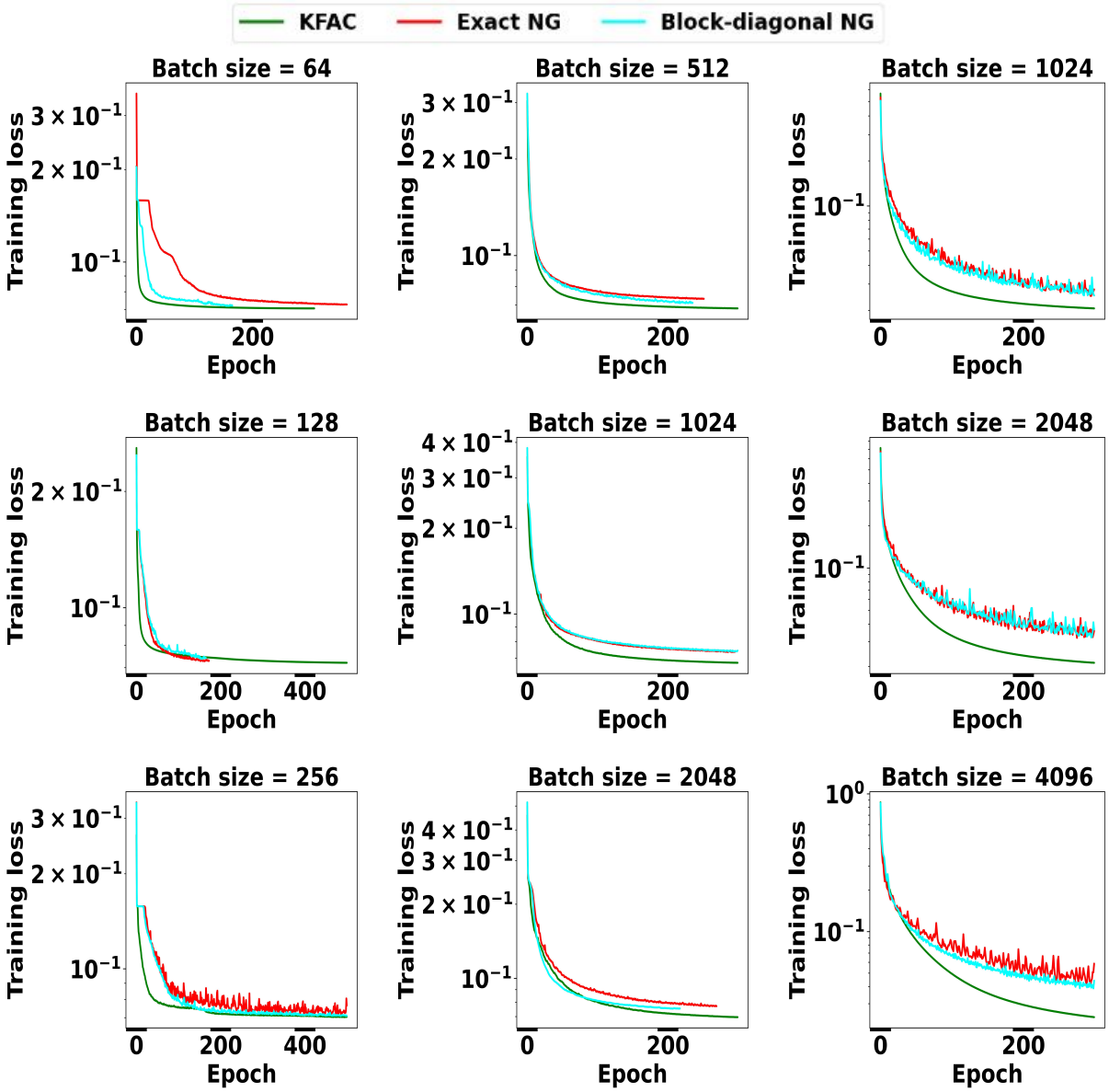
As for the fact that KFAC is more efficient than the exact block-diagonal, this remains puzzling since both (KFAC and exact block-diagonal) ignore extra-diagonal blocks and KFAC is an approximation of the exact block-diagonal. To understand this unexpected observation, we propose to further investigate the comparison between KFAC and block-diagonal NG. We find that these two methods do not use the same regularization technique. This leads us to study the impact of regularization in the next subsection.

4.4.2 Impact of regularization

Here, we seek to understand why KFAC outperforms block-diagonal NG in terms of optimization performance. Since KFAC uses a heuristic regularization technique that is different from the normal regularization used for block-diagonal NG, we propose to study the impact of this type of regularization on KFAC.

In the KFAC method (see §3.1.2), each block is regularized as

$$[F_{\bullet \text{KFAC}}]_{i,i} = (\bar{A}_{i-1} + \lambda_{\bar{A}_{i-1}} I) \otimes (G_i + \lambda_{G_i} I). \tag{4.53}$$



(a) CURVES (b) MNIST (c) FACES

Figure 4.6: Comparison of Exact NG and block-diagonal NG against KFAC on the three deep auto-encoder problems (CURVES **first** column, MNIST **middle** column and FACES **last** column). Three different batch sizes are considered for each problem (each row corresponds to a different batch size).

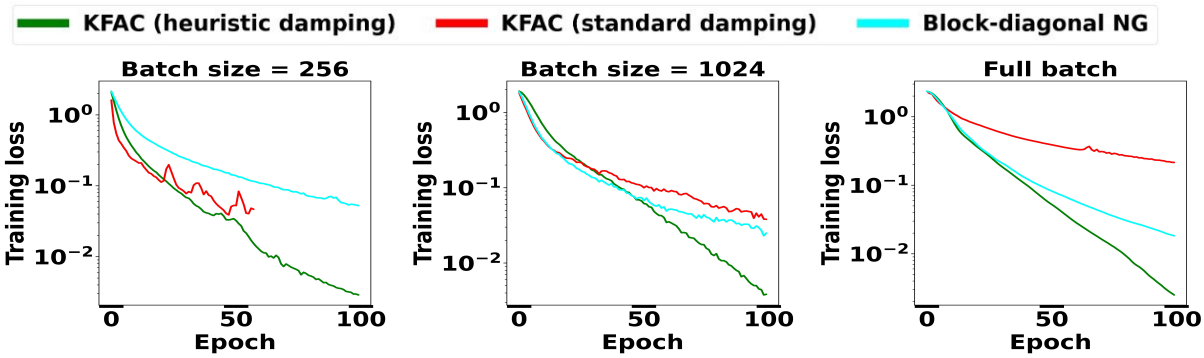


Figure 4.7: Evaluation of the impact of damping techniques on KFAC with an MLP optimization problem.

This regularization technique is purely heuristic (no theoretical basis) and is solely motivated by computation purposes. In the case of the classical damping technique, each block of KFAC should be regularized as follows

$$[F_{\bullet\,\mathrm{KFAC}}]_{i,i} = \bar{A}_{i-1} \otimes G_i + \lambda I_{p_i}. \tag{4.54}$$

The heuristic technique used in KFAC therefore adds the following two additional terms

$$\lambda_{\bar{A}_{i-1}} I \otimes G_i \quad \text{and} \quad \bar{A}_{i-1} \otimes \lambda_{G_i} I$$

to the original curvature matrix. To evaluate whether these additional terms explain the success of KFAC over block-diagonal NG, we implement KFAC with normal damping and compare its performance against block-diagonal NG. To do so, we consider the optimization of two toy networks, namely a shallow MLP of 5 layers trained with a subset of 5000 images from MNIST dataset and a CNN containing 2 convolution layers and 1 MLP layer trained with subset of 1000 images from CIFAR10 dataset. For each network, we consider two different batch sizes and the case of full batch (i.e. deterministic optimization).

Figures 4.7 and 4.8 illustrate the results obtained for the MLP and CNN respectively. We observe that in each problem, regardless of the size of the batch, KFAC with heuristic damping outperforms significantly block-diagonal NG, which in turn does better than KFAC with normal damping. These results explain that the success of KFAC over block-diagonal NG relies on the heuristic damping technique used for it. This is not surprising because both KFAC and block-diagonal NG rely on the block-diagonal approximation of the FIM and KFAC is a further approximation of block-diagonal NG. And therefore, the success of KFAC compared with block-diagonal NG could only be attributed to the regularization technique used.

These findings suggest regularizing KFAC-type methods using heuristic damping instead of normal damping.

4.5 Conclusion

In this study, we sought to improve KFAC by incorporating extra-diagonal blocks using two-level decomposition methods. To this end, we proposed several two-level KFAC methods, carefully designing coarse corrections. Through several experiments, we came to the conclusion that

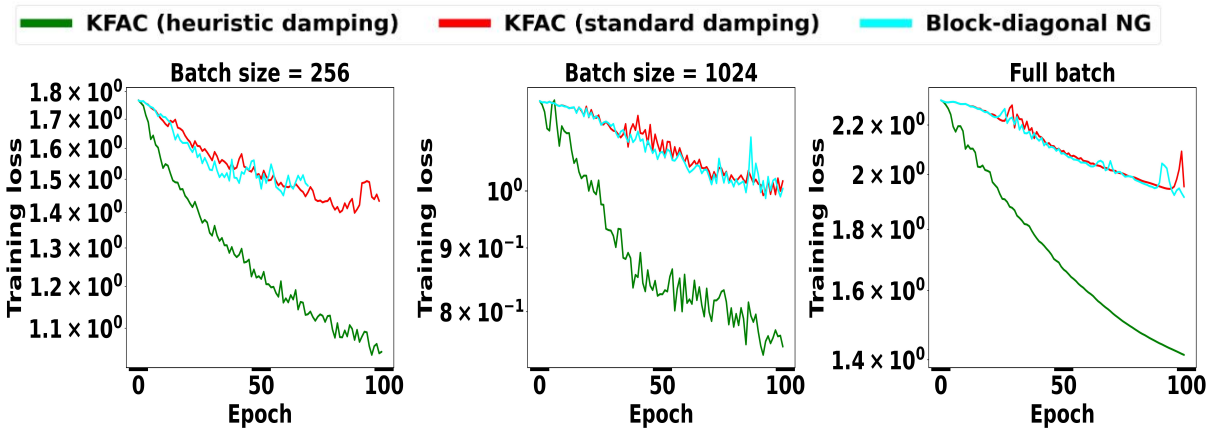


Figure 4.8: Evaluation of the impact of damping techniques on KFAC with a CNN optimization problem.

two-level KFAC methods do not generally outperform the original KFAC method in terms of optimization performance of the objective function. This implies that taking into account the interactions between the layers is not useful for the optimization process.

We also numerically verified that, at the level of the linear system of each iteration, the increment provided by any two-level method is much closer to the exact natural gradient solution than that obtained with KFAC, in a norm naturally associated with the FIM. This reveals that closeness to the exact natural gradient does not necessarily results in a more efficient algorithm. This observation is consistent with Benzing's previous claim [14] (corroborated by our experiments in §4.4) that KFAC outperforms the exact natural gradient in terms of optimization performance.

The fact that incorporating extra-diagonal blocks does not improve or often even hurts the optimization performance of the initial diagonal approximation could be explained by a negative interaction between different layers of the neural network. This suggests ignoring extra-diagonal blocks of the FIM and keeping the block-diagonal approximation, and if one seeks to improve the block-diagonal approximation, one should focus on diagonal blocks as attempted in chapter 3 and in many recent works [17, 54, 56].

It is worth pointing out that the conclusion of Tpselepedis et al. [167] on the performance of their proposed two-level method seems a little hasty. Indeed, the authors only ran two different experiments: the optimization of a CNN and a simple linear network. For the CNN network, they did not observe any improvement. For the linear network, they obtain some improvement in the optimization performance. Their conclusion is therefore based on this single observation.

4.A Efficient computation of $F_\bullet u$ and F_{coarse}

4.A.1 Notations

We consider the same network architectures (MLP and CNN) and notations introduced in chapter 1. For two matrices A and B of same sizes, $A \odot B$ denotes the Hadamard (element-wise) product of A and B. We will also write $\langle u, v \rangle$ for the inner (dot) product between vectors u and v. We recall that "vec" is the operator that turns a matrix into a vector by stacking its columns together and "MAT," the converse of "vec," turns a vector into a matrix.

For a big vector $u \in \mathbb{R}^p$, where p is the number of parameters contained in θ, the notation $u[i] \in \mathbb{R}^{p_i}$ stands for the part of u corresponding to layer i, whose number of parameters is p_i. For example, if $u = \theta = [\text{vec}(W_1)^T, \text{vec}(W_2)^T, \dots, \text{vec}(W_\ell)^T]^T$, then $u[i] = \text{vec}(W_i)$ for all $i \in \{1, \dots, \ell\}$.

Finally, We recall that m is the number of data points in a mini-batch $\mathcal{S}$.

4.A.2 Computation of $F_\bullet u$

In view of the regularization (4.2), it is plain that

$$F_\bullet u = Fu + \lambda u. \tag{4.55}$$

Since the regularization term does not cause any trouble, we only have to deal with Fu. It is notoriously knwon that the matrix-vector product involving the Fisher matrix can be carried out without explicitly forming F thanks to an algorithm by Schraudolph [152]. However, this approach requires to perform additional forward/backward passes. Here, we present an efficient way to evaluate Fu by re-using the quantities computed during the traditional backward/forward pass.

Suppose that we have a mini-batch $\mathcal{S} = \{(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})\}$ sampled from the training set $\mathcal{D}$ where targets y's are sampled from the model predictive distribution $P_{y|x}(\theta)$. Then, F is computed using a Monte Carlo estimation, i.e.,

$$F = [\mathcal{D}\theta(\mathcal{D}\theta)^T] \approx \frac{1}{m} \sum_{b=1}^{m} (\mathcal{D}\theta)^{(b)} ((\mathcal{D}\theta)^{(b)})^T = \frac{1}{m} J J^T, \tag{4.56}$$

where $J \in \mathbb{R}^{p \times m}$ is the matrix whose b-th column is $(\mathcal{D}\theta)^{(b)}$. Thus,

$$Fu = \frac{1}{m} J J^T u. \tag{4.57}$$

The computation of Fu can therefore be divided into two steps: (1) matrix-vector product $v = J^T u$; (2) matrix-vector product $\frac{1}{m} J v$.

Step 1: multiplying by J^T. Since $J^T \in \mathbb{R}^{m \times p}$ and $u \in \mathbb{R}^p$, we have $v = J^T u \in \mathbb{R}^m$. For all $b \in \{1, \dots, m\}$, the b-th entry v_b of v is none other than the dot product between the b-th column $(\mathcal{D}\theta)^{(b)}$ of J and u. This dot product can be split into layer-wise dot products and then summed up together. Formally, we have

$$v_b = \left\langle (\mathcal{D}\theta)^{(b)}, u \right\rangle = \sum_{i=1}^{\ell} \left\langle \text{vec}((\mathcal{D}W_i)^{(b)}), u[i] \right\rangle. \tag{4.58}$$

We now distinguish two cases according to the type of layer i.

1. If layer i is an MLP, we have $\mathcal{D}W_i = g_i \bar{a}_{i-1}^T$ and then

$$\begin{aligned}\left\langle \mathrm{vec}((\mathcal{D}W_i)^{(b)}), u_{[i]} \right\rangle &= \left\langle \mathrm{vec}(g_i^{(b)}(\bar{a}_{i-1}^{(b)})^T), u_{[i]} \right\rangle \\ &= \left\langle \bar{a}_{i-1}^{(b)} \otimes g_i^{(b)}, u_{[i]} \right\rangle \\ &= \left((\bar{a}_{i-1}^{(b)})^T \otimes (g_i^{(b)})^T\right) u_{[i]} \\ &= (g_i^{(b)})^T \mathrm{MAT}(u_{[i]}) \bar{a}_{i-1}^{(b)}. \end{aligned} \tag{4.59}$$

We obtain thus in matrix form

$$J^T u = \sum_{i=1}^{\ell} \mathrm{diag}\big(\hat{\mathcal{G}}_i^T \mathrm{MAT}(u_{[i]}) \hat{\mathcal{A}}_{i-1}\big),$$

where

$$\hat{\mathcal{A}}_{i-1} = (\bar{a}_{i-1}^{(1)}, \ldots, \bar{a}_{i-1}^{(m)}) \in \mathbb{R}^{d_{i-1} \times m} \tag{4.60}$$

is the matrix of activations, and

$$\hat{\mathcal{G}}_i = (g_i^{(1)}, \ldots, g_i^{(m)}) \in \mathbb{R}^{d_i \times m} \tag{4.61}$$

is the matrix containing pre-activations derivatives.

2. If layer i is a CNN layer, we have $\mathcal{D}W_i = \sum_{t \in \mathcal{T}_i} g_{i,t} \bar{a}_{i-1,t}^T$ and therefore

$$\begin{aligned}\left\langle \mathrm{vec}((\mathcal{D}W_i)^{(b)}), u_{[i]} \right\rangle &= \left\langle \mathrm{vec}\left(\sum_{t \in \mathcal{T}_i} g_{i,t}^{(b)} (\bar{a}_{i-1,t}^{(b)})^T \right), u_{[i]} \right\rangle \\ &= \left\langle \sum_{t \in \mathcal{T}_i} \mathrm{vec}(g_{i,t}^{(b)} (\bar{a}_{i-1,t}^{(b)})^T), u_{[i]} \right\rangle \\ &= \sum_{t \in \mathcal{T}_i} (g_{i,t}^{(b)})^T \mathrm{MAT}(u_{[i]}) \bar{a}_{i-1,t}^{(b)}. \end{aligned} \tag{4.62}$$

We therefore obtain

$$J^T u = \sum_{i=1}^{\ell} \sum_{t \in \mathcal{T}_i} \mathrm{diag}\big(\hat{\mathcal{G}}_{i,t}^T \mathrm{MAT}(u_{[i]}) \hat{\mathcal{A}}_{i-1,t}\big),$$

where for a spatial position $t \in \mathcal{T}_i$,

$$\hat{\mathcal{A}}_{i-1,t} = (\bar{a}_{i-1,t}^{(1)}, \ldots, \bar{a}_{i-1,t}^{(m)}) \in \mathbb{R}^{d_{i-1} \times m} \tag{4.63}$$

is the matrix of activations corresponding to t, and

$$\hat{\mathcal{G}}_{i,t} = (g_{i,t}^{(1)}, \ldots, g_{i,t}^{(m)}) \in \mathbb{R}^{d_i \times m} \tag{4.64}$$

is the matrix containing pre-activations derivatives corresponding to t.

Step 2: multiplying by J. Here, we detail how to compute Jv, but one should not forget to multiply the result by the scaling factor $1/m$. Let $v \in \mathbb{R}^m$. Jv is a weighted sum of the columns of J with the weight coefficients corresponding to the entries of v. In other words,

$$Jv = \sum_{b=1}^{m} v_b (\mathcal{D}\theta)^{(b)}. \tag{4.65}$$

For all $i \in \{1, \dots, \ell\}$, the part of Jv corresponding to layer i is a linear combination of those of columns of J corresponding to layer i, that is,

$$(Jv)[i] = \sum_{b=1}^{m} v_b \operatorname{vec}((\mathcal{D}W_i)^{(b)}). \tag{4.66}$$

As in step 1, we have to distinguish two cases according the type of layer i.

1. If layer i is an MLP, we have

$$(Jv)[i] = \sum_{b=1}^{m} v_b \operatorname{vec}(g_i^{(b)} (\bar{a}_{i-1}^{(b)})^T) = \operatorname{vec}\Big(\sum_{b=1}^{m} v_b g_i^{(b)} (\bar{a}_{i-1}^{(b)})^T \Big), \tag{4.67}$$

and then

$$\operatorname{MAT}\big((Jv)_{[i]}\big) = \sum_{b=1}^{m} v_b g_i^{(b)} (\bar{a}_{i-1}^{(b)})^T = \big[(\mathbb{1} v^T) \odot \hat{\mathcal{G}}_i\big] \hat{\mathcal{A}}_{i-1}^T, \tag{4.68}$$

where $\mathbb{1} \in \mathbb{R}^{d_i}$ is a vector of all one's, $\hat{\mathcal{A}}_{i-1}$ and $\hat{\mathcal{G}}_i$ are matrices defined respectively in (4.60) and (4.61). Note that the last equality in (4.68) is due to the fact that for two matrices $A = [A_1, \dots, A_m]$ and $B = [B_1, \dots, B_m]$, we have

$$AB^T = \sum_k A_k B_k^T. \tag{4.69}$$

2. If layer i is a CNN, we have

$$\begin{aligned} (Jv)_{[i]} &= \sum_{b=1}^{m} v_b \operatorname{vec}\Big(\sum_{t \in \mathcal{T}_i} g_{i,t}^{(b)} (\bar{a}_{i-1,t}^{(b)})^T \Big) \\ &= \sum_{b=1}^{m} \operatorname{vec}\Big(\sum_{t \in \mathcal{T}_i} v_b g_{i,t}^{(b)} (\bar{a}_{i-1,t}^{(b)})^T \Big), \end{aligned} \tag{4.70}$$

and then

$$\begin{aligned} \operatorname{MAT}\big((Jv)_{[i]}\big) &= \sum_{b=1}^{m} v_b \sum_{t \in \mathcal{T}_i} g_{i,t}^{(b)} (\bar{a}_{i-1,t}^{(b)})^T \\ &= \big[(\mathbb{1} V^T) \odot \hat{\mathcal{G}}_i\big] [\![\hat{\mathcal{A}}_{i-1}]\!]^T, \end{aligned} \tag{4.71}$$

where here $\mathbb{1}$ is a vector of all one's of size c_i (the number of output channels), whereas

$$V = (v_1, \dots v_1 | \dots, | v, \dots v_m)^T \tag{4.72a}$$

$$[\![\hat{\mathcal{A}}_{i-1}]\!] = \big(\bar{a}_{i-1,1}^{(1)}, \dots, \bar{a}_{i-1,|\mathcal{T}_i|}^{(1)} | \bar{a}_{i-1,1}^{(2)}, \dots, \bar{a}_{i-1,|\mathcal{T}_i|}^{(2)} | \dots | \bar{a}_{i-1,1}^{(m)}, \dots, \bar{a}_{i-1,|\mathcal{T}_i|}^{(m)}\big), \tag{4.72b}$$

$$\hat{\mathcal{G}}_i = \big(g_{i,1}^{(1)}, \dots, g_{i,|\mathcal{T}_i|}^{(1)} | g_{i,1}^{(2)}, \dots, g_{i,|\mathcal{T}_i|}^{(2)} | \dots | g_{i,1}^{(m)}, \dots, g_{i,|\mathcal{T}_i|}^{(m)}\big), \tag{4.72c}$$

are respectively in $\mathbb{R}^{|\mathcal{T}_i| m}$ (a duplicated version of v), $\mathbb{R}^{(c_{i-1}\Delta_i + 1) \times |\mathcal{T}_i| m}$ and $\mathbb{R}^{c_i \times |\mathcal{T}_i| m}$.

4.A.3 Computation of F_{coarse}

From definition (4.14) of the coarse operator and the a priori form (4.23) of the coarse space, it follows that

$$[F_{\text{coarse}}]_{i,j} = V_i^T [F_\bullet]_{i,j} V_j \tag{4.73}$$

for all $(i,j) \in \{1,\dots,\ell\} \times \{1,\dots,\ell\}$. In view of the regularization (4.2), the entry (4.73) becomes

$$[F_{\text{coarse}}]_{i,j} = \begin{cases} V_i^T F_{i,j} V_j & \text{if } i \neq j, \\ V_i^T F_{i,i} V_i + \lambda V_i^T V_i & \text{if } i = j. \end{cases} \tag{4.74}$$

Since the regularization term $\lambda V_i^T V_i$ does not cause any trouble, we only have to deal with the elementary products $v^T F_{i,j} w$, where $v \in \mathbb{R}^{p_i}$ (a column of V_i) and $w \in \mathbb{R}^{p_j}$ (a column of V_j). We reall that $F_{i,j} \in \mathbb{R}^{p_i \times p_j}$ is computed using a Monte Carlo estimation on a mini-batch, i.e.,

$$F_{i,j} \approx \frac{1}{m} \sum_{b=1}^{m} \text{vec}(\mathcal{D}W_i)^{(b)} (\text{vec}(\mathcal{D}W_j)^{(b)})^T = \frac{1}{m} J_i J_j^T, \tag{4.75}$$

with

$$J_i = \left(\text{vec}(\mathcal{D}W_i)^{(1)}, \text{vec}(\mathcal{D}W_i)^{(2)}, \dots, \text{vec}(\mathcal{D}W_i)^{(m)}\right) \in \mathbb{R}^{p_i \times m}, \tag{4.76a}$$

$$J_j = \left(\text{vec}(\mathcal{D}W_j)^{(1)}, \text{vec}(\mathcal{D}W_j)^{(2)}, \dots, \text{vec}(\mathcal{D}W_j)^{(m)}\right) \in \mathbb{R}^{p_j \times m}. \tag{4.76b}$$

Then, $v^T F_{i,j} w$ is given by

$$v^T F_{i,j} w = \frac{1}{m} v^T J_i J_j^T w. \tag{4.77}$$

The computation of $v^T F_{i,j} w$ can therefore be performed in three steps: (1) matrix-vector product $\mu = J_j^T w$; (2) matrix-vector product $\gamma = J_i \mu$; (3) dot product $\frac{1}{m}\langle v, \gamma \rangle$. The computation of $\mu = J_j^T v$ is done in the same way as $J^T u$ in the previous subsection §4.A.2. The only difference is that here we do not sum over layers. Likewise, the computation of $\gamma = J_i \mu$ is done exactly in the same way as $(Jv)[i]$ in §4.A.2. Finally, step 3 is the classical dot product and is straightforward.

4.B Details about the computation of exact natural gradient

Here, we give details on the calculation of $J^T u$, $J^T J$ and Jv from §4.4.1. The computations of $J^T u$ and Jv are already detailed in §4.A.2. As for $J^T J$, let $(b_1, b_2) \in [\![1; m]\!]^2$, then it is easy to notice that the (b_1, b_2) entry of $J^T J$ is given by the dot product between the b_1^{th} column and b_2^{th} column of J. Formally,

$$\begin{aligned} 2.)\ \forall (b_1, b_2) \in [\![1; m]\!]^2, (J^T J)_{b_1, b_2} &= \left\langle (\mathcal{D}\theta)^{(b_1)}, (\mathcal{D}\theta)^{(b_2)} \right\rangle \\ &= \sum_{i=1}^{\ell} \left\langle \text{vec}((\mathcal{D}W_i)^{(b_1)}), \text{vec}((\mathcal{D}W_i)^{(b_2)}) \right\rangle \\ &= \sum_{i=1}^{\ell} \mu_i, \end{aligned}$$

with $\mu_i = \left\langle \text{vec}((\mathcal{D}W_i)^{(b_1)}), \text{vec}((\mathcal{D}W_i)^{(b_2)}) \right\rangle$. For the computation of μ_i, we need to distinguish to cases according the type of layer i.

1. If layer i is an MLP, we have

$$\begin{aligned}\mu_i =&\big\langle \operatorname{vec}(g_i^{(b_1)}(\bar{a}_{i-1}^{(b_1)})^T), \operatorname{vec}(g_i^{(b_2)}(\bar{a}_{i-1}^{(b_2)})^T)\big\rangle\\ =&\big(\operatorname{vec}(g_i^{(b_1)}(\bar{a}_{i-1}^{(b_1)})^T)\big)^T \operatorname{vec}(g_i^{(b_2)}(\bar{a}_{i-1}^{(b_2)})^T)\\ =&\big((\bar{a}_{i-1}^{(b_1)})^T \otimes (g_i^{(b_1)})^T\big)\big(\bar{a}_{i-1}^{(b_2)} \otimes g_i^{(b_2)}\big)\\ =&[(\bar{a}_{i-1}^{(b_1)})^T \bar{a}_{i-1}^{(b_2)}][(g_i^{(b_1)})^T g_i^{(b_2)}].\end{aligned}$$

   Thus in matrix form, we obtain

$$J^T J = \sum_{i=1}^{\ell} (\hat{\mathcal{A}}_{i-1}^T \hat{\mathcal{A}}_{i-1}) \odot (\hat{\mathcal{G}}_i^T \hat{\mathcal{G}}_i), \tag{4.78}$$

   where $\hat{\mathcal{A}}_{i-1}$ is the matrix of activations defined by (4.60), and $\hat{\mathcal{G}}_i$ is the matrix containing pre-activations derivatives defined by equation (4.61).

2. If layer i is a convolution layer, we have

$$\begin{aligned}\mu_i =&\big\langle \operatorname{vec}\big(\sum_{t\in\mathcal{T}_i} g_{i,t}^{(b_1)}(\bar{a}_{i-1,t}^{(b_1)})^T\big), \operatorname{vec}\big(\sum_{t\in\mathcal{T}_i}^{T_i} g_{i,t}^{(b_2)}(\bar{a}_{i-1,t}^{(b_2)})^T\big)\big\rangle\\ =&\sum_{t\in\mathcal{T}_i}\sum_{t'\in\mathcal{T}_i}\big[(\bar{a}_{i-1,t}^{(b_1)})^T \bar{a}_{i-1,t'}^{(b_2)}\big]\big[(g_{i,t}^{(b_1)})^T g_{i,t'}^{(b_2)}\big].\end{aligned}$$

   Therefore, we obtain

$$J^T J = \sum_{i=1}^{\ell}\sum_{t\in\mathcal{T}_i}\sum_{t'\in\mathcal{T}_i} (\hat{\mathcal{A}}_{i-1,t}^T \hat{\mathcal{A}}_{i-1,t'}) \odot (\hat{\mathcal{G}}_{i,t}^T \hat{\mathcal{G}}_{i,t'}), \tag{4.79}$$

   with $\hat{\mathcal{A}}_{i-1,t}$ and $\hat{\mathcal{G}}_{i,t}$ are defined according to (4.63) and (4.64) respectively.

Chapter 5

An empirical analysis of generative adversarial networks training with curvature information

Contents

This chapter is concerned with the application of the KFAC method to the training of generative adversarial networks (GANs). Our goal is to empirically analyze the impact of incorporating curvature information on convergence speed, stability and overall performance for various datasets. We consider only the basic KFAC approximation, without any of the enhancements suggested in §3–§4.

The mathematical framework of GANs is sketched out in §5.1, at the end of which we describe the DCGAN (deep convolutional GAN) architecture that will be of interest to us. The latter involves the crucial notion of transposed convolution layers, which we shall elaborate on in §5.2 and to which we shall extend to KFAC approximation in §5.2.3. Finally, numerical tests are provided in §5.3 along with observations on the effectiveness of the proposed method.

5.1 Generative adversarial networks

5.1.1 Continuous framework

In less than a decade, *generative adversarial networks* (GANs) have established themselves as an elegant and powerful paradigm for generating synthetic data achieving a very high degree of "resemblance" to real samples. Initially proposed by Goodfellow [64, 66] and improved by

many subsequent authors [6, 70, 114, 134, 188, 192], GANs have met with tremendous success in many applications such as image-to-image translation [1, 87], image in-painting [81, 184], image super-resolution [102, 175], text-to-image generation [182, 189] and many more. Mention should be made that GANs are part of a wider family of generative models including *variational autoencoders* [89] and *autoregressive models* [168, 169].

Let $\mathscr{X}$ be the (finite-dimensional) space of real data samples. These objects have some degree of consistency, insofar as they are generated as samples from a common probability distribution P_r defined over $\mathscr{X}$. We assume that P_r has a density function $\mathsf{p}_r(x)$. We do not know either P_r or $\mathsf{p}_r(x)$. The only information available to us is the data samples. This is why we would like to find a probability distribution P_G with density $\mathsf{p}_G(x)$, also defined on $\mathscr{X}$, to approximate P_r and $\mathsf{p}_r(x)$. Once this is done, we can take samples from the distribution P_G to generate other "similar" objects in $\mathscr{X}$.

To build P_G, we can start from an initial probability distribution P_z, with density $\mathsf{p}_z(z)$, defined on another (finite-dimensional) space $\mathscr{Z}$ whose dimension may not be the same as that of $\mathscr{X}$. This initial distribution P_z governs the random noise that will be later used to create samples. For instance, we can set p_z to be the standard normal distribution, although other choices are possible. The idea of GANs is to determine a mapping $G : \mathscr{Z} \to \mathscr{X}$, called *generator*, such that if a random variable Z has distribution P_z, then the random variable $G(Z)$ has distribution P_G. In other words, $\mathsf{P}_G = \mathsf{P}_z \circ G^{-1}$.

Vanilla GAN. For P_G to satisfactorily mimick P_r, the vanilla GAN sets up an adversarial system from which G receives feedback to improve itself. More specifically, in addition to G, it introduces another function $D : \mathscr{X} \to [0, 1]$, called *discriminator*, whose role is

- to demote the samples generated by G by assigning a low score to objects of the form $G(z)$; thus $D(G(z))$ must be close to 0;
- to promote the samples coming from P_r by assigning a high score to them; thus, $D(x)$ must be close to 1.

Simultaneously, the generator G tries to fool the discriminator D by maximizing $D(G(z))$. One approach to attaining all these goals, depicted in Figure 5.1, is to solve the minimax problem

$$\min_G \max_D \mathbb{E}_{x\sim\mathsf{P}_r}\big[\log\big(D(x)\big)\big] + \mathbb{E}_{z\sim\mathsf{P}_z}\big[\log\big(1 - D(G(z))\big)\big] \tag{5.1}$$

with respect to all functions (G, D). Plainly, the minimax problem (5.1) can be reformulated as

$$\min_{\mathsf{P}_G} \max_D \mathbb{E}_{x\sim\mathsf{P}_r}\big[\log\big(D(x)\big)\big] + \mathbb{E}_{x\sim\mathsf{P}_G}\big[\log\big(1 - D(x)\big)\big], \tag{5.2}$$

the interest of which is to work directly with the distribution $\mathsf{P}_G \in \{\mathsf{P}_z \circ G^{-1},\ G : \mathscr{Z} \to \mathscr{X}\}$. The following result sheds new light on (5.2) and paves the way for its generalization.

Theorem 5.1 (Goodfellow et al. [66])**.** *The minimax problem* (5.2) *of the vanilla GAN has the same minimizers as*

$$\min_{\mathsf{P}_G} \mathrm{JS}\big[\mathsf{P}_r \,\|\, \mathsf{P}_G\big], \tag{5.3}$$

where the Jensen-Shannon divergence

$$\mathrm{JS}\big[\mathsf{P} \,\|\, \mathsf{Q}\big] = \frac{1}{2}\mathrm{KL}\left[\mathsf{P} \,\middle\|\, \frac{\mathsf{P}+\mathsf{Q}}{2}\right] + \frac{1}{2}\mathrm{KL}\left[\mathsf{Q} \,\middle\|\, \frac{\mathsf{P}+\mathsf{Q}}{2}\right] \tag{5.4}$$

is a symmetrized version of the Kullback-Leibler divergence (2.47).

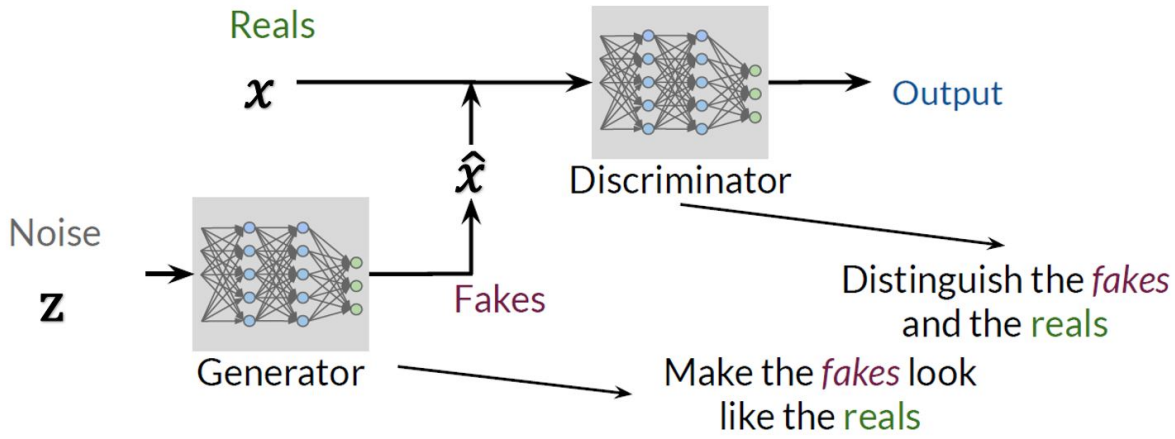


Figure 5.1: GAN diagram.

Proof. Let us write down the objective function of (5.2) as

$$V(\mathsf{P}_G, D) := \int_{\mathscr{X}} \{\mathsf{p}_r(x) \log(D(x)) + \mathsf{p}_G(x) \log(1 - D(x))\} \, \mathrm{d}x. \tag{5.5}$$

For any $(p, q) \in \mathbb{R}^2 \backslash (0, 0)$, the function $d \mapsto p \log d + q \log(1 - d)$ reaches its maximum over $[0, 1]$ at $p/(p + q)$. Since the discriminator does not need to be defined when both p_r and p_G vanish, we can consider

$$D^*[\mathsf{P}_G](x) = \frac{\mathsf{p}_r(x)}{\mathsf{p}_r(x) + \mathsf{p}_G(x)} \tag{5.6}$$

and see that it achieves the maximum with respect to D of V at fixed P_G. The maximal value

$$V(\mathsf{P}_G, D^*[\mathsf{P}_G]) = \sup_D V(\mathsf{P}_G, D) \tag{5.7}$$

can be evaluated as

$$V^*(\mathsf{P}_G) := \int_{\mathscr{X}} \left\{ \mathsf{p}_r(x) \log \frac{\mathsf{p}_r(x)}{\mathsf{p}_r(x) + \mathsf{p}_G(x)} + \mathsf{p}_G(x) \log \frac{\mathsf{p}_G(x)}{\mathsf{p}_r(x) + \mathsf{p}_G(x)} \right\} \mathrm{d}x \tag{5.8a}$$

$$= \mathrm{KL}\left[\mathsf{P}_r \,\middle\|\, \frac{\mathsf{P}_r + \mathsf{P}_G}{2} \right] - \log \frac{1}{2} + \mathrm{KL}\left[\mathsf{P}_G \,\middle\|\, \frac{\mathsf{P}_r + \mathsf{P}_G}{2} \right] - \log \frac{1}{2} \tag{5.8b}$$

$$= 2\,\mathrm{JS}\big[\mathsf{P}_r \,\|\, \mathsf{P}_G\big] - 2 \log \frac{1}{2}. \tag{5.8c}$$

This implies that

$$\arg\min_{\mathsf{P}_G} V^*(\mathsf{P}_G) = \arg\min_{\mathsf{P}_G} \mathrm{JS}[\mathsf{P}_r \,\|\, \mathsf{P}_G], \tag{5.9}$$

hence the desired equivalence. □

Theorem 5.1 certifies the appropriateness of formulation (5.2). Indeed, let us relax the constraint $\mathsf{P}_G = P_z \circ G^{-1}$ and assume that P_G can be any probability distribution over $\mathscr{X}$. Then, the solution of (5.3) is obviously $\mathsf{P}_G = \mathsf{P}_r$, because $\mathrm{JS}[\mathsf{P} \,\|\, \mathsf{Q}] \geqslant 0$ for all pair of probability

distributions (P, Q) and equality holds if and only if $\mathsf{P} = \mathsf{Q}$. This is exactly what we have set out to accomplish (except for the fact that since we do not know P_r, this solution has a purely theoretical value). Inserting then $\mathsf{p}_G = \mathsf{p}_r$ into (5.6), we end up with the optimal discriminator

$$D(x) = \frac{1}{2} \qquad \text{for all} \quad x \in \operatorname{supp} \mathsf{p}_G. \tag{5.10}$$

Put another way, we have reached a Nash equilibrium [137] at which the discriminator D is no longer able to tell apart generated samples from training ones.

f-GAN. As a straightforward generalization of the vanilla GAN, we can replace the Jensen-Shannon divergence in (5.4) by another notion of divergence between probability distributions. Let f be a strictly convex with domain $I \subset \mathbb{R}$ such that $f(1) = 0$. The *f-divergence* [2] between two probability distributions (P, Q) on $\mathscr{X}$ with densities (p, q) is defined as

$$\operatorname{div}_f[\mathsf{P} \,\|\, \mathsf{Q}] = \mathbb{E}_{x\sim\mathsf{Q}}\Big[f\Big(\frac{\mathsf{p}}{\mathsf{q}}\Big)\Big] = \int_{\mathscr{X}} f\Big(\frac{\mathsf{p}(x)}{\mathsf{q}(x)}\Big)\mathsf{q}(x)\,\mathrm{d}x, \tag{5.11}$$

where we agree on the convention that the integrand in the integral of the last equality (5.11) is zero whenever $\mathsf{q}(x) = 0$. The Kullback-Leibler and Jensen-Shannon divergences are both special cases of the f-divergence [176]. Given an f-divergence, the f-GAN problem is defined as

$$\min_{\mathsf{P}_G} \operatorname{div}_f[\mathsf{P}_r \,\|\, \mathsf{P}_G]. \tag{5.12}$$

Again, if the implicit form $\mathsf{P}_G = \gamma \circ G^{-1}$ were put aside, the solution of this minimization problem would be $\mathsf{P}_G = \mathsf{P}_r$, since an f-divergence is always non-negative and vanishes only when the two probability distributions are equal almost everywhere. For computational purpose, however, formulation (5.12) has no practical value. Fortunately, under some mild restrictions, we can express it in an equivalent way that is reminiscent of a minimax problem.

Theorem 5.2. *Let f be strictly convex and continuously differentiable on $I \subset \mathbb{R}$. The f-GAN problem (5.12) subject to the additional constraint $\mathsf{P}_G \gg \mathsf{P}_r$, that is, P_r be absolutely continuous with respect to P_G, has the same minimizers as*

$$\min_{\mathsf{P}_G \gg \mathsf{P}_r} \sup_T \mathbb{E}_{x\sim\mathsf{P}_r}[T(x)] - \mathbb{E}_{x\sim\mathsf{P}_G}[f^*(T(x))], \tag{5.13}$$

where f^* denotes the Legendre-Fenchel transform of f, the sup being taken over all measurable function $T : \mathscr{X} \to \operatorname{Dom}(f^*) \subset \mathbb{R}$.

Proof. Nguyen et al. [120] showed that if f meets the stated assumptions and if $\mathsf{P}_r \ll \mathsf{P}_G$, then

$$\operatorname{div}_f[\mathsf{P}_r \,\|\, \mathsf{P}_G] = \sup_T \mathbb{E}_{x\sim\mathsf{P}_r}[T(x)] - \mathbb{E}_{x\sim\mathsf{P}_G}[f^*(T(x))], \tag{5.14}$$

where the sup is taken over all measurable function $T : \mathscr{X} \to \operatorname{Dom}(f^*)$. Minimizing $\operatorname{div}_f[\mathsf{P}_r \,\|\, \mathsf{P}_G]$ over $\mathsf{P}_G \gg \mathsf{P}_r$ therefore yields the same minimizers as (5.13). □

In the f-GAN approach, a function T appearing in (5.13) is called a *critic*. With a slight abuse, it is also referred to as the *discriminator*. An alternative but similar formulation, in which the absolute continuity constraint $\mathsf{P}_r \ll \mathsf{P}_G$ is removed, gives rise to the category of *variational divergence minimization* [176].

Wasserstein GAN (WGAN). A common pitfall with vanilla GAN and f-GAN is the lack of continuity of the divergence, which causes the optimizer to stagnate. To illustrate this point, let us consider the following example, due to Arjovsky et al. [6]. Let $Z \sim \mathcal{U}((0,1))$ be the uniform distribution on the open interval $(0,1)$. For $\zeta \in \mathbb{R}$, define

$$\mathsf{P} = (0, Z), \qquad \mathsf{P}_\zeta = (\zeta, Z), \tag{5.15}$$

on $\mathbb{R}^2$. Then, P and P_ζ are singular distributions with disjoint support if $\zeta \neq 0$. It is then easy to check that

$$\mathrm{JS}[\mathsf{P} \,\|\, \mathsf{P}_\zeta] = \begin{cases} \log 2 & \text{if } \zeta \neq 0; \\ 0 & \text{if } \zeta = 0, \end{cases} \tag{5.16}$$

and

$$\mathrm{KL}[\mathsf{P} \,\|\, \mathsf{P}_\zeta] = \begin{cases} +\infty & \text{if } \zeta \neq 0; \\ 0 & \text{if } \zeta = 0, \end{cases} \tag{5.17}$$

Therefore, for a positive sequence $\zeta_n \downarrow 0$, P_{ζ_n} does not converge to P in the sense of these two divergences. If we train the vanilla GAN with an intial source distribution P_{ζ_0} with $\zeta_0 \neq 0$, we would be stuck with a flat gradient.

To address this issue, Arjovsky et al. [6] advocated to replace (5.3) by the new formulation

$$\min_{\mathsf{P}_G} \mathrm{W}\big[\mathsf{P}_r \,\|\, \mathsf{P}_G\big] \tag{5.18}$$

where W denotes the *Wasserstein-1* distance, also known as the *Earth-mover* distance. The latter is defined as

$$\mathrm{W}\big[\mathsf{P} \,\|\, \mathsf{Q}\big] = \inf_{\pi \in \Pi(\mathsf{P},\mathsf{Q})} \int_{\mathscr{X}\times\mathscr{X}} \|x - y\| \,\mathrm{d}\pi(x,y) = \inf_{\pi \in \Pi(\mathsf{P},\mathsf{Q})} \mathbb{E}_{(x,y)\sim\pi}[\|x - y\|], \tag{5.19}$$

where $\Pi(\mathsf{P},\mathsf{Q})$ stands for the set of all probability distributions $\pi(x,y)$ on $\mathscr{X} \times \mathscr{X}$ whose marginals of π coincide with P and Q. Intuitively, $\pi(x,y)$ indicates how much "mass" must be transported from x to y in order to transform distribution P into distribution Q. The minimal value $\mathrm{W}[\mathsf{P} \,\|\, \mathsf{Q}]$ quantifies the least amount of work required to do the overall transport. Note that the notion of Wasserstein distance hinges upon a preexisting vector norm $\|\cdot\|$ in $\mathscr{X}$.

For the example given in (5.15), it can be verified that

$$\mathrm{W}\big[\mathsf{P} \,\|\, \mathsf{P}_\zeta\big] = |\zeta| \qquad \text{for all } \zeta \in \mathbb{R}, \tag{5.20}$$

which ensures continuity for $\zeta \downarrow 0$ and gives the optimizer a better chance to converge. More generally, it can be shown [6, 10] that the topology associated to the Wasserstein distance is weaker than the one associated with the JS or KL divergence in the following sense.

Proposition 5.1. *Let $\{\mathsf{P}_n\}_{n\in\mathbb{N}}$ be a sequence probability distributions on $\mathscr{X}$ and P a possible limit. Then,*

1. *If $\mathrm{JS}[\mathsf{P} \,\|\, \mathsf{P}_n] \to 0$, then $\mathrm{W}[\mathsf{P} \,\|\, \mathsf{P}_n] \to 0$.*
2. *If $\mathrm{KL}[\mathsf{P} \,\|\, \mathsf{P}_n] \to 0$ or $\mathrm{KL}[\mathsf{P}_n \,\|\, \mathsf{P}] \to 0$, then $\mathrm{W}[\mathsf{P} \,\|\, \mathsf{P}_n] \to 0$.*
3. *$\mathrm{W}[\mathsf{P} \,\|\, \mathsf{P}_n] \to 0$ if and only if $\mathsf{P}_n \xrightarrow{d} \mathsf{P}$, where $\xrightarrow{d}$ denotes convergence in distribution.*

Proof. See Barnett [10]. □

Proposition 5.1 and example (5.15) suggest that the Wasserstein distance is likety to be a more suitable loss function for training GANs. But, as was the case for f-GANs, this loss function $\mathrm{W}\big[\mathsf{P}_r \,\|\, \mathsf{P}_G\big]$ cannot be computed in practice, since we do not know P_r. It turns out that we can recast it under a computable "adversarial" form.

Theorem 5.3. *The WGAN problem* (5.18) *has the same minimizers as*

$$\min_{\mathsf{P}_G} \sup_{T \in \mathrm{Lip}_1(\mathscr{X})} \mathbb{E}_{x\sim\mathsf{P}_r}[T(x)] - \mathbb{E}_{x\sim\mathsf{P}_G}[T(x)], \tag{5.21}$$

where $\mathrm{Lip}_1(\mathscr{X})$ *is the set of all Lipschitz functions* $T : \mathscr{X} \to \mathbb{R}$ *with Lipschitz constant 1.*

Proof. According to the Kantorovich-Rubinstein duality [174], we have

$$\mathrm{W}\big[\mathsf{P} \,\|\, \mathsf{Q}\big] = \sup_{T \in \mathrm{Lip}_1(\mathscr{X})} \mathbb{E}_{x\sim\mathsf{P}}[T(x)] - \mathbb{E}_{x\sim\mathsf{Q}}[T(x)] \tag{5.22}$$

for all pairs of probability distributions (P, Q) on $\mathscr{X}$. □

Similarly to f-GANs, although a function T appearing in (5.22) should be called *critic* in all rigor, it is common to refer to it as *discriminator* and even to write D in place of T.

WGAN with gradient penalty (WGAN-GP). The trouble with WGAN lies in the constraint $T \in \mathrm{Lip}_1(\mathscr{X})$, which also reads

$$\|\nabla_x T(x)\|_2 \leqslant 1 \tag{5.23}$$

and which is really delicate to enforce, especially when T is approximated by a neural network. The authors of [6] recommended to clip the weights of the discriminator, that is, all weights of the discriminator are restricted to remain in a certain interval $[-c, c]$, where $c > 0$ is a hyperparameter. However, clipping the weights severely damages the capacity of the discriminator to learn correctly. Gulrajani et al. [70] came up with a more subtle way to handle this problem. Their method rests upon a constraint saturation property of the optimal critic.

Lemma 5.1. *Let* $T_\sharp \in \mathrm{Lip}_1(\mathscr{X})$ *be an optimal solution of*

$$\max_{\|\nabla T\|_2 \leqslant 1} \mathbb{E}_{x\sim\mathsf{P}_r}[T(x)] - \mathbb{E}_{x\sim\mathsf{P}_G}[T(x)]. \tag{5.24}$$

If $T_\sharp$ *is differentiable and if* $\pi_\sharp(x = y) = 0$[1]*, where* $\pi_\flat$ *is the optimal joint distribution arising in the definition of* $\mathrm{W}[\mathsf{P}_r \,\|\, \mathsf{P}_g]$*, then*

$$\|\nabla_x T_\sharp(x)\|_2 = 1 \tag{5.25}$$

almost everywhere in the sense of both P_r *and* P_G*.*

Proof. See Gulrajani et al. [70, Corollary 1]. □

[1] This technical assumption aims to exclude the case when the matching point of sample x is x itself. It is satisfied in the case that P_r and P_G have supports that intersect in a set of measure 0.

Lemma 5.1 is a strong incentive to approximate the constrained problem (5.21) by the unconstrained but penalized problem

$$\min_{\mathsf{P}_G} \max_{D} \mathbb{E}_{x\sim\mathsf{P}_r}[D(x)] - \mathbb{E}_{\widetilde{x}\sim\mathsf{P}_G}[D(\widetilde{x})] - \lambda \mathbb{E}_{\widehat{x}\sim\mathsf{P}_{\widehat{x}}}[(\|\nabla_x D(\widehat{x})\|_2 - 1)^2], \tag{5.26}$$

where $\lambda > 0$ is the regularization weight and

$$\widehat{x} = \epsilon x + (1-\epsilon)\widetilde{x}, \qquad \epsilon \sim \mathcal{U}([0,1]). \tag{5.27}$$

is a mixture of real (x) and generated ($\widetilde{x}$) images. Note that we have used the symbol D instead of T and that the minus sign in front of the penalty term is due to the fact that we are maximizing with respect to D.

WGAN-GP has significantly improved the stability of GANs by fully unleashing the potential of WGAN. Nowadays, it is used in almost all modern GAN architectures. It should be kept in mind that training a GAN remains a difficult task. The two most frequently encountered issues are [10, 176]:

- *Vanishing gradient.* Illustrated by example (5.15), this also occurs when the discriminator becomes too good at telling fake from real. As a result, the generator receives too weak gradient information, which impedes its improvement.
- *Mode collapse.* This refers to the phenomenon by which the generator fails to capture the full diversity of the target distribution and produces a small set of outputs over and over again or collapses to just a few modes while the target distribution is multi-modal.

By acting on the continuous level, WGAN-GP helps to fight the above plagues. Nevertheless, there are other failure modes which are associated to the discrete level.

Finally, we would like to point out that several approaches other than modifying the loss function have been developed with the aim of stabilizing GANs training. Such methods include new network architectures design [134, 188], regularization [70, 115] and heuristic tricks [148].

5.1.2 Approximation by networks and optimization

So far, we have been working in the continuous framkework where minimization and maximization are carried out with respect to all possible functions G and D. In practice, the generator and the discriminator/critic must be restricted to be neural networks $G_{\theta_G} : \mathscr{Z} \to \mathscr{X}$ and $D_{\theta_D} : \mathscr{X} \to \mathbb{R}$ parametrized by θ_G and θ_D. The WGAN-GP problem (5.26) is now approximated by

$$\min_{\theta_G} \max_{\theta_D} V(\theta_G, \theta_D), \tag{5.28}$$

where $V(\theta_G, \theta_D)$ represents an approximation of the loss function in (5.26). To work out such an approximation, we consider

- $\mathcal{S}_x = \{x^{(1)}, \dots, x^{(m)}\}$ a minibatch of real samples from the training data in $\mathscr{X}$;
- $\mathcal{S}_z = \{z^{(1)}, \dots, z^{(m)}\}$ a minibatch of samples in $\mathscr{Z}$ drawn from the distribution γ.

Note that the two sets have the same size m. Then, we set

$$V(\theta_G, \theta_D) = \frac{1}{m}\sum_{b=1}^{m} D_{\theta_D}(x^{(b)}) - \frac{1}{m}\sum_{b=1}^{m} D_{\theta_D}(G_{\theta_G}(z^{(b)})) - \frac{\lambda}{m}\sum_{b=1}^{m} \left(\|\nabla_x D_{\theta_D}(\widehat{x}^{(b)})\|_2 - 1\right)^2, \tag{5.29}$$

in which

$$\widehat{x}^{(b)} = \epsilon^{(b)} x^{(b)} + (1 - \epsilon^{(b)}) G_{\theta_G}(z^{(b)}) \tag{5.30}$$

with $\epsilon^{(b)}$ randomly drawn from $\mathcal{U}([0,1])$.

When it comes to minimax problems, the *gradient descent ascent* (GDA) or the *stochastic gradient descent ascent* (SGDA) are workhorse methods. The basic tenet is to alternate between maximizing in θ_D and minimizing in θ_G, so that the two players are trained simultaneously and improve together. Algorithm 4 describes the minibatch SGDA method for solving (5.28). This algorithm depends on an integer hyperparameter K, which is the number of gradient ascent steps to be applied to the discriminator before an gradient descent step is performed to update the generator. Indeed, it is quite usual that the discriminator needs to be trained faster than the generator. The reason for this seems to be that the feedback of the discriminator is crucial for the performance of the whole GAN. However, it was said earlier that the vanishing gradient issue may occur when the discriminator is too good. Therefore, there is a balance to be struck between training the discriminator faster then the generator and overtraining it!

Algorithm 4: Minibatch SGDA for WGAN-GP

Input: $K \in \mathbb{N}^*$, $\alpha_D > 0$, $\alpha_G > 0$

for *number of training iterations* **do**

  for K *steps* **do**

    Sample minibatch of m noise samples $\{z^{(1)}, \dots, z^{(m)}\}$ from distribution γ;

    Sample minibatch of m real data samples $\{x^{(1)}, \dots, x^{(m)}\}$ from training set;

    Update the discriminator by ascending gradient with respect to θ_D

$$\theta_D \leftarrow \theta_D + \frac{\alpha_D}{m} \nabla_{\theta_D} \Big\{ \sum_{b=1}^{m} D_{\theta_D}(x^{(b)}) - D_{\theta_D}(G_{\theta_G}(z^{(b)})) - \lambda \big(\|\nabla_x D_{\theta_D}(\widehat{x}^{(b)})\|_2 - 1 \big)^2 \Big\}$$

  end

  Sample minibatch of m noise samples $\{z^{(1)}, \dots, z^{(m)}\}$ from distribution γ;

  Update the generator by descending gradient with respect to θ_G

$$\theta_G \leftarrow \theta_G - \frac{\alpha_G}{m} \nabla_{\theta_G} \Big\{ \sum_{b=1}^{m} -D_{\theta_D}(G_{\theta_G}(z^{(b)})) \Big\}$$

  (neglecting the penalty term so that the generator does not see the real data)

end

In the context of GANs training, adaptive gradient methods such as Adam [88] have demonstrated their effectiveness compared to SGDA and remain therefore the methods of choice for many practitioners. Several previous works [82, 106] attributed Adam's superior performance over SGDA to its higher convergence speed. Meanwhile, more recent works [15, 48, 50] studied the training behavior of GANs using only eigenvalues of the Hessian and reached to the conclusion that curvature information can help with GANs training. However, the performance of second-order methods for GANs training has not been investigated in depth.

We believe that second-order methods are worth considering for GANs. Indeed, it has empirically been observed that the limitations of first-order methods for GANs were caused by the presence of irregularities (saddle points, local optima, plateaus) in the loss landscape

of GANs, which account for the non-convexity and non-concavity of the objective function. Through the curvature matrix, second-order methods gain access to some pieces of information about the landscape of the loss function and can hopefully escape saddle points and achieve a faster convergence to an optimal solution. In light of Algorithm 4, a second-order descent ascent algorithm takes a similar form, with

$$\theta_D \leftarrow \theta_D - \frac{\alpha_D}{m}[C_D(\theta_G, \theta_D)]^{-1}\nabla_{\theta_D} V(\theta_G, \theta_D) \tag{5.31}$$

for the discriminator update and

$$\theta_G \leftarrow \theta_G - \frac{\alpha_G}{m}[C_G(\theta_G, \theta_D)]^{-1}\widetilde{\nabla}_{\theta_G} V(\theta_G, \theta_D) \tag{5.32}$$

for the generator update (with $\widetilde{\nabla}_{\theta_G} V$ approximating $\nabla_{\theta_G} V$ by neglecting the penalty term). Here, the curvature matrix C_D is an approximation of the partial Hessian $\nabla^2_{\theta_D\theta_D} V$ and is expected to be negative semi-definite, while the curvature matrix C_G is an approximation of the partial Hessian $\nabla^2_{\theta_G\theta_G} V$ and is expected to be positive semi-definite. The SGDA algorithm is recovered by setting $C_D = -I$ and $C_G = I$.

In this chapter, we propose to investigate the performance of a natural gradient optimizer, in which C_D and C_G result from the KFAC approximation of the corresponding Fisher matrices. We consider the original KFAC instead of the KPSVD-like methods developed in Chapter 3 or the two-level methods in Chapter 4, because we have seen that the former generally perform worse than KFAC on convolutional neural networks, while the latter have comparable performance to KFAC despite incurring additional computational costs.

Before doing so, we need to know about two features of the GAN under study.

5.1.3 Characteristics of the GAN under study

Evaluation of GANs. Despite the overwhelming success of GANs in numerous applications, it is challenging to correctly evaluate them. This is because in GANs, and more generally in generative models, there is no explicit ground truth labels with which one can quantitatively and objectively compare the quality of generated samples. On top of that, there is a lack of meaningful metrics that allow users to measure similarity between images (or pixels). Evaluation of GANs is based on two important properties: fidelity (quality of generated images, i.e., how realistic they look) and diversity (variety of images the generator is able to produce, i.e., how well the generated images cover all classes presented in the training data). Several metrics for evaluating GANs have been introduced in the literature, but there is no consensus as to which measure best captures the model's strength and weakness and should be used for fair comparison [16].

Here, we present the *Fréchet inception distance* (FID) [76], which is the most widely used metric in the community. FID is based on Wasserstein-2 distance between generated samples distribution and the distribution of training samples. Concretely, FID embeds generated samples and training samples into a feature space using the *inception network*, a convolutional neural network introduced in [162]. Assuming that both generated and real embedded features follow multivariate normal distributions $\mathcal{N}(\mu_G, \Sigma_G)$ and $\mathcal{N}(\mu_r, \Sigma_r)$, FID is defined as the Wasserstein-2 distance between these two Gaussians, given by

$$\begin{aligned} \text{FID} &= \text{W}_2\big[\mathcal{N}(\mu_G, \Sigma_G) \,\|\, \mathcal{N}(\mu_r, \Sigma_r)\big] && \text{(5.33a)} \\ &= \|\mu_G - \mu_r\|_2^2 + \text{tr}\big(\Sigma_G + \Sigma_r - 2\sqrt{\Sigma_G\Sigma_r}\big). && \text{(5.33b)} \end{aligned}$$

In practical computations, the quantities (μ_g, Σ_g) and (μ_r, Σ_r) must be estimated using Monte-Carlo method. To this end, let us consider

- $\mathcal{S}_x = \{x^{(1)}, \dots, x^{(m)}\}$ a minibatch of real samples from the training data in $\mathscr{X}$;
- $\mathcal{S}_z = \{z^{(1)}, \dots, z^{(m)}\}$ a minibatch of samples in $\mathscr{Z}$ drawn from the distribution γ;
- the inception network [162] $F : \mathscr{X} \to \mathbb{R}^N$, which embeds an image from $\mathscr{X}$ to the space of features $\mathbb{R}^N$.

Then, we set

$$\mu_r = \frac{1}{m}\sum_{b=1}^{m} F(x^{(b)}), \qquad \Sigma_r = \frac{1}{m}\sum_{b=1}^{m}[F(x^{(b)}) - \mu_r][F(x^{(b)}) - \mu_r]^T, \tag{5.34a}$$

$$\mu_G = \frac{1}{m}\sum_{b=1}^{m} F(G_{\theta_G}(z^{(b)})), \quad \Sigma_G = \frac{1}{m}\sum_{b=1}^{m}[F(G_{\theta_G}(z^{(b)})) - \mu_G][F(G_{\theta_G}(z^{(b)})) - \mu_G]^T. \tag{5.34b}$$

Lower FID means smaller distance between learned and real data distributions. The major concern with FID is its presumption that embedded features obey Gaussian distributions, which is far from being granted. Furthermore, it takes into account only the first two moments of the distributions. Notwithstanding these concerns, FID appears to be a relevant measure of fidelity and diversity. It has been observed that FID is consistent with qualitative evaluation by human judgment [16]. This explains why FID is widely adopted in the community.

Deep convolutional generative adversarial networks (DCGAN). This extension of the original GAN framework was introduced by Radford et al. [134] in order to improve the quality and stability of generated samples, particularly in the domain of image generation. The primary innovation of DCGANs lies in their architecture, which incorporates convolutional layers into the GAN framework. Convolutional neural networks (CNNs) have demonstrated exceptional performance in various image processing tasks, by effectively capturing spatial hierarchies and extracting meaningful features from images. By leveraging the power of CNNs, DCGANs enhance the capability of GANs to generate realistic and high-quality images.

DCGANs introduce several architectural guidelines and design choices to ensure stable training and better sample generation:

- *Transposed convolution layers*: DCGANs utilize transposed convolutional layers in the generator network. These layers help upsample the low-resolution input noise into high-resolution image-like outputs.
- *Strided convolutions*: in the discriminator network, DCGANs use strided convolutions instead of pooling layers to downsample the input images. This helps in preserving spatial information while reducing the image resolution.
- *Batch normalization*: DCGANs apply batch normalization to both the generator and discriminator networks. This technique normalizes the inputs to each layer, which helps in mitigating issues related to internal covariate shift and contributes to more stable training.
- *Activation functions*: DCGANs use Rectified Linear Units (ReLU) as the activation function for the generator network, except for the output layer, which typically employs a hyperbolic tangent (tanh) activation function. In the discriminator network, Leaky ReLU is used to introduce non-linearity and prevent sparse gradients.

These architectural choices help address the common challenges faced by traditional GANs, such as training instability, mode collapse and low-quality or blurry generated images.

DCGANs have been successfully applied to various image generation tasks, including generating realistic human faces, creating artistic images, and synthesizing new objects. They have also been used in domain adaptation, image super-resolution, and other applications that involve generating or transforming images.

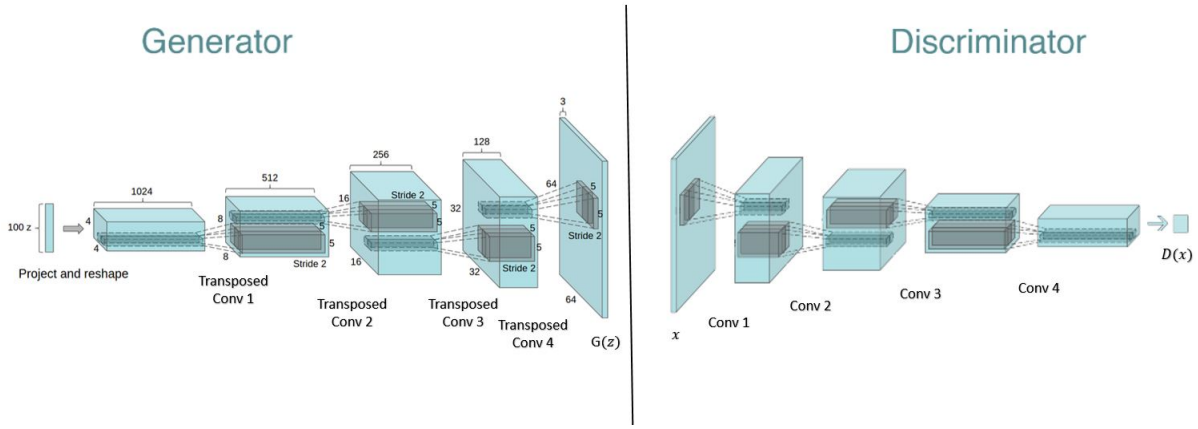


Figure 5.2: DCGAN architecture as proposed in [134]. The input noise z is usually a vector of size 100.

Since transposed convolution are an essential part of GAN architectures, we first extend the KFAC method to transposed convolution layers. Then, we evaluate the performance of the obtained optimizer on GANs training tasks. To the best of our knowledge, we are the first to consider KFAC for transposed convolution layers and also to study its performance for GANs training.

5.2 Extension of KFAC to transposed convolution layers

5.2.1 Transposed convolution layers

Transposed convolution layers [46], also referred as *fractionally strided convolution* or *deconvolution*[2] layers are a fundamental block in many deep learning models, including GANs, convolutional auto-encoder, etc. They are thus crucial for many computer vision tasks such as image segmentation, image generation, and image-to-image translation. Their role is to upsample or increase the spatial dimensions of the inputs.

In a standard convolutional layer, a set of filters are applied to input feature maps to produce output feature maps with reduced spatial dimensions. Transposed convolutions work in the opposite direction. Instead of reducing the spatial dimensions of input feature maps, they aim to expand them. The idea behind a transposed convolution is to learn to increase the spatial resolution of the input feature maps using a set of learnable parameters.

[2]The term *deconvolution* is sometimes wrongly used as a synonym of *transposed convolution*. A deconvolution is the inverse of a convolution. A deconvolution is thus applied to the output of some convolution to recover its input. This is not the case of a transposed convolution, which only recovers the shape, but not the input itself.

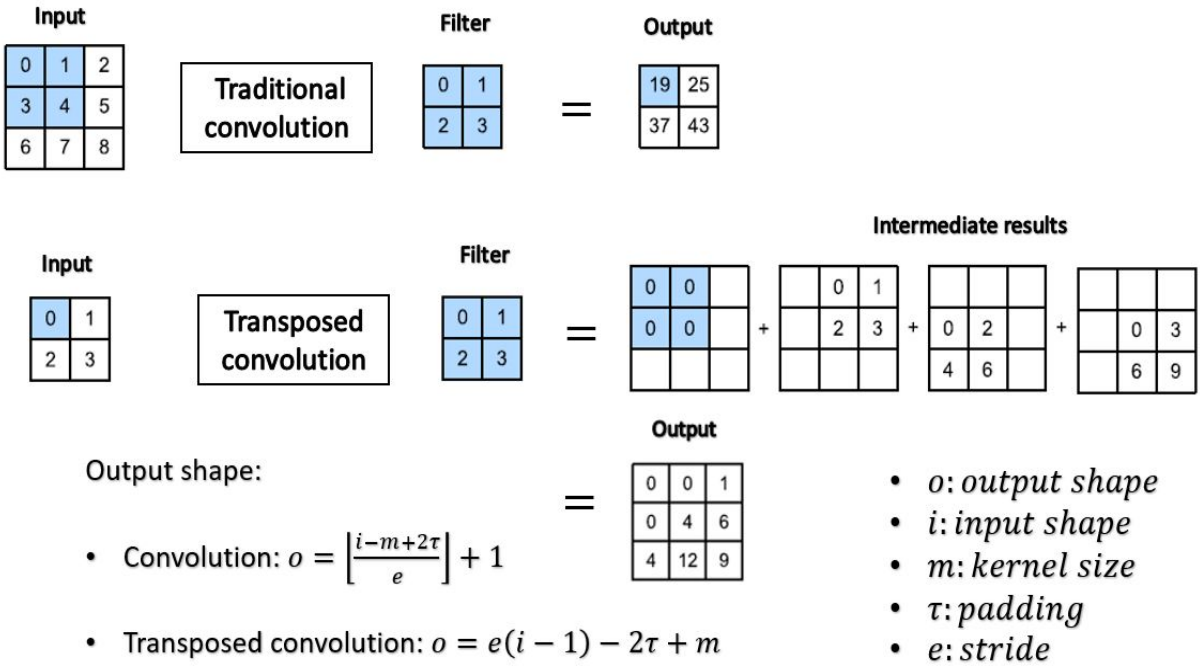


Figure 5.3: Comparison of transposed convolution against traditional convolution.

Transposed convolutions are closely related to classical convolutions. They use the same concepts such as filters, padding and stride (see Figure 5.3). A transposed convolution operation can be viewed of as the gradient of some traditional convolution with respect to its input, which is usually how a transposed convolution is implemented in practice [46]. Finally, applying some transformation to the input, with a proper reshape of filters, a transposed convolution can be turned into traditional convolution operation (cf. §5.2.2).

5.2.2 Reformulation as traditional convolution

A transposed convolution can be performed using a traditional convolution operation [46]. This requires a few modifications to the original inputs of the transposed convolution, as shown in the following relationship.

Relationship 1 (Dumoulin et Visin [46])**.** *Suppose we want to perform a transposed convolution operation between an input features map* $\mathcal{A}_{i-1} \in \mathbb{R}^{c_{i-1} \times h_{i-1} \times w_{i-1}}$ *and a set of filters* $\mathcal{F} \in \mathbb{R}^{c_i \times c_{i-1} \times m_i \times m_i}$*, using a padding* τ *and a stride* e*. Then, considering new versions* $\mathcal{A}'_{i-1} \in \mathbb{R}^{c_{i-1} \times h'_{i-1} \times c'_{i-1}}$*,* $\mathcal{F}' \in \mathbb{R}^{c_i \times c_{i-1} \times m_i \times m_i}$*,* τ' *and* $(e', \bar{e})$ *of the initial input features map, filters, padding and stride respectively, we can perform a traditional convolution on the new variables and obtain the same result as if we performed a transposed convolution operation on*

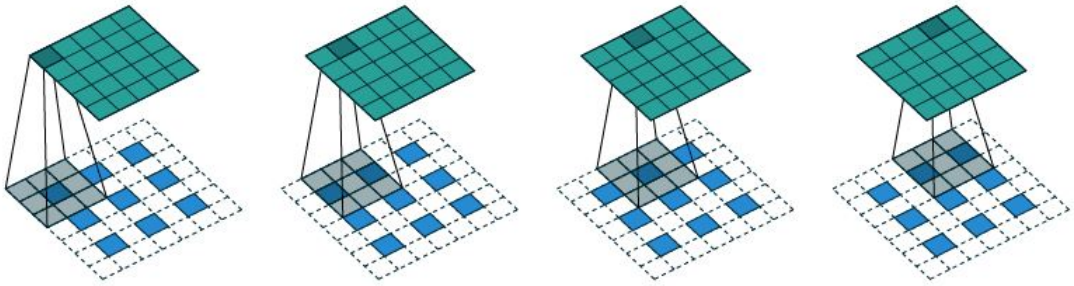


Figure 5.4: Example of a transposed convolution turned into a traditional convolution [46]. The original variables are an input feature map of shape 3×3 (represented by blue color), a filter of size 3×3, a padding $\tau = 1$, and a stride $e = 2$. The transposed convolution is turned into a traditional convolution using new variables defined as follows: padding $\tau' = 3 - 1 - 1 = 1$, strides $e' = 1$, $\bar{e} = 2 - 1 = 1$, fractionally strided version of the input obtained by inserting $\bar{e} = 1$ row and $\bar{e} = 1$ column of zero's between rows and columns of the initial input, new filter obtained by performing a rotation of angle π of the initial filter.

the original variables, i.e.,

$$\textit{transposed convolution}\big(\mathcal{A}_{i-1}, \mathcal{F}, \tau, e\big) = \textit{convolution}\big(\mathcal{A}'_{i-1}, \mathcal{F}', \tau', e', \bar{e}\big).$$

The new variables are defined as follows:

- $\tau' = m_i - \tau - 1$.
- $e' = 1$, $\bar{e} = e - 1$.
- *$\mathcal{F}'$ is obtained by performing a rotation of angle π of $\mathcal{F}$ from the axis corresponding to the height of a kernel (penultimate axis) towards the axis corresponding to the width the kernel (last axis). This operation is a kind of reflection of the values of each matrix constituting the tensor $\mathcal{F}$ (see Figure 5.5 for an illustration). We will denotes by $\mathcal{R}$ this operation and by $\mathcal{R}^{-1}$ its inverse (i.e., rotation of angle $-\pi$) such that*

$$\begin{aligned} \mathcal{R} : \mathbb{R}^{c_i \times c_{i-1} \times m_i \times m_i} &\longrightarrow \mathbb{R}^{c_i \times c_{i-1} \times m_i \times m_i} \\ \mathcal{F} &\longmapsto \mathcal{F}', \end{aligned}$$

  and

$$\begin{aligned} \mathcal{R}^{-1} : \mathbb{R}^{c_i \times c_{i-1} \times m_i \times m_i} &\longrightarrow \mathbb{R}^{c_i \times c_{i-1} \times m_i \times m_i} \\ \mathcal{F}' &\longmapsto \mathcal{F}. \end{aligned}$$

- *$\mathcal{A}'_{i-1}$ is obtained by inserting $\bar{e}$ number of rows and columns of zero's between rows and columns of $\mathcal{A}_{i-1}$ respectively. $\mathcal{A}'_{i-1}$ is said to be a **fractionally strided** version of $\mathcal{A}_{i-1}$. This is why transposed convolutions are also called fractionally strided convolutions.*

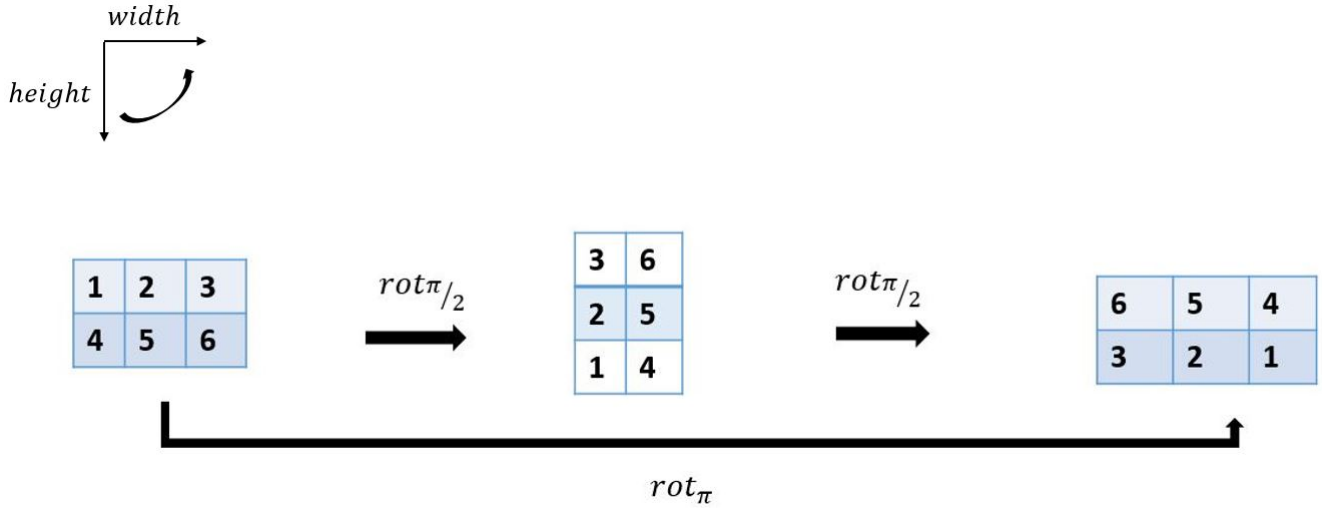


Figure 5.5: An illustration of $\mathcal{R}$ operator. Here both input and output channels are equal to 1.

5.2.3 KFAC approximation

To establish KFAC for transposed convolutions, we will proceed by analogy with the case of KFAC for traditional convolutions. From now on, to simplify notation, we will omit the the subscript i relating to a layer. Let us consider a traditional convolution layer which receives as input a features map $\mathcal{A}' \in \mathbb{R}^{c\times h'\times w'}$ and has a set of kernels represented by a tensor $\mathcal{F}' \in \mathbb{R}^{c_o\times c_\times m\times m}$. Let call by $\mathcal{V}$ the reshape operator that turns the tensor $\mathcal{F}'$ into a weight matrix $W' \in \mathbb{R}^{c_o\times(cm^2+1)}$ (cf. §2.1.2) and $\mathcal{V}^{-1}$ its inverse such that

$$\begin{aligned}\mathcal{V} : \mathbb{R}^{c_o\times c_\times m\times m} &\longrightarrow \mathbb{R}^{c_o\times(cm^2+1)}\\ \mathcal{F}' &\longmapsto W',\end{aligned}$$

and

$$\begin{aligned}\mathcal{V}^{-1} : \mathbb{R}^{c_o\times(cm^2+1)} &\longrightarrow \mathbb{R}^{c_o\times c_\times m\times m}\\ W' &\longmapsto \mathcal{F}'.\end{aligned}$$

The KFAC iteration for that convolution layer is given by (cf. §3.4.1)

$$\mathrm{vec}(W'_{k+1}) = \mathrm{vec}(W'_k) - \alpha_k\big(\bar{A}'_k \otimes G'_k\big)^{-1}\nabla_{\mathrm{vec}(W'_k)}h(\theta'_k), \tag{5.35}$$

We recall that "vec" is the operator that turns a matrix in a vector by stacking its columns all together. Iteration (5.35) can be expressed in matrix form as

$$W'_{k+1} = W'_k - \alpha_k(G'_k)^{-1}\nabla_{W'_k}h(\theta'_k)(\bar{A}'_k)^{-1}. \tag{5.36}$$

The Kronecker factors are defined as (cf. §3.4.1)

$$\bar{A}'_k = \mathbb{E}\Big[\sum_{t\in\mathcal{T}'}(\bar{a}'_t)_k(\bar{a}'_t)_k^T\Big], \qquad G'_k = \frac{1}{|\mathcal{T}'|}\mathbb{E}\Big[\sum_{t\in\mathcal{T}'}(g'_t)_k(g'_t)_k^T\Big]. \tag{5.37}$$

Using the unrolling approach presented in §2.1.2, and the matrix implementation of a convolution operation (see equation (2.6)), the Kronecker factors can be rewritten as

$$\bar{A}'_k = \mathbb{E}\big[[\![\mathcal{A}'_k]\!][\![\mathcal{A}'_k]\!]^T\big], \qquad G'_k = \mathbb{E}\big[\nabla_{S'_k} L[\nabla_{S'_k} L]^T\big]. \tag{5.38}$$

We recall that in (5.38), L denotes the loss function, S'_k refers to the matrix of pre-activations and $[\![\mathcal{A}'_k]\!]$ is the unrolled version of the matrix of activations $\mathcal{A}'_k$. Note that $S'_k = S_k$ (this is because S_k is the output of the direct transposed convolution and S'_k is the output of a convolution which is equivalent to the transposed convolution)

Now, suppose we have a transposed convolution layer i, which receives as input a features map $\mathcal{A}$, a set of filters $\mathcal{F}$, a padding τ and a stride e. To derive a KFAC iteration to that layer, we follow the steps below:

1. We build new versions $\mathcal{A}'$, $\mathcal{F}'$, τ' and $(e', \bar{e})$ of the inputs as defined by Relationship 1.
2. We treat layer i as a traditional convolution by applying the KFAC iteration to the new variables as defined by (5.36).
3. We go back to the original variables by setting

$$\begin{aligned} W_{k+1} &= \mathcal{V}(\mathcal{F}_{k+1}) && \text{(5.39a)}\\ &= \mathcal{V}(\mathcal{R}^{-1}(\mathcal{F}'_{k+1})) && \text{(5.39b)}\\ &= \mathcal{V}(\mathcal{R}^{-1}(\mathcal{V}^{-1}(W'_{k+1}))) && \text{(5.39c)}\\ &= \mathcal{V}(\mathcal{R}^{-1}(\mathcal{V}^{-1}(W'_k - \alpha_k (G'_k)^{-1}\nabla_{W'_k} h(\theta'_k)(\bar{A}'_k)^{-1}))). && \text{(5.39d)} \end{aligned}$$

Operators $\mathcal{R}$ and $\mathcal{V}$ and their inverses are obviously linear. We thus obtain

$$\begin{aligned} W_{k+1} &= \mathcal{V}(\mathcal{R}^{-1}(\mathcal{V}^{-1}(W'_k))) - \alpha_k \mathcal{V}(\mathcal{R}^{-1}(\mathcal{V}^{-1}((G'_k)^{-1}\nabla_{W'_k} h(\theta'_k)(\bar{A}'_k)^{-1}))) && \text{(5.40a)}\\ &= W_k - \alpha_k \mathcal{V}(\mathcal{R}^{-1}(\mathcal{V}^{-1}((G'_k)^{-1}\nabla_{W'_k} h(\theta'_k)(\bar{A}'_k)^{-1}))). && \text{(5.40b)} \end{aligned}$$

Noticing that $h(\theta_k) = h(\theta'_k)$, we have $\nabla_{W'_k} h(\theta'_k) = \nabla_{W'_k} h(\theta_k)$. Since $W'_k = \mathcal{V}(\mathcal{R}(\mathcal{V}^{-1}(W_k)))$, we obtain

$$\nabla_{W'_k} h(\theta'_k) = \mathcal{V}(\mathcal{R}(\mathcal{V}^{-1}(\nabla_{W_k} h(\theta_k)))).$$

The KFAC iteration for the transposed convolution is therefore given by

$$W_{k+1} = W_k - \alpha_k \mathcal{V}(\mathcal{R}^{-1}(\mathcal{V}^{-1}((G'_k)^{-1}[\mathcal{V}(\mathcal{R}(\mathcal{V}^{-1}(\nabla_{W_k} h(\theta_k))))](\bar{A}'_k)^{-1}))). \tag{5.41}$$

5.3 Numerical tests

5.3.1 Deep convolutional auto-encoders

In this subsection, we evaluate our extension of KFAC for transposed convolution layers on deep convolutional auto-encoders. Before considering GANs, we first test the optimization performance on the objective function of the proposed method using convolutional deep auto-encoders. This is because, in GANs, because of the nature of the optimization problem, it is difficult to evaluate the optimization speed of an optimizer. Furthermore, in GANs, the metric of interest is FID rather than the value of the objective function.

The basic structure of a convolutional auto-encoder consists of an encoder and a decoder. The encoder takes an input image and gradually reduces its spatial dimensions while increasing the number of channels or feature maps. This process captures hierarchical representations of the input image, with higher-level features representing more abstract concepts. The encoder typically consists of convolutional layers followed by pooling layers. The decoder performs the inverse operation, taking the learned encoded representation and reconstructing an output image that is as close as possible to the original input. It uses transposed convolution to progressively upsample the features and reconstruct the image. The number of channels or feature maps is reduced in the decoder to match the original input. See Figure 5.6 for an illustration.

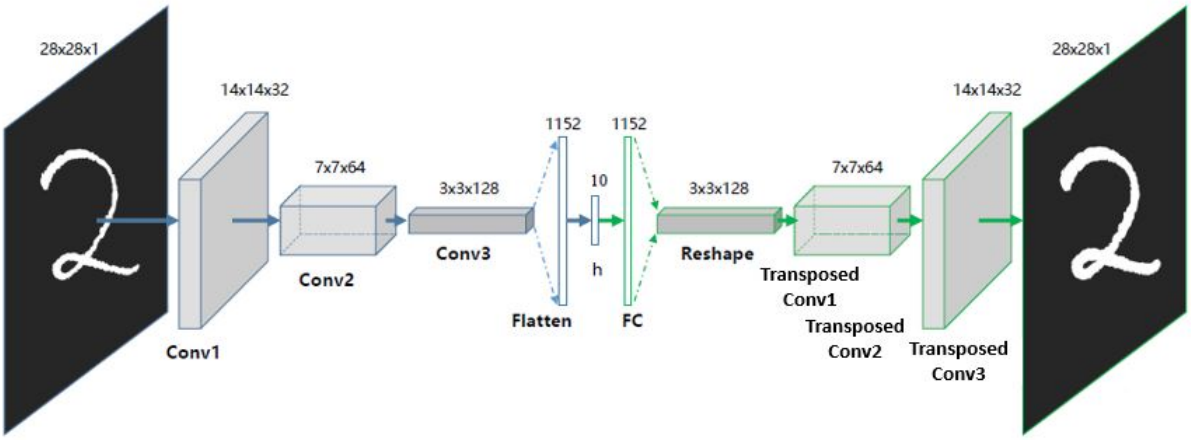
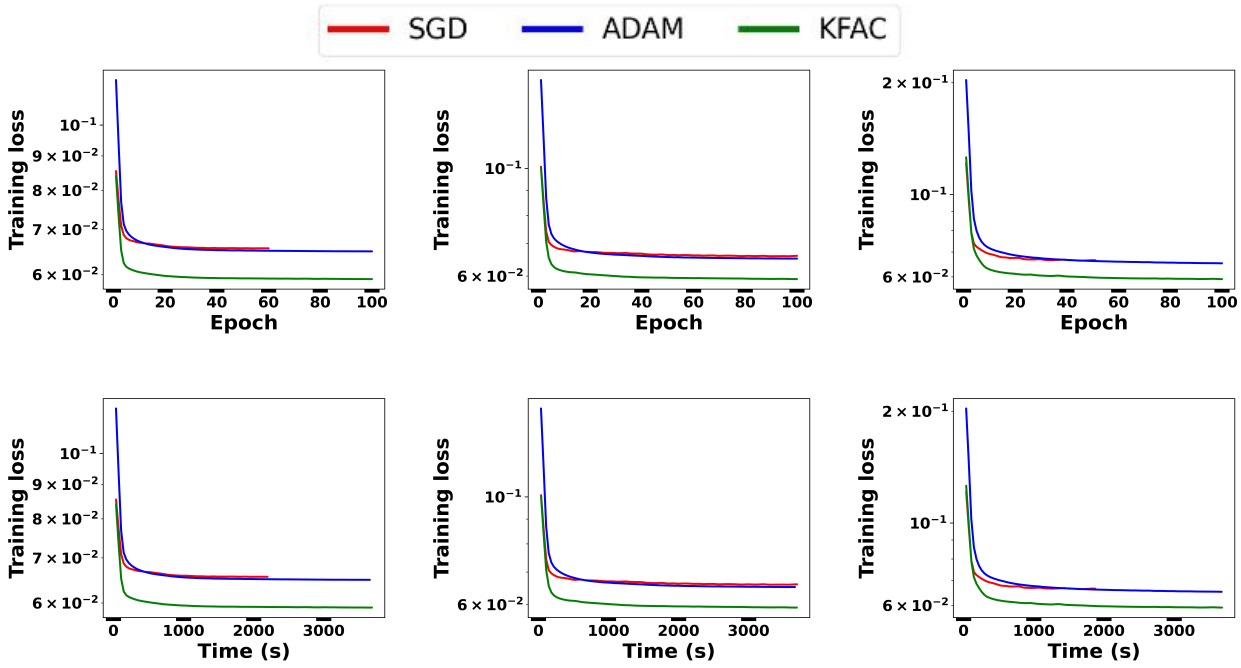


Figure 5.6: Architecture of a standard convolutional auto-encoder [71].

In our experiments, we consider three different convolutional auto-encoders, each trained with a different dataset (MNIST, CIFAR10 and SVHN). All three architectures are four layers deep and contain batch normalizations. For each problem, we train the network with three different batch sizes. Hyper-parameters (learning rate, damping) are selected according to the lowest value of the training loss. We apply early stopping with a patience of 10 epochs which means that we stop training if there is no an improvement of the training loss after 10 consecutive epochs. Figures 5.7, 5.8, 5.9 show the results obtained for MNIST, CIFAR10 and SVHN datasets respectively. We observe that for each problem, regardless of the batch size, KFAC outperforms SGD and ADAM both with respect to epoch and time. This implies that our extension of KFAC for transposed convolution is efficient both in terms of optimization performance and computational cost. We also observe that, in each problem, the gap between KFAC and baseline optimizers (ADAM and SGD) increases as the batch size increases. This is a desirable property insofar as models can be trained faster by considering larger batches.

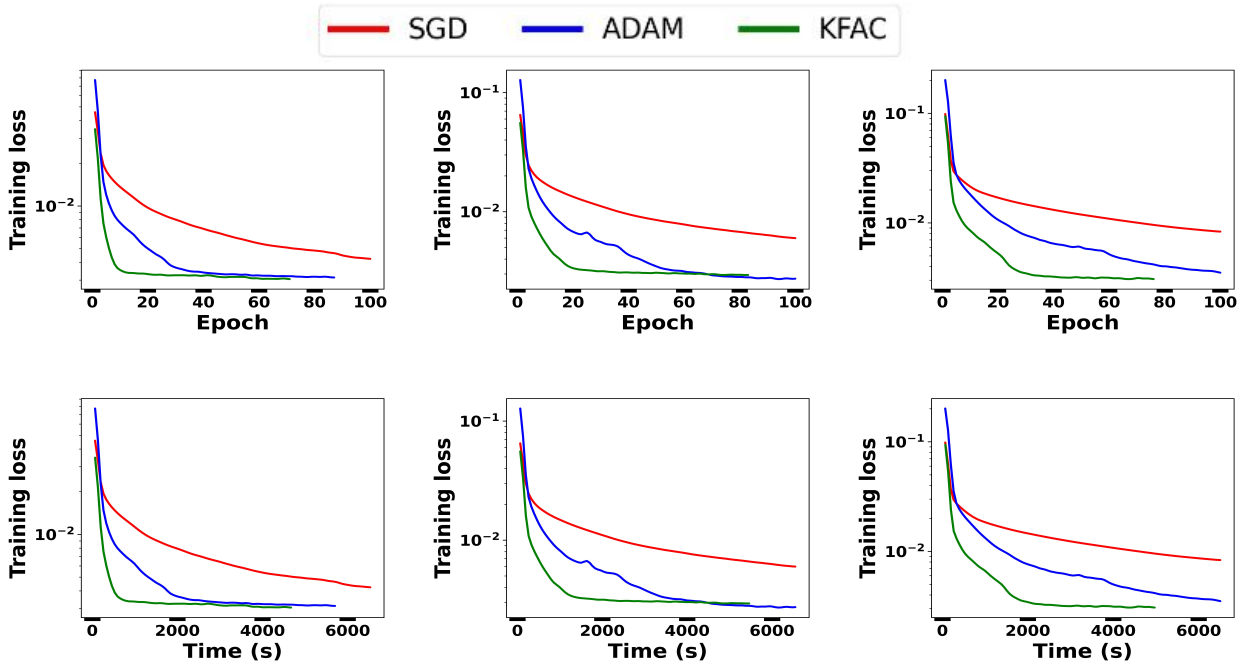
5.3.2 Generative adversarial networks

Here, we present a set of experiments that empirically evaluate the performance of GANs trained using curvature information. Our aim is not to achieve state-of-the-art performance, but rather to evaluate how KFAC-type methods perform on GANs compared to baselines (SGDA and Adam). In our experiments, we consider two different GANs, each trained with a different dataset. The first one trained with MNIST dataset, consists of a generator and a discriminator containing both 5 layers and batch normalizations. For the second one, we use the ResNet GAN



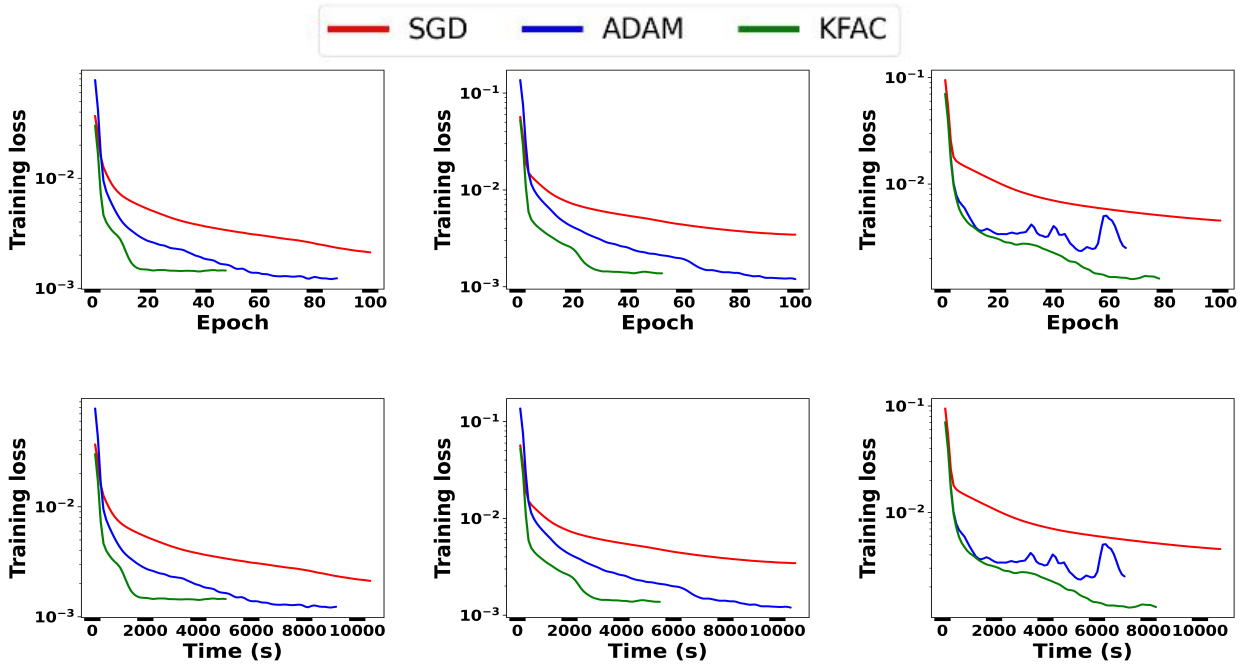
(a) Batch size = 128 (b) Batch size = 256 (c) Batch size = 512

Figure 5.7: Comparison of KFAC against SGD and ADAM on a deep convolutional auto-encoder trained with MNIST dataset. Three different batch sizes are considered (128 **first** column, 256 **middle** column and 512 **last** column). In each column, first Figure displays training loss vs epoch while second ones depicts training loss vs time.



(a) Batch size = 256 (b) Batch size = 512 (c) Batch size = 1024

Figure 5.8: Optimization performance evaluation of KFAC, ADAM and SGD on a deep convolutional auto-encoder trained with CIFAR10 dataset. Three different batch sizes are used (256 **first** column, 512 **middle** column and 1024 **last** column). For each batch size, first Figure shows training loss vs epoch while second ones depicts training loss vs time.



(a) Batch size = 512 (b) Batch size = 1024 (c) Batch size = 2048

Figure 5.9: Optimization performance evaluation of KFAC, ADAM and SGD on a deep convolutional auto-encoder trained with SVHN dataset. Three different batch sizes are used (512 **first** column, 1024 **middle** column and 2048 **last** column). For each batch size, first Figure shows training loss vs epoch while second ones displays training loss vs time.

as in [82] and train it with CIFAR10. It should be noted that both generator and discriminator of ResNet GAN contain 11 layers each and use residual blocks and batch normalizations. In both problems, we use Wasserstein loss with gradient penality. We choose FID to quantitatively assess the performance of each model. Note that for each optimizer, we apply Grid-Search and choose hyperparameters that give the best FID score. We also apply an early-stop procedure using FID as criterion. This means that we stop training the model if the FID score starts to deteriorate after a few consecutive epochs. For KFAC optimizer, in order to conduct a comprehensive analysis of the impact of curvature information on each player (generator or discriminator), we consider the following optimization settings:

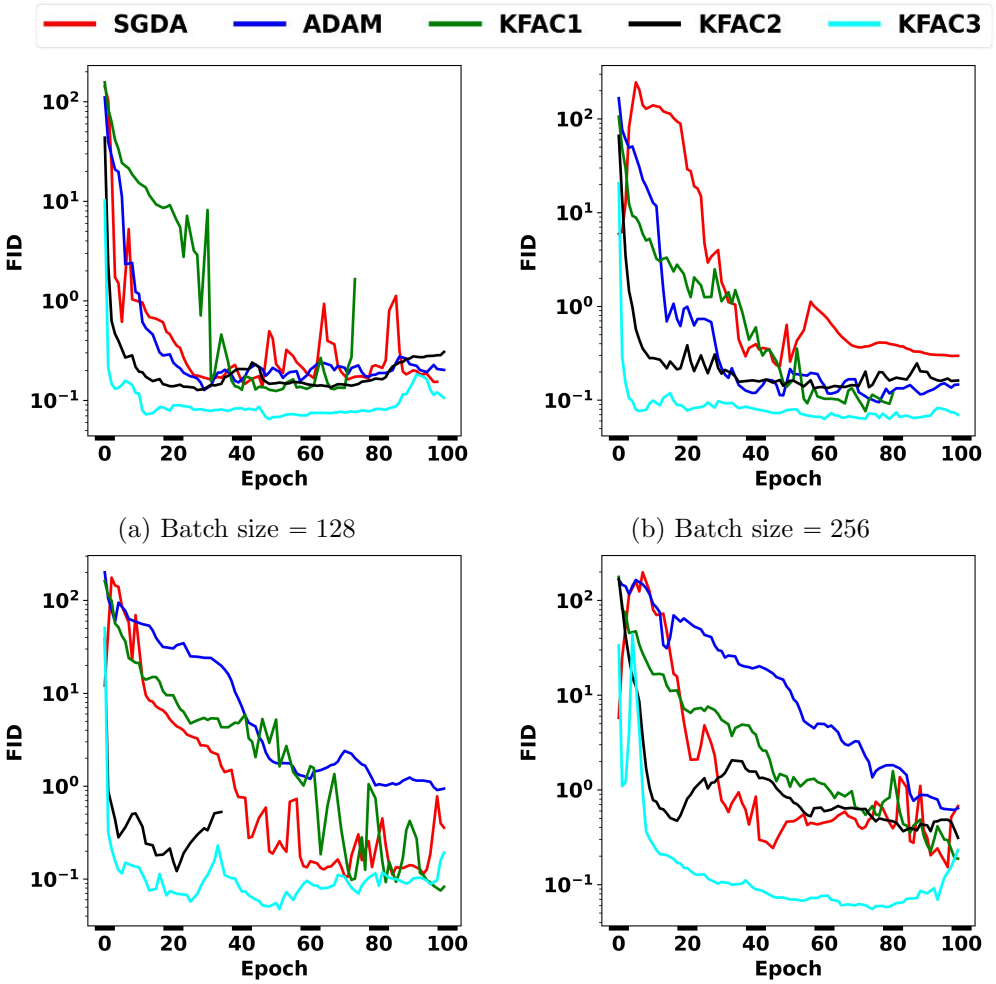
- KFAC1: the generator is trained with KFAC optimizer and the discriminator is trained with Adam optimizer.
- KFAC2: both generator and discriminator networks are trained with KFAC optimizer.
- KFAC3: the generator is trained with Adam optimizer while the discriminator is trained with KFAC optimizer.

Figures 5.10 and 5.11 show the obtained results for MNIST and CIFAR10 problems respectively. For MNIST, we observe that KFAC3 significantly outperforms other optimizers and the performance gap widens as batch size increases. It is followed by KFAC2 and KFAC1 which globally do better than baseline optimizers. Adam outperforms SGDA on batch sizes 128 and 256 but underperforms on batch sizes 512 and 1024. As for CIFAR10, again KFAC3 displays a better performance than other optimizers, regardless of batch size. But here, Adam is better than KFAC1 and KFAC2 which in turn outperform SGDA.

The conclusion from these two experiments is that KFAC3 is the most effective method among the considered optimizers. As for the other optimizers, there is no defined order of performance, as the performance of one optimizer compared to another depends on the network architecture, data, batch size, and can reverse if the configuration is changed.

The fact that KFAC3 consistently displays a good performance is an expected result. Indeed, several studies [51, 82, 83] have shown that in order to achieve a better performance for a GAN, it is necessary to keep the discriminator relatively well-trained compared to the generator. This requirement is met with KFAC3 since the discriminator is trained with the KFAC optimizer, allowing it to converge relatively faster than the generator, which is trained with Adam. This argument could be used to explain why KFAC1 has unstable performance and sometimes performs worse than standard optimizers. In fact, in the case of KFAC1, unlike KFAC3, it is the generator that is trained with KFAC while the discriminator is trained with Adam, contradicting the condition of effectively training the discriminator more than the generator. Regarding KFAC2, it is difficult to find a precise explanation for its unstable performance, as both players have been trained with KFAC. Nevertheless, the case of KFAC2 proves that optimizing GANs remains highly a complex task and that it is not sufficient to simply train both players with efficient optimizers to achieve better performance.

Figures 5.12, 5.13, 5.14, 5.15 and 5.16 display MNIST generated images for the first 10 epochs of training using SGDA, Adam, KFAC1, KFAC2 and KFAC3 optimizers respectively. Every optimizer uses batch size of 128, and in each epoch, 25 images are generated. By analyzing these early results, we aim to assess the ability of optimization methods to facilitate rapid generation of meaningful and visually appealing images. The early-stage images generated by the GANs using the SGDA, Adam, KFAC1 and KFAC2 optimizers show moderate progress in quality. While the initial images lack clarity, there is a discernible improvement over the first



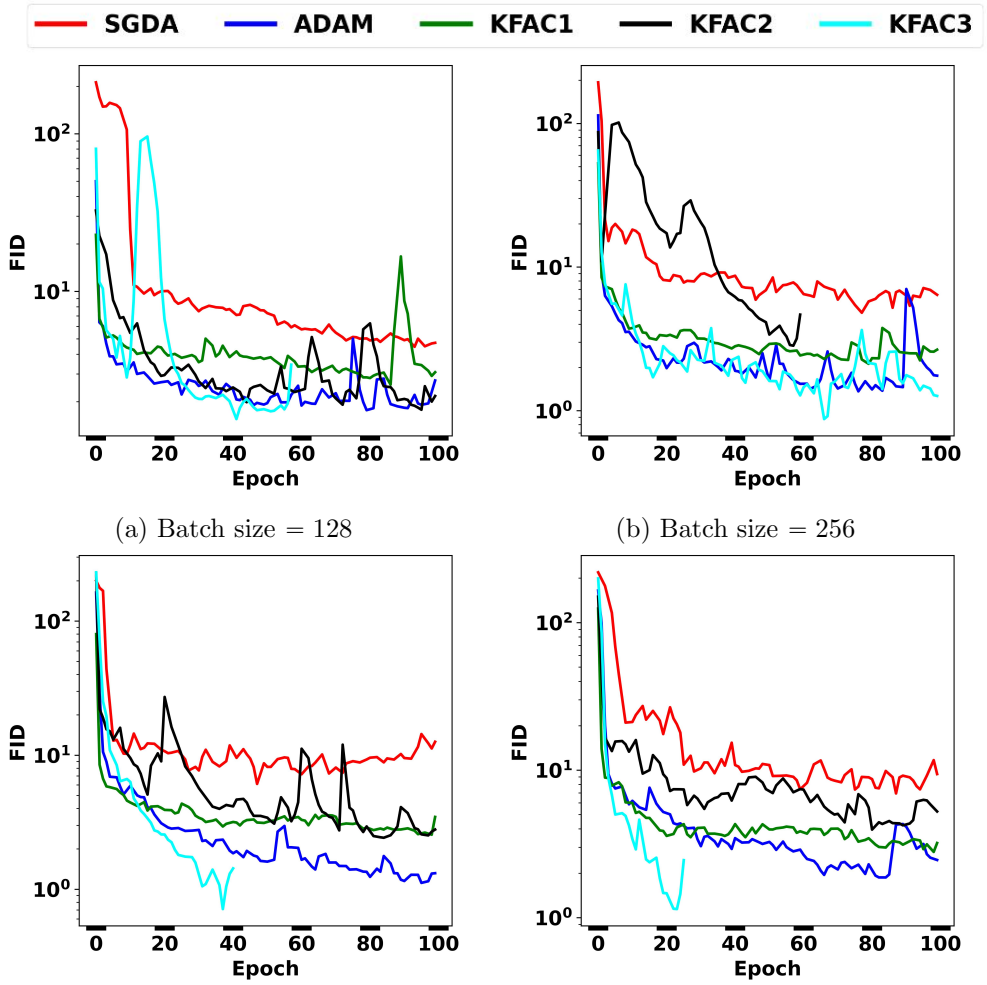
(a) Batch size = 128

(b) Batch size = 256

(c) Batch size = 512

(d) Batch size = 1024

Figure 5.10: Performance evaluation of different optimizers on a GAN model trained with MNIST dataset. Four different batch sizes are considered.



(a) Batch size = 128

(b) Batch size = 256

(c) Batch size = 512

(d) Batch size = 1024

Figure 5.11: FID curves of different optimizers on ResNet GAN trained with CIFAR10 dataset. Four different batch sizes are considered.

10 epochs. However, GANs tarined with SGDA and KFAC1 demonstrate a slower improvement compared to Adam and KFAC2 optimizers. As for KFAC3 optimizer, the GAN trained with it exhibits a great performance right from the early stages. The images generated during the first epoch show remarkable detail and realism. As training progresses, the quality continues to improve significantly compared to other optimizers, demonstrating the effectiveness of KFAC3 in rapidly producing relevant images. This implies that one can train a GAN with KFAC3 for a short time and be able to generate high-quality images, thereby saving time. Furthermore, a careful observation of the generated images of each optimizer and FID curves associated with it corroborates the positive correlation between the FID score and the quality of the generated images.

In a nutshell, these experiments suggest that to achieve a better GAN performance, one can train the discriminator with a second-order optimizer like KFAC and train the generator with an adaptive method such as Adam. It is important to note that these results need to be confirmed on large scale datasets such as ImageNet [36] using more complex state-of-the-art architectures such as StyleGAN [86], BigGAN [21], Self-attention GAN [188], etc.

5.4 Conclusion

In this chapter, the objective was to investigate the performance of the KFAC optimizer in the context of GANs training. We introduced an extension to the KFAC method to handle transposed convolution layers, which are important components of GAN architectures.

To evaluate the effectiveness of the proposed optimizer, three different convolutional auto-encoder problems were considered, each trained with a different dataset. The results showed that the extended KFAC optimizer outperformed baseline optimizers such as SGD and Adam in terms of optimization performance, both in terms of iterations and time. This demonstrates the effectiveness of the proposed method for optimizing the objective function.

Moving on to GAN training, the findings revealed promising results regarding the effectiveness of the KFAC optimizer. Indeed, when the generator was trained with Adam optimizer and the discriminator with KFAC, the model achieved faster convergence, as indicated by a good FID score, and its ability to generate high-quality images early in the training process. However, experiments also highlighted a concern when both players (generator and discriminator) were trained with the KFAC optimizer. This resulted in an unstable performance of the model and sometimes led to worse performance compared to models where both players were trained with baseline optimizers like SGD or Adam. This emphasizes the difficulty of effectively training GANs and raises questions about how to effectively incorporate curvature information in GAN optimization to achieve better performance.

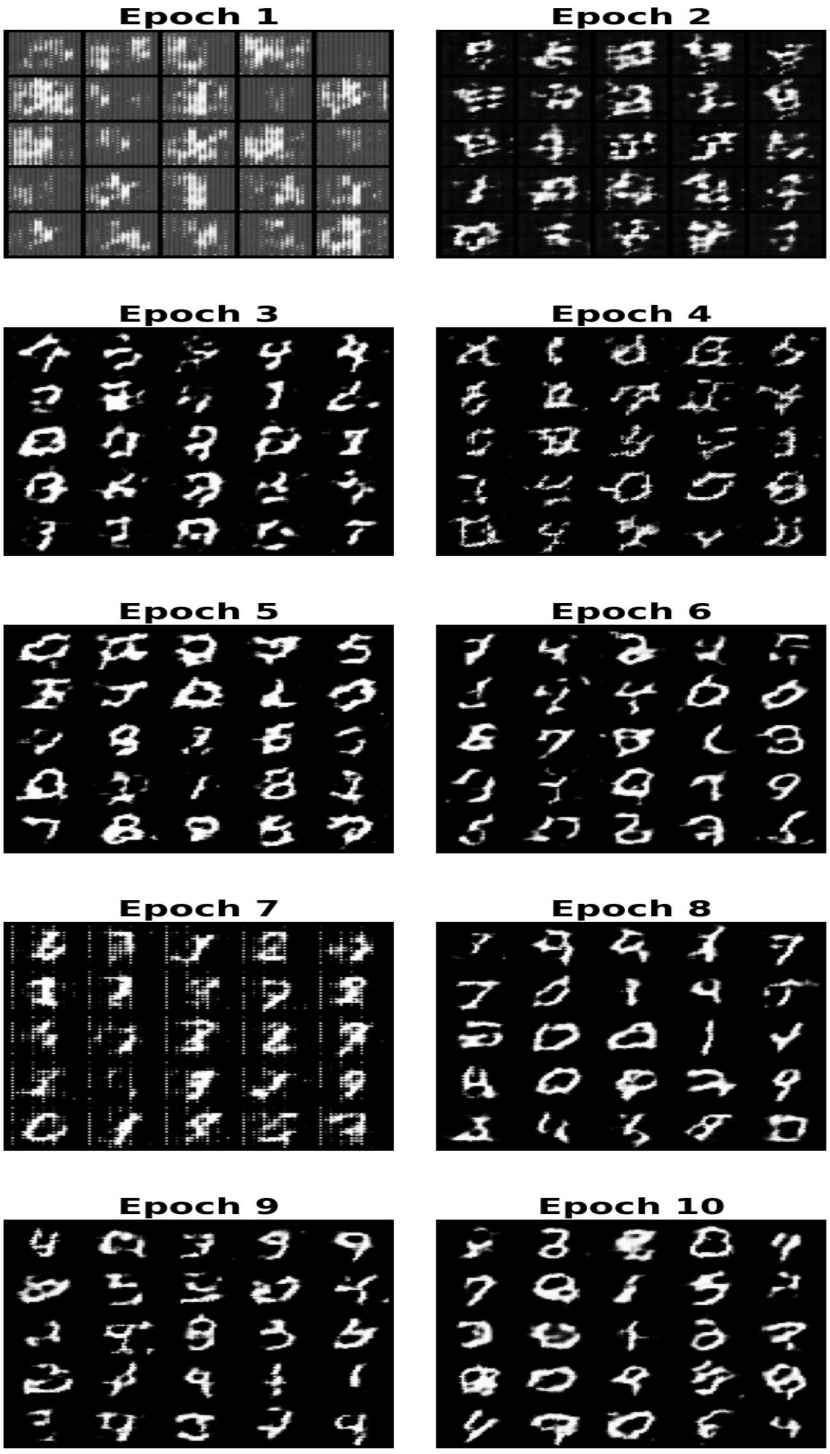


Figure 5.12: MNIST images randomly generated by a GAN trained with SGDA optimizer for the first 10 epochs.

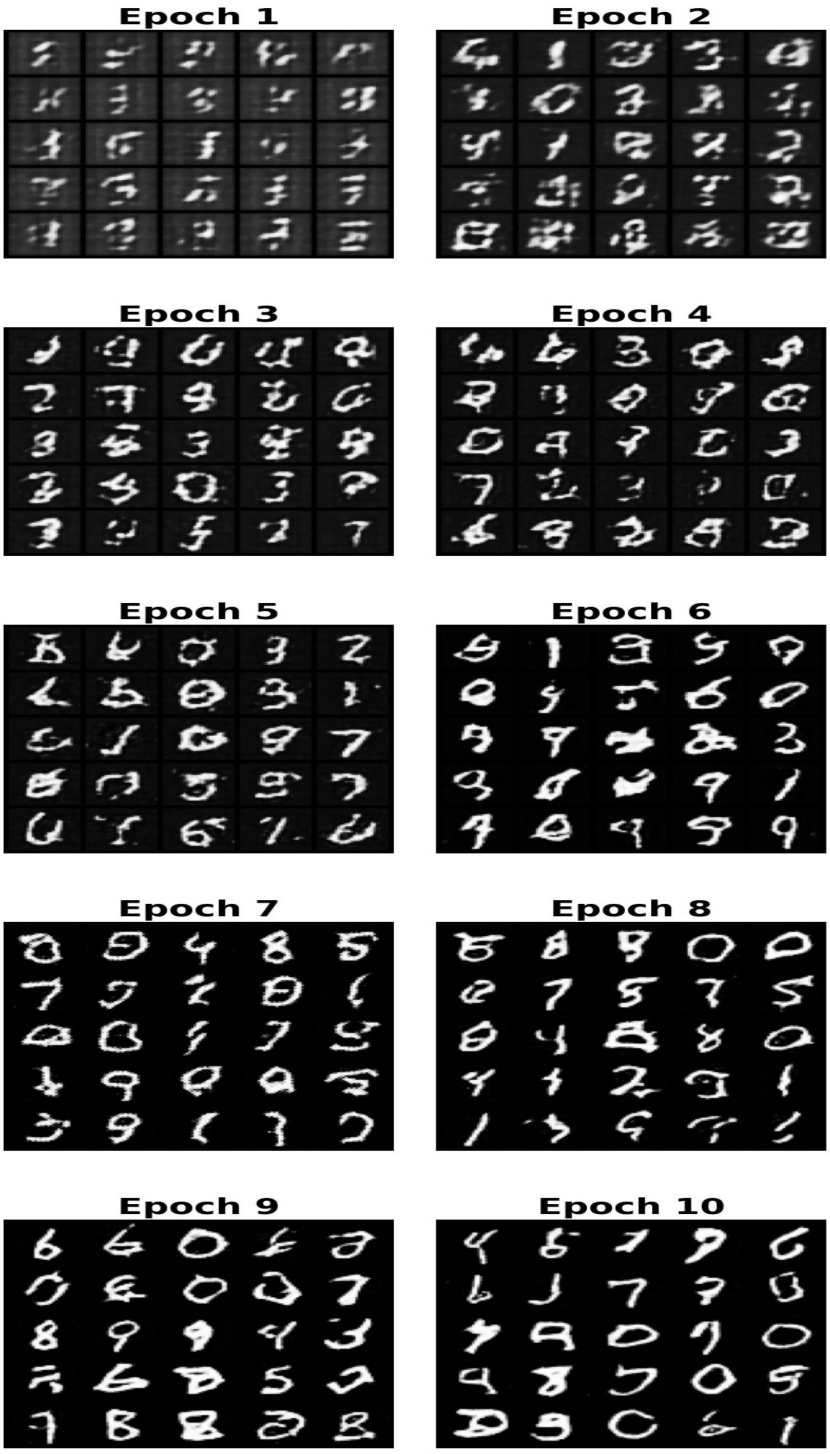


Figure 5.13: MNIST images randomly generated by a GAN trained with ADAM optimizer for the first 10 epochs.

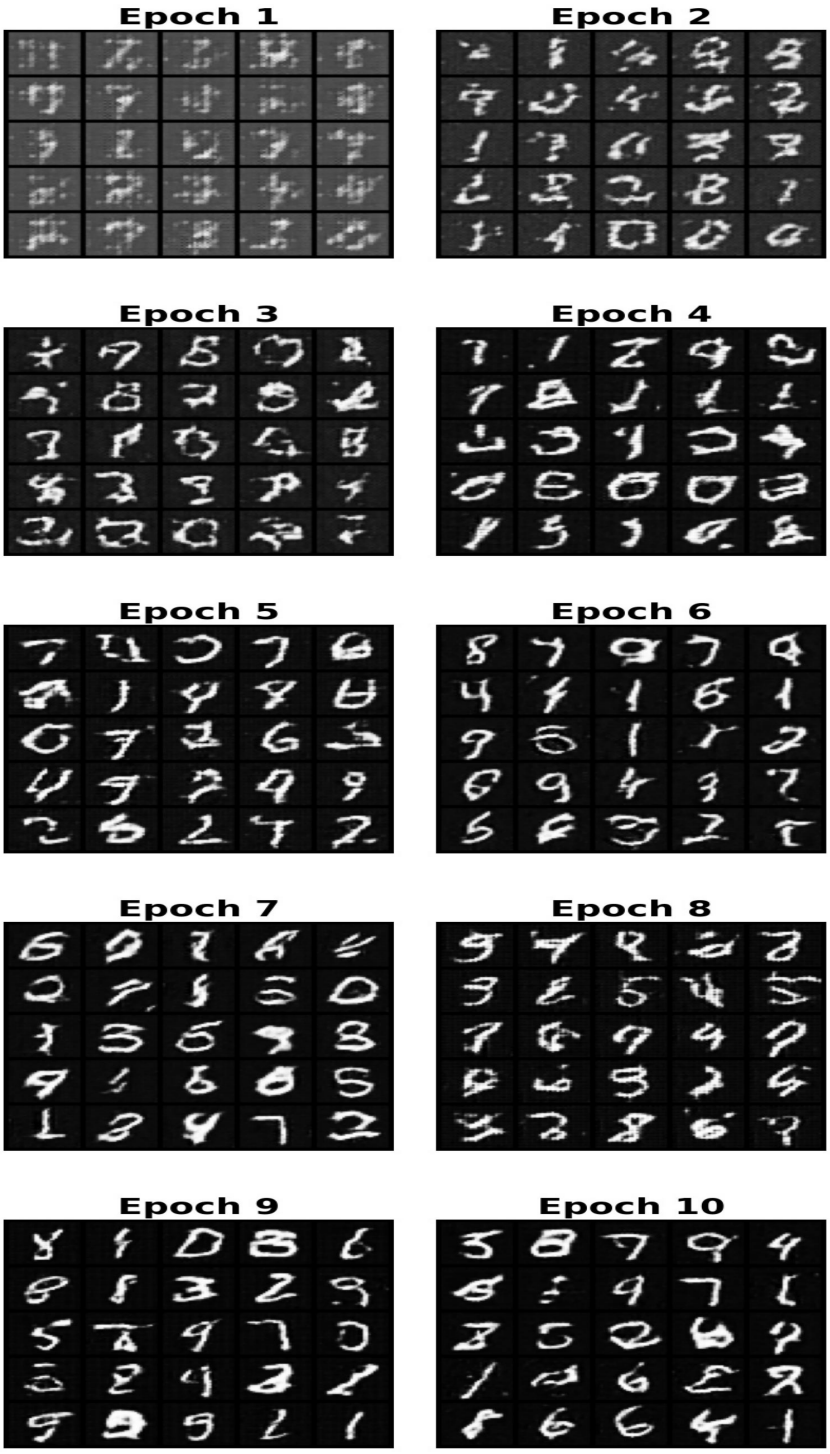


Figure 5.14: MNIST images randomly generated by a GAN trained with KFAC1 optimizer for the first 10 epochs.

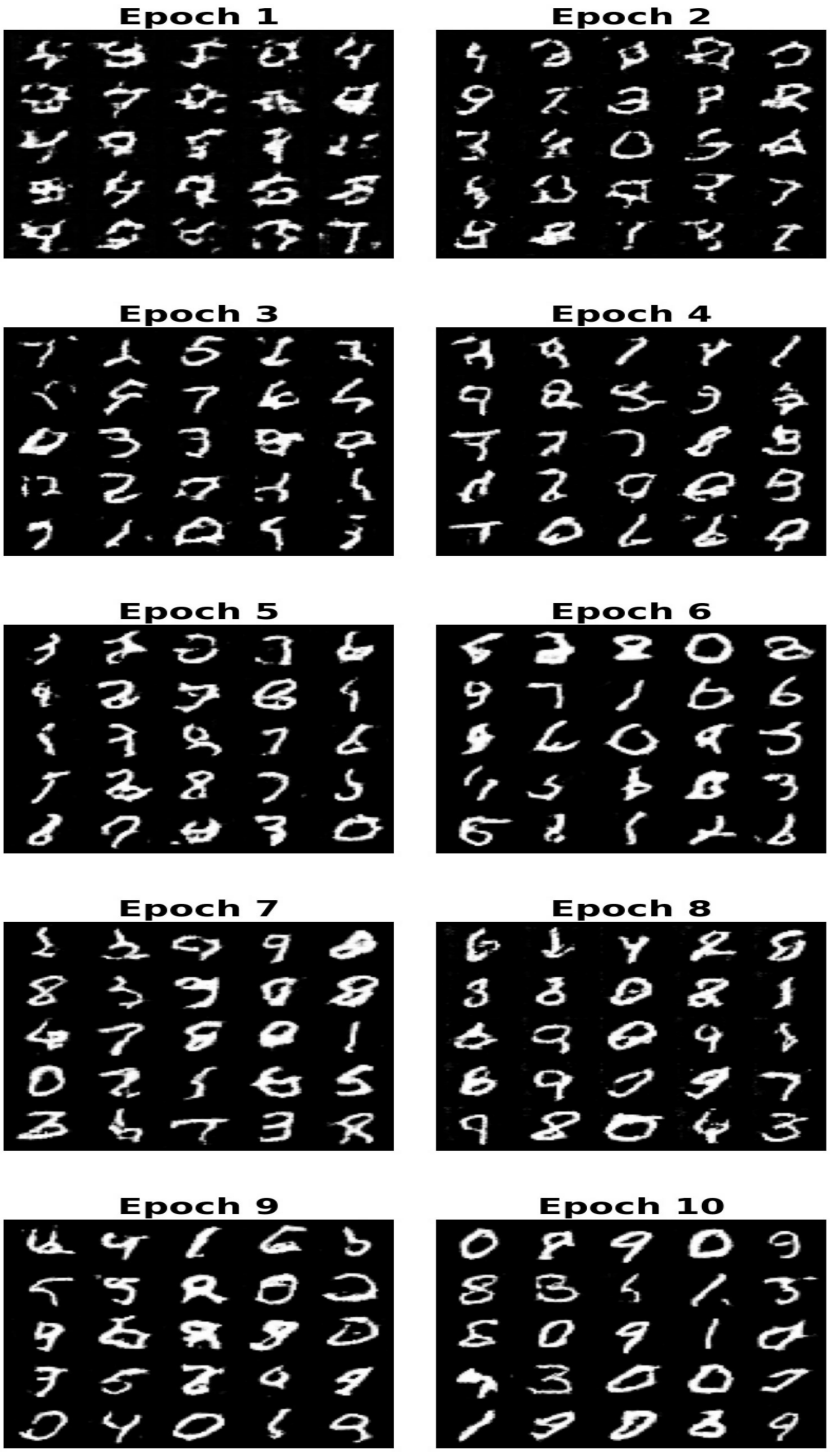


Figure 5.15: MNIST images randomly generated by a GAN trained with KFAC2 optimizer for the first 10 epochs.

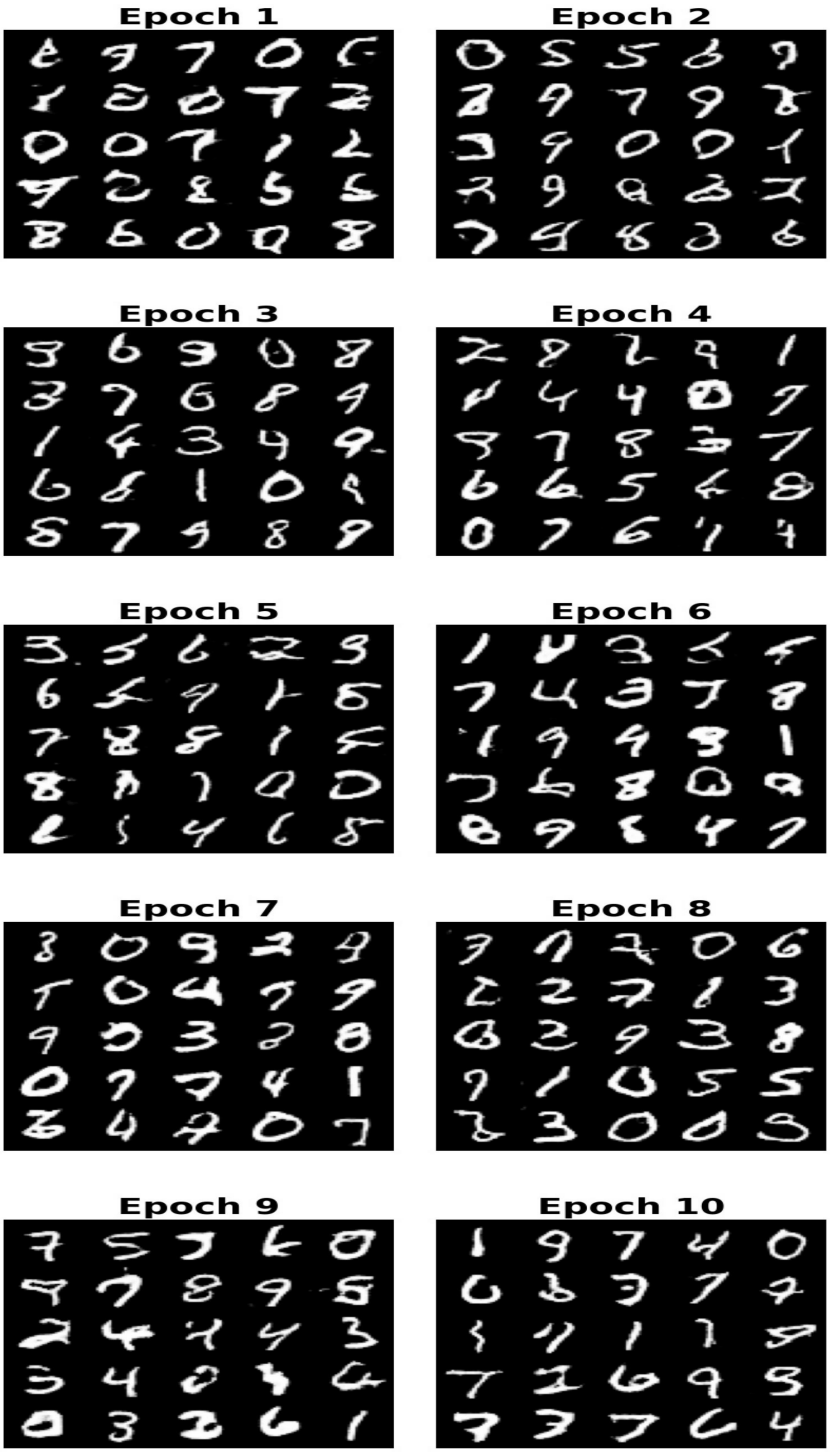


Figure 5.16: MNIST images randomly generated by a GAN trained with KFAC3 optimizer for the first 10 epochs.

Chapter 6

Conclusion and perspectives

Contents

6.1 Summary of key results

As stated in §1.2.3, the objective of this thesis was to bring some improvements to KFAC-related methods for DNNs. More precisely, it was about breaking free from the assumptions underlying the KFAC approximation and extending it to new network architectures. In a broader sense, we wished to design more robust and computationally efficient natural gradient methods that would ultimately converge faster than baseline optimizers such as SGD and ADAM.

Literature review. We started by conducting a review of the existing literature regarding DNN optimization, of which §2 is a summary. This demonstrated the crucial role of algorithms in the context of deep learning and highlighted the specific difficulties associated with training of DNNs. The overview also shed light on the limitations of first-order methods, such as their slow rate of convergence, their sensitivity to the choice of learning rate and their inherent inefficiency in navigating the parameter space.

This motivated us to consider second-order methods with a special emphasis on natural gradient methods, for which there are sound theoretical arguments for their being able to circumvent the difficulties encountered with first-order method. It soon appeared that the bottleneck lies in finding efficient approximations to the curvature matrix. The breakthrough achieved by KFAC for the Fisher matrix naturally led us to work on improving it further.

Alternative block-diagonal approximations of the FIM. In §3.2, we proposed four novel approximations to the diagonal blocks of the Fisher matrix for MLP, called *KPSVD, Deflation, Bi-diagonalization* and *KFAC-CORRECTED*. All of them take the form of a Kronecker product or the sum of two Kronecker products, but unlike KFAC, the factors are not inferred from any statistical assumption. Instead, they are defined more rigorously as minimizers of a least Frobenius-norm error between the original block and the approximant. The four minimization problems can all be solved efficiently using the Kronecker-product singular value decomposition. The numerical results obtained in §3.3 with three standard deep auto-encoders testified to the

superiority of the proposed methods over KFAC, both in terms of Fisher approximation quality and optimization speed of the objective function.

We also applied the same approximation paradigm to convolutional layers in §3.4, where three hypotheses are required for KFC. This time, for the sake of computational time, we had to make one single assumption in order to turn a double sum into a sum over a diagonal band. However, it turned out that even with quite large band widths, our methods did not generally outperform KFC.

Correction by coarse layer interactions. In §4.2, we put forward four coarse corrections to the KFAC approximation, called *Nicolaides, spectral, Krylov* and *residuals*. The first three are inspired from two-level domain decomposition, while the last one stems from a slightly different philosophy. All of them aim to enrich the KFAC matrix with macroscopic information about the interaction between the layers, so as to make it closer to the true FIM. Through several numerical tests in §4.3, we came to the rather disappointing conclusion that two-level KFAC methods does not generally outperform the original KFAC method in terms of optimization performance of the objective function. In other words, enhancing the solution of the linear system at each iteration (as carefully checked in §4.3.4) as a reference does not necessarily produce a more efficient optimization process.

To push the analysis to the extreme, we compared KFAC with the exact natural gradient (using the true and full FIM) and the block diagonal natural gradient in §4.4.1. Much to our surprise, KFAC does better than the block diagonal natural gradient, which in turn outperforms the exact natural gradient. These unexpected results corroborate Benzing's claim [14] that KFAC outperforms the exact natural gradient. So far, we do not have a clear explanation for this negative interlayer correlation which degrades the optimization performance. But at least this goes in the same direction as what occurred for two-level methods. Finally, in order to understand the superiority of KFAC over block natural gradient, the study undertaken in §4.4.2 showed that KFAC owes its success to its very peculiar regularization technique.

Application of KFAC to the training of GANs. In §5.2, we introduced an extension of the KFAC method to transposed convolution layers, which are core components of GAN models. This was successfully handled by recasting a transposed convolution as a traditional convolution. Numerical results obtained in §5.3.1 on three different deep convolutional auto-encoders demonstrated a superior performance of the proposed method compared to baseline optimizers such as SGD and Adam.

Moving on to GANs training, the experiments conducted in §5.3.2 revealed promising results regarding the effectiveness of the KFAC optimizer. Indeed, when the generator was trained with Adam optimizer and the discriminator with KFAC, the model achieved faster convergence, as indicated by a good FID score and the ability of the model to generate relevant images early in the training process. However, the experiments also highlighted a concern when both players (generator and discriminator) were trained with the KFAC optimizer: such a set-up resulted in an unstable performance of the model and sometimes led to worse performance compared to models where both players were trained with baseline optimizers like SGD or Adam. This underlines the challenge of effectively training GANs and calls for further investigations into better ways of incorporating curvature information.

6.2 Recommendations for future research

The insights gained from this work lay the groundwork for further exploration and offer potential directions for future research. Here, we outline some perspectives that might be considered.

Theoretical investigations. While this thesis and many previous works supplied compelling numerical results on the behavior of KFAC-related methods, there is a real need to comprehend such methods on a theoretical level. Future works could for instance focus on analyzing their convergence properties and assessing the impact of network architectures and activation functions, so as to solidify their theoretical foundation.

As a concrete case of study, it is most urgent to find a convincing explanation to the fact that KFAC, which is an approximation of the natural gradient, outperforms the latter in terms of optimization performance. It would also be highly valuable to develop arguments to account for the success of the heuristic damping technique used in the KFAC method.

Generalization properties. As was the case for almost every previous work related to natural gradient and KFAC methods [14, 17, 68, 112], the one in this thesis is limited to the optimization performance of the objective function. More precisely, all our endeavors were targeted at achieving the lowest value for the training loss. Of course, the question arises as to the generalization capacity of these methods. Since the study of generalization requires a different experimental framework [14, 187, 190], we leave it as a prospect.

Scalability and freedom from network architecture. The scalability of natural gradient methods to large-scale datasets and complex neural network architectures such as transformers and diffusion models is an important area for future research. Furthermore, KFAC and all methods based on a Kronecker block-diagonal approximation to a curvature matrix strongly depend on the network architecture. This has the shortcoming of users having to modify and to adapt these methods every time they switch to another network architecture. It would therefore be a major advance to design a new method that is free from the assumption about the nature of the network to be optimized, and that works fine for all models of interest.

Such an idea has recently been scrutinized. Indeed, by assuming that each block of the FIM corresponds to the covariance of a tensor normal distribution in the model, the TNT methods [139] proposes a Kronecker block-diagonal approximation to the FIM that has the advantage to be free from the structure of layers. However, their evaluation was limited to MLPs and CNNs. It would therefore be interesting to study this promising novel method for more complex architectures such as RNNs, transformers and diffusion models.

Bibliography

[1] R. Abdal, Y. Qin, and P. Wonka, *Image2StyleGAN: How to embed images into the StyleGAN latent space?*, in 2019 IEEE/CVF International Conference on Computer Vision, Seoul, South Korea, 27 Oct–2 Nov 2019, pp. 4431–4440, https://doi.org/10.1109/ICCV.2019.00453.

[2] S. M. Ali and S. D. Silvey, *A general class of coefficients of divergence of one distribution from another*, J. R. Stat. Soc.: Ser. B Methodol., 28 (1966), pp. 131–142, https://doi.org/10.1111/j.2517-6161.1966.tb00626.x.

[3] S.-I. Amari, *Natural gradient works efficiently in learning*, Neur. Comput., 10 (1998), pp. 251–276, https://doi.org/10.1162/089976698300017746.

[4] S.-I. Amari, *Information Geometry and Its Applications*, Applied Mathematical Sciences, Springer, Tokyo, 2016, https://doi.org/10.1007/978-4-431-55978-8.

[5] S.-I. Amari and H. Nagaoka, *Methods of Information Geometry*, vol. 191 of Translations of Mathematical Monographs, American Mathematical Society, Providence, Rhode Island, 2000, https://doi.org/10.1090/mmono/191.

[6] M. Arjovsky, S. Chintala, and L. Bottou, *Wasserstein generative adversarial networks*, in 34th International Conference on Machine Learning, D. Precup and Y. W. Teh, eds., vol. 70 of Proceedings in Machine Learning Research, Sydney, Australia, 6–11 August 2017, pp. 214–223, https://proceedings.mlr.press/v70/arjovsky17a.html.

[7] N. Ay, J. Jost, H. V. Lê, and L. Schwachhöfer, *Information Geometry*, vol. 64 of Ergebnisse der Mathematik und ihrer Grenzgebiete. 3. Folge / A Series of Modern Surveys in Mathematics, Springer, 2017, https://doi.org/10.1007/978-3-319-56478-4.

[8] J. Ba, R. Grosse, and J. Martens, *Distributed second-order optimization using Kronecker-factored approximations*, in 5th International Conference on Learning Representations, Conference Track Proceedings, Toulon, France, 24–26 Apr 2017, https://openreview.net/forum?id=SkkTMpjex.

[9] D. Bahdanau, K. Cho, and Y. Bengio, *Neural machine translation by jointly learning to align and translate*, in 3rd International Conference on Learning Representations, Conference Track Proceedings, Y. Bengio and Y. LeCun, eds., San Diego, California, May 2015, https://arxiv.org/abs/1409.0473.

[10] S. A. Barnett, *Convergence problems with generative adversarial networks*, master's thesis, Mathematical Institute, University of Oxford, 2018, https://arxiv.org/abs/1806.11382.

[11] J. Barzilai and J. M. Borwein, *Two-point step size gradient methods*, IMA J. Numer. Anal., 8 (1988), pp. 141–148, https://doi.org/10.1093/imanum/8.1.141.

[12] S. Becker and Y. LeCun, *Improving the convergence of back-propagation learning with second-order methods*, in Proceedings of the 1988 Connectionist Models Summer School, D. Touretzky, G. Hilton, and T. Sejnowski, eds., San Mateo, 1988, Morgan Kaufman, pp. 29–37, https://www.researchgate.net/publication/216792889_Improving_the_Convergence_of_Back-Propagation_Learning_with_Second-Order_Methods.

[13] Y. Bengio, L. Yao, G. Alain, and P. Vincent, *Generalized denoising auto-encoders as generative models*, in 27th Annual Conference on Neural Information Processing Systems, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, eds., vol. 26 of Advances in Neural Information Processing Systems, Lake Tahoe, Nevada, 5–10 Dec 2013, pp. 899–907, https://proceedings.neurips.cc/paper/2013/hash/559cb990c9dffd8675f6bc2186971dc2-Abstract.html.

[14] F. Benzing, *Gradient descent on neurons and its link to approximate second-order optimization*, in 39th International Conference on Machine Learning, K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, and S. Sabato, eds., vol. 162 of Proceedings of Machine Learning Research, Baltimore, Maryland, 17–23 Jul 2022, https://proceedings.mlr.press/v162/benzing22a.html.

[15] H. Berard, G. Gidel, A. Almahairi, P. Vincent, and S. Lacoste-Julien, *A closer look at the optimization landscapes of generative adversarial networks*, in 8th International Conference on Learning Representations, Conference Track Proceedings, virtual, 26 Apr–1 May 2020, https://openreview.net/forum?id=HJeVnCEKwH.

[16] A. Borji, *Pros and cons of GAN evaluation measures*, Comput. Vis. Image Underst., 179 (2019), pp. 41–65, https://doi.org/10.1016/j.cviu.2018.10.009.

[17] A. Botev, H. Ritter, and D. Barber, *Practical Gauss-Newton optimisation for deep learning*, in 34th International Conference on Machine Learning, D. Precup and Y. W. Teh, eds., vol. 70 of Proceedings of Machine Learning Research, Sydney, Australia, 6–11 Aug 2017, pp. 557–565, https://proceedings.mlr.press/v70/botev17a.html.

[18] L. Bottou, F. E. Curtis, and J. Nocedal, *Optimization methods for large-scale machine learning*, SIAM Review, 60 (2018), pp. 223–311, https://doi.org/10.1137/16M1080173.

[19] L. Bottou and Y. Le Cun, *On-line learning for very large data sets*, Appl. Stoch. Models Bus. Ind., 21 (2005), pp. 137–151, https://doi.org/10.1002/asmb.538.

[20] N. Boumal, *An Introduction to Optimization on Smooth Manifolds*, Cambridge University Press, Cambridge, 2023, https://doi.org/10.1017/9781009166164.

[21] A. Brock, J. Donahue, and K. Simonyan, *Large scale GAN training for high fidelity natural image synthesis*, in 7th International Conference on Learning Representations, Conference Track Proceedings, New Orleans, Louisiana, 6–9 May 2019, https://openreview.net/forum?id=B1xsqj09Fm.

[22] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, *Language models are few-shot learners*, in 34th Conference on Neural Information Processing Systems, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, eds., vol. 33 of Advances in Neural Information Processing Systems, virtual, 6–12 Dec 2020, pp. 1877–1901, https://proceedings.neurips.cc/paper_files/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html.

[23] C. G. Broyden, *The convergence of a class of double-rank minimization algorithms 1. General considerations*, IMA J. Appl. Math., 6 (1970), pp. 76–90, https://doi.org/10.1093/imamat/6.1.76.

[24] R. H. Byrd, G. M. Chin, and J. Nocedal, *Sample size selection in optimization methods for machine learning*, Math. Program., 134 (2012), pp. 127—155, https://doi.org/10.1007/s10107-012-0572-5.

[25] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu, *A limited memory algorithm for bound constrained optimization*, SIAM J. Sci. Comp., 16 (1995), pp. 1190–1208, https://doi.org/10.1137/0916069.

[26] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, *End-to-end object detection with transformers*, in 16th European Conference on Computer Vision, A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, eds., virtual, 23–28 Aug 2020, pp. 213–229, https://doi.org/10.1007/978-3-030-58452-8_13.

[27] K. Chellapilla, S. Puri, and P. Y. Simard, *High performance convolutional neural networks for document processing*, in 10th International Workshop on Frontiers in Handwriting Recognition, G. Lorette, H. Bunke, and L. Schomaker, eds., La Baule, France, 23–26 Oct 2006, https://hal.inria.fr/inria-00112631.

[28] T. Chen, S. Saxena, L. Li, D. J. Fleet, and G. Hinton, *Pix2seq: A language modeling framework for object detection*, 2022, https://arxiv.org/abs/2109.10852.

[29] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio, *On the properties of neural machine translation: Encoder-decoder approaches*, in Proceedings of the 8th Workshop on Syntax, Semantics and Structure in Statistical Translation (SSST 2014), D. Wu, M. Carpuat, X. Carreras, and E. M. Vecchi, eds., Doha, Qatar, 25 Oct 2014, Association for Computational Linguistics, pp. 103–111, https://doi.org/10.3115/v1/W14-4012.

[30] J. Chung, C. Gülçehre, K. Cho, and Y. Bengio, *Empirical evaluation of gated recurrent neural networks on sequence modeling*, 2014, https://arxiv.org/abs/1412.3555.

[31] A. R. Conn, N. I. M. Gould, and P. L. Toint, *Trust-Region Methods*, MOS-SIAM Series on Optimization, SIAM, Philadelphia, 2000, https://doi.org/10.1137/1.9780898719857.

[32] H. Cramér, *Mathematical Methods of Statistics*, vol. 9 of Princeton Mathematical Series, Princeton University Press, Princeton, 1946, https://doi.org/10.1515/9781400883868.

[33] B. C. Csáji, *Approximation with artificial neural networks*, master's thesis, Faculty of Sciences, Eötvös Loránd University, Budapest, Hungary, 2001, https://www.researchgate.net/publication/357888472_Approximation_with_Artificial_Neural_Networks.

[34] A. Defazio, F. Bach, and S. Lacoste-Julien, *SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives*, in 28th Conference on Neural Information Processing Systems, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, eds., vol. 27 of Advances in Neural Information Processing Systems, Montreal, Canada, 8–13 Dec 2014, https://proceedings.neurips.cc/paper/2014/hash/ede7e2b6d13a41ddf9f4bdef84fdc737-Abstract.html.

[35] R. S. Dembo, S. C. Eisenstat, and T. Steihaug, *Inexact Newton methods*, SIAM J. Numer. Anal., 19 (1982), pp. 400–408, https://doi.org/10.1137/0719025.

[36] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and F.-F. Li, *ImageNet: A large-scale hierarchical image database*, in 2009 IEEE Conference on Computer Vision and Pattern Recognition, Miami, Florida, 20–25 Jun 2009, pp. 248–255, https://doi.org/10.1109/CVPR.2009.5206848.

[37] J. E. Dennis and R. B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, vol. 16 of Classics in Applied Mathematics, SIAM, Philadelphia, 1996, https://doi.org/10.1137/1.9781611971200.

[38] G. Desjardins, K. Simonyan, R. Pascanu, and K. Kavukcuoglu, *Natural neural networks*, in 29th Conference on Neural Information Processing Systems, vol. 28 of Advances in Neural Information Processing Systems, Montreal, Canada, 7–12 Dec 2015, pp. 2071–2079, https://papers.nips.cc/paper_files/paper/2015/hash/2de5d16682c3c35007e4e92982f1a2ba-Abstract.html.

[39] P. Deuflhard and G. Heindl, *Affine invariant convergence theorems for Newton's method and extensions to related methods*, SIAM J. Numer. Anal., 16 (1979), pp. 1–10, https://doi.org/10.1137/0716001.

[40] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, *BERT: Pre-training of deep bidirectional transformers for language understanding*, in 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), Minneapolis, Minnesota, Jun 2019, pp. 4171–4186, https://doi.org/10.18653/v1/N19-1423.

[41] B. Ding, H. Qian, and J. Zhou, *Activation functions and their characteristics in deep neural networks*, in 2018 Chinese Control And Decision Conference, Shenyang, China, 9–11 Jun 2018, pp. 1836–1841, https://doi.org/10.1109/CCDC.2018.8407425.

[42] V. Dolean, P. Jolivet, and F. Nataf, *An Introduction to Domain Decomposition Methods: Algorithms, Theory, and Parallel Implementation*, SIAM, Philadelphia, 2015, https://doi.org/10.1137/1.9781611974065.

[43] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, *An image is worth* 16×16 *words: Transformers for image recognition at scale*, in 9th International Conference on Learning Representations, Conference Track Proceedings, virtual, 3–7 May 2021, https://openreview.net/forum?id=YicbFdNTTy.

[44] T. Dozat, *Incorporating Nesterov momentum into Adam*, in 4th International Conference on Learning Representations, Conference Track Proceedings, San Juan, Puerto Rico, 2–4 May 2016, https://openreview.net/forum?id=OM0jvwB8jIp57ZJjtNEZ.

[45] J. Duchi, E. Hazan, and Y. Singer, *Adaptive subgradient methods for online learning and stochastic optimization*, J. Mach. Learn. Res., 12 (2011), pp. 2121–2159, https://doi.org/10.5555/1953048.2021068.

[46] V. Dumoulin and F. Visin, *A guide to convolution arithmetic for deep learning*, 2016, https://arxiv.org/abs/1603.07285.

[47] W. J. Duncan, *Some devices for the solution of large sets of simultaneous equations (with an appendix on the reciprocation of partitioned matrices)*, Lond. Edinb. Dubl. Phil. Mag., 35 (1944), pp. 660–670, https://doi.org/10.1080/14786444408520897.

[48] R. Durall, A. Chatzimichailidis, P. Labus, and J. Keuper, *Combating mode collapse in GAN training: An empirical analysis using Hessian eigenvalues*, 2020, https://arxiv.org/abs/2012.09673.

[49] Y. Fang, B. Liao, X. Wang, J. Fang, J. Qi, R. Wu, J. Niu, and W. Liu, *You only look at one sequence: Rethinking transformer in vision through object detection*, in 35th Conference on Neural Information Processing Systems, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. S. Liang, and J. W. Vaughan, eds., vol. 34 of Advances in Neural Information Processing Systems, virtual, 6–14 Dec 2021, pp. 26183–26197, https://proceedings.neurips.cc/paper/2021/hash/dc912a253d1e9ba40e2c597ed2376640-Abstract.html.

[50] T. Fiez, B. Chasnov, and L. J. Ratliff, *Convergence of learning dynamics in Stackelberg games*, 2019, https://arxiv.org/abs/1906.01217.

[51] T. Fiez and L. Ratliff, *Gradient descent-ascent provably converges to strict local minmax equilibria with a finite timescale separation*, 2020, https://arxiv.org/abs/2009.14820.

[52] R. Fletcher, *A new approach to variable metric algorithms*, Comput. J., 13 (1970), pp. 317–322, https://doi.org/10.1093/comjnl/13.3.317.

[53] K. Gao, Z.-H. Huang, X. Liu, M. Wang, S. Wang, Z. Wang, D. Xu, and F. Yu, *Eigenvalue-corrected natural gradient based on a new approximation*, Asia Pac. J. Oper. Res., 40 (2023), p. 2340005, https://doi.org/10.1142/S0217595923400055.

[54] K.-X. Gao, X.-L. Liu, Z.-H. Huang, M. Wang, Z. Wang, D. Xu, and F. Yu, *A trace-restricted Kronecker-factored approximation to natural gradient*, in 35th AAAI Conference on Artificial Intelligence, vol. 35 of AAAI-21 Technical Tracks, virtual, 2–9 Feb 2021, https://doi.org/10.1609/aaai.v35i9.16921.

[55] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin, *Convolutional sequence to sequence learning*, in 34th International Conference on Machine Learning, D. Precup and Y. W. Teh, eds., vol. 70 of Proceedings of Machine Learning Research, Sydney, Australia, 06–11 Aug 2017, pp. 1243–1252, https://proceedings.mlr.press/v70/gehring17a.html.

[56] T. George, C. Laurent, X. Bouthillier, N. Ballas, and P. Vincent, *Fast approximate natural gradient descent in a Kronecker-factored eigenbasis*, in 32th Conference on Neural Information Processing Systems, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, eds., vol. 31 of Advances in Neural Information Processing Systems, Montreal, Canada, 3–8 Dec 2018, pp. 9550–9560, https://papers.nips.cc/paper_files/paper/2018/hash/48000647b315f6f00f913caa757a70b3-Abstract.html.

[57] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik, *Rich feature hierarchies for accurate object detection and semantic segmentation*, in 2014 IEEE Conference on Computer Vision and Pattern Recognition, Columbus, Ohio, 23–28 Jun 2014, pp. 580–587, https://doi.org/10.1109/CVPR.2014.81.

[58] X. Glorot and Y. Bengio, *Understanding the difficulty of training deep feedforward neural networks*, in 13th International Conference on Artificial Intelligence and Statistics, Y. W. Teh and M. Titterington, eds., vol. 9 of Proceedings of Machine Learning Research, Sardinia, Italy, 13–15 May 2010, pp. 249–256, https://proceedings.mlr.press/v9/glorot10a.html.

[59] X. Glorot, A. Bordes, and Y. Bengio, *Deep sparse rectifier neural networks*, in 14th International Conference on Artificial Intelligence and Statistics, G. Gordon, D. Dunson, and M. Dudík, eds., vol. 15 of Proceedings of Machine Learning Research, Fort Lauderdale, Florida, 11–13 Apr 2011, pp. 315–323, https://proceedings.mlr.press/v15/glorot11a.html.

[60] D. Goldfarb, *A family of variable-metric methods derived by variational means*, Math. Comp., 24 (1970), pp. 23–26, https://doi.org/10.1090/S0025-5718-1970-0258249-6.

[61] D. Goldfarb, Y. Ren, and A. Bahamou, *Practical quasi-Newton methods for training deep neural networks*, in 34th Conference on Neural Information Processing Systems, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, eds., vol. 33 of Advances in Neural Information Processing Systems, virtual, 6–12 Dec 2020, pp. 2386–2396, https://proceedings.neurips.cc/paper/2020/hash/192fc044e74dffea144f9ac5dc9f3395-Abstract.html.

[62] G. Golub and W. Kahan, *Calculating the singular values and pseudo-inverse of a matrix*, J. Soc. Ind. Appl. Math.: Ser. B, Numer. Anal., 2 (1965), pp. 205–224, https://doi.org/10.1137/0702016.

[63] G. H. Golub and C. F. van Loan, *Matrix Computations*, Johns Hopkins University Press, Baltimore, Maryland, 1996.

[64] I. Goodfellow, *NIPS 2016 tutorial: Generative adversarial networks*, 2016, https://arxiv.org/abs/1701.00160.

[65] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, Adaptive Computation and Machine Learning series, MIT Press, Cambridge, Massachusetts, 2016, http://www.deeplearningbook.org.

[66] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, *Generative adversarial nets*, in 28th Conference on Neural Information Processing Systems, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, eds., vol. 27 of Advances in Neural Information Processing Systems, Montreal, Canada, 2014, pp. 2672–2680, https://papers.nips.cc/paper_files/paper/2014/hash/5ca3e9b122f61f8f06494c97b1afccf3-Abstract.html.

[67] P. Goyal, P. Dollár, R. B. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, *Accurate, large minibatch SGD: Training ImageNet in 1 hour*, 2017, https://arxiv.org/abs/1706.02677.

[68] R. Grosse and J. Martens, *A Kronecker-factored approximate Fisher matrix for convolution layers*, in 33rd International Conference on Machine Learning, M. F. Balcan and K. Q. Weinberger, eds., vol. 48 of Proceedings of Machine Learning Research, New York, 19–24 Jun 2016, pp. 573–582, http://proceedings.mlr.press/v48/grosse16.html.

[69] R. Grosse and R. Salakhutdinov, *Scaling up natural gradient by sparsely factorizing the inverse Fisher matrix*, in 32nd International Conference on Machine Learning, F. Bach and D. Blei, eds., vol. 37 of Proceedings in Machine Learning Research, Lille, France, 2015, pp. 2304–2313, http://proceedings.mlr.press/v37/grosse15.html.

[70] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, *Improved training of Wasserstein GANs*, in 31st Conference on Neural Information Processing Systems, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds., vol. 30 of Advances in Neural Information Processing Systems, Long Beach, California, 2017, Curran Associates, Inc., pp. 5767–5777, https://papers.nips.cc/paper_files/paper/2017/hash/892c3b1c6dccd52936e27cbd0ff683d6-Abstract.html.

[71] X. Guo, X. Liu, E. Zhu, and J. Yin, *Deep clustering with convolutional autoencoders*, in Neural Information Processing, D. Liu, S. Xie, Y. Li, D. Zhao, and E.-S. M. El-Alfy, eds., vol. 10635 of Lecture Notes in Computer Science, Guangzhou, China, 14–18 Nov 2017, Springer, pp. 373–382, https://doi.org/10.1007/978-3-319-70096-0_39.

[72] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick, *Mask R-CNN*, in 2017 IEEE International Conference on Computer Vision, Venice, Italy, 22–29 Oct 2017, pp. 2980–2988, https://doi.org/10.1109/ICCV.2017.322.

[73] K. He, X. Zhang, S. Ren, and J. Sun, *Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification*, in 2015 IEEE International Conference on Computer Vision, Santiago, Chile, 7–13 Dec 2015, pp. 1026–1034, https://doi.org/10.1109/ICCV.2015.123.

[74] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, in 2016 IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, Nevada, 27–30 Jun 2016, pp. 770–778, https://doi.org/10.1109/CVPR.2016.90.

[75] T. Heskes, *On "natural" learning and pruning in multilayered perceptrons*, Neur. Comput., 12 (2000), pp. 881–901, https://doi.org/10.1162/089976600300015637.

[76] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, *GANs trained by a two time-scale update rule converge to a local Nash equilibrium*, in 31st Conference on Neural Information Processing Systems, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds., vol. 30 of Advances in Neural Information Processing Systems, Long Beach, California, 4–9 Dec 2017, Curran Associates, Inc., pp. 6629–6640, https://papers.nips.cc/paper_files/paper/2017/hash/8a1d694707eb0fefe65871369074926d-Abstract.html.

[77] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury, *Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups*, IEEE Signal Process. Mag., 29 (2012), pp. 82–97, https://doi.org/10.1109/MSP.2012.2205597.

[78] G. E. Hinton and R. R. Salakhutdinov, *Reducing the dimensionality of data with neural netwoks*, Science, 313 (2006), pp. 504–507, https://doi.org/10.1126/science.1127647.

[79] S. Hochreiter and J. Schmidhuber, *Long short-term memory*, Neur. Comput., 9 (1997), pp. 1735–1780, https://doi.org/10.1162/neco.1997.9.8.1735.

[80] G. Huang, Z. Liu, and K. Q. Weinberger, *Densely connected convolutional networks*, in 2017 IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, Hawai, 21–26 Jul 2017, pp. 2261–2269, https://doi.org/10.1109/CVPR.2017.243.

[81] S. Iizuka, E. Simo-Serra, and H. Ishikawa, *Globally and locally consistent image completion*, ACM Trans. Graph., 36 (2017), https://doi.org/10.1145/3072959.3073659.

[82] S. Jelassi, D. Dobre, A. Mensch, Y. Li, and G. Gidel, *Dissecting adaptive methods in GANs*, 2022, https://arxiv.org/abs/2210.04319.

[83] C. Jin, P. Netrapalli, and M. I. Jordan, *What is local optimality in nonconvex-nonconcave minimax optimization?*, in 37th International Conference on Machine Learning, H. Daumé III and A. Singh, eds., vol. 119 of Proceedings of Machine Learning Research, virtual, 2020, pp. 4880–4889, https://proceedings.mlr.press/v119/jin20e.html.

[84] R. Johnson and T. Zhang, *Accelerating stochastic gradient descent using predictive variance reduction*, in 27th Conference on Neural Information Processing Systems, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, eds., vol. 26 of Advances in Neural Information Processing Systems, Lake Tahoe, Nevada, 5–10 Dec 2013, pp. 315–323, https://papers.nips.cc/paper/2013/hash/ac1dd209cbcc5e5d1c6e28598e8cbbe8-Abstract.html.

[85] A. Karpathy, *The unreasonable effectiveness of recurrent neural networks*. Blog post, 2015, http://karpathy.github.io/2015/05/21/rnn-effectiveness.

[86] T. Karras, S. Laine, and T. Aila, *A style-based generator architecture for generative adversarial networks*, in 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Long Beach, California, 15–20 Jun 2019, pp. 4396–4405, https://doi.org/10.1109/CVPR.2019.00453.

[87] T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, and T. Aila, *Analyzing and improving the image quality of StyleGAN*, in 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Seattle, Washington, 13–19 Jun 2020, pp. 8107–8116, https://doi.org/10.1109/CVPR42600.2020.00813.

[88] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, in 3rd International Conference on Learning Representations, Conference Track Proceedings, Y. Bengio and Y. LeCun, eds., San Diego, California, 7–9 May 2015, http://arxiv.org/abs/1412.6980.

[89] D. P. Kingma and M. Welling, *Auto-encoding variational Bayes*, in 2nd International Conference on Learning Representations, Conference Track Proceedings, Banff, Canada, 14–16 Apr 2014, https://openreview.net/forum?id=33X9fd2-9FyZd.

[90] A. Koroko, A. Anciaux-Sedrakian, I. Ben Gharbia, V. Garès, M. Haddou, and Q. H. Tran, *Efficient approximations of the Fisher matrix in neural networks using Kronecker product singular value decomposition*, 2022, https://arxiv.org/abs/2201.10285.

[91] A. Koroko, A. Anciaux-Sedrakian, I. Ben Gharbia, V. Garès, M. Haddou, and Q. H. Tran, *Analysis and comparison of two-level KFAC methods for training deep neural networks*, 2023, https://arxiv.org/abs/2303.18083.

[92] A. Krizhevsky, *Learning multiple layers of features from tiny images*, tech. report, University of Toronto, Toronto, Ontario, 2009, https://www.cs.toronto.edu/~kriz/cifar.html.

[93] A. Krizhevsky, I. Sutskever, and G. E. Hinton, *ImageNet classification with deep convolutional neural networks*, in 26th Conference on Neural Information Processing Systems, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds., vol. 25 of Advances in Neural Information Processing System, Lake Tahoe, Nevada, 3–8 Dec 2012, pp. 1097–1105, https://papers.nips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html.

[94] F. Kunstner, P. Hennig, and L. Balles, *Limitations of the empirical Fisher approximation for natural gradient descent*, in 33th Conference on Neural Information Processing Systems, H. Wallach, H. Larochelle, A. Beygelzimer, F. Alché-Buc, E. Fox, and R. Garnett, eds., vol. 32 of Advances in Neural Information Processing Systems, Vancouver, Canada, 8–14 Dec 2019, pp. 4156–4167, https://proceedings.neurips.cc/paper/2019/hash/46a558d97954d0692411c861cf78ef79-Abstract.html.

[95] Y. Le Cun, L. Bottou, Y. Bengio, and P. Haffner, *Gradient-based learning applied to document recognition*, IEEE Proc., 86 (1998), pp. 2278–2324, https://doi.org/10.1109/5.726791.

[96] N. Le Roux, P.-A. Manzagol, and Y. Bengio, *Topmoumoute online natural gradient algorithm*, in 21st Conference on Neural Information Processing Systems, J. Platt, D. Koller, Y. Singer, and S. Roweis, eds., vol. 20 of Advances in Neural Information Processing Systems, Vancouver, Canada, 3–6 Dec 2007, pp. 849–856, https://proceedings.neurips.cc/paper_files/paper/2007/hash/9f61408e3afb633e50cdf1b20de6f466-Abstract.html.

[97] Y. LeCun, *A theoretical framework for back-propagation*, in 1988 Connectionist Models Summer School, G. Hinton and T. Sejnowski, eds., vol. 1, Pittsburg, Philadelphia, 1988, Morgan Kaufmann, pp. 21–28, https://www.researchgate.net/publication/2360531_A_Theoretical_Framework_for_Back-Propagation.

[98] Y. LeCun and Y. Bengio, *Convolutional networks for images, speech, and time-series*, in The handbook of brain theory and neural networks, M. A. Arbib, ed., MIT Press, Cambridge, Massachusetts, 1995, pp. 255–258, http://www.iro.umontreal.ca/~lisa/pointeurs/handbook-convo.pdf.

[99] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, *Backpropagation applied to handwritten zip code recognition*, Neur. Comput., 1 (1989), pp. 541–551, https://doi.org/10.1162/neco.1989.1.4.541.

[100] Y. LeCun, P. Haffner, L. Bottou, and Y. Bengio, *Object recognition with gradient-based learning*, in Shape, Contour and Grouping in Computer Vision, D. A. Forsyth, J. L. Mundy, V. di Gesu, and R. Cipolla, eds., vol. 1681 of Lecture Notes in Computer Science, Springer Verlag, 1999, pp. 319–345, https://doi.org/10.1007/3-540-46805-6_19.

[101] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, *Efficient backprop*, in Neural Networks: Tricks of the Trade: Second Edition, G. Montavon, G. B. Orr, and K.-R. Müller, eds., Springer Berlin Heidelberg, Berlin, Heidelberg, 1998, pp. 9–48, https://doi.org/10.1007/978-3-642-35289-8_3.

[102] C. Ledig, L. Theis, F. Huszar, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, and W. Shi, *Photo-realistic single image super-resolution using a generative adversarial network*, in 2017 IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, Hawai, 21–26 Jul 2017, pp. 105–114, https://doi.org/10.1109/CVPR.2017.19.

[103] K. Levenberg, *A method for the solution of certain non-linear problems in least squares*, Quart. Appl. Math., 2 (1944), pp. 164–168, https://doi.org/10.1090/qam/10666.

[104] J. Liang, Y. Xu, C. Bao, Y. Quan, and H. Ji, *Barzilai-Borwein-based adaptive learning rate for deep learning*, Patt. Recogn. Letters, 128 (2019), pp. 197–203, https://doi.org/10.1016/j.patrec.2019.08.029.

[105] D. C. Liu and J. Nocedal, *On the limited memory BFGS method for large scale optimization*, Math. Prog., 45 (1989), pp. 503–528, https://doi.org/10.1007/BF01589116.

[106] M. Liu, Y. Mroueh, J. Ross, W. Zhang, X. Cui, P. Das, and T. Yang, *Towards better understanding of adaptive gradient algorithms in generative adversarial nets*, in 8th International Conference on Learning Representations, Conference Track Proceedings, virtual, 26 Apr–1 May 2020, https://openreview.net/forum?id=SJxIm0VtwH.

[107] D. W. Marquardt, *An algorithm for least-squares estimation of nonlinear parameters*, SIAM J. Appli. Math., 11 (1963), pp. 431–441, https://doi.org/10.1137/0111030.

[108] J. Martens, *Deep learning via Hessian-free optimization*, in 27th International Conference on Machine Learning, J. Fürnkranz and T. Joachims, eds., Haifa, Israel, 21–24 Jun 2010, pp. 735–742, https://icml.cc/Conferences/2010/papers/458.pdf.

[109] J. Martens, *Second-order optimization for neural networks*, PhD thesis, University of Toronto, Ontario, Canada, 2016, http://hdl.handle.net/1807/71732.

[110] J. Martens, *New insights and perspectives on the natural gradient method*, J. Mach. Learn. Res., 21 (2020), pp. 1–76, https://jmlr.csail.mit.edu/papers/v21/17-678.html.

[111] J. Martens, J. Ba, and M. Johnson, *Kronecker-factored curvature approximations for recurrent neural networks*, in 6th International Conference on Learning Representations, Conference Track Proceedings, Vancouver, Canada, 30 Apr–3 May 2018, https://openreview.net/forum?id=HyMTkQZAb.

[112] J. Martens and R. Grosse, *Optimizing neural networks with Kronecker-factored approximate curvature*, in 32nd International Conference on Machine Learning, F. Bach and D. Blei, eds., vol. 37 of Proceedings in Machine Learning Research, Lille, France, 6–11 Jul 2015, pp. 2408–2417, http://proceedings.mlr.press/v37/martens15.html.

[113] J. Martens and I. Sutskever, *Learning recurrent neural networks with Hessian-free optimization*, in 28th International Conference on Machine Learning, L. Getoor and T. Scheffer, eds., Bellevue, Washington, 28 Jun–2 Jul 2011, pp. 1033–1040, https://icml.cc/2011/papers/532_icmlpaper.pdf.

[114] M. Mirza and S. Osindero, *Conditional generative adversarial nets*, 2014, https://arxiv.org/abs/1411.1784.

[115] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida, *Spectral normalization for generative adversarial networks*, in 6th International Conference on Learning Representations, Conference Track Proceedings, Vancouver, Canada, 30 Apr–3 May 2018, https://openreview.net/forum?id=B1QRgziT-.

[116] N. Murata, *A statistical study of on-line learning*, in Online Learning and Neural Networks, D. Saad, ed., Cambridge University Press, Cambridge, 1998, ch. 4, pp. 63–92, https://doi.org/10.1017/CBO9780511569920.005.

[117] F. Nataf, H. Xiang, V. Dolean, and N. Spillane, *A coarse space construction based on local Dirichlet-to-Neumann maps*, SIAM J. Sci. Comput., 33 (2011), pp. 1623–1642, https://doi.org/10.1137/100796376.

[118] Y. E. Nesterov, *A method for solving the convex programming problem with convergence rate* $O(1/k^2)$, Dokl. Akad. Nauk SSSR, 269 (1983), pp. 543–547, http://mi.mathnet.ru/dan4600.

[119] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Ng, *Reading digits in natural images with unsupervised feature learning*, in NIPS Workshop on Deep Learning and Unsupervised Feature Learning, Granada, December 2011, http://ufldl.stanford.edu/housenumbers/nips2011_housenumbers.pdf.

[120] X. L. Nguyen, M. J. Wainwright, and M. I. Jordan, *Estimating divergence functionals and the likelihood ratio by convex risk minimization*, IEEE Trans. Inf. Theory, 56 (2010), pp. 5847–5861, https://doi.org/10.1109/TIT.2010.2068870.

[121] R. A. Nicolaides, *Deflation of conjugate gradients with applications to boundary value problems*, SIAM J. Numer. Anal., 24 (1987), pp. 355–365, https://doi.org/10.1137/0724027.

[122] F. Nielsen, *The many faces of information geometry*, Not. Am. Math. Soc., 69 (2022), pp. 36–45, https://doi.org/10.1090/noti2403.

[123] J. Nocedal and S. J. Wright, *Numerical Optimization*, Springer Series in Operations Research and Financial Engineering, Springer, New York, 2006, https://doi.org/10.1007/978-0-387-40065-5.

[124] Y. Ollivier, *Riemannian metrics for neural networks I: feedforward networks*, Inform. Infer., 4 (2015), pp. 108–153, https://doi.org/10.1093/imaiai/iav006.

[125] Y. Ollivier, L. Arnold, A. Auger, and N. Hansen, *Information-geometric optimization algorithms: A unifying picture via invariance principles*, J. Mach. Learn. Res., 18 (2017), pp. 1–65, https://doi.org/10.5555/3122009.3122027.

[126] K. Osawa, Y. Tsuji, Y. Ueno, A. Naruse, R. Yokota, and S. Matsuoka, *Large-scale distributed second-order optimization using Kronecker-factored approximate curvature for deep convolutional neural networks*, in 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Long Beach, California, 15–20 Jun 2019, pp. 12351–12359, https://doi.org/10.1109/CVPR.2019.01264.

[127] H. Park, S.-i. Amari, and K. Fukumizu, *Adaptive natural gradient learning algorithms for various stochastic models*, Neural Networks, 13 (2000), pp. 755–764, https://doi.org/10.1016/S0893-6080(00)00051-4.

[128] R. Pascanu and Y. Bengio, *Revisiting natural gradient for deep networks*, 2013, https://arxiv.org/abs/1301.3584.

[129] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, *PyTorch: An imperative style, high-performance deep learning library*, in 33rd Conference on Neural Information Processing Systems, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, eds., vol. 32 of Advances in Neural Information Processing Systems, Vancouver, Canada, 8–14 Dec 2019, pp. 8026–8037, https://proceedings.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html.

[130] B. A. Pearlmutter, *Fast exact multiplication by the Hessian*, Neur. Comput., 6 (1994), pp. 147–160, https://doi.org/10.1162/neco.1994.6.1.147.

[131] B. Polyak, *Some methods of speeding up the convergence of iteration methods*, USSR Comput. Math. Math. Phys., 4 (1964), pp. 1–17, https://doi.org/10.1016/0041-5553(64)90137-5.

[132] B. Polyak and A. B. Juditsky, *Acceleration of stochastic approximation by averaging*, SIAM J. Control Optim., 30 (1992), pp. 838–855, https://doi.org/10.1137/0330046.

[133] D. Povey, X. Zhang, and S. Khudanpur, *Parallel training of DNNs with natural gradient and parameter averaging*, 2014, https://arxiv.org/abs/1410.7455.

[134] A. Radford, L. Metz, and S. Chintala, *Unsupervised representation learning with deep convolutional generative adversarial networks*, 2015, https://arxiv.org/abs/1511.06434.

[135] T. Raiko, H. Valpola, and Y. Lecun, *Deep learning made easier by linear transformations in perceptrons*, in 15th International Conference on Artificial Intelligence and Statistics, N. D. Lawrence and M. Girolami, eds., vol. 22 of Proceedings of Machine Learning Research, La Palma, Canary Islands, 21–23 Apr 2012, PMLR, pp. 924–932, https://proceedings.mlr.press/v22/raiko12.html.

[136] C. R. Rao, *Information and accuracy attainable in the estimation of statistical parameters*, Bull. Calcutta Math. Soc., 37 (1945), pp. 81–91, https://doi.org/10.1007/978-1-4612-0919-5_16.

[137] L. J. Ratliff, S. A. Burden, and S. S. Sastry, *Characterization and computation of local Nash equilibria in continuous games*, in 51st Annual Allerton Conference on Communication, Control and Computing, Allerton, Illinois, 2–4 Oct 2013, pp. 917–924, https://doi.org/10.1109/Allerton.2013.6736623.

[138] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, *You only look once: Unified, real-time object detection*, in 2016 IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, Nevada, 27–30 Jun 2016, pp. 779–788, https://doi.org/10.1109/CVPR.2016.91.

[139] Y. Ren and D. Goldfarb, *Tensor normal training for deep learning models*, in 35th Conference on Neural Information Processing Systems, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, eds., vol. 34 of Advances in Neural Information Processing Systems, virtual, 6–14 Dec 2021, pp. 26040–26052, https://proceedings.neurips.cc/paper/2021/hash/dae3312c4c6c7000a37ecfb7b0aeb0e4-Abstract.html.

[140] H. Ritter, A. Botev, and D. Barber, *A scalable Laplace approximation for neural networks*, in 6th International Conference on Learning Representations, Conference Track Proceedings, Vancouver, Canada, 2018, https://openreview.net/forum?id=Skdvd2xAZ.

[141] H. Robbins and S. Monro, *A stochastic approximation method*, Ann. Math. Statist., 22 (1951), pp. 400–407, https://www.jstor.org/stable/2236626.

[142] O. Ronneberger, P. Fischer, and T. Brox, *U-Net: Convolutional networks for biomedical image segmentation*, in Medical Image Computing and Computer-Assisted Intervention, N. Navab, J. Hornegger, W. Wells, and A. Frangi, eds., vol. 9351 of Lecture Notes in Computer Science, 2015, https://doi.org/10.1007/978-3-319-24574-4_28.

[143] F. Rosenblatt, *The perceptron: a probabilistic model for information storage and organization in the brain*, Psychol. Rev., 65 (1958), pp. 386–408, https://doi.org/10.1037/h0042519.

[144] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning representations by back-propagating errors*, Nature, 323 (1986), pp. 533–536, https://doi.org/10.1038/323533a0.

[145] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and F.-F. Li, *ImageNet large scale visual recognition challenge*, Int. J. Comput. Vis., 115 (2015), pp. 211–252, https://doi.org/10.1007/s11263-015-0816-y.

[146] Y. Saad, *Numerical Methods for Large Eigenvalue Problems*, Classics in Applied Mathematics, SIAM, Philadelphia, 2011, https://doi.org/10.1137/1.9781611970739.

[147] H. Sak, A. Senior, and F. Beaufays, *Long short-term memory recurrent neural network architectures for large scale acoustic modeling*, in 15th Annual Conference of the International Speech Communication Association. Celebrating the Diversity of Spoken Languages, H. Li and P. Ching, eds., Singapore, 14–18 Sep 2014, pp. 338–342, https://arxiv.org/abs/1402.1128.

[148] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, X. Chen, and X. Chen, *Improved techniques for training GANs*, in 13th Conference on Neural Information Processing Systems, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, eds., vol. 29 of Advances in Neural Information Processing Systems, Barcelona, Spain, 5–10 Dec 2016, Curran Associates, Inc., pp. 2226–2234, https://papers.nips.cc/paper_files/paper/2016/hash/8a3363abe792db2d8761d6403605aeb7-Abstract.html.

[149] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, *The graph neural network model*, IEEE Trans. Neural Netw., 20 (2009), pp. 61–80, https://doi.org/10.1109/TNN.2008.2005605.

[150] D. Scherer, A. Müller, and S. Behnke, *Evaluation of pooling operations in convolutional architectures for object recognition*, in Artificial Neural Networks – ICANN 2010, K. Diamantaras, W. Duch, and L. S. Iliadis, eds., vol. 6354 of Lecture Notes in Computer Science, Thessaloniki, Greece, 15–18 Sep 2010, Springer, pp. 92–101, https://doi.org/10.1007/978-3-642-15825-4_10.

[151] M. W. Schmidt, N. Le Roux, and F. R. Bach, *Minimizing finite sums with the stochastic average gradient*, Math. Prog., 162 (2017), pp. 83–112, https://doi.org/10.1007/s10107-016-1030-6.

[152] N. N. Schraudolph, *Fast curvature matrix-vector products for second-order gradient descent*, Neur. Comput., 14 (2002), pp. 1723–1738, https://doi.org/10.1162/08997660260028683.

[153] N. N. Schraudolph, J. Yu, and S. Günter, *A stochastic quasi-newton method for online convex optimization*, in 11th International Conference on Artificial Intelligence and Statistics, M. Meila and X. Shen, eds., vol. 2 of Proceedings of Machine Learning Research, San Juan, Puerto Rico, 21–24 Mar 2007, pp. 436–443, https://proceedings.mlr.press/v2/schraudolph07a.html.

[154] M. Schuster and K. Paliwal, *Bidirectional recurrent neural networks*, IEEE Trans. Signal Process., 45 (1997), pp. 2673–2681, https://doi.org/10.1109/78.650093.

[155] T. Sercu, C. Puhrsch, B. Kingsbury, and Y. LeCun, *Very deep multilingual convolutional neural networks for LVCSR*, in 2016 IEEE International Conference on Acoustics, Speech and Signal Processing, Shanghai, China, 20–25 Mar 2016, pp. 4955–4959, https://doi.org/10.1109/ICASSP.2016.7472620.

[156] D. F. Shanno, *Conditioning of quasi-Newton methods for function minimization*, Math. Comp., 24 (1970), pp. 647–656, https://doi.org/10.1090/S0025-5718-1970-0274029-X.

[157] K. Simonyan and A. Zisserman, *Very deep convolutional networks for large-scale image recognition*, in 3rd International Conference on Learning Representations, Conference Track Proceedings, Y. Bengio and Y. LeCun, eds., San Diego, California, 7–9 May 2015, https://arxiv.org/abs/1409.1556.

[158] Y. Song, J. Song, and S. Ermon, *Accelerating natural gradient with higher-order invariance*, in 35th International Conference on Machine Learning, J. Dy and A. Krause, eds., vol. 80 of Proceedings of Machine Learning Research, Stockholm, Sweden, 10–15 Jul 2018, pp. 4713–4722, https://proceedings.mlr.press/v80/song18a.html.

[159] R. Strudel, R. Garcia, I. Laptev, and C. Schmid, *Segmenter: Transformer for semantic segmentation*, in 2021 IEEE/CVF International Conference on Computer Vision, Montreal, Canada, 10–17 Oct 2021, pp. 7242–7252, https://doi.org/10.1109/ICCV48922.2021.00717.

[160] R. Sun, *Optimization for deep learning: theory and algorithms*, 2019, https://arxiv.org/abs/1912.08957.

[161] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, *On the importance of initialization and momentum in deep learning*, in 30th International Conference on Machine Learning, S. Dasgupta and D. McAllester, eds., vol. 28 of Proceedings of Machine Learning Research, Atlanta, Georgia, 17–19 Jun 2013, pp. 1139–1147, https://proceedings.mlr.press/v28/sutskever13.html.

[162] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, *Going deeper with convolutions*, in 2015 IEEE Conference on Computer Vision and Pattern Recognition, Boston, Massachusetts, 7–12 Jun 2015, pp. 1–9, https://doi.org/10.1109/CVPR.2015.7298594.

[163] R. Takahashi, T. Matsubara, and K. Uehara, *Scale-invariant recognition by weight-shared CNNs in parallel*, in Proceedings of the 9th Asian Conference on Machine Learning, vol. 77 of Proceedings of Machine Learning Research, Seoul, South Korea, 15–17 Nov 2017, pp. 295–310, https://proceedings.mlr.press/v77/takahashi17a.html.

[164] C. Tan, S. Ma, Y.-H. Dai, and Y. Qian, *Barzilai-Borwein step size for stochastic gradient descent*, in Advances in Neural Information Processing Systems, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, eds., vol. 29, 2016, pp. 685–693, http://papers.nips.cc/paper/6286-barzilai-borwein-step-size-for-stochastic-gradient-descent.

[165] T. Tieleman and G. Hinton, *Lecture 6.5 RMSProp: Divide the gradient by a running average of its recent magnitude*, COURSERA: Neural Networks for Machine Learning, 4 (2012), pp. 26–31.

[166] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jegou, *Training data-efficient image transformers & distillation through attention*, in 38th International Conference on Machine Learning, M. Meila and T. Zhang, eds., vol. 139 of Proceedings of Machine Learning Research, virtual, 18–24 Jul 2021, pp. 10347–10357, https://proceedings.mlr.press/v139/touvron21a.html.

[167] N. Tselepidis, J. Kohler, and A. Orvieto, *Two-level K-FAC preconditioning for deep learning*, in 12th Annual Workshop on Optimization for Machine Learning, virtual, 2020, https://www.opt-ml.org/papers/2020/paper_63.pdf.

[168] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, *WaveNet: A generative model for raw audio*, 2016, https://arxiv.org/abs/1609.03499.

[169] A. van den Oord, N. Kalchbrenner, and K. Kavukcuoglu, *Pixel recurrent neural networks*, in 33rd International Conference on Machine Learning, M.-F. Balcan and K. Q. Weinberger, eds., vol. 48 of JMLR Workshop and Conference Proceedings, New York, 19–24 Jun 2016, pp. 1747–1756, http://proceedings.mlr.press/v48/oord16.pdf.

[170] C. F. van Loan, *The ubiquitous Kronecker product*, J. Comput. Appl. Math., 123 (2000), pp. 85–100, https://doi.org/10.1016/S0377-0427(00)00393-9.

[171] C. F. Van Loan and N. Pitsianis, *Approximation with Kronecker products*, in Linear Algebra for Large Scale and Real-Time Applications, M. S. Moonen, G. H. Golub, and B. L. R. De Moor, eds., vol. 232 of NATO ASI Series, Dordrecht, 1993, Springer, https://doi.org/10.1007/978-94-015-8196-7_17.

[172] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, *Attention is all you need*, in 31st Conference on Neural Information Processing Systems, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds., vol. 30 of Advances in Neural Information Processing Systems, Long Beach, California, 4–9 Dec 2017, pp. 5998–6008, https://proceedings.neurips.cc/paper_files/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

[173] T. Vatanen, T. Raiko, H. Valpola, and Y. LeCun, *Pushing stochastic gradient towards second-order methods – backpropagation learning with transformations in nonlinearities*, in International Conference on Neural Information Processing, M. Lee, A. Hirose, Z.-G. Hou, and R. M. Kil, eds., vol. 8226 of Lecture Notes in Computer Science, Berlin, 2013, Springer, pp. 442–449, https://doi.org/10.1007/978-3-642-42054-2_55.

[174] C. Villani, *Optimal Transport: Old and New*, vol. 338 of Grundlehren der mathematischen Wissenschaften, Springer, Berlin, 2009, https://doi.org/10.1007/978-3-540-71050-9.

[175] X. Wang, K. Yu, S. Wu, J. Gu, Y. Liu, C. Dong, Y. Qiao, and C. C. Loy, *ESRGAN: Enhanced super-resolution generative adversarial networks*, in Computer Vision – ECCV 2018 Workshops, L. Leal-Taixé and S. Roth, eds., vol. 11133 of Lecture Notes in Computer Science, Cham, 2019, Springer, pp. 63–79, https://doi.org/10.1007/978-3-030-11021-5_5.

[176] Y. Wang, *A mathematical introduction to generative adversarial nets (GAN)*, 2020, https://arxiv.org/abs/2009.00169.

[177] Y. Wang, X. Zhang, T. Yang, and J. Sun, *Anchor DETR: Query design for transformer-based object detection*, 2022, https://arxiv.org/abs/2109.07107.

[178] S. Wiesler, A. Richard, R. Schlüter, and H. Ney, *Mean-normalized stochastic gradient for large-scale deep learning*, in 2014 IEEE International Conference on Acoustics, Speech and Signal Processing, Florence, Italy, 4–9 May 2014, pp. 180–184, https://doi.org/10.1109/ICASSP.2014.6853582.

[179] Y. Wu, E. Mansimov, R. B. Grosse, S. Liao, and J. Ba, *Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation*, in 31st Conference on Neural Information Processing Systems, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds., vol. 30 of Advances in Neural Information Processing Systems, Long Beach, California, 4–9 Dec 2017, pp. 5285–5294, https://proceedings.neurips.cc/paper/2017/hash/361440528766bbaaaa1901845cf4152b-Abstract.html.

[180] H. Xiao, K. Rasul, and R. Vollgraf, *Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms*, 2017, https://arxiv.org/abs/1708.07747.

[181] E. Xie, W. Wang, Z. Yu, A. Anandkumar, J. M. Alvarez, and P. Luo, *SegFormer: Simple and efficient design for semantic segmentation with transformers*, in 35th Conference on Neural Information Processing Systems, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, eds., vol. 34 of Advances in Neural Information Processing Systems, virtual, 6–14 Dec 2021, pp. 12077–12090, https://proceedings.neurips.cc/paper/2021/hash/64f1f27bf1b4ec22924fd0acb550c235-Abstract.html.

[182] T. Xu, P. Zhang, Q. Huang, H. Zhang, Z. Gan, X. Huang, and X. He, *AttnGAN: Fine-grained text to image generation with attentional generative adversarial networks*, in 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, Utah, 18–23 Jun 2018, pp. 1316–1324, https://doi.org/10.1109/CVPR.2018.00143.

[183] H. H. Yang and S.-i. Amari, *Complexity issues in natural gradient descent method for training multilayer perceptrons*, Neur. Comput., 10 (1998), pp. 2137–2157, https://doi.org/10.1162/089976698300017007.

[184] J. Yu, Z. Lin, J. Yang, X. Shen, X. Lu, and T. S. Huang, *Free-form image inpainting with gated convolution*, in 2019 IEEE/CVF International Conference on Computer Vision, Seoul, South Korea, 27 Oct–2 Nov 2019, pp. 4470–4479, https://doi.org/10.1109/ICCV.2019.00457.

[185] M. D. Zeiler, *ADADELTA: An adaptive learning rate method*, 2012, https://arxiv.org/abs/1212.5701.

[186] G. Zhang, S. Sun, D. Duvenaud, and R. Grosse, *Noisy natural gradient as variational inference*, in 35th International Conference on Machine Learning, J. Dy and A. Krause, eds., vol. 80 of Proceedings of Machine Learning Research, Stockholm, Sweden, 10–15 Jul 2018, pp. 9308–9321, https://proceedings.mlr.press/v80/zhang18l.html.

[187] G. Zhang, C. Wang, B. Xu, and R. Grosse, *Three mechanisms of weight decay regularization*, in 7th International Conference on Learning Representations, Conference Track Proceedings, New Orleans, Louisiana, 6–9 May 2019, https://openreview.net/forum?id=B1lz-3Rct7.

[188] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena, *Self-attention generative adversarial networks*, 2019, https://arxiv.org/abs/1805.08318.

[189] H. Zhang, T. Xu, H. Li, S. Zhang, X. Wang, X. Huang, and D. Metaxas, *StackGAN: Text to photo-realistic image synthesis with stacked generative adversarial networks*, in 2017 IEEE International Conference on Computer Vision, Venice, Italy, 22–29 Oct 2017, pp. 5908–5916, https://doi.org/10.1109/ICCV.2017.629.

[190] J. Zhang, *Gradient descent based optimization algorithms for deep learning models training*, 2019, https://arxiv.org/abs/1903.03614v1.

[191] S. Zheng, J. Lu, H. Zhao, X. Zhu, Z. Luo, Y. Wang, Y. Fu, J. Feng, T. Xiang, P. S. Torr, and L. Zhang, *Rethinking semantic segmentation from a sequence-to-sequence perspective with transformers*, in 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Nashville, Tennessee, 20–25 Jun 2021, pp. 6877–6886, https://doi.org/10.1109/CVPR46437.2021.00681.

[192] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, *Unpaired image-to-image translation using cycle-consistent adversarial networks*, in 2017 IEEE International Conference on Computer Vision, Venice, Italy, 22–29 Oct 2017, pp. 2242–2251, https://doi.org/10.1109/ICCV.2017.244.

[193] X. Zhu, W. Su, L. Lu, B. Li, X. Wang, and J. Dai, *Deformable DETR: Deformable transformers for end-to-end object detection*, 2021, https://arxiv.org/abs/2010.04159.

université PARIS-SACLAY

ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre : Méthodes d'optimisation basées sur le gradient naturel pour les réseaux de neurones profonds

Mots clés : réseau de neurones, gradient stochastique, gradient naturel, matrice de Fisher, produit de Kronecker, décomposition de domaines

Résumé : La méthode du gradient stochastique est la technologie actuellement prédominante pour effectuer la phase d'entraînement des réseaux de neurones. Par rapport à une descente classique, le calcul du vrai gradient comme une moyenne sur les données est remplacé par un élément aléatoire de la somme. En présence de données massives, cette approximation audacieuse permet de diminuer le nombre d'évaluations de gradients élémentaires et d'alléger le coût de chaque itération. Le prix à payer est l'apparition d'oscillations et la lenteur de convergence souvent excessive en nombre d'itérations.
L'objectif de cette thèse est de concevoir une approche à la fois : (i) plus robuste, en faisant appel aux méthodes fondamentales qui ont fait leur preuve en optimisation classique, i.e., en dehors du cadre de l'apprentissage ; et (ii) plus rapide en termes de vitesse de convergence
Nous nous intéressons en particulier aux méthodes de second ordre, connues pour leur stabilité et leur rapidité de convergence. Pour éviter le goulot d'étranglement de ces méthodes, qui est le coût exorbitant d'une itération où intervient un système linéaire à matrice pleine, nous tentons d'améliorer une approximation récemment introduite sous le nom de *Kronecker-Factorized Approximation of Curvature* (KFAC) pour la matrice de Fisher, laquelle remplace la matrice hessienne dans ce contexte.
Plus précisément, nos axes de travail sont : (i) construire de nouvelles factorisations de Kronecker fondées sur une justification mathématique plus rigoureuse que KFAC ; (ii) prendre en compte l'information issue des blocs hors diagonaux de la matrice de Fisher, qui représentent l'interaction entre les différentes couches ; (iii) généraliser KFAC à une architecture de réseau autre que celles pour lesquelles elle a été initialement développée.

Title: Natural gradient-based optimization methods for deep neural networks

Keywords: neural networks, stochastic gradient, natural gradient, Fisher matrix, Kronecker product, domain decomposition

Abstract: The stochastic gradient method is currently the prevailing technology for training neural networks. Compared to the classical gradient descent method, the calculation of the true gradient as an average over the data is replaced by a random element of the sum. When dealing with massive data, this bold approximation enables one to decrease the number of elementary gradient evaluations and to alleviate the cost of each iteration. The price to be paid is the appearance of oscillations and the slowness of convergence, which is often excessive in terms of number of iterations.
The aim of this thesis is to design an approach that is both: (i) more robust, using the fundamental methods that have been successfully proven in classical optimization, i.e., outside the machine learning framework; and (ii) faster in terms of convergence speed.
We are especially interested in second-order methods, known for their stability and speed of convergence. To circumvent the bottleneck of these methods, which lies in the prohibitive cost of an iteration involving a linear system with a full matrix, we attempt to improve an approximation recently introduced as *Kronecker-Factorized Approximation of Curvature* (KFAC) for the Fisher matrix, which replaces the Hessian matrix in this context.
More specifically, our work focuses on: (i) building new Kronecker factorizations based on a more rigorous mathematical justification than in KFAC; (ii) taking into account the information from the off-diagonal blocks of the Fisher matrix, which represent the interaction between the different layers; (iii) generalizing KFAC to a network architecture other than those for which it had been initially developed.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France