Deep learning in event-based neuromorphic systems

Johannes C. Thiele

▶ To cite this version:

Johannes C. Thiele. Deep learning in event-based neuromorphic systems. Artificial Intelligence [cs.AI]. Université Paris Saclay (COmUE), 2019. English. NNT : 2019SACLS403 . tel-02417462

HAL Id: tel-02417462

https://theses.hal.science/tel-02417462

Submitted on 18 Dec 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

université
PARIS-SACLAY

UNIVERSITÉ PARIS SUD
Comprendre le monde, construire l'avenir

NNT : 2019SACLS403

Deep Learning in Event-Based Neuromorphic Systems

Thèse de doctorat de l'Université Paris-Saclay
préparée à l'Université Paris-Sud

Ecole doctorale n°575
Electrical, Optical, Bio : PHYSICS-AND-ENGINEERING (EOBE)
Spécialité de doctorat : Sciences de l'information et de la communication

Thèse présentée et soutenue à Palaiseau, le 22.11.2019, par

JOHANNES CHRISTIAN THIELE

Composition du Jury :

Antoine Manzanera Enseignant-Chercheur, ENSTA ParisTech	Président
Simon Thorpe Directeur de Recherche, CNRS	Rapporteur
Shih-Chii Liu Privatdozent, ETH Zurich	Rapporteur
Emre Neftci Assitant Professor, UC Irvine	Examinateur
Antoine Dupret Ingénieur-Chercheur, CEA/LIST	Directeur de thèse
Olivier Bichler Ingénieur-Chercheur, CEA/LIST	Encadrant

Thèse de doctorat

Acknowledgments

This thesis was hosted by the DACLE department of CEA, LIST. I want to thank Thierry Collette, the former head of the department, and the current head Jean René Lequepeys, for offering me the possibility to perform my doctoral project in their department. I want to thank all members of the CEA laboratories that hosted me during my thesis, LCE and L3A, and their respective heads Nicolas Ventroux and Thomas Dombek, for the great personal and scientific support they offered me throughout the whole duration of my doctorate. In particular, I want to express my gratitude towards all members of the neural network group, for answering a physicist his many questions about software design and hardware. It was a great pleasure to work together with all of you and I am happy to be able to continue with our excellent work in the years to come. I am grateful for your great patience and support in those months when I was still getting accustomed to the French language.

Most of all, I want to thank my supervisors Olivier Bichler and Antoine Dupret for supporting me with their extensive technical knowledge and research experience. I am grateful for their great trust in me and the autonomy they offered me during my thesis, while always being present to answer my questions with great patience and expertise.

I want to thank all jury members for studying my manuscript and participating in my defense. In particular, I want to thank my thesis reviewers Shih-Chii Liu and Simon Thorpe for providing me such precise and helpful feedback on my manuscript. Many thanks to Antoine Manzanera and Emre Neftci for participating as jury members in my defense.

I also want to thank our collaborators at INI in Zurich for our productive work, in particular Giacomo Indiveri for offering me the opportunity to spend time in his research group during my thesis.

Finally, I want to thank all my loved ones for supporting me during my doctorate. In particular, I want to express my gratitude towards my parents Walter and Elvira Thiele, for enabling me to pursue my studies for all of these years, up to this final academic degree.

Résumé en français

Les réseaux de neurones profonds jouent aujourd'hui un rôle de plus en plus important dans de nombreux systèmes intelligents. En particulier pour les applications de reconnaissance des objets et de traduction automatique, les réseaux profonds, dits «deep neural networks», représentent l'état de l'art. Cependant, inférence et apprentissage dans les réseaux de neurones profonds nécessitent une grande quantité de calcul, qui dans beaucoup de cas limite l'intégration des réseaux profonds dans les environnements en ressources limitées. Étant donné que le cerveau humain est capable d'accomplir des tâches complexes avec un budget énergétique très restreint, cela pose la question si on ne pourrait pas améliorer l'efficacité des réseaux de neurones artificiels en imitant certains aspects des réseaux biologiques. Les réseaux de neurones évènementiels de type «spiking neural network (SNN)», qui sont inspirés par le paradigme de communication d'information dans le cerveau, représentent une alternative aux réseaux de neurones artificiels classiques. Au lieu de propager des nombres réels, ce type de réseau utilise des signaux binaires, appellés «spikes», qui sont des évènements déclenchés en fonction des entrées du réseau. Contrairement aux réseaux classiques, pour lequels la quantité de calcul effectué est indépendante des informations reçues, ce type de réseau évènementiel est capable de traiter des informations en fonction de leur quantité. Cette caractéristique entraine qu'une grande partie d'un réseau évènementiel restera inactif en fonction des données. Malgré ces avantages potentiels, entraîner les réseaux spike reste un défi. Dans la plupart des cas, un réseau spike, pour une même architecture, n'est pas capable de fournir une précision d'inférence égale au réseau artificiel classique. Cela est particulièrement vrai dans les cas où l'apprentissage doit être exécuté sur un matériel de calcul bio-inspiré, dit matériel *neuromorphique*. Cette thèse présente une étude sur les algorithmes d'apprentissage et le codage d'informations dans les réseaux de neurones évènementiels, en mettant un accent sur les algorithmes qui sont capables d'un apprentissage sur une puce neuromorphique. Nous commençons notre étude par la conception d'une architecture optimisée pour l'ap-

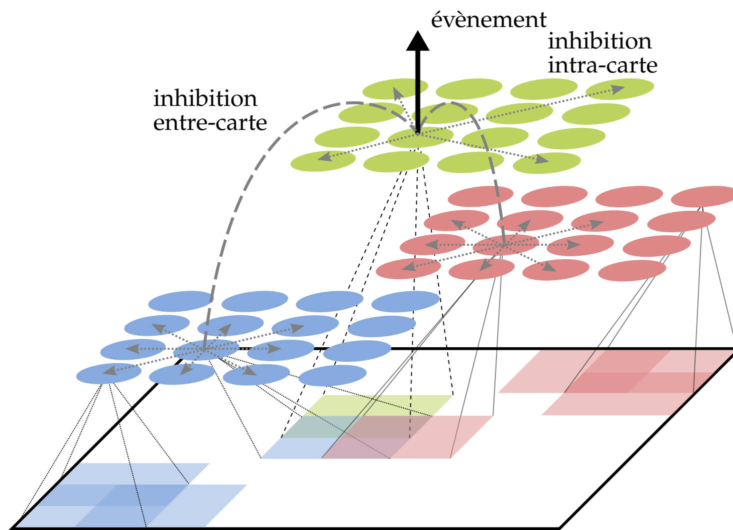

Figure 0.1: Structure basique d'une couche convolutive avec inhibition WTA entre-carte et intra-carte. Chaque carte de charactéristiques (montrées en différentes couleurs) détecte des charactéristiques dans l'image entière. Si un neurone déclenche un évènement, tous les neurones des autres couches, se trouvant au voisinage du neurone déchlenché, sont inhibés, de même que les neurones de la carte du neurone déchlenché.

prentissage continu, qui apprend avec une règle d'apprentissage type STDP («spike-timing dependent plasticity», littéralement *plasticité dépendant de l'instant de déclenchement*). Le STDP est une règle d'apprentissage non-supervisée qui permet, avec un mécanisme compétitif du type WTA («winner-takes-all», littéralement *le vainqueur prend tout*), une extraction des caractéristiques des données non-étiquettées (figure 0.7). Cette règle est facilement réalisable dans les matériels neuromorphiques. Ce mécanisme d'apprentissage nous permet de construire un système capable d'un apprentissage interne dans une puce neuromorphique, potentiellement avec une haute efficacité énergétique. Cette approche est utilisée pour l'apprentissage des caractéristiques dans un CNN («convolutional neural network», réseau de neurones convolutif). Ce réseau consiste en deux couches convolutives, deux couches de «pooling»(qui regroupent les entrées en faisant la moyenne d'une région spatiale) et une couche entièrement connectée. Dans les approches précédentes, le mécanisme de WTA permettait seulement une apprentissage couche par couche. En utilisant un modèle de neurone avec deux accumulateur (figure 0.2), nous sommes capables d'entraîner toutes les couches du réseau simultanément. Ce modèle de neurone représente une caractéristique innovante de notre architecture, qui facilite largement l'apprentissage continu. De plus, nous montrons que le choix spécifique de la règle STDP, qui mesure les temps d'évènements en fonction des dernières remises à zéro et qui n'utilise que des temps absolus, per-

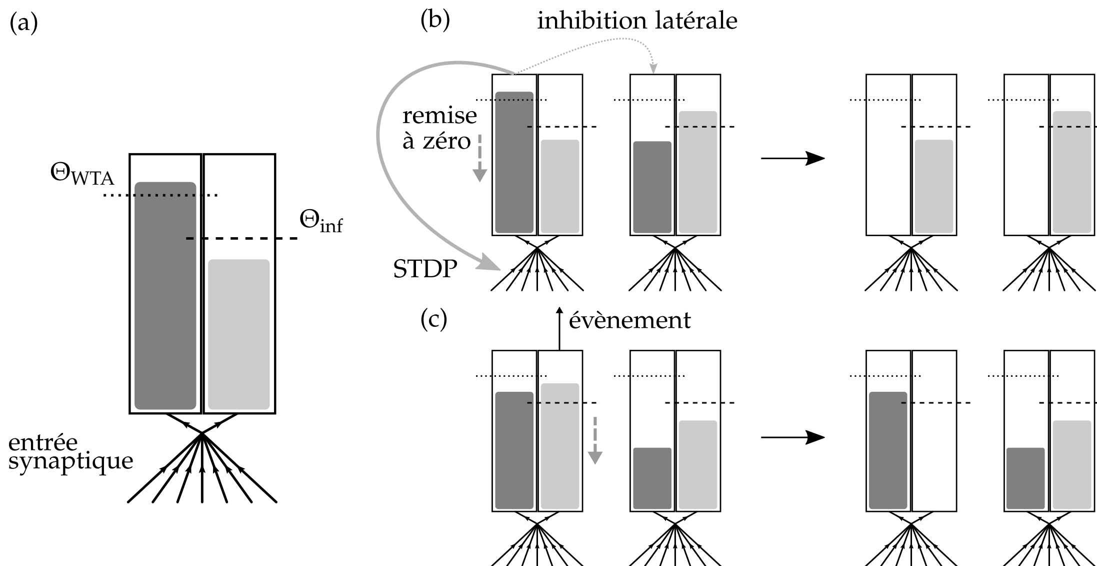

FIGURE 0.2: (a) Description schématique du neurone en deux accumulateurs. Le neurone intègre tous les entrées reçues simultanément dans deux accumulateurs, l'un pour le déclenchement STDP et WTA, l'autre pour l'inférence (cela veut dire la propagation d'évènements).(b) Interaction entre deux neurones liés par inhibition latérale. Si l'intégration d'accumulateur WTA dépasse le seuil, aucun évènement n'est propagé à la couche suivante. Cependant, le neurone déclenche STDP et l'inhibition latérale, qui remet à zéro les intégrations des autres neurones. (c) Si l'intégration dépasse le seuil dans l'accumulateur d'inférence, un évènement est propagé et l'intégration est remise à zéro. Les accumulateurs d'inférence des autres neurones restent inaffectés, ou sont alternativement influencés par une inhibition de courte distance (plus courte que l'inhibition des accumulateurs d'apprentissage). Cela permet de contrôler combien de cartes par position contribuent à la propagation des évènements.

met d'obtenir une invariance temporelle de fonctionnement du réseau. Nous montrons que notre architecture est capable d'une extraction des caractéristiques pertinentes (figure 0.3), qui permettent une classification des images des chiffres de la base de données MNIST. La classification est faite avec une simple couche entièrement connectée dont les neurones correspondent aux prototypes des chiffres. Notre architecture démontre une meilleure performance de classification que les réseaux de l'état de l'art qui consistent seulement en une couche entièrement connectée, qui montre l'avantage de passer des informations par des couches convolutives. On parvient à une précision de classification maximale de 96,58%, qui est la meilleure précision pour un réseau où toutes les couches sont entrainées avec un mécanisme STDP. Nous montrons la robustesse du réseau face aux variations du taux d'apprentissage et le temps de présentation des stimuli. En changeant la taille des différentes couches du réseau, nous montrons que la performance du réseau peut être améliorée en augmentant le nombre des neurones dans la couche entièrement

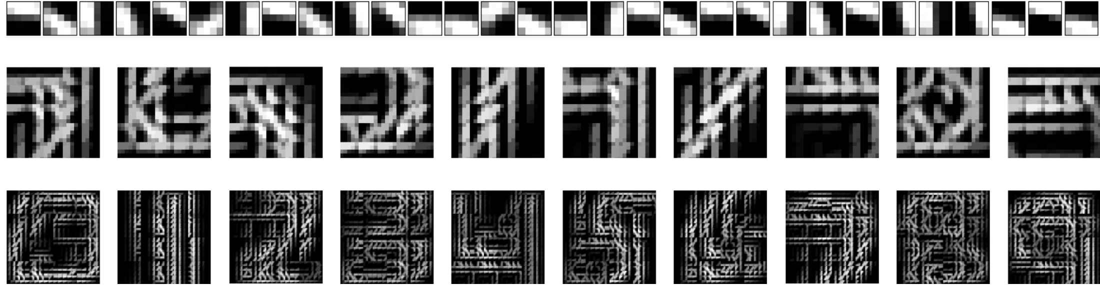

FIGURE 0.3: Visualization des caractéristiques apprises par les différentes couches du réseau. Dans la première couche, les caractéristiques correspondent simplement aux poids du noyau de convolution. On voit que cette couche apprend à détecter des différences de contraste locales. Les caractéristiques du haut niveau sont construites en choisissant pour chaque neurone la caractéristique de la couche précédente avec le poids le plus fort. Parce que les filtres dans des positions différentes se chevauchent, les caractéristiques ont une apparence floue. On voit que dans la deuxième couche convolutive, les neurones deviennent sensibles aux parties de chiffres. Finalement, dans la couche entièrement connectée, chaque neurone a appris une caractéristique qui ressemble à un prototype spécifique d'un chiffre faisant partie d'une des différentes classes (chiffre de 0 à 9 de gauche à droite).

connectée et la deuxième couche convolutive. Comme extension de ces travaux, nous montrons que le réseau peut aussi être appliqué à l'apprentissage des caractéristiques d'une base de données dynamique, N-MNIST, qui représente une version de MNIST avec des chiffres en mouvement. Avec quelques changements mineurs des paramètres de notre architecture, il est capable d'apprendre des caractéristiques qui maintiennent l'information du mouvement (figure 0.4). En ajoutant une deuxième couche entièrement connectée avec un classificateur supervisé, nous sommes capables d'obtenir une précision de classification de 95,77%. Cela représente la première démonstration d'un apprentissage de N-MNIST dans un réseau convolutif où toutes les caractéristiques sont apprises par un mécanisme STDP. Nous montrons également que l'architecture est capable d'une inférence basée seulement sur des sous-mouvement. Nos résultats montrent alors que notre architecture possède toutes les capacités nécessaires pour un système neuromorphique qui doit effectuer un apprentissage sur un flux de données évènementielles.

En comparaison avec des approches supervisées fondées sur l'algorithme BP («backpropagation», la rétro-propagation du gradient), notre architecture démontre une performance de classification inférieure. Nous présentons une analyse des différents algorithmes d'apprentissage pour les réseaux spike d'un point de vue localité de l'information. Reposant sur cette synthèse, nous expliquons que les règle d'apprentissage du type STDP, qui sont spatialement et temporellement locales, sont limitées dans leur capacité d'apprendre certaines caractéristiques nécessitant une optimisation globale du réseau. Cette

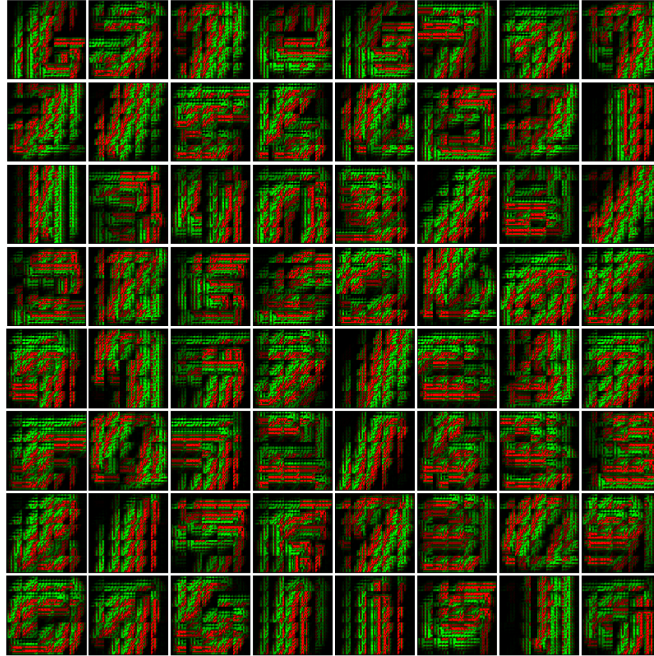

FIGURE 0.4: Visualisation des caractéristiques apprises par la première couche entièrement connectée du réseau.

faiblesse pourrait expliquer leurs performances inférieures par rapport aux mécanismes d'optimisation non-locaux type rétro-propagation du gradient.

Pour dépasser ces limites du STDP, nous élaborons un nouvel outil pour l'apprentissage dans les réseaux spike, *SpikeGrad*, qui représente une implémentation entièrement évènementielle de la rétro-propagation du gradient. Cet algorithme est construit à partir d'un modèle de neurone spécial possédant un deuxième accumulateur pour l'intégration des informations du gradient. Cet accumulateur est mis à jour d'une manière pratiquement équivalente à l'accumulateur ordinaire du neurone, avec la différence que la dérivée de la fonction d'activation est prise en compte pour le calcul du gradient. En utilisant un accumulateur d'activation de neurone qui est pondéré par le taux d'apprentissage, *SpikeGrad* est capable d'effectuer un apprentissage qui repose uniquement sur des accumulations et des comparaisons. De plus, *SpikeGrad* possède un autre grand avantage. En utilisant un modèle de neurone type IF («integrate-and-fire», littéralement *intègre-et-déclenche*) avec une fonction d'activation résiduelle, nous montrons que les activations accumulées des neurones sont équivalentes à celles d'un réseau de neurones artificiels. Cela permet la simulation de l'apprentissage du réseau évènementiel en tant que réseau de neurones artificiels classique, qui est facile à effectuer avec le matériel de calcul haute performance typiquement utilisé dans l'apprentissage profond.

Nous montrons comme cette approche peut être utilisée pour l'entraine-

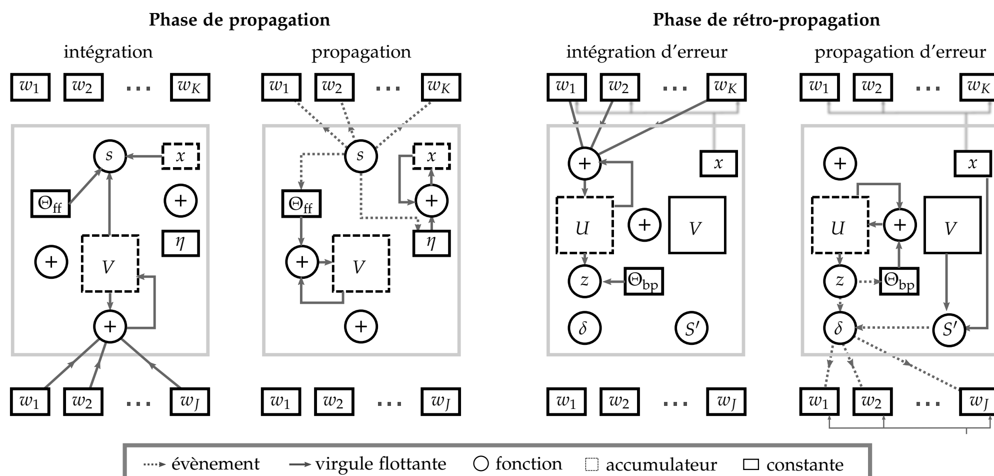

FIGURE 0.5: Phases de propagation et rétro-propagation dans un seul neurone dans l'algorithme *SpikeGrad*. **Intégration :** Chaque fois qu'un signal évènementiel arrive à une des synapses $\{w_1, ..., w_J\}$, la valeur du poid est ajoutée à la variable d'intégration V ponderée par le signe de l'évènement. Après chaque mise à jour, la variable d'intégration est comparée au seuil $\pm\Theta_{\text{ff}}$ et la trace synaptique x par la fonction d'activation évènementielle s, qui décide si un évènement sera déclenché. **Propagation :** Si un évènement est déclenché, il augmente la trace x par le taux d'apprentissage $\pm\eta$, et il est transmis aux connexions sortantes. L'intégration est augmentée par $\pm\Theta_{\text{ff}}$ selon le signe de l'évènement. **Intégration d'erreur :** Des évènements signés sont reçus par les synapses $\{w_1, ..., w_K\}$ des connexions sortantes. La valeur du poid est ajoutée à la variable d'intégration d'erreur U, et elle est comparée au seuil $\pm\Theta_{\text{bp}}$ par la fonction z. **Propagation d'erreur :** Si le seuil est dépassé, un évènement signé est émis et U est incrémentée par $\pm\Theta_{\text{bp}}$. L'évènement est pondéré par la dérivée de la fonction d'activation effective (qui est calculée avec V et x), et propagé à travers les connexions entrantes. Les poids des neurones sont mis à jour en utilisant ce signal d'erreur et les traces synaptiques des neurones de la couche précédente.

ment d'un réseau spike qui est capable d'inférer des relations entre des valeurs numériques et des images. Notre première démonstration concerne un réseau qui apprend la relation $A + B = C$. Nous montrons que SpikeGrad permet d'entraîner ce type de réseau plus rapidement, avec moins de neurones, et avec une meilleure précision d'inférence que les approches bio-inspirés de l'état de l'art. La deuxième tache examinée est l'apprentissage d'une version visuelle de la relation logique XOR, sur les images de 0 et 1 de MNIST. Nous montrons également que le réseau est capable d'une inférence visuelle dans ce scenario, et qu'il peut produire des stimuli artificiels qui ressemblent aux exemples qu'il a reçus pendant l'apprentissage.

La deuxième démonstration des capacités de *SpikeGrad* est faite sur une tâche de classification. Il est montré que l'outil est capable d'entraîner un

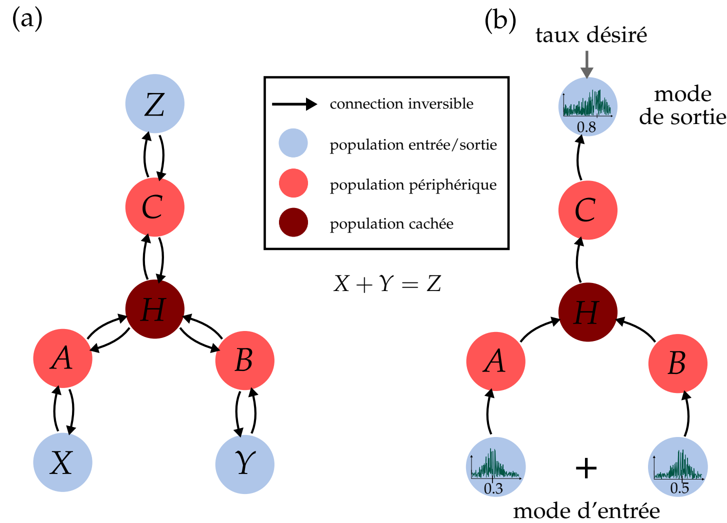

FIGURE 0.6: (a) Architecture du réseaux relationnel avec trois variables. Les populations d'entrée/sortie sont marquées par X, Y, Z. Les populations périphériques sont marquées par A, B, C, et la population cachée par H. (b) Entraînement du réseau. Pendant l'apprentissage, deux populations servent comme populations d'entrée et fournissent un profil des taux de déclenchement désiré. La troisième population est entraînée pour reproduire ce profil, à partir des entrées. Ces différents rôles des populations sont interchangés pendant l'apprentissage pour permettre une inférence dans tous les sens de la relation.

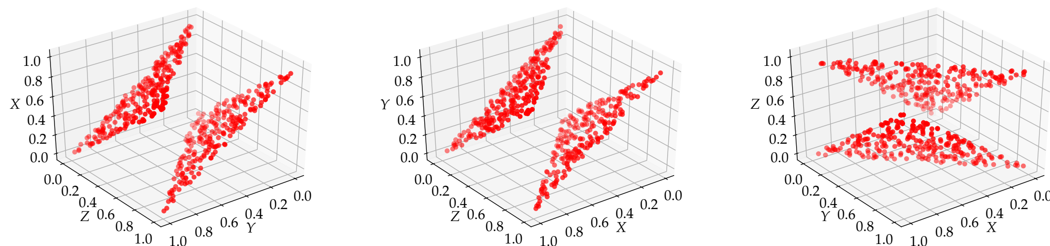

FIGURE 0.7: Inférence des trois variables de la relation. Les deux axes du fond marquent les deux variables d'entrée, tandis que l'axe vertical représente la valeur inférée. On voit que le réseau a appris à représenter précisement toutes les directions possibles de la relation.

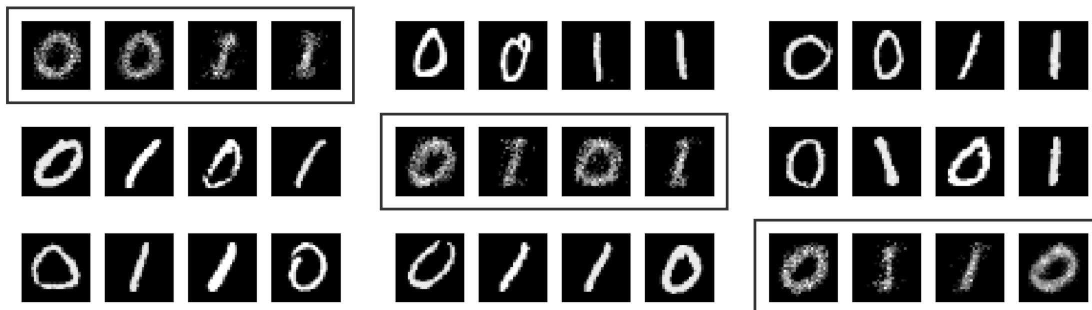

FIGURE 0.8: Inférence des trois variables de la relation visuelle. Les stimuli inférés sont marqués par des boîtes. On voit que le réseau a appris à représenter toutes les directions de la relation et est capable de construire des stimuli artificiels qui ressemblent aux autres stimuli de la base de données.

Architecture	Méthode	Taux Rec. (**max**[moyen±dev.])
Rueckauer *et al.* [2017]	CNN converti à SNN	$99,44\%$
Wu *et al.* [2018b]*	SNN entraîné avec BP float.	$99,42\%$
Jin *et al.* [2018]*	Macro/Micro BP	$99,49\%$
Ces travaux*	SNN entraîné avec BP float.	$\mathbf{99,48}[99,36 \pm 0,06]\%$
Ces travaux*	SNN entraîné avec BP évènementiel	$\mathbf{99,52}[99,38 \pm 0,06]\%$

TABLE 0.1: Comparaison des différentes méthodes de l'état de l'art des architectures CNN évènementielles sur MNIST. * marque l'utilisation de la même topologie (28x28-15C5-P2-40C5-P2-300-10).

grand réseau spike convolutif, qui donne des taux de reconnaissance d'image compétitive (figures 0.1 et 0.2). Sur MNIST, nous obtenons une précision de classification de $99,52\%$, et sur CIFAR10, nous obtenons $89,99\%$. Dans les deux cas, ces précisions sont aussi bonnes que celles des réseaux spike entraînés avec la rétro-propagation en virgule flottante. Elles sont aussi comparables avec celles des réseaux de neurones classiques possédant une architecture comparable. De plus, nos résultats montrent que ce type de rétro-propagation évènementielle du gradient pourrait facilement exploiter la grande parcimonie qui se trouve dans le calcul du gradient. Particulièrement vers la fin d'apprentissage, quand l'erreur commence à diminuer fortement, la quantité de calcul pourrait diminuer fortement en utilisant un codage évènementiel.

Nos travaux introduisent alors plusieurs mécanismes d'apprentissage puissants. La première partie de nos travaux avait comme objectif d'utiliser un mécanisme d'apprentissage bio-inspiré pour construire un système neuromorphique capable d'un apprentissage continue. Pour franchir plusieurs limitations de cette approche, nous avons ensuite développé une implémentation évènementielle de la rétro-propagation du gradient. Ces approches pourraient alors promouvoir l'utilisation des réseaux spike pour des problèmes réels. Dans notre opinion, les approches comme *SpikeGrad*, qui sont basées sur

Architecture	Méthode	Taux Rec. (**max**[moyen±dev.])
Rueckauer *et al.* [2017]	CNN converti à SNN (avec BatchNorm)	$90,85\%$
Wu *et al.* [2018a]*	SNN entraîné avec BP float. (sans «NeuNorm»)	$89,32\%$
Sengupta *et al.* [2019]	SNN (VGG-16) converti à SNN SNN	$91,55\%$
Ces travaux*	SNN entraîné avec BP float.	**89.72**$[89,38 \pm 0,25]\%$
Ces travaux*	SNN entraîné avec BP évènementiel	$\mathbf{89,99}[89,49 \pm 0,28]\%$

TABLE 0.2: Comparaison des différentes méthodes de l'état de l'art des architectures CNN évènementielles sur CIFAR10. * marque l'utilisation de la même topologie (32x32-128C3-256C3-P2-512C3-P2-1024C3-512C3-1024-512-10).

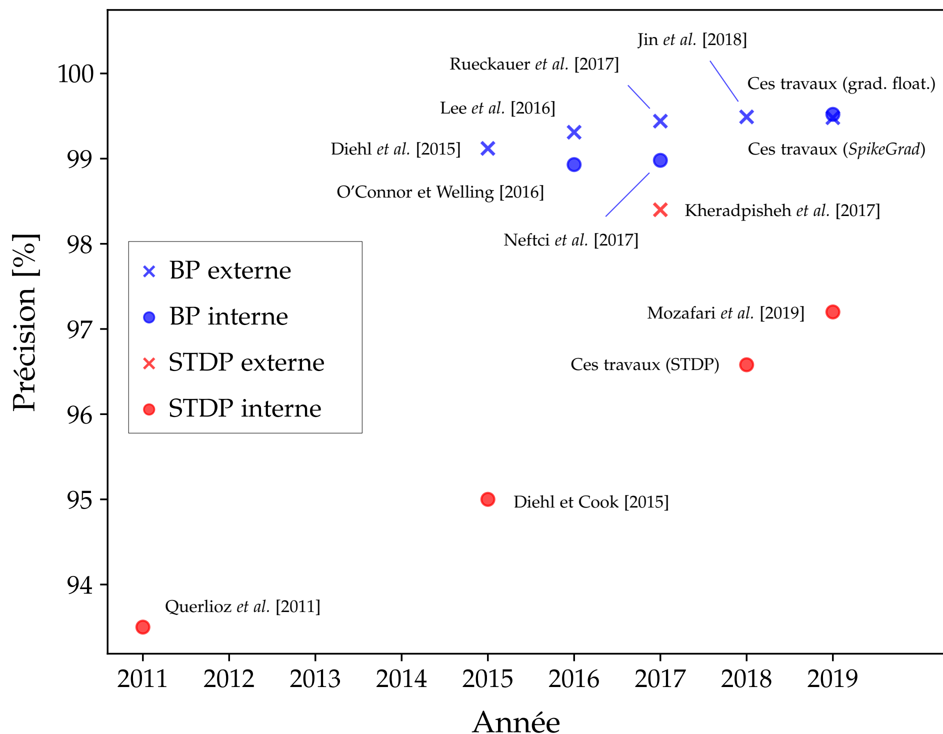

FIGURE 0.9: Développement des précisions d'inférence de l'état de l'art sur MNIST pendant les dernières années. On voit que les travaux initiaux utilisaient surtout des approches STDP. Alors qu'il y a encore de la recherche sur l'amélioration des performances des SNNs entraînés avec STDP, les meilleurs résultats des dernières années sont fournis par des méthodes qui utilisent des approximations de la rétro-propagation. *SpikeGrad* est la première méthode pour l'apprentissage interne sur puce qui donne des précisions comparables aux méthodes d'apprentissage externes. Veuillez noter que cette présentation est simplifiée et montre seulement les tendances générales qui sont observables dans la domaine. Surtout, nous n'avons pas pris en compte l'efficacité énergétique ou des différents contraintes matérielles, objectifs d'apprentissage et méthodes. Kheradpisheh *et al.* [2017] est marqué comme méthode externe à cause de l'utilisation d'un classifieur type «support vector machine».

l'apprentissage automatique, sont les plus adaptées pour obtenir des réseaux spike de haute performance d'inférence (figure 0.9). *SpikeGrad* est le premier algorithme d'apprentissage évènementiel qui permet d'obtenir, même dans des grands réseaux, des précisions d'inférence comparables aux réseaux évènementiels entraînés avec des méthodes d'apprentissage externe.

Cependant, nous croyons que la future proche des applications des réseaux spike sera plutôt l'inférence efficace que l'apprentissage continu. Bien que l'apprentissage continu soit intéressant d'un point de vue théorique, il est difficile aujourd'hui d'imaginer des systèmes industriels avec un tel degré d'autonomie. Le problème principal des réseaux spike, qui ralentit leur application industrielle, est l'absence d'un matériel pouvant implémenter un tel système évènementiel d'une manière efficace. De plus, le développement de ce genre de matériel nécessite des outils de production et des expertises très spécialisées. Cette complexité empêche aujourd'hui une standardisation industrielle.

Bibliographie

Peter U. Diehl et Matthew Cook : Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Frontiers in Computational Neuroscience*, 9:Article 99, August 2015.

Peter U. Diehl, Daniel Neil, Jonathan Binas, Matthew Cook, Shih-Chii Liu et Michael Pfeiffer : Fast-Classifying, High-Accuracy Spiking Deep Networks Through Weight and Threshold Balancing. *In International Joint Conference on Neural Networks (IJCNN)*, 2015.

Yingyezhe Jin, Peng Li et Wenrui Zhang : Hybrid Macro/Micro Level Backpropagation for Training Deep Spiking Neural Networks. *In Advances in Neural Information Processing Systems (NIPS)*, pages 7005–7015, 2018.

Saeed Reza Kheradpisheh, Mohammad Ganjtabesh, Simon J. Thorpe et Timothée Masquelier : STDP-based spiking deep convolutional neural networks for object recognition. *Neural Networks*, 2017.

Jun Haeng Lee, Tobi Delbruck et Michael Pfeiffer : Training Deep Spiking Neural Networks Using Backpropagation. *Frontiers in Neuroscience*, (10 :508), 2016.

Milad Mozafari, Mohammad Ganjtabesh, Abbas Nowzari-Dalini, Simon J. Thorpe et Timothée Masquelier : Bio-inspired digit recognition using reward-modulated spike-timing-dependent plasticity in deep convolutional networks. *to appear in : Pattern Recognition*, 2019.

Emre O. Neftci, Charles Augustine, Paul Somnath et Georgios Detorakis : Event-Driven Random Backpropagation : Enabling Neuromorphic Deep Learning Machines. *Frontiers in Neuroscience*, 11(324), 2017.

Peter O'Connor et Max Welling : Deep Spiking Networks. *arXiv :1602.08323v2, NIPS 2016 workshop "Computing with Spikes"*, 2016.

Damien Querlioz, Olivier Bichler et Christian Gamrat : Simulation of a Memristor-Based Spiking Neural Network Immune to Device Variations. *In International Joint Conference on Neural Networks (IJCNN)*, numéro 8, pages 1775–1781, 2011.

Bodo Rueckauer, Iulia-Alexandra Lungu, Yuhuang Hu, Michael Pfeiffer et Shih-Chii Liu : Conversion of Continuous-Valued Deep Networks to Efficient Event-Driven Networks for Image Classification. *Frontiers in Neuroscience*, 11(682), 2017.

Abhronil Sengupta, Yuting Ye, Robert Wang, Chiao Liu et Kaushik Roy : Going Deeper in Spiking Neural Networks : VGG and Residual Architectures. *Frontiers in Neuroscience*, 13:95, 2019.

Yujei Wu, Lei Deng, Guoqi Li, Jun Zhu et Luping Shi : Direct Training for Spiking Neural Networks : Faster, Larger, Better. *arXiv :1809.05793v1*, 2018a.

Yujei Wu, Lei Deng, Guoqi Li, Jun Zhu et Luping Shi : Spatio-Temporal Backpropagation for Training High-Performance Spiking Neural Networks. *Frontiers in Neuroscience*, (12 :331), 2018b.

Contents

1 Introduction **1**
1.1 High level introduction . . . 1
1.1.1 The rise of deep learning . . . 1
1.1.2 The energy problems of deep learning . . . 1
1.1.3 Event-based neuromorphic systems . . . 2
1.2 Thesis outline . . . 3
1.3 Document structure . . . 4

2 Problem Statement and State of the Art **7**
2.1 Basic properties of deep neural networks . . . 7
2.1.1 Frame-based artificial neurons . . . 8
2.1.2 Basic layer types . . . 8
2.1.3 The backpropagation algorithm . . . 11
2.2 Spiking neurons: a bio-inspired artificial neuron model . . . 15
2.2.1 Biological neurons . . . 15
2.2.2 The integrate-and-fire neuron . . . 16
2.2.3 Spiking neuromorphic hardware . . . 18
2.2.4 Information encoding in spiking networks . . . 20
2.3 Simulating event-based computation on classical computers . . . 25
2.3.1 Event-based simulation . . . 26
2.3.2 Clock-based simulation . . . 27
2.3.3 Our implementation choices . . . 28
2.4 Training algorithms for deep spiking networks . . . 28
2.4.1 Off-chip training . . . 29
2.4.2 On-chip training . . . 32
2.4.3 Performance comparison . . . 36

3 Online Learning in Deep Spiking Networks with Spike-Timing Dependent Plasticity **39**
3.1 Introduction . . . 39

3.1.1 Problem statement 39
3.1.2 Our approach 40
3.2 Methodology 42
3.2.1 Online learning constraints 42
3.2.2 Neuron model and STDP learning rule 43
3.2.3 Network topology and inhibition mechanism 46
3.2.4 Dual accumulator neuron 47
3.3 Experiments 48
3.3.1 Learning on a converted dataset 48
3.3.2 Extension to dynamic data 57
3.4 Discussion 65
3.4.1 Analysis of the architecture characteristics 65
3.4.2 Comparison to other approaches 69
3.4.3 Future outlook 70

4 On-chip Learning with Backpropagation in Event-based Neuromorphic Systems 73
4.1 Is backpropagation incompatible with neuromorphic hardware? 73
4.1.1 Local vs. non-local learning algorithms 74
4.1.2 The weight transport problem 75
4.1.3 Coding the gradient into spikes 77
4.1.4 Method comparison and implementation choice 77
4.2 The *SpikeGrad* computation model 78
4.2.1 Mathematical description 79
4.2.2 Event-based formulation 84
4.2.3 Reformulation as integer activation ANN 84
4.2.4 Compatible layer types 90
4.3 Experiments 91
4.3.1 A network for inference of relations 91
4.3.2 MNIST and CIFAR10 image classification 100
4.4 Discussion 104
4.4.1 Computational complexity estimation 104
4.4.2 Hardware considerations 106
4.4.3 Conclusion and future outlook 106

5 Summary and Future Outlook 109
5.1 Summary of our results 109
5.2 The future of SNNs 110
5.2.1 Training by mapping SNNs to ANNs 110
5.2.2 On-chip vs. off-chip optimization 111
5.2.3 Hardware complexity as a major bottleneck of SNNs . . 112

Bibliography 115

Chapter 1

Introduction

1.1 High level introduction

1.1.1 The rise of deep learning

While tremendous advances in computing power in the last decades have lead to machines that can handle increasingly sophisticated algorithms, even modern computers still lack the capacity to perform tasks in domains that seem intuitive to humans, such as vision and language. Creating machines that learn like humans from experience, and which can apply their knowledge to solve these kind of problems, has for a long time been a desire with comparably large investments of effort and relatively little practical success.

The field of artificial intelligence (AI) aims to build machines that are capable of performing exactly this type of tasks. In recent years, approaches to AI have become increasingly dominated by a class of algorithms loosely inspired by the human brain. *Deep Learning* (DL) is a class of *artificial neural network* (ANN) models that represents information by a hierarchy of layers of simple computational units, that are called artificial neurons. Deep learning has nowadays become a successful and well-established method in machine learning and provides state-of-the-art solutions for a wide range of challenging engineering problems (for recent reviews, see LeCun et al. [2015] or Schmidhuber [2014]). In particular, convolutional neural networks (CNNs) and recurrent neural networks (RNNs) are now used in a huge variety of practical applications, most notably computer vision and natural language processing.

1.1.2 The energy problems of deep learning

The development of deep learning has tremendously profited from the parallel improvement of Graphics Processing Units (GPUs). Although GPUs were primarily developed to process the expensive matrix operations used in com-

puter graphics and video game applications, it happens to be exactly the same type of computation that is necessary to execute large artificial neural network structures. The extremely high parallelism provided by this specialized hardware has allowed the implementation of extremely powerful systems for large-scale AI problems. While deep learning algorithms have shown impressive performance, they require an extreme amount of computation, in particular for the training procedure. The most advanced machine learning systems have become so calculation intensive that only a small number of industrial or academic laboratories are able to perform their training (Strubell et al. [2019]). This is why many of these systems are so far mostly used for centralized, high performance computing applications on large-scale servers (in the "Cloud").

Besides the huge investments in hardware and energy that are necessary to train deep neural networks, the calculation intensity also prevents their integration into systems with tight resource constraints. These are for instance Internet-of-Things (IoT) applications, autonomous systems and other systems that process data close to where it is acquired (at the "Edge"). Modern GPUs typically have a power consumption of several hundred Watts. Integrating AI algorithms in resource constrained environments however requires power consumption that is orders of magnitude lower. Recent years have seen a rise in specialized AI chips that are able to perform operations more efficiently by a combination of optimized processing infrastructure, precision reduction and topology optimizations (see for instance Jacob et al. [2017] and Howard et al. [2017]). These do however not alter the parallel, vectorized principle of computation. In particular, they cannot easily exploit the high level of sparsity found in many deep neural network architectures (see for instance Loroch et al. [2018] or Rhu et al. [2018] for recent empirical studies).

1.1.3 Event-based neuromorphic systems

This leads us to the foundational idea of *neuromorphic engineering*, a field of engineering that aims to construct AI systems inspired by the human brain. Despite some loose analogies to information processing in the brain, the aforementioned deep learning approaches are orders of magnitude less efficient in performing many cognitive tasks than our very own biological hardware. To achieve a more fundamental paradigm shift, it may thus be necessary to investigate a rather fundamental aspect in which GPU-based DL implementations differ from the real brain: while deep learning models are based on parallel floating point calculations, biological neurons communicate via asynchronous binary signals. So-called spiking neural network (SNN) models, which imitate this binary firing behavior found in the human brain, have emerged in recent years as a promising approach to tackle some of the challenges of modern AI hardware. These models could potentially be more energy efficient

due to the lower complexity of their fundamental operations, and because their event-driven nature allows them to process information in a dynamic, data dependent manner. The later aspect allows spiking neural networks to exploit the high temporal and spatial sparsity that can be found in the computational flow of many DL topologies, and which is difficult to be exploited by GPUs.

Because event-based systems work so differently from standard computers, mastering this kind of systems requires considering the intimate relationship between algorithm and computational substrate. The aim of this thesis is to design algorithms that take into account these particular properties of spiking neuromorphic systems.

1.2 Thesis outline

This thesis will present an investigation of SNNs and their training algorithms. Typically these algorithms can be divided into two subcategories: off-chip learning and on-chip learning. In the first case, the actual training of the SNN is performed on an arbitrary computing system. The only constraint is that the final, trained system has to be implementable on spiking neuromorphic hardware. The learning process therefore does not have to be compatible with the SNN hardware infrastructure. In the second case, the training of the system will also be performed on a spiking neuromorphic chip. This typically complicates the optimization procedure. Most efforts to reduce the energy footprint of both ANNs and SNNs have focused on the inference phase, since most current applications which utilize deep neural networks are trained on servers and then exported to mobile platforms. In the long term, it would however be desirable to have systems with the capability to learn on a low power budget and without the need of a working connection to a high performance computing system. Additionally, embedded learning is interesting from a theoretical perspective, since also biological brains are able to perform learning and inference in the same system simultaneously.

This is why this thesis focuses in particular on designing learning algorithms for this kind of embedded on-chip learning, with a particular focus on learning hierarchical representations in deep networks. While we will often talk in this context about spiking neuromorphic hardware, this thesis has a strong focus on algorithm design. This means that we take into account the constraints that are common to most potential spiking neuromorphic chips regarding the type of operations that can be used and how information is communicated between units. However, the main objective of this work is to find rather generic algorithms that are agnostic about the exact hardware implementation of the spiking neural network. While we try to design algorithms that could be suitable for both digital and analog implementations,

there will be a focus on digital implementations, since these allow an easier theoretical analysis and comparison to standard ANNs. If some fundamental aspects of the implementation depend clearly on the type of hardware that is being used, we provide a discussion that points out the potential issues.

As our general paradigm, our approach will be *bio-inspired*, but not *bio-mimetic*: while neuromorphic hardware is generally inspired by the human brain, we will only consider biological plausibility where it could prove useful from an engineering perspective. Our conceptual focus and the relationship of our research to other domains is visualized in figure 1.1.

1.3 Document structure

The remainder of this document is structured as follows:

- Chapter 2 explains general concepts and prior work that are necessary to understand the context of our research. It provides a brief overview of standard artificial neural networks and the backpropagation algorithm. It describes the basic properties of spiking neurons and how they represent information. We briefly discuss the main types of neuromorphic hardware and approaches to computer simulation of spiking neural networks. Finally, we present the state of the art of training algorithms for spiking neural networks.

- Chapter 3 presents an approach to online learning in neuromorphic systems, using a biologically inspired learning rule based on spike-timing dependent plasticity. The chapter will close with a discussion of the advantages and limitations of local, biologically inspired learning rules.

- Chapter 4 is concerned with the implementation of a neuromorphic version of the backpropagation algorithm. We propose *SpikeGrad*, a solution for an on-chip implementation of backpropagation in spiking neural networks.

- Chapter 5 features a discussion of all results in the context of the full thesis and gives an outlook on promising future research directions.

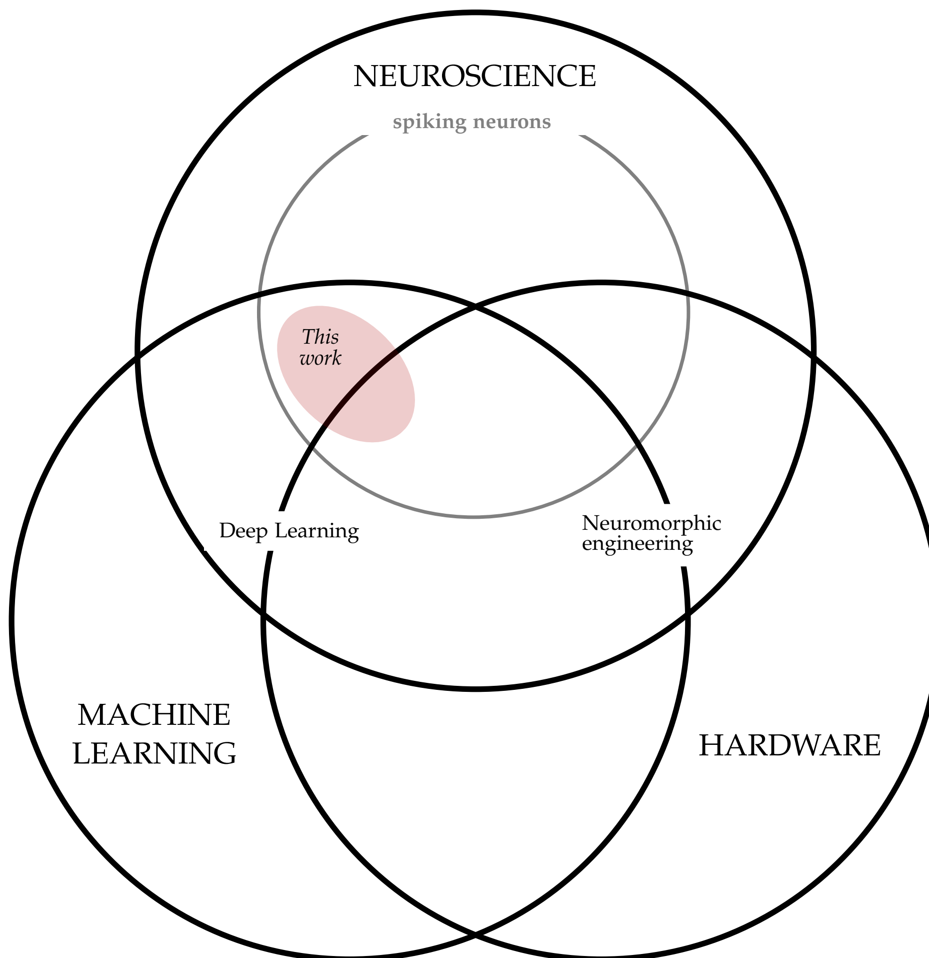

Figure 1.1: Schematic outline of the relationship of this thesis with other research domains. Deep learning can be seen as the application of some ideas from neuroscience in machine learning. Neuromorphic engineering aims to build hardware that mimics the workings of the human brain. We will focus on the utilization of artificial spiking neurons, a bio-inspired neuron model, for building deep neural networks. While this thesis has a clear focus on learning algorithm development, possible dedicated hardware implementations are taken into account in mechanism design.

Chapter 2

Problem Statement and State of the Art

This chapter gives a detailed overview over the main research problems that are addressed by this thesis. It contains a high level, self-contained description of the necessary concepts in deep learning and spiking neural networks that are necessary to understand the experiments in the later chapters of the document. Additionally, it features a detailed discussion of prior research in the field of spiking neural network training.

2.1 Basic properties of deep neural networks

This section gives a brief overview of traditional deep neural networks, how they are currently used to solve machine learning problems. It introduces the notions that are necessary to understand the differences between standard artificial neural networks and spiking neural network models, which are the main topic of this thesis. Additionally, it features a detailed derivation of the backpropagation algorithm that will be frequently referenced in later chapters. From now on, we refer to standard ANNs as *frame-based* ANNs or simply ANNs, while spiking neural networks are referred to as SNNs.

If not stated otherwise, the notation introduced in this chapter is based on a local, single neuron point of view. This allows a formulation that is more consistent with the local computation paradigm of neuromorphic hardware, which takes the single neuron and its connections as a reference point. The layer of a neuron is labeled by the index $l \in \{0, \ldots, L\}$, where $l = 0$ corresponds to the input layer and $l = L$ to the output layer of the network. Neurons in each layer are labeled by the index i. The incoming feedforward connections of each neuron are labeled by j and the incoming top-down connections in the context of backpropagation by k. Which neurons are exactly referenced by these indices depends on the topology of the neural network.

2.1.1 Frame-based artificial neurons

The output (or activation) y_i of a basic artificial neuron is described by the following equation:

$$y_i = a\left(\sum_j w_{ij}x_j + b_i\right). \tag{2.1}$$

The term in parentheses is a linear function of the inputs x_j. The parameters w_{ij} are called the *weights*. The term b_i is called *bias* and represents the offset of the linear function that is described by the weighted sum. The weights and biases are the main parameters of basically every artificial neural network. For a more compact representation, the bias value is sometimes represented as an element of the weights that always receives an input of 1. The *activation function a* is typically a non-linear function that is applied to the weighted sum.

In standard ANNs, all these variables are typically real-valued. We explain later that this stands in contrast to biological neurons, which typically output a binary spike signals. The biological justification for using real numbers as activations is based on the idea that the information propagated by the neuron is not encoded in the individual spikes, but in the firing rate of the neuron, which is real valued. However, ANNs used in machine learning have been mostly detached from biology, and real valued (floating point) numbers are simply used to represent larger variable ranges.

2.1.2 Basic layer types

We briefly summarize the properties of the most common ANN layer types, which are also used in almost all SNN implementations.

Fully connected

Fully connected (or "linear") layers represent the simplest form of connectivity. All Z_{out} neurons in a layer are connected to all Z_{in} neurons of the previous layer. Adding one bias value for each neuron, the number of parameters in a fully connected layer is therefore:

$$N_{\text{params}} = Z_{\text{in}} \cdot Z_{\text{out}} + Z_{\text{out}}. \tag{2.2}$$

Fully connected layers can be used in all layers of a neural network, but are typically found in the higher layers of a deep convolutional neural network.

Convolutional

Convolutional layers are a more specialized type of neural network layer and particularly popular in image recognition. In contrast to fully connected lay-

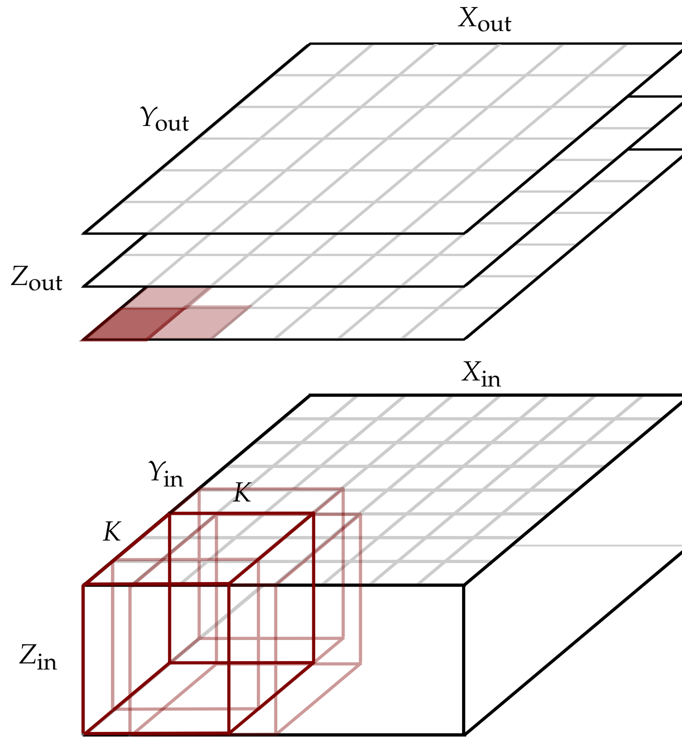

Figure 2.1: Visualization of a convolutional layer with parameters $K = 3$, $S = 1$ (without padding). The colored volume represents the weight kernel of size $K^2 \cdot Z_{in}$. The parameters K and S implicitly set the dimensions X_{out} and Y_{out} based on the dimensions X_{in} and Y_{in}.

ers, they preserve spatial information in the x and y dimensions of the image. Neurons in a convolutional layer are grouped into Z_{out} so-called *feature maps* (see figure 2.1). A feature map is parameterized by a weight kernel of size $K^2 \cdot Z_{in}$, where K is the kernel dimension in x and y direction, and Z_{in} the number of input channels (the z dimension of the input). For the first layer, Z_{in} is usually 3 (one dimension for each RGB channel). For higher layers, the number of input channels Z_{in} is the number of feature maps of the previous layer. This weight kernel is shared by all neurons that are part of the same feature map, which is called *weight sharing*. Each neuron in a feature map is only connected to a sub-region of size K^2 in the x and y dimensions of the input, but to the full depth Z_{in} in the z dimension. By connecting each neuron in the feature map to a different sub-region, the same kernel is applied to the full image. The output of a convolutional layer is of size $X_{out} \cdot Y_{out} \cdot Z_{out}$, where $X_{out} \cdot Y_{out}$ is the number of neurons in each feature map. While Z_{out} is chosen manually as a hyperparameter, X_{out} and Y_{out} are given implicitly by the dimension K of the kernel and the number of positions where it is applied in the input. These positions are calculated by shifting the kernel by a fixed number of pixels in the x and y directions. The number of pixels by

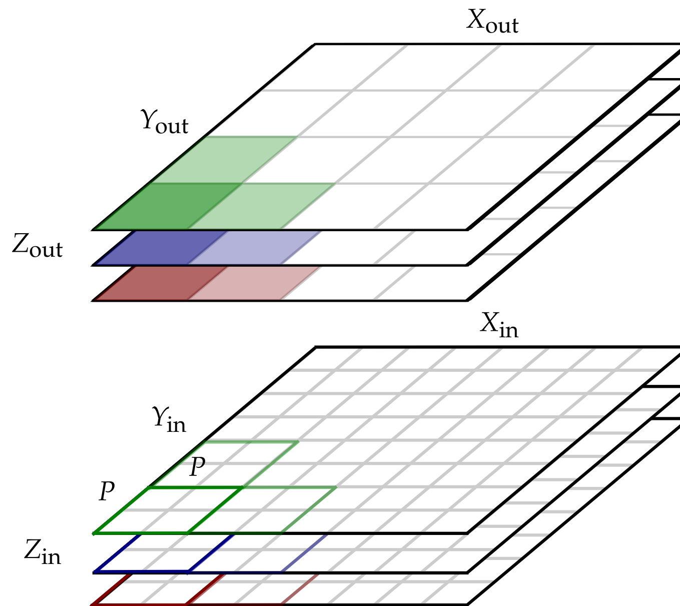

Figure 2.2: Visualization of a pooling layer with parameters $P = 2$, $S = 2$. In contrast to convolutional layers, pooling layers only connect to one feature map (represented by the 3 colors) and thus reduce the input only in the x and y dimensions.

which the kernel is shifted is called the *stride* S of the convolutional layer. A convolutional layer is therefore fully characterized by the kernel dimension K, the stride S and the number of feature maps Z_{out}. The number of learnable parameters in a layer is given by the number of feature maps times the size of the kernel, plus one bias value for each feature map:

$$N_{params} = K^2 \cdot Z_{in} \cdot Z_{out} + Z_{out}. \tag{2.3}$$

Often the input to a convolutional layer is artificially expanded by adding constant pixel values (typically zero) in the x and y dimensions. This process called *padding* is usually used to enforce certain dimensions X_{out} and Y_{out} of the convolutional layer, or to ensure that kernels size and stride fit exactly into the input dimensions X_{in} and Y_{in}. For instance, a kernel size of $K = 3$ with a stride of 1 is typically used with padding of 1 zero valued pixel at the boundaries of the input. This way we obtain $X_{out} = X_{in}$ and $Y_{out} = Y_{in}$.

Pooling

Convolutional layers are often combined with so-called pooling layers (see figure 2.2). Pooling layers serve to reduce the size of the input, typically in the x and y dimensions. This means that for the channel dimension of the pooling neurons we have $Z_{out} = Z_{in}$, and each pooling neuron connects only to neurons of a single feature map (other pooling connection patterns exist

but will be omitted here). The *pooling window* dimension P describes the sub-region of size P^2 to which the pooling operation is applied. Like convolutional layers, pooling layers have a stride parameter S. For most networks, pooling windows are non-overlapping and thus $P = S$. We will focus here on two types of pooling operations that are by far the most common: *max pooling* and *average pooling*. Max pooling propagates the activity of the neuron with the highest activity in the pooling window. Average pooling takes the average of all neuron activations in the pooling window.

Pooling layers are different from other neural network layers as they do not possess learnable parameters. They only perform a differentiable operation on the output of a previous convolutional layer.

2.1.3 The backpropagation algorithm

The tremendous success of modern deep learning techniques is to a large extent boosted by the backpropagation algorithm (Rumelhart et al. [1986]). The backpropagation algorithm can be seen as an efficient algorithm for the optimization of differentiable multi-layer structures. From an optimization perspective, it represents an implementation of the gradient descent algorithm, which is a standard method for function optimization in machine learning.

Gradient descent optimization

Let us define an artificial neural network structure as a differentiable function f with parameters (weights and biases) $\boldsymbol{w}$, which maps an input vector $\boldsymbol{X}$ (the data) to an output vector $\boldsymbol{y}$:

$$\boldsymbol{y} = f(\boldsymbol{w}, \boldsymbol{X}) \tag{2.4}$$

We now wish to find the optimal parameters $\boldsymbol{w}$ to produce a desired output of the network. For this purpose, we define a cost function $\mathcal{C}$ which serves as a measure of how close we are to this objective. The optimal parameters $\hat{\boldsymbol{w}}$ are those that minimize the cost function over the training patterns $\boldsymbol{X}$:

$$\hat{\boldsymbol{w}} = \arg\min_{\boldsymbol{w}} \mathcal{C}(\boldsymbol{y}(\boldsymbol{w}, \boldsymbol{X}), \boldsymbol{X}). \tag{2.5}$$

The gradient descent training algorithm tries to solve this problem by iteratively changing all parameters by a small amount in the negative (descending) direction of the gradient of the cost function with respect to the corresponding parameter:

$$\Delta \boldsymbol{w} = -\eta \frac{\partial \mathcal{C}(\boldsymbol{y}(\boldsymbol{w}, \boldsymbol{X}), \boldsymbol{X})}{\partial \boldsymbol{w}}, \tag{2.6}$$

or in a neuron based notation:

$$\Delta w_{ij}^{l} = -\eta \frac{\partial \mathcal{C}(\boldsymbol{y}(\boldsymbol{w}, \boldsymbol{X}), \boldsymbol{X})}{\partial w_{ij}^{l}}, \quad \Delta b_{i}^{l} = -\eta \frac{\partial \mathcal{C}(\boldsymbol{y}(\boldsymbol{w}, \boldsymbol{X}), \boldsymbol{X})}{\partial b_{i}^{l}}. \tag{2.7}$$

The learning rate η represents the step width by which we move in the direction of descending gradient. To solve the optimization problem, we therefore have to find these partial derivatives for all network parameters.

The standard approach is to calculate the gradient over the full training set $\boldsymbol{X}$ (a *batch*). This is however computationally expensive, since the whole training set has to be processed for a single parameter update. The other extreme is to calculate the gradient over a single, random training example, and update the parameters for each of these examples. Due to the randomness of the samples that are drawn from the training set, this approach is called *stochastic gradient descent* (SGD). SGD has lower memory requirements than batch gradient descent and can perform more updates given the same computational resources (Bottou and LeCun [2004]).

The drawback of SGD is that it cannot exploit the massive parallelism of GPUs. The gradient is therefore usually calculated with respect to a small number of randomly drawn training examples (a *mini-batch*) simultaneously, which allows parallelizing the computation. The optimal size of the mini-batch depends on the GPU memory constraints, but can also be an important hyperparameter of the optimization process.

Derivation of the backpropagation algorithm

Gradient descent is a general optimization algorithm that is in principle independent of the function that is optimized. As long as the function is differentiable, gradient descent can be applied to find at least a local minimum. We will now demonstrate that, in the case of an ANN, this derivative can be calculated exactly and computationally efficient. The fundamental idea is to use the chain rule from differential calculus to calculate exactly the derivative of the cost function $\mathcal{C}$ with respect to each network parameter. We start in the final layer of the network, which provides the input to the cost function:

$$\frac{\partial \mathcal{C}}{\partial w_{ij}^{L}} = \frac{\partial \mathcal{C}}{\partial y_{i}^{L}} \frac{\partial y_{i}^{L}}{\partial w_{ij}^{L}}, \qquad \frac{\partial \mathcal{C}}{\partial b_{i}^{L}} = \frac{\partial \mathcal{C}}{\partial y_{i}^{L}} \frac{\partial y_{i}^{L}}{\partial b_{i}^{L}}. \tag{2.8}$$

The first factor in both equations depends only on the choice of the cost function. The second term can be calculated based on the neuron model of the layer:

$$y_{i}^{l} = a^{l}(I_{i}^{l}), \quad I_{i}^{l} = \sum_{j} w_{ij}^{l} y_{j}^{l-1} + b_{i}^{l}. \tag{2.9}$$

We call the variable I_i^l the *integration* of the neuron, which represents the total weighted input transmitted to the neuron. We can see that the output y_i^l of each neuron depends on the outputs y_j^{l-1} of the previous layer. Using the chain rule, this gives for any layer:

$$\frac{\partial y_i^l}{\partial w_{ij}^l} = \frac{\partial a^l(I_i^l)}{\partial I_i^l} \frac{\partial I_i^l}{\partial w_{ij}^l} = a_i^{\prime l} y_j^{l-1}, \qquad \frac{\partial y_i^l}{\partial b_{ij}^l} = a_i^{\prime l}, \tag{2.10}$$

where we have used the short notation $a_i^{\prime l} := \partial a^l(I_i^l)/\partial I_i^l$. Note that this derivative is evaluated on I_i^l of the forward propagation phase, which requires us to store the I_i^l or $a_i^{\prime l}$ for each neuron so that it is available during the backpropagation phase. Substitution of (2.10) into (2.8) allows us to obtain the gradients for the top layer:

$$\frac{\partial \mathcal{C}}{\partial w_{ij}^L} = \frac{\partial \mathcal{C}}{\partial y_i^L} a_i^{\prime L} y_j^{L-1}, \qquad \frac{\partial \mathcal{C}}{\partial b_i^L} = \frac{\partial \mathcal{C}}{\partial y_i^L} a_i^{\prime L}. \tag{2.11}$$

As an example for a cost function that can be used, consider a cost function that consists only of the L_2 loss between the y_i^L of the top layer and the targets t_i:

$$\mathcal{C}(\boldsymbol{y}^L, \boldsymbol{t}) = \mathcal{L}_2(\boldsymbol{y}^L, \boldsymbol{t}) = \frac{1}{2} \sum_i (y_i^L - t_i)^2. \tag{2.12}$$

This yields for the gradients (2.11):

$$\frac{\partial \mathcal{C}}{\partial w_{ij}^L} = (y_i^L - t_i) a_i^{\prime L} y_j^{L-1}, \qquad \frac{\partial \mathcal{C}}{\partial b_i^L} = (y_i^L - t_i) a_i^{\prime L} \tag{2.13}$$

We now continue by calculating the gradient for the penultimate layer $L-1$:

$$\frac{\partial \mathcal{C}}{\partial w_{ij}^{L-1}} = \left(\sum_k \frac{\partial \mathcal{C}}{\partial y_k^L} \frac{\partial y_k^L}{\partial I_k^L} \frac{\partial I_k^L}{\partial y_i^{L-1}} \right) \frac{\partial y_i^{L-1}}{\partial w_{ij}^{L-1}}, \tag{2.14}$$

$$\frac{\partial \mathcal{C}}{\partial w_{ij}^{L-1}} = \left(\sum_k \frac{\partial \mathcal{C}}{\partial y_k^L} a_k^{\prime L} w_{ki}^L \right) a_i^{\prime L-1} y_j^{L-2}. \tag{2.15}$$

A comparison of (2.15) and (2.11) shows that term $\partial \mathcal{C}/\partial y_i^L$ was replaced by the sum in parentheses. These two terms therefore take the same role in both equations and can be seen as an error signal. In the top layer, this error signal is directly calculated based on the cost function. In the penultimate layer, the error term is calculated based on a weighted sum that weights the errors

in the top layer by the activation function derivatives $a_k'^L$ and the connecting weights w_{ki}^L. The error in the top layer is therefore "back-propagated" through the network. To write this more explicitly, we define the *error* in the top layer as:

$$\delta_i^L := \frac{\partial \mathcal{C}}{\partial y_i^L} a_i'^L \tag{2.16}$$

and for all other layers by the recursive relation:

$$\delta_i^l = \left(\sum_k \delta_k^{l+1} w_{ki}^{l+1} \right) a_i'^l. \tag{2.17}$$

Applying these definitions to equation (2.15) gives:

$$\frac{\partial \mathcal{C}}{\partial w_{ij}^{L-1}} = \left(\sum_k \delta_k^L w_{ki}^L \right) a_i'^{L-1} y_j^{L-2} = \delta_i^{L-1} y_j^{L-2}. \tag{2.18}$$

This can be generalized to any layer of the network. (2.16) and (2.17) can thus be used to obtain the expressions of the gradients in each layer:

$$\frac{\partial \mathcal{C}}{\partial w_{ij}^l} = \delta_i^l y_j^{l-1}, \quad \frac{\partial \mathcal{C}}{\partial b_i^l} = \delta_i^l. \tag{2.19}$$

This can now be substituted into (2.7) to obtain the update rule for all network parameters:

$$\Delta w_{ij}^l = -\eta \delta_i^l y_j^{l-1}, \quad \Delta b_i^l = -\eta \delta_i^l. \tag{2.20}$$

Backpropagation is thus a way to exactly solve the so-called *credit assignment* problem, that consists in identifying the exact contribution of a learnable network parameter to the output of the network. Only by identifying the contribution of each parameter, it is possible to globally optimize a multi-layer neural network to produce a desired output. We can see that the training of an ANN can thus be described by the set of basic equations (2.9), (2.17) and (2.20), including (2.16) for the top layer. Equations (2.9) and (2.17) describe the operations that have to be performed by each individual neuron. Since these operations are independent of other neurons in the same layer, they can be performed in parallel for all neurons of a layer. The computation that has to be performed by each neuron is in both cases a weighted sum, which is performed in hardware as a sequence of multiply-accumulate (MAC) operations.

The power of backpropagation lies in the fact that it can, like the feed-forward computations in ANNs, be implemented very efficiently on modern computing hardware, in particular GPUs. This allows the optimization of the network over many iterations of a large dataset, which is crucial for the training of very deep artificial neural networks.

2.2 Spiking neurons: a bio-inspired artificial neuron model

This remainder of this chapter motivates the use of spiking neural networks, which can be seen as an extension of ANNs that offers higher biological realism (Maass [1997]). We discuss major potential advantages of using this type of neuron, and a brief overview of hardware implementations of spiking neurons. Additionally, we discuss how spiking neurons can be simulated in computers. Finally, we present the state-of-the-art solutions for training SNNs.

2.2.1 Biological neurons

We review in this section the basic properties of biological neurons, as presented in introductory neuroscience books (such as Bear et al. [2007]). The description is rather high-level and omits most of the complexity associated with biological neurons. Since our goal is to be bio-inspired rather than biomimetic, we focus on those aspects of biological neurons that serve as the inspiration for the algorithms described in the later chapters of this work.

Neurons are a special type of cells that are found, among other cell types, in the human brain. They represent the cell type that is thought to be responsible for the execution of most cognitive tasks. On a high level of abstraction, a biological neuron consists of three fundamental parts: dendrites, soma and axon (see figure 2.3). The dendrites represent the incoming connections of a neuron that relay and process information received from other neurons. In particular, the dendrites are the part of the neuron where information is received via the synapses. Strictly speaking, a synapse does not belong to only one neuron, but represents the point of interaction between two neurons. It is the place where the axon of one neuron (the *presynaptic* neuron) connects to the dendrites of other neurons (the *postsynaptic* neurons). The axon is the *single* outgoing connection of a neuron, which transmits signals emitted from the neuron body. This body of the neuron, the soma, is the place where most of the typical cell organs are found. It can be imagined as the place where incoming information from the dendrites is integrated and channeled through the axon.

Information in the neuron is stored and transmitted through electrical charge. Changes in the synapses induced by signals from other neurons will change the ion concentration in the neuron, and therefore its *membrane potential*, which is produced by a charge difference between the interior of the neuron and the surrounding environment. If the membrane potential surpasses a certain threshold value, a rapid discharge of the neuron takes place, which resets the potential to its resting value. This discharge is propagated as a cascade via the axon of the neuron to synapses connecting to other neu-

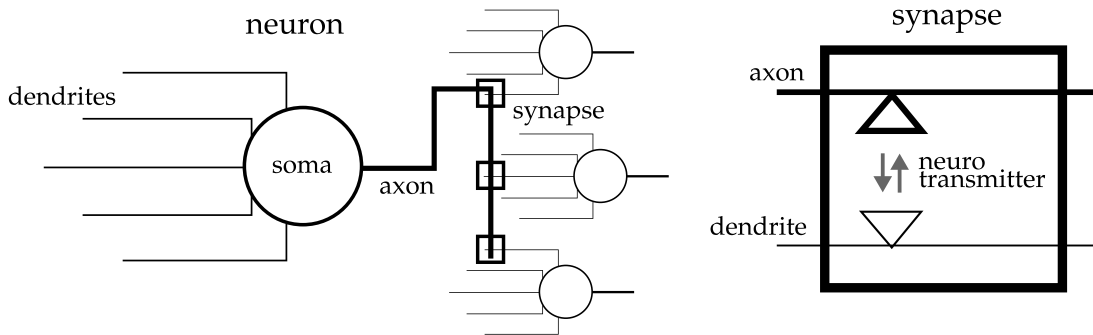

Figure 2.3: Abstract description of a biological spiking neuron

rons. There the discharge signal will cause changes in the synapses which will trigger an exchange of chemical neurotransmitters between the two synaptic terminals. These neurotransmitters then again lead to electrical changes in the receiving neuron. Additionally, synaptic plasticity may occur, which is the change of synaptic behavior triggered by electrical and chemical interaction between the sending and the receiving neuron. Due to the instantaneous, event-like nature of the discharges of the neurons, which are observable as a spike in the recorded membrane potential of a neuron, these signals are often simply referred to as *spikes*. The important point in this description is the fact that neurons are only able to propagate binary events. How these events are processed and how information can be effectively communicated with them can depend on a large number of factors, from the synaptic behavior up to the firing dynamics of the full neural systems.

This description of biological neurons will be sufficient for most of the discussions in this thesis. Neuromorphic systems using a more detailed neuron model exist (for example dendritic multi-compartment models such as used in Sacramento et al. [2018]), but will be left out here for clarity.

2.2.2 The integrate-and-fire neuron

Like traditional artificial neurons, spiking neurons can be seen as an abstract representation of biological neurons. The by far most popular mathematical model of spiking neurons is the integrate-and-fire (IF) model. IF models describe the membrane potential by a variable $V(t)$ that changes over time. To abstract from the biological inspiration for $V(t)$ as a potential, we refer from now on to $V(t)$ as the *integration variable*. This allows us to use a purely mathematical description without having to take physical units into account. There is a large number of alternative formulations of the integrate-and-fire model, which all put an emphasis on different properties of biological neurons. We restrict ourselves in this discussion to the formulation which we consider the most common in the related literature, and the most relevant for this thesis.

In particular, we do not consider any spike transmission delays or refractory periods.

Leaky integrate-and-fire neuron

The so-called leaky integrate-and-fire (LIF) neuron has the special feature that all variables that are integrated over time are subject to leakage: they decay to zero over time, with time constants λ and λ_{syn}. This decay serves as a kind of high-pass filter for incoming spike trains. A basic LIF neuron can for instance be modeled by a differential equation of the form:

$$\frac{\mathrm{d}V(t)}{\mathrm{d}t} = -\lambda V(t) + I(t), \quad I(t) = \sum_j w_j x_j(t). \tag{2.21}$$

The variables w_j represent the weights of the incoming connections of the neuron. The change of the integration variable is modeled by an input current $I(t)$, which is the weighted sum of so-called *synaptic traces* $x_j(t)$:

$$x_j(t) = \sum_{t_j^{\mathrm{s}}} e^{-\lambda_{\mathrm{syn}}(t - t_j^{\mathrm{s}})}. \tag{2.22}$$

In the case presented here, both synaptic traces and the integration variable are modeled as decaying variables. The traces are instantaneously increased by 1 every time a spike arrives at a time t_j^{s} at a synapse j, and then decay exponentially to 0. Every time the integration variable surpasses a threshold value Θ, a spike signal is emitted and $V(t)$ is reset to its base value (typically 0). Since the trace has an explicit dependence on the time of presynaptic spike arrival, this model can take into account the exact spike timing.

The LIF model is very popular for analog hardware implementations, since the integration and decay dynamics of the neuron can easily be modeled by the behavior of sub-threshold transistors and capacitors (see section 2.2.3).

Non-leaky integrate-and-fire neuron

While the LIF model is the most popular model for analog implementations, it is less popular for digital implementations, since the computation of the differential equation for every point in time can be very costly. This is especially true for the exponential function that has to be calculated for each synaptic trace for every point in time. For this reason, spiking neuron models for digital implementations often use an even higher abstraction of spiking neurons, which has no leakage and is only based on accumulations:

$$V(t + \Delta t) = V(t) + \sum_j w_j s_j(t). \tag{2.23}$$

Every input spike $s_j(t)$ will lead to an instantaneous (denoted by Δt) increase in the membrane potential by the value of the synaptic weight w_j. Spikes are typically modeled by a step function that is 1 when the integration variable surpasses the threshold Θ and 0 otherwise:

$$s\left(V(t)\right) = \begin{cases} 1 & \text{if } V(t) \geq \Theta \\ 0 & \text{otherwise} \end{cases}. \tag{2.24}$$

After each spike, the neuron is reset to 0 or decreased by a fixed value (typically Θ), which gives $s\left(V(t)\right)$ effectively a behavior similar to a delta function.

Because of the absence of leakage terms, this model is often simply called IF neuron or non-leaky integrate-and-fire neuron. The attractiveness of this neuron model for digital implementations is the absence of multiplication operations if $s_j(t)$ is the output of another IF neuron, since $s\left(V(t)\right)$ is either one or zero. Every input spike $s_j(t)$ will only trigger an accumulation that increments the integration variable by the weight value w_j.

2.2.3 Spiking neuromorphic hardware

Neuromorphic hardware can roughly be subdivided into analog and digital approaches. A detailed review of neuromorphic engineering can be found in Schuman et al. [2017].

Analog hardware

Analog hardware uses physical processes to model certain computational functions of artificial neurons. The advantage of this approach is that operations that might be costly to implement as an explicit mathematical operation can be realized very efficiently by the natural dynamics of the system. Additionally, real valued physical variables could in principle have almost infinite precision. An example are the decaying potential variables or synaptic traces in the LIF neuron model, which can be modeled by the dynamics of discharging capacitors, as described in pioneering work of Mead [1990] on analog neuromorphic hardware.

Analog hardware implementations differ on the degree to which analog elements are used. Many analog implementations only perform the computation in the neuron with analog elements, but keep the communication of spike signals digital. Examples of such *mixed-signal* neuromorphic hardware can be found for instance in Moradi et al. [2018], Qiao et al. [2015], Schmitt et al. [2017] or Neckar et al. [2019].

Major drawbacks of hardware using analog elements are high variability due to imperfections in the designed circuits (in particular when using memristors), thermal noise and difficulties in memory retention. Many of these problems grow with increasing miniaturization. Another problem is that the

timescale of the physical dynamics has to be matched with the timescale of the input, which depends for example on the size of electronic elements (Chicca et al. [2014]). To the best of our knowledge, basically all neuromorphic analog implementations of spiking neurons have so far been limited to the research domain.

Digital hardware

Digital hardware represents all variables of the neuron by bits, just like a classical computer. This means that the precision of variables depends on the number of bits that is used to represent the variables. This precision also strongly influences the energy consumption of the basic operations and the memory requirements for variable storage.

The great advantage of digital designs compared to analog hardware is that the precision of variables is controllable and guaranteed. Additionally, digital hardware can be designed with established state-of-the-art techniques for chip design and manufacturing.

A large number of digital designs exists for spiking neuromorphic hardware. Examples of industry designs of Application Specific Integrated Circuits (ASICs) include Davies et al. [2018] and Merolla et al. [2014], while Frenkel et al. [2018] and Lorrain [2018] represent academic research chips. Even the industrial designs have however not yet been integrated into many commercial products. Alternatively, due to the high production costs of ASICs, other research groups have focused on implementing SNNs in Field Programmable Gate Arrays (FPGAs) (for instance Yousefzadeh et al. [2017])

One drawback of digital designs is that they are suitable for the implementation of spiking neurons only to a limited extent. Classical high performance computing hardware, such as GPUs, is highly vectorized and heavily exploits parallelism in data processing. This kind of processing is very suitable for the operations that have to be performed in standard artificial neural networks. Spiking neural networks operate however in a strongly asynchronous fashion, with high spatial and temporal sparsity. An efficient implementation of a spiking neural network therefore requires a design that differs strongly from most existing high performance computing hardware. Typically, digital neuromorphic hardware uses the so-called *address-event representation* (AER) protocol, where values are communicated by events that consist of an address and a timestamp.

A particular problem when implementing the IF neuron in digital hardware is the large number of memory accesses that are necessary to update the integration variable with each incoming spike. In particular if the memory is not located in direct vicinity of the processing element that performs the operations, the high number of memory transfers can impact the energy budget more strongly that the computing operation itself (Horowitz [2014]).

In this thesis, we chose to design most algorithms with a digital implementation in mind. The main reason for this are the high precision requirements currently imposed by most applications that use deep learning solutions. Additionally, computation in digital hardware can be exactly modeled by computer simulation. Designing algorithms for analog circuits would require us to make a large number of assumptions on the potential hardware and its physical properties. Additionally, all state-of-the-art standard deep neural network solutions are implemented in digital hardware. It is therefore easier to compare a spiking network with a standard ANN in a simulated digital framework, since this does not necessarily require an actual dedicated hardware implementation.

2.2.4 Information encoding in spiking networks

This section briefly reviews the different approaches to encode information in spiking neural networks. We show that the optimal coding strategy is deeply connected to the choice of the neuron model, the hardware constraints and the application target.

Rate coding

The probably most common and best studied approach to encode information in SNNs is to use the firing rate. This rate can be either defined with respect to a certain explicit physical time (e.g. events per second), as it is usually done in neuroscience, or with respect to an implicit time (e.g. per stimulus or per example). The basic idea is to represent information in an accumulated quantity that has the same value irrespective of the exact time of spike arrival. This implies that the information representation capacity of the spiking neuron increases with the number of spikes it is able to emit in a certain time. Figure 2.4 demonstrates the principle of rate coding for different measurement interval sizes. It can be seen that depending on how long the observation window is, the measured firing rates can deviate quite strongly from each other. If the neuron should be able to represent the same number independent of the observation time, the firing should be as regular as possible. In a counting-based representation, where the observation window always has the same size, the timing of spikes is irrelevant and only the total number of spikes is required to represent a number.

In particular in digital implementations, the efficiency of rate coding depends on the cost of the operation induced by a single spike. If the cost of such an operation is rather high, a spiking network can be less efficient than its frame-based counterpart, since many spikes may be necessary to communicate the same information. For instance, we require n spike events to enable a neuron to represent integer numbers in the range $[0, n]$. To represent all pos-

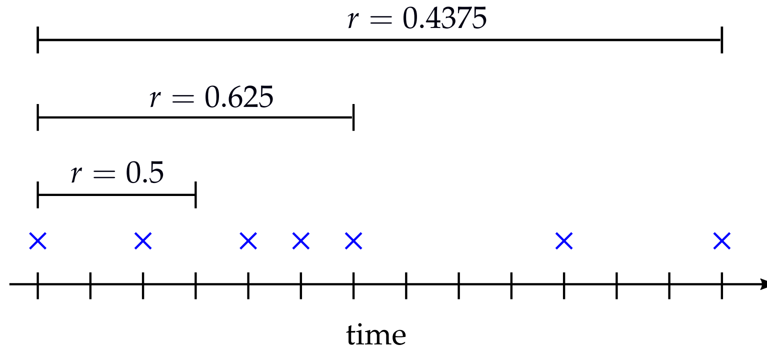

Figure 2.4: Demonstration of the rate coding principle. Blue crosses represent spike events. Firing rates r are given for three different observation windows.

sible values of an unsigned n-bit integer variable, $2^n - 1$ events are required (zero is represented by no event). This exponential growth in the number of events that is required to represent a value, relative to communicating the value as a single n-bit signal, means that rate coding can only be efficient if the required precision of the activation values is not very high.

Rate coding is still an interesting coding principle since it allows to dynamically adjust the number of computations in the network. This has two desirable effects: Firstly, the event-based processing of the SNN automatically roots the computation through the network topology. For instance, neurons that remain inactive will not trigger any operations in their outgoing connections. This process automatically distributes computations to the parts of the network that are the most relevant. The second advantage lies in the possibility to adjust the number of computations depending on the desired precision. In the IF model, it is possible for a signal to propagate to the next layers independently of the activity of other neurons in the same layer. This property is called *depth-first* propagation. This stands in contrast to traditional ANNs, where processing is usually layer-wise, or *breadth-first*. The advantage of depth-first processing is that all layers of the network can perform computation simultaneously. This can be an advantage in particular in a rate coding framework, since it allows the final layer to perform approximate inference based on the partial integration of spike signal. If only few input spikes are available, or if an approximate, low-latency response is desired, inference can be performed on the basis of a few output spikes. If higher precision is required, the integration time can be extended. As already mentioned, the condition for these dynamic precision properties to function properly is that spikes are approximately distributed homogeneously in time for all time interval sizes, i.e. the firing rate is constant.

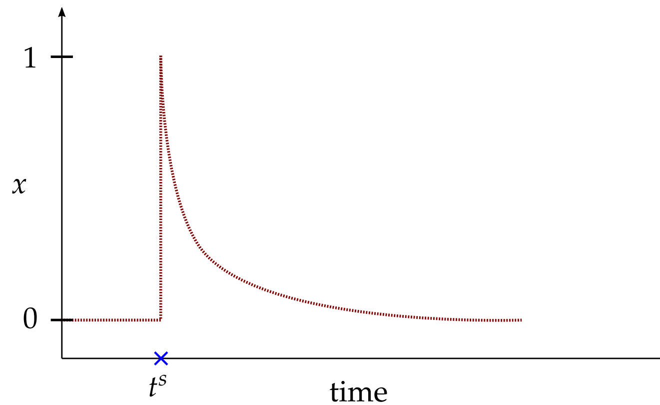

Figure 2.5: Demonstration of the temporal coding principle. At time t^s, the value of the synaptic trace x jumps from 0 to 1, and then decays continuously back to zero. The value of x is unique for any time $t > t^s$.

Temporal coding

A potentially more efficient coding strategy for SNNs is temporal coding, which uses the exact spike timing to encode information. In principle, the spike timing of a single spike (with respect to a reference point) is able to encode a real number with arbitrary precision. However, in practice it has proven to be rather difficult to build systems that use an effective temporal coding. This has several reasons: one of the most relevant is related to hardware, in particular, which type of hardware should be used for such a system. Consider for instance the leaky integrate-and-fire neuron (2.21) from the previous section. The crucial part of the model is equation 2.22, which describes the temporal dependency of the synaptic trace $x_j(t)$. It allows us to obtain a value for each trace that depends on the recent spike history. As it can be seen in figure 2.5, the trace x jumps from 0 to 1 at time t^s. It then decays continuously back to zero. For a single spike per synapse, the value of the trace is thus unique for each possible presynaptic spike arrival time t^s. The question is how this trace can be computed efficiently in hardware. Using digital hardware, calculating the exponential function is rather costly, and there is the risk to lose many of the computational advantages of the temporal coding scheme. Indeed, most of the potential advantages of temporal coding rely on the assumption that the trace and neuron dynamics can be calculated efficiently by the dynamics of a physical system. This requires the use of analog hardware, which means that the systems has to be tolerant with respect to all the typical defects and constraints of analog systems (mismatch of electronic parts, temperature dependence, etc.). This limits the use of analog hardware using temporal coding for practical neuromorphic systems in many cases, despite the large gains that could be achieved in principle. Another disadvantage of temporal coding is that the full spike pattern might be necessary to encode a

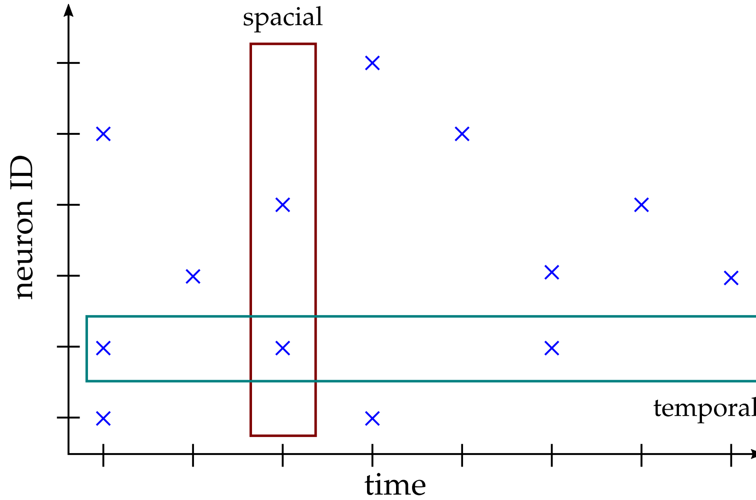

Figure 2.6: Visualization of spatial and temporal sparsity. Blue crosses represent non-zero values of the neuron output.

specific stimulus, since every spike can carry relevant information. This takes away the possibility to use only a sub-sequence of the stimulus for inference, as it can be done with rate coding. Another specific weakness of model (2.21) is that it is only able to high-pass filter a spike train: If a spike pattern arrives on a timescale much slower than the synaptic or neural decay dynamics, the trace or integration will decay to zero before integration reaches the firing threshold. If the same pattern arrives however much faster, the neuron will still become active, and even have a higher activity. This behavior is undesirable for a system that is supposed to be sensitive to a very specific timescale.

An alternative to using exact spike times is to consider only the order of spikes. This approach was inspired by the observation that recognition in the visual cortex seems to be too fast compared to the typical firing rates of neurons (Thorpe et al. [1996]). Most neurons would effectively only have the possibility to spike a few times, which makes it impossible to use their rate for accurate information representation. This is why Thorpe et al. [2001] suggested a coding scheme where every neuron spikes only once, and information is encoded in the order of these spikes. Compared to the rate coding approach, this leads to a large increase in the information that can be encoded using only a few spikes, since for N spikes, there are $N!$ possible ways to arrange their firing order.

Exploiting spatial and temporal sparsity

A fundamental advantage of spiking neural networks is their capacity to naturally exploit sparsity in computation. Sparsity in this context means than the number of operations that has to be performed is effectively much small

than the number of operations that is theoretically possible. We will distinguish here two types of sparsity which are both exploitable by SNNs: the first type of sparsity, which we call *spatial sparsity*, is the ratio of inactive units compared to the total number of units that could be active at a given point in time. In the context of an artificial neural network, this is the number of neurons that produce a non-zero signal at a given point in time. In a frame-based ANN, this means that the output of a neuron is exactly 0. In an SNN, this means that no spike event is produced. This concept of sparsity is typically used in linear algebra, where the sparsity of a matrix is the relative number of elements that are equal to zero. The second type of sparsity, which we refer to as *temporal sparsity*, is the number of times a specific unit is active in a given time window, relative to the number of times it could be active in principle. In the context of an ANN, this would be the number of times a neuron is active for a sequence of inputs, divided by the sequence length. In an SNN, this is the number of times a neuron emits a spike relative to the number of times this would have been possible. Note that for an SNN, this concept is only clearly defined if we assume discrete time with a finite number of time points at which the neuron can be active. If time is continuous, the maximal number of times that a neuron can be active is given implicitly by the physical constraints of the hardware.

In an ANN, both types of sparsity are potentially difficult to exploit. Spatial sparsity is difficult to exploit since most ANN hardware is optimized to perform the operations of all neurons in parallel. As we have mentioned before, (2.17) and (2.9) are usually calculated for all neurons of a layer in parallel. The computation is always performed in the same way, even if most or all of the inputs to the neuron are zero. Checking for each neuron which inputs are inactive would break the homogeneity of computation and destroy many of the benefits of high parallelism. It is therefore usually faster to perform the useless multiplications of the zero elements. Temporal sparsity is difficult to exploit since this would require an additional operation that checks at each point in time if the output of a neuron is zero or not. If most of the network is inactive, this requires a large number of these control operations.

SNNs can exploit both types of sparsity in a natural way. A computation is only triggered if a specific neuron at a specific time emits a spike event. A low number of active neurons at a specific point in time, or a low activity of a neuron in a specific time window, will therefore automatically lead to a lower number of operations in the network compared to an ANN.

We want to note here that this discussion is simplified and does not take into account many of the potential subtleties of actual hardware implementations. It serves only as a high-level description of the type of problems that are related to exploiting sparsity in hardware that can be addressed by spiking neurons. In particular, we do not take into account potential mechanisms in ANNs to ignore zero values (such as Aimar et al. [2019]).

Matching learning objective with hardware and coding strategy

These different coding mechanisms demonstrate the importance of matching the coding mechanism of a spiking neural networks with application objective, input encoding and hardware constraints.

For instance, a network using an exact time coding is necessarily sensitive to the time scale of the input. While this is exactly what is desired for the detection of a specific temporal pattern, it may be undesirable in the case of an input which is characterized by spatial rather than temporal dependencies, such as an image. Even in the case of patterns where some dynamic information is desired, it may be more useful to encode information in accumulated spike patterns over characteristic time windows rather than using the exact spike times. This is actually the case for the LIF neuron (2.21) and learning algorithms that learn on the values of accumulated synaptic traces rather than the exact spike times. The exponentially decaying synaptic trace loses temporal information as soon as several spikes can arrive at a synapse, since its value is simply a sum of the temporally weighted contributions of each spike. An almost infinite number of spike patterns can lead to the same value of the synaptic trace at a specific point in time. The learning algorithm thus has no possibility to distinguish the contribution of each spike if it is only provided the value of the trace, without the explicit timing information. In this case, the trace is rather a presentation of a short-term presynaptic firing rate. This is desirable for scenarios where the system should be sensitive to a specific timescale of the input signal, which is however still encoded in noisy short-term firing rates, rather than exact spike times. This is the type of input that is typically provided by event-based visions sensors, such as described by Posch and Wohlgenannt [2008] and Lichtsteiner et al. [2008].

For all presented event-based information encoding mechanisms, a major advantage is the possibility to naturally perform event-based processing, i.e. communication and computation that is solely driven by external input stimuli. The system remains idle if no stimulus is provided, which can be a key element in achieving low power consumption.

2.3 Simulating event-based computation on classical computers

While the algorithms developed in this thesis are designed for dedicated spiking neuromorphic hardware, our aim is to stay as agnostic as possible to the specific type of hardware implementation. Additionally, since the field is still developing rapidly, a general hardware that would allow us to easily implement these algorithms does not exist. We therefore rely on computer simulations to test the performance of the algorithms. We analyze two approaches.

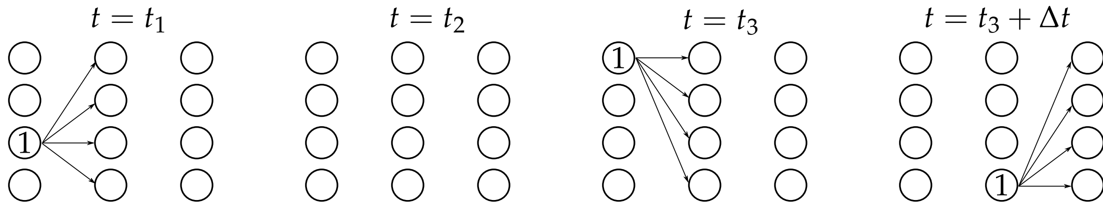

Figure 2.7: Processing in an event-based simulator. Two spike are emitted by the first layer at $t = t_1$ and $t = t_3$. The second events leads to a spike in the second layer at $t = t_3 + \Delta t$.

2.3.1 Event-based simulation

Event-based simulators explicitly simulate each event that is triggered in the network using a type of AER protocol. Each event, with its address and timestamp, is propagated to all neurons that connect to the emitting neuron. These neurons are updated and then potentially emit events by themselves. This recursive propagation of events is continued until all events have been propagated and no new events are triggered. The event-based simulator is therefore highly sequential. Additionally, it simulates realistically the event-based spike routing of a spiking neuromorphic system. A visualization can be seen in figure 2.7: two spike events arrive from the lowest layer at times $t = t_1$ and $t = t_3$. During t_2, no events are triggered and the system remains inactive. The second event at t_3 triggers an event in the second layer at $t_3 + \Delta t$. Δt represents the minimal time it takes for a neuron to respond to an incoming spike. In hardware, there can be a minimal response time imposed by the constraints of the hardware system or by transmission delays.

An event-based simulation offers the advantage that the timing of spikes can be represented with very high precision. Additionally, the number of computational operations is proportional to the number of events. This can however become a problem when the number of events becomes very large. Also, because of their sequential nature, event-based simulators are difficult to parallelize and have problems leveraging the speedups brought by GPUs.

In general, it can be said that event-based simulators are very suitable for simulations of small networks, with a small number of events, which require high temporal precision. This can be for instance the simulation of networks using temporal coding or neuromorphic hardware with complex event-based learning rules. The event-based simulation paradigm loses its beneficial properties if the simulation requires operations that are not event-based, but simultaneously performed for a large number of neurons (e.g. the continuous integration of synaptic traces in (2.21)).

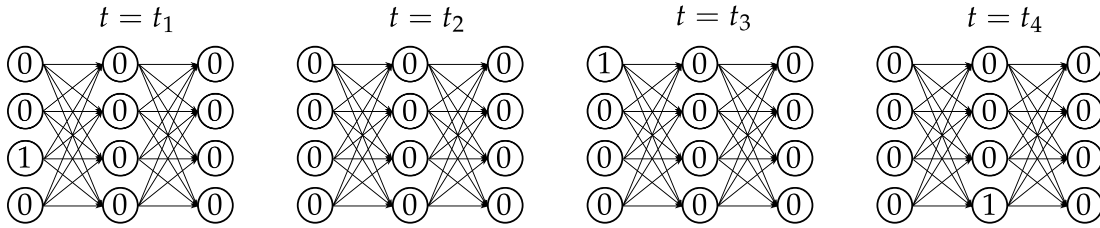

Figure 2.8: Processing in a clock-based simulator. Spikes are emitted by the first layer at $t = t_1$ and $t = t_3$, and by the second layer at $t = t_4$. At each time step, sparse matrices are propagated through the network, where 1 indicates an event and 0 indicates that no event was emitted.

2.3.2 Clock-based simulation

By far the largest number of spiking neural network simulators are clock-based (see Hazan et al. [2018] for an overview). Time is discretized into a finite number of points when all neurons are updated simultaneously. Spikes are propagated in the form of binary matrices that contain 1s and 0s (see figure 2.8).

Clock-based simulators are highly parallel and their computational complexity scales linearly with the number of time steps. At the same time, their computational complexity is independent of the number of events per time step. Because of the propagation of spike matrices, clock-based simulations are structurally similar to simulating a frame-based ANN. This is why many clock-based simulators are built on-top of existing deep learning frameworks. In particular if activations are sparse, a large number of redundant operations is performed, since many of the matrix elements will be zero. Also, the number of operations can be very large if high temporal precision and many time steps are required. In figure 2.8 at time $t = t_3$, no spike events are triggered in the network. The simulator will however still process all the zeros, although they do not necessarily change the neuron state variables.

Clock-based simulators are useful for simulating large networks, in particular when a large number of events is triggered. They are very efficient in cases where the temporal resolution can be rather coarse, since then only a small number of time steps is required. This makes them suitable for simulation of large spiking networks based on a rate or accumulation code as discussed in 2.4. Because of their high parallelism, they can profit from GPU acceleration. Clock-based simulators are also advantageous if the network is not fully event-based. This can be for instance the case when the neuron integration variable is described by a differential equation which has to be integrated over time for all neurons in the network (for instance the neuron model (2.21)).

2.3.3 Our implementation choices

Which simulator is optimal depends on the type of network that should be simulated. In particular for counting-based spike processing, where no exact temporal information is required and a large number of spikes is emitted, the clock-based simulator is much faster. The current version of the N2D2 library of Bichler et al. [2019], which we used for all simulations in this work, contains an event-based simulator for spiking networks. This is why our first experiments in chapter 3 use an event-based simulator for the static MNIST dataset. During these experiments, we became aware of the limitations of the event-based simulator for this type of experiments, and a CUDA accelerated clock-based simulator was implemented. This simulator was then used for the dynamic dataset. Even for the comparably small networks used in chapter 3, we found a speedup when using the clock-based simulator, even when no parallelization over mini-batches was used. For the simulation of the algorithm in chapter 4, we further accelerated the simulation by switching from hand-designed CUDA kernels to the specialized operations provided by the cuDNN library (Chetlur et al. [2014]). The clock-based simulation paradigm allows us to use the cuDNN primitives for convolutions and average pooling. All other matrix operations are directly implemented with cuBLAS. This lead again to a considerable speedup of the simulation.

2.4 Training algorithms for deep spiking networks

In the previous section, we explained the types of coding mechanisms that can be used to represent information in spiking neural network. This leaves the open question how such a representation can be learned.

The main purpose of this section is to give the context of the problems we consider in the remainder of this thesis and how our work can be positioned with respect to previous results. It will present a high level discussion of the current state of the art for learning (hierarchical) representations in spiking networks. Since in this thesis we do only consider image recognition problems, we limit the discussion here to results on the MNIST and CIFAR10 benchmarks.

The number of learning rules for spiking neurons that has been conceived in the context of theoretical neuroscience is enormous. We will limit our discussion here to learning rules that have been demonstrated to be suitable for multi-layer neural networks and which are thus interesting for deep learning problems. For an alternative presentation of the state of the art of deep learning in spiking neural networks, the reader can refer to recent review articles of Pfeiffer and Pfeil [2018], Tavanaei et al. [2019] and Bouvier et al. [2019].

2.4.1 Off-chip training

Off-chip training describes the training of a network using a different computing infrastructure than the neuromorphic chip that implements the network after training. For instance, training is performed on a high-performance computing cluster and the network is subsequently deployed on a substrate with low power consumption. The advantage of off-chip training is that the optimization algorithm can be computationally much more complex than the computation that has to be performed during inference. The obvious drawback of this approach is that learning is impossible once the system is deployed on the chip. The SNN can therefore not adapt its parameters without access to the system used for optimization.

Conversion of ANNs to SNNs

One of the oldest approaches for training SNNs is to train an ANN with backpropagation and map the parameters of the ANN to the SNN after training (see for instance Diehl et al. [2015], Esser et al. [2016], Neil and Liu [2016], Cao et al. [2015], Rueckauer et al. [2017] and Sengupta et al. [2019]). This approach is often called "spike transcoding" or "spike conversion". The main advantage of this approach is that the optimization process can be performed on an ANN. This permits the use of state-of-the-art optimization procedures and GPUs for training. The best results for most benchmark tasks are currently provided by this approach. A simulation of the exact spike dynamics in a large network can be computationally expensive, in particular if high firing rates and precise spike times are required (see section 2.3). This is why training an SNN as an ANN is so far the only approach that allows to scale SNNs to complex benchmark tasks that require large networks, such as ImageNet (Russakovsky et al. [2015]), since only the test phase has to be explicitly simulated with spikes. The main disadvantage of this approach is that some particularities of SNNs, which do not exist in the corresponding ANN, cannot be taken into account during training. For this reason the inference performance of the SNN is typically lower than that of the original ANN.

Most conversion approaches are based on the firing rate. Rueckauer and Liu [2018] present a conversion approach based on exact spike time coding and demonstrate that the number of spikes can be greatly reduced by this approach, however with a certain loss in accuracy.

Gradient descent on simulated spike dynamics

The most popular optimization algorithm for ANNs is gradient descent using backpropagation of errors. Because of the success of the backpropagation algorithm in training of standard ANNs, the recent years have seen a large increase in methods trying to apply backpropagation directly to SNNs.

Architecture	Method	Rec. Rate
Diehl et al. [2015]	CNN converted SNN	99.12%
Rueckauer et al. [2017]	CNN converted to SNN	99.44%
Rueckauer and Liu [2018]	CNN converted to SNN (temporal coding)	98.53%

Table 2.1: Comparison of different state-of-the-art spiking architectures converted from ANNs, and their inference performance on MNIST.

Architecture	Method	Rec. Rate
Esser et al. [2016]	CNN converted to SNN	89.32%
Rueckauer et al. [2017]	CNN converted to SNN (with BatchNorm)	90.85%
Sengupta et al. [2019]	CNN (VGG-16) converted to SNN	91.55%

Table 2.2: Comparison of different state-of-the-art spiking CNN architectures converted from ANNs, and their inference performance on CIFAR10.

Backpropagation based on accumulated information This class of algorithms uses quantities that are accumulated over several spikes and maps them to the parameters of an ANN. One of the main problems in applying backpropagation to spiking networks is the non-differentiability of the activation function. Take for instance the IF model (2.2.2). Both the integration variable $V(t)$ and the activation $s(t)$ are discontinuous when a spike is fired. Most of these algorithms use therefore a form of approximation of the derivative of the activation function to circumvent this problem. This allows to backpropagate gradient information through the whole network just as in a standard deep neural network. Lee et al. [2016] and Panda and Roy [2016] use the integration variable as a surrogate for the neuron activation and ignore the discontinuities at spike time. O'Connor and Welling [2016] use the total accumulated input. Yin et al. [2017] and Neftci et al. [2017] use a so-called straight-through estimator (see Bengio [2013] for the original use in stochastic neural networks and Yin et al. [2019] for a theoretical analysis). Zenke and Ganguli [2018], Huh and Sejnowski [2018], Wu et al. [2018a], Samadi et al. [2017] and Wu et al. [2019b] use smooth surrogates defined on the integration variable. An alternative approach is used by Severa et al. [2019]: instead of using a surrogate derivative, the network starts with a continuous activation function that is slowly transformed into a step function during training. Binas et al. [2016], Wu et al. [2019a] and Tavanaei and Maida [2019] use the accumulated neuron outputs as a representation of the activation function during backpropagation.

This class of approaches generally achieves good classification accuracy on static benchmark tasks, such as image recognition. One advantage is that some particularities of the spike coding can be taken into account (for instance the typical firing rates or activation ranges that are possible). These approaches do however not take explicit timing into account, since they operate only on quantities accumulated during forward propagation. Therefore

they cannot learn temporal patterns. They can be said to emphasize the dynamic precision and sparse coding aspect of SNNs.

Backpropagation using temporal information Another axe of research tries to view spiking neural networks as recurrent neural networks (RNN). This term is used here for all neural network types where the state of a neuron depends on its previous states. The discretized dynamics of recurrent neural networks have to be set into relationship with the timescale of the input, which is typically presented in the form of frames (i.e. arrays of numbers). This is exactly the same process which is used in the discrete time clock-based simulation of SNNs. It is therefore natural to model the discrete time dynamics of an SNN as a recurrent neural network. In the case of an IF neuron, the dependency on previous network states is represented by the previous values of the integration variable and the reset term. The reset value can be modeled by a (learnable) recurrent weight. By using backpropagation through time (BPTT), the recurrent weights can be trained by unrolling the temporal dynamics of the SNN, just like it is usually done in RNNs (see for instance Wu et al. [2018a], Wu et al. [2018b], Bellec et al. [2018] and Neftci et al. [2019]). Huh and Sejnowski [2018] map an SNN to an RNN that is modeled by differential equations and use control theory to derive the gradient. Zenke and Ganguli [2018], Jin et al. [2018] and Shrestha and Orchard [2018] represent approaches that include spike history into the gradient calculation. As in the case of non-recurrent SNNs, these approaches have to deal with the non-differentiability of the activation function. This is typically done in a similar way as in the non-temporal case, by using a differentiable surrogate.

The number of learning algorithms that are capable of using continuous time to represent information is much more limited. Bohte et al. [2000], Mostafa [2017], Mostafa et al. [2017] and Liu et al. [2017] use approximations of the backpropagation algorithm directly defined on the spike times, while Shrestha and Orchard [2018] use learnable transmission delays. The main difficulty is the need to find a learning algorithm that is compatible with the neuron model that is imposed by the hardware constraints. In particular, gradient descent optimization requires all operations to be differentiable. The large advantage of these networks is that they can utilize very sparse codes based on a few spikes. However, the strong constraints imposed on the learning mechanism often lead to lower inference performance compared to models based on firing rates. Additionally, while rate codes can still perform an approximate inference with a low number of spikes, temporal networks often rely on the full spike pattern to perform inference correctly.

Architecture	Method	Rec. Rate
Panda and Roy [2016]	Unsupervised/direct BP	99.08%
Lee et al. [2016]	BP rate coding	99.31%
Yin et al. [2017]	BP rate coding	99.44%
Liu et al. [2017]	BP temporal coding	99.1%
Wu et al. [2018a]	BP rate/temporal coding	99.42%
Jin et al. [2018]	BP rate/temporal coding	99.49%
Shrestha and Orchard [2018]	BP rate/temporal coding	99.36%

Table 2.3: Comparison of different state-of-the-art spiking architectures trained offline on MNIST.

Architecture	Method	Rec. Rate
Panda and Roy [2016]	Unsupervised/direct BP	70.16%
Wu et al. [2018b]	BP rate/temporal coding (no NeuNorm)	89.32%
Wu et al. [2018b]	BP rate/temporal coding (NeuNorm)	90.53%

Table 2.4: Comparison of different state-of-the-art spiking architectures trained offline on CIFAR10.

Other off-chip optimization techniques

Like frame-based ANNs, the parameters of an SNN can be found with a large number of alternative optimization procedures. This encompasses probabilistic methods, genetic algorithms or even analytically exact solutions. In particular the computational neuroscience literature contains a large number of methods that find an optimal configuration of a frame-based or spiking neural network for a specific task (for an introductory presentation, see for instance Dayan and Abbott [2001]). However, most of these methods are computationally expensive and do not scale as well to large networks as gradient descent techniques using the backpropagation algorithm. We will therefore not present an extensive analysis of these approaches in this thesis.

2.4.2 On-chip training

On-chip training methods are designed to be applicable on the same chip architecture that is used for inference. They therefore have to be compatible with its hardware constraints. Since typically neuromorphic chips are designed to be applicable in resource constrained environments, this imposes strong restrictions on the type of learning mechanism that can be used.

Hardware constraints for on-chip learning

Designing dedicated hardware is a complex process which requires to take into account a large number of interdependent considerations. In this thesis,

we restrict our analysis to a number of constraints that are typically required to make an on-chip implementation of an algorithm at least plausible:

- Memory: Spiking neural networks tend to have high memory requirements, since at least the integration variable of each neuron has to be saved. If any other additional variables are used, such as synaptic traces, these requirements increase.
- Calculation infrastructure: One of the main objectives of spiking neural networks, in particular in digital implementations, is to avoid multiplication operations. On-chip algorithms should therefore use additions and comparisons only, or operations that can be easily implemented by analog computation.
- Communication: Spiking networks communicate by events, which typically have a polarity and a timestamp. We therefore constrain the range of values that can be communicated by an event to $\{-1, 0, 1\}$, where 0 represents no event. Many implementations restrict spike events to binary values only, which is an even stronger constraint.

Biologically inspired approaches

For many years, spike-timing dependent plasticity (STDP) has been the algorithm of choice for the implementation of machine learning tasks in spiking neuromorphic systems. STDP is a phenomenon observed in the brain (Bi and Poo [1998]) and describes how the efficacy (loosely speaking, the weight) of a synapse changes as a function of the relative timing of presynaptic and postsynaptic spikes (see figure 2.9 for a visualization). A presynaptic spike in this context is the spike arriving at the synapse of the neuron. The postsynaptic spike is the spike emitted by the neuron itself. The mechanism of STDP is based on the idea that those synapses that are likely to have contributed to the firing of the neuron should be reinforced. Similarly, those synapses that did not contribute, or contributed in a negative way, should be weakened. This selection is performed, as the name of the mechanism suggests, on the basis of temporal correlation. The temporal signature of presynaptic and postsynaptic spiking is therefore used as a proxy for causality. In this aspect STDP, differs from another well-known learning paradigm from theoretical neuroscience that is also based on the reinforcement of correlated firing: Hebbian learning (Hebb [1949]). The basic concept of Hebbian learning is often summarized with the quotation: "*What fires together, wires together*", and is based on the correlation of firing rates between neurons with respect to an arbitrary reference window. All information regarding the exact timing of spikes is lost. STDP, on the other hand, tries to take into account the timing of every spike in the learning rule. The obvious advantage of this approach is that

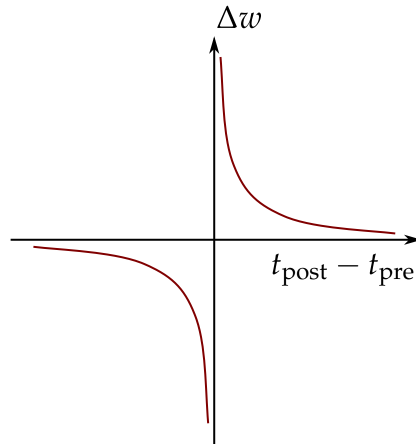

Figure 2.9: Schematic description of STDP. Δw describes the change of the synaptic weight, which depends on the relative timing of presynaptic and postsynaptic spike times t_{pre} and t_{post}. In the version shown here, synapses that received a spike before the postsynaptic spike of the neuron are strengthened, while synapses that receive a spike afterwards are weakened. The rule therefore can be seen as reinforcing causality.

the potential information representation capacity is significantly higher, since temporal spike coding can be used.

The popularity of STDP in the neuromorphic community has several reasons: first of all, the field of neuromorphic computing has traditionally been inspired by biology. This is the reason why early approaches for learning in neuromorphic hardware have been inspired by mechanisms observed in the brain. Additionally, STDP is easy to implement in analog neuromorphic hardware. The time dependence of STDP is often modeled by an exponential decay, which can easily be calculated by analog electronic elements. This is why many early works on learning in neuromorphic hardware have used STDP for feature extraction. Based on the particular learning objective and hardware implementation that is envisioned, architectures using STDP can differ to a large extent, which makes their comparison difficult. In the context of memristive and analog implementations, Querlioz et al. [2011], Querlioz et al. [2015], Bichler et al. [2011] and Diehl and Cook [2015] use deterministic versions of the LIF neuron in combination with a winner-takes-all (WTA) mechanism. In the framework of stochastic neurons, Nessler et al. [2013], Habenschuss et al. [2012] and Habenschuss et al. [2013] show that a specific version of STDP in a winner-takes-all circuit can approximate the Expectation Maximization algorithm. More recent spiking deep network models trained with STDP have as objective to train multi-layer spiking networks with increasing precision. The deep spiking architectures of Kheradpisheh et al. [2016], Kheradpisheh et al. [2017], Panda et al. [2017], Mozafari et al. [2019a], Mozafari et al. [2019b], Tavanaei et al. [2016] and Tavanaei and Maida [2017]

Architecture	Method	Rec. Rate
Querlioz et al. [2011]	Shallow FC network	93.5%
Neftci et al. [2014]	Shallow FC network (contrastive divergence)	91.9%
Diehl and Cook [2015]	Shallow FC network	95.0%
Kheradpisheh et al. [2017]	Deep Conv. network + SVM	98.4%
Tavanaei and Maida [2017]	Deep Conv. network + SVM	98.36%
Mozafari et al. [2019a]	Deep Conv. network + R-STDP	97.2%

Table 2.5: Comparison of different STDP-based state-of-the-art spiking architectures on MNIST.

use a simplified unsupervised STDP rule in combination with a WTA mechanism to extract hierarchical features in CNN-like architectures. This enables them to process large-scale images of natural objects (for instance handwritten digits or human faces). These networks use STDP however only for learning the features of the convolutional layers. For the final layer, a more complex supervised classifier, based on an SVM or reinforcement learning, is trained on the extracted features. While most of these approaches are only implemented as computer simulations, Yousefzadeh et al. [2017] demonstrate for instance the possibility to extract simple features with competitive STDP in an FPGA implementation of a spiking neural network.

While we have focused our analysis here on visual recognition, reservoir computing (Lukoševičius and Jaeger [2009]) using STDP has also been used successfully for speech recognition (see for instance Jin and Li [2016] and Jin et al. [2016]).

Stochastic sampling approaches

Alternative optimization procedures for spiking networks, that are to some extent bio-inspired, include so-called *sampling approaches*. The theory of these approaches is based on a stochastic interpretation of neuron firing. Examples of this approach include Petrovici et al. [2013], Neftci et al. [2014] and Probst et al. [2015].

Gradient-based approaches

Several works have explored training shallow spiking networks on-chip with gradient descent (for instance Nair et al. [2017]). As long as only one layer has to be trained, the gradient does not have to be propagated through the network. The propagation of the gradient in backpropagation implies several challenges in SNN-like hardware, which are discussed in more detail in chapter 4. While the forward processing of an SNN operates on asynchronous, binary events, the backpropagation step requires the propagation of high-precision floating point numbers. Using events also for propagation of the gradient could however be a promising way to reduce the energy foot-

print of the backpropagation algorithm. Additionally, it would allow using the same hardware infrastructure for processing of the forward pass and for backpropagation.

In this spirit, Neftci et al. [2017], Samadi et al. [2017] and Kaiser et al. [2018] have developed event-based versions of the backpropagation algorithm that operate directly on spikes. Errors are propagated directly through random feedback weights to each layer of the network. These algorithms therefore offer the possibility to train an SNN fully using spikes as the only means of communication. Additionally, these mechanisms are considered more biologically plausible, since errors do not have to be propagated in the inverse direction of synapses.

A disadvantage of these methods is that they remain limited in the precision of the optimization procedure. While the error that is propagated through random weights seems to provide an approximation of the gradient that is sufficient for convergence on simple problems, it is questionable if the gradient propagated this way is precise enough for large networks (see for instance Bartunov et al. [2018] for a review of biologically inspired backpropagation techniques in ANNs). O'Connor and Welling [2016] demonstrate that the gradient can be discretized into spikes and backpropagated through the network using bidirectional synapses. However, the feasibility is only demonstrated for a rather small, fully connected network.

2.4.3 Performance comparison

Tables 2.3 and 2.1 show the results of the state-of-the-art off-chip training methods on the MNIST dataset (LeCun et al. [1998]). It can be seen that in recent years, inference performance of SNNs has increased continuously and is basically on-par with ANNs for the MNIST benchmark when using similar network architectures (typically around 99.4%). However, the MNIST benchmark is nowadays considered extremely easy because its elements have little resemblance to natural images. This is why recent implementations have shifted their focus towards the more challenging CIFAR10 benchmark, which consists of RGB images of 10 classes of real world objects. Tables 2.4 and 2.2 show the state-of-the-art results of SNNs for the CIFAR10 benchmark. Also in this case, SNNs are able to yield performances comparable to ANNs with a similar architecture. The gap to the best performing ANNs is however larger as in the case of MNIST. This is mainly because modern architectures on CIFAR10 use layer types and mechanisms that are difficult to implement in spiking neural networks, such as max pooling and Batch Normalization (Ioffe and Szegedy [2015]), or are extremely large and therefore difficult to simulate as an SNN (the currently best result on CIFAR10 is 99.0% by Huang et al. [2019] using a network with 557 Mio. parameters). This gap becomes even larger for realistic image recognition benchmarks such as ImageNet (Russakovsky

et al. [2015]) where, despite recent improvements, SNNs still have difficulties to approach the state of the art (Sengupta et al. [2019]).

Comparing the results of STDP-based algorithms with the results obtained by gradient descent optimization shows that STDP fails to match the performance of the latter. It will be discussed in chapter 4 why this is the case. However, due to the lack of adequate algorithms that can be implemented on neuromorphic hardware, STDP is typically the algorithm of choice for spiking systems with on-chip learning. It is easy to implement and has been demonstrated to be able to learn hierarchical representations in convolutional spiking networks. Our first approach to building a deep network for online learning on-chip, which is presented in the following chapter, therefore uses STDP. In the subsequent chapter, we demonstrate how to obtain a high-performance on-chip implementation of backpropagation in SNNs.

Chapter 3

Online Learning in Deep Spiking Networks with Spike-Timing Dependent Plasticity

The work in this chapter is published in Thiele et al. [2017a], Thiele et al. [2018a] and Thiele et al. [2018b].

3.1 Introduction

3.1.1 Problem statement

The series of experiments described in this chapter treats the problem of online learning in spiking network. Our definition of an online learning scenario in this context is the following: learning should be possible in an autonomous way, in the sense that the system should be able to perform learning continuously on an event stream, while being provided a minimum of structural information about the dataset. It treats the question how a network of spiking neurons, equipped with a biologically inspired learning rule, can learn hierarchical representations in such a scenario. Due to the choice of the learning mechanism, it can be seen in the tradition of biologically motivated neuromorphic engineering. Our main concern is the possibility to implement our network on an event-based neuromorphic hardware platform, and we considering biological plausibility mainly where it could offer potential benefits to our architecture.

This kind of autonomous learning requires that the system should use a minimum of external feedback. One principal constraint therefore is that features shall be extracted without any label information. For this reason, we decide to use an unsupervised spike-timing dependent plasticity (STDP) (Bi and Poo [1998]) learning rule. STDP has several advantageous properties: it

is one of the few unsupervised algorithms for spiking neural networks that has been shown to be able to extract hierarchical features. At the same time, it is a learning algorithm that has been extensively studied for on-chip learning (see for instance Querlioz et al. [2011] and Qiao et al. [2015]). A large number of previous studies, that have already been presented in the previous chapter, have thus focused on unsupervised learning frameworks using STDP. Most notably, Kheradpisheh et al. [2017] demonstrate how a simplified unsupervised STDP rule in combination with a winner-takes-all (WTA) mechanism is able to extract hierarchical features in a CNN-like architecture. The best classification results for this type of network are provided by Kheradpisheh et al. [2017], Tavanaei and Maida [2017]. However, these architectures are not fully compatible with the event-driven online learning paradigm. For instance, Kheradpisheh et al. [2017] and Tavanaei and Maida [2017] use a supervised support vector machine classifier on the extracted features to evaluate final classification performance. In a more neuromorphic fashion, a supervised spike-based classifier, based on reinforcement learning, was tested in the same framework by Mozafari et al. [2019a].

However, learning of hierarchical features in these previous works has mostly been investigated in the database centered framework of standard deep learning systems. Most of the aforementioned unsupervised architectures are trained in a greedy-layer wise fashion, where each layer is optimized for unsupervised feature extraction on the full dataset, before the next layer is trained in the same way. This only partially exploits the properties of neuromorphic systems that could be interesting for learning online from continuous sensor data in real-world settings. For instance, it takes away the possibility to perform approximate inference already during the learning process. It also limits the possibility to use the output of higher layers to influence the features learned in the layers below, making it for instance difficult to combine the network with mechanisms that involve feedback from higher layers to lower ones. Almost all existing STDP-based deep networks are in this sense still based on the classical machine learning paradigm, where classification performance is optimized over multiple iterations of a possibly fully labeled dataset, with well separated training examples of fixed presentation time.

3.1.2 Our approach

In contrast to these previous works, we focus on an event-based online learning setting, i.e. a scenario where the network receives a continuous and asynchronous stream of unlabeled event-based data, from which it extracts features and performs approximate online classification. This requires a system that is able to learn features from a constantly changing scene with objects appearing at different timescales. By introducing a mechanism to decouple WTA dynamics from spike propagation, all layers of our network can be

trained simultaneously. This enables us to perform approximate inference during the learning process. Furthermore, we introduce a STDP learning mechanism which removes any notion of absolute time from our network, such that all spike times are measured relative to the dynamics of postsynaptic spikes. This makes our network fully event-based and timescale invariant. We show that it is possible to perform feature learning without providing any information about the structure of the input data (such as the number of classes) and treating the training set as a continuous stream of event-based input. This also includes implicit information, such as the duration of image presentation. We demonstrate the high convergence speed and robustness of learning with respect to some of the typical problems that might occur in an online learning setting. Despite these constraints, our network yields a test accuracy of $(96.58 \pm 0.16)\%$ on the spike-converted MNIST benchmark after a single presentation of the training set, using only a single neuron spike count classifier. This demonstrates the high specificity of neurons in the top layer and is the highest reported score so far for a network where all connections up to the final features are trained solely with unsupervised STDP.

Additionally, we show that the same architecture, with minor modifications, can perform feature extraction directly from an AER data stream. This is demonstrated using the event-based N-MNIST dataset by Orchard et al. [2015]. Also in this context of dynamic data, the architecture remains solely accumulation-based and is able to learn without having to be adapted to the input timescale. We show how the extracted features preserve dynamic information up to the highest hierarchical level, where the full object prototypes can be used for online inference or movement analysis. For this purpose, we introduce a simple event-based supervised classifier for the top level features, which reaches a maximal test set performance of 95.77%. This demonstrates that the extracted dynamic features allow effective discrimination of the examples in the dataset.

All mechanisms used in the network are constructed to be simple and generic, and should therefore be compatible with a large number of neuromorphic learning platforms. Together with the parallel work of Iyer et al. [2018], our network is the first neuromorphic architecture able to extract hierarchical features directly from a stream of AER data in an unsupervised online-learning setting. Additionally, it is the first neuromorphic system which can be used to learn complex hierarchical features unsupervised from a continuous stream of AER data, operating outside of a database framework and on multiple timescales.

3.2 Methodology

The notation in this chapter is the same as used in chapter 2. To simplify notation, we omit the layer index l in the following description.

3.2.1 Online learning constraints

A main objective of our approach is to make the system completely event-driven and remove any notion of absolute time. In particular, this means that the network should not be given any explicit or implicit knowledge about the timescale of the input data. This includes for example the long term firing rates of individual pixels as well as the information when a training example is exchanged for the next one. This has several consequences for the setup of our network:

- No leakage currents: we use a simple integrate-and-fire neuron
- No refractory periods: the firing of a neuron is solely driven by its integration, spike and reset dynamics
- No inhibitory refractory periods: WTA inhibition directly reduces the integration variable of other neurons to prevent them from firing
- No reset of neuron values when a new training example is presented, since this provides the network with implicit information about the training set
- No restriction of neuron firing that relates to the presentation of a training examples, for instance the restriction of neuron firing to once per training example
- No homeostatic mechanisms that use implicit knowledge of the training set statistics, for example constraining the neurons in the last layer to have equal firing rates
- No additional preprocessing of stimuli (for instance input normalization). The network has to be able to deal with strongly varying numbers of spikes for each stimulus.

These changes allow our network to be only driven by the timescale of its input, which can in principle even change during the learning process, without affecting the network dynamics. The only requirement for the network to be able to learn from input spikes is that sufficient spikes are produced in the network to trigger the STDP mechanism. Our network therefore fully embraces the paradigm of asynchronous event-based processing without depending on a clock-based mechanism.

3.2.2 Neuron model and STDP learning rule

All layers of the architecture use a simple version of the non-leaky integrate-and-fire neuron. The integration variable V_i is updated every time a presynaptic spike arrives at time t at neuron i via synapse j, which leads to an increase of the integration variable by the weight value w_{ij}:

$$V_i \leftarrow V_i + w_{ij}, \quad t_{ij}^{\text{pre}} \leftarrow t. \tag{3.1}$$

Subsequently, t_{ij}^{pre} is set to t, such that it always reflects the time of last spike arrival at synapse j. This update rule is fully event-based and only driven by the time of presynaptic events. If an event leads to the integration variable surpassing the threshold Θ of the neuron at time t_i^{post}, the neuron triggers a postsynaptic spike.

As weight update rule, we use a variant of the STDP learning rule introduced in Querlioz et al. [2011]. Every time a neuron triggers a postsynaptic spike at time t_i^{post}, its weights are updated as follows:

$$\Delta w_{ij} = \begin{cases} \alpha_+ \cdot \exp\left(-\beta_+ \cdot \frac{w_{ij}-w_{\min}}{w_{\max}-w_{\min}}\right) & \text{if } t_i^{\text{ref}} < t_{ij}^{\text{pre}} < t_i^{\text{post}} \\ \alpha_- \cdot \exp\left(-\beta_- \cdot \frac{w_{\max}-w_{ij}}{w_{\max}-w_{\min}}\right) & \text{otherwise} \end{cases}, \tag{3.2}$$

with learning rates $\alpha_+ > 0$ and $\alpha_- < 0$ and damping factors $\beta_-, \beta_+ \geq 0$. Synapses that received a spike at time t_{ij}^{pre} since the reference time t_i^{ref} are potentiated, and depressed otherwise. Afterwards, the integration variable is reset to 0 and the reference time t_i^{ref} of the neuron is set to the time of the postsynaptic spike:

$$V_i \leftarrow 0, \quad t_i^{\text{ref}} \leftarrow t_i^{\text{post}}. \tag{3.3}$$

Since the integration variable of the postsynaptic neuron is reset after each spike, this ensures that only neurons that directly contributed to the current postsynaptic spike are potentiated.

Our experiments show that this learning rule works best if we use a rather strong damping of $\beta_+ = 3$ for the LTP (long-term potentiation) term and no damping ($\beta_- = 0$) for LTD (long-term depression):

$$\Delta w_{ij} = \begin{cases} \alpha_+ \cdot \exp\left(-\beta_+ \cdot w_{ij}\right) & \text{if } t_i^{\text{ref}} < t_{ij}^{\text{pre}} < t_i^{\text{post}} \\ \alpha_- & \text{otherwise} \end{cases}. \tag{3.4}$$

Weights are constrained to be in the range $[0, 1]$. In contrast to Querlioz et al. [2011], we do not use an explicit STDP time window to decide between LTP and LTD. A similar learning mechanism was used in Kheradpisheh et al. [2017], however with a different weight dependence. The main reason for

adding the exponential weight dependence is the tendency of STDP to converge too quickly to the minimal and maximal weight values. As argued in Querlioz et al. [2011], the computationally expensive exponential function in our learning rule could be implicitly implemented by the device physics of a memristive synapse. Besides these practical considerations, our experiments show that the architecture does not depend too strongly on the exact details of the STDP rule and also works with minor performance losses with a simpler version, which does not include the exponential weight dependency.

The rule (3.4) is qualitatively very similar to the optimal STDP rule for stochastic neurons introduced by Nessler et al. [2013] and explored in more detail by Habenschuss et al. [2012] and Habenschuss et al. [2013]. This similarity has been pointed out already by Querlioz et al. [2015] in their analysis of rule (3.2), in the context of a memristive hardware implementation. Although the optimality condition does not strictly apply here due to the non-stochastic firing dynamics of the spiking neurons, it is interesting that this very similar rule seems to empirically yield the best results in our network. Tavanaei et al. [2016] demonstrate for a similar learning rule a probabilistic interpretation of the weights, which converge to the log odds of the firing probability of the neuron. We observed that only the ratio of α_+ and α_- seems to influence the quality of the learned features (with the absolute values still guiding the speed of learning). A similar behavior has been observed in Kheradpisheh et al. [2017] and Querlioz et al. [2015], which indicates that this could be a general property of this class of simple postsynaptic STDP learning rules. An interesting observation is that the same learning rule with the same ratios can be applied to all convolutional layers, as well as the fully connected layer at the top of the network. Only the magnitudes of the ratios have to be adapted to account for the different learning speed of each layer, which is a consequence of the different number of spikes that is triggered in each layer. Since our rule is driven by postsynaptic spikes, the number of spikes caused in a layer will strongly influence the learning speed. We observed that there is a strong connection between the ratio of the rates and the threshold values of the neurons. Since our learning rule only distinguishes between synapses that have spiked after t^{ref} and those which have not, a high threshold value will allow more synapses to contribute before the postsynaptic spike is triggered. To obtain a feature that is sensitive to spikes from those synapses that actually spike more often, we will have to use a learning rate with rather strong depression, since there will be statistically only a few situations where a synapse did not spike at all and is therefore subject to LTD. The inverse is true for a low threshold value, where only a few synapses will contribute on average to a postsynaptic spike and depression is therefore the more common scenario for a synapse, which is why it should be rather weak. For our choice of parameters, we found that a ratio of $\alpha_+ = -8\alpha_-$ works best in practice. This leads to a learning rule with high LTP for small weight

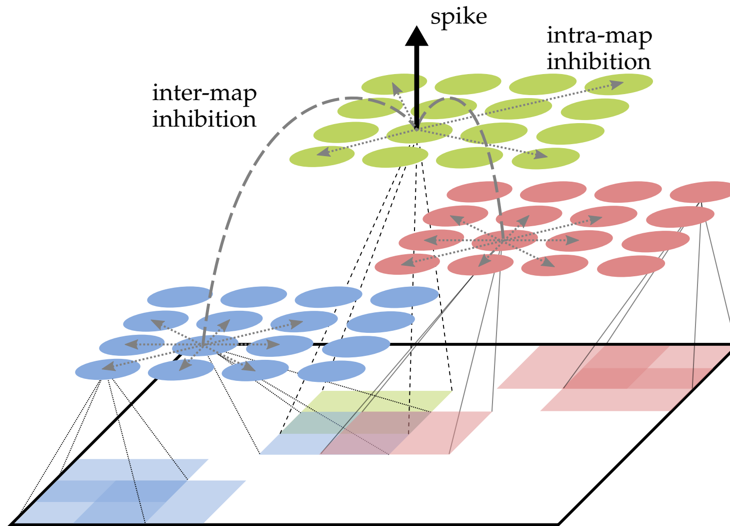

Figure 3.1: Basic structure of a convolutional layer with inter-map and intra-map WTA. Each feature map (shown in different colors) detects features in an overlapping manner over the full image. If a neuron spikes in a feature map, it inhibits neurons in all feature maps around a small neighborhood of its position, and all other neurons in the same feature map.

values that is strongly damped if the value increases. Already for a weight value above $w = -1/\beta_+ \log(-\alpha_-/\alpha_+) \approx 0.7$, depression becomes on average stronger than potentiation, which will effectively prevent weights from saturating. This behavior is important for the online learning capabilities of our architecture, since a saturation of weights could prevent the system from learning on additional examples. At the same time, the learning rule stays highly sensitive even if the weight is close to zero, which allows the system to adapt to changing input statistics.

In contrast to Querlioz et al. [2011] and Diehl and Cook [2015], our network does not use a homeostatic mechanism to adjust the firing rates of the neurons. Although such a mechanism was shown to greatly improve performance on this classification task for one-layered networks, we decided against such a mechanism since it makes implicit assumptions about the statistics of the training set. For instance, enforcing equal firing rates for neurons over long timescales, which is a typical objective for homeostatic mechanisms, imposes that the features represented by each neuron occur approximately equally often. This is in general not true for an online learning task and in particular the intermediate level features learned by the convolutional layers do not have equal probability of occurrence.

3.2.3 Network topology and inhibition mechanism

The setup of our network is similar to other competitive convolutional architectures trained with STDP (such as Kheradpisheh et al. [2017], Panda et al. [2017], Tavanaei and Maida [2017] and Mozafari et al. [2019a]). We use two convolutional layers with a varying number of feature maps, depending on the experiment (for a general overview over ANN topologies, please refer to chapter 2). If not stated otherwise, all simulations use by default 16 feature maps in the first and 32 maps in the second convolutional layer, as well as 1000 neurons in the fully connected top layer (see figure 3.3). Both convolutional layers use a filter size of 5×5 with a stride of 1. The neuron firing threshold values are 8 (first convolutional layer), 30 (second convolutional layer) and 30 (fully connected layer) respectively. The first type of inhibition is a intra-map WTA mechanism, that inhibits all other neurons in a feature map at different positions as soon as one neuron in the map releases a spike (see figure 3.1). Inhibition is performed by resetting the integration variables and reference times t^{ref} of the inhibited neurons (such that spike times are always measured with respect to the last reset of the integration variable). This mechanism prevents a single map from dominating the learning competition at all position of the input volume by learning a feature that is too general. The second inhibitory mechanism acts in an identical way between feature maps (inter-map) and inhibits neurons in a small neighborhood of the position of the spiking neuron in all feature maps. This neighborhood will typically be chosen such that all neurons whose filters strongly overlap with the filter of the firing neuron will be inhibited (in our case the two next neighbors). This competitive mechanism is essential to diversify the features learned by different feature maps.

After each convolutional layer, the network performs a pooling operation over non-overlapping windows of size 2×2 in each feature map to reduce the dimensionality of the input. In contrast to the architecture introduced in Kheradpisheh et al. [2017], where a form of max-pooling is performed that only propagates the first spike in a pooling window, our pooling layer propagates all spikes that are emitted in the pooling window. This is necessary if the network should be defined independent of the input timescale, since else we would have to define a point at which the pooling neuron is unlocked again (which is usually done when a new example is presented). Additionally, this allows propagation of a more flexible number of spikes to the following layers. In our implementation, the pooling neurons are not actual neurons since they simply propagate all spikes from a certain input region, but in principle the pooling neurons could be replaced by a more complex neuron model which has a more specific selectivity or a threshold value. The basic module of convolutional layer followed by pooling layer can in principle be arbitrarily copied to form a multi-layer deep network.

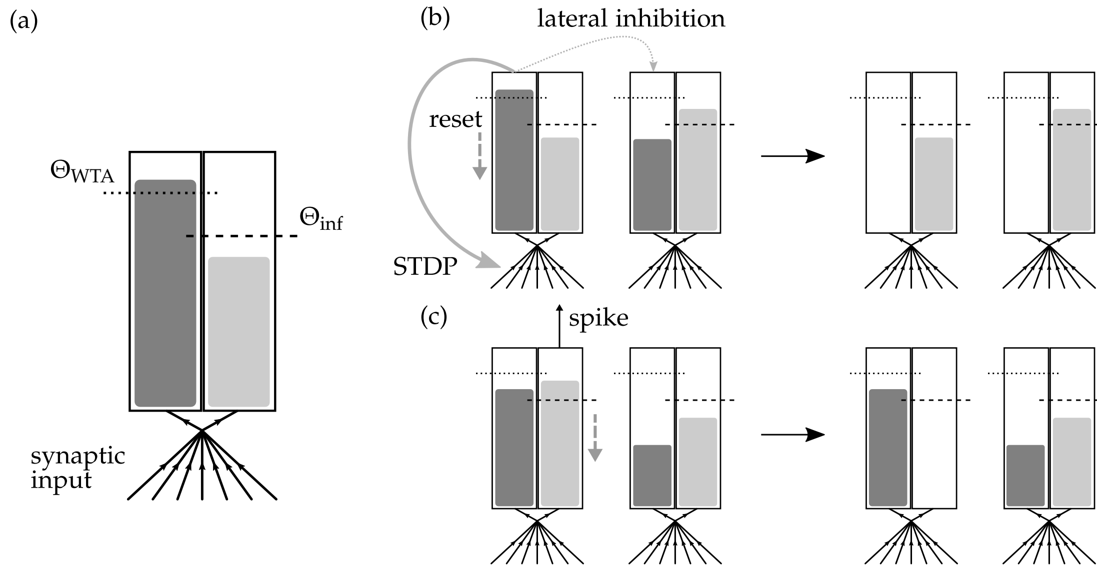

Figure 3.2: (a) Schematic description of the dual-accumulator neuron. The neuron integrates all inputs it receives simultaneously in two accumulators, one for STDP triggering and WTA, and the second one for inference (i.e. spike propagation). (b) Interaction between two neurons coupled via lateral inhibition. If the integration in the WTA accumulator reaches the threshold value, no actual spike is propagated to the next layer. However, the neuron will trigger STDP and lateral inhibition, which resets the WTA accumulator of the other neuron. (c) If integration reaches the threshold value in the inference accumulator, a spike will be propagated and integration will be reset. The inference accumulators of other neurons remain unaffected or, as in this work, can be subject to a small range inhibition (smaller than the inhibiton in the learning accumulator), to control how many maps at this position contribute to spike propagation.

Similar to frame-based convolutional neural networks, the convolutional layers are followed by a fully connected layer, which is trained with the same STDP mechanism as the convolutional layers. It merges the features from all positions and features maps to learn global, position independent representations of the classes present in the input data. This distinguishes our architecture from other multi-layer competitive CNN architectures and makes the last layer conceptually similar to the single layer networks of Diehl and Cook [2015] and Querlioz et al. [2011]. This type of representation enables us to obtain spikes in the last layer which are direct indicators of the class the network detects.

3.2.4 Dual accumulator neuron

To make our architecture suitable for online learning, another paradigm has to be reconsidered that was present in previous work. A major problem when

training all layers simultaneously comes from the WTA mechanism. Although inhibition is necessary to diversify the learned features in each layer, it significantly reduces the number of spikes that is emitted by the neurons and prevents spikes from different maps at the same input position. This limits the amount of information that is received by the higher layers and also prevents the propagation of spike codes that are a combination of several features maps. In the layer-wise training approach, this problem is resolved by disabling inhibition after the layer has been trained and then training the next layer on the output of all lower layers.

In our work, we take a different approach, which enables us to maintain lateral inhibition and still obtain sufficient spikes to train higher layers. We introduce a second integration accumulator in the neuron, which receives the same input, but is not affected by lateral inhibition and whose "spikes" do not trigger STDP updates (see figure 3.2). This corresponds to a separation of the competitive learning dynamics from the inference dynamics. Since the inference accumulator receives the same feedforward input as the learning accumulator, the spiking of the inference accumulator is still a good representation of how well the input matches the receptive field of the neuron. Both accumulators can in principle have different threshold values. By tuning the threshold of the inference accumulator, the firing rate of the neurons can be adjusted without affecting the learning mechanism. More spikes will generally lead to faster convergence in higher layers and higher inference precision, but also require more computational resources.

Our experiments show that it can still be beneficial for learning in higher layers to enable some inter-map inhibition between inference accumulators. It also helps to regulate the total number of spikes in the layer. This mostly depends on the need to have several features contributing to the emitted spike code at a particular position or only the most salient feature. It therefore enables us to smoothly switch between a one-hot feature representation, where only one feature can be active at a given position, and a more continuous representation, where multiple features can contribute partially. The dual accumulator neuron allows us to treat the competitive aspects of learning and coding independently.

3.3 Experiments

3.3.1 Learning on a converted dataset

The first series of experiments is performed on a classic machine learning benchmark, the MNIST dataset (LeCun et al. [1998]). The dataset consist of a collection of 70000 grayscale images of handwritten digits of size 28×28. 60000 of these images are used for training, while the remaining 10000 represent the test set.

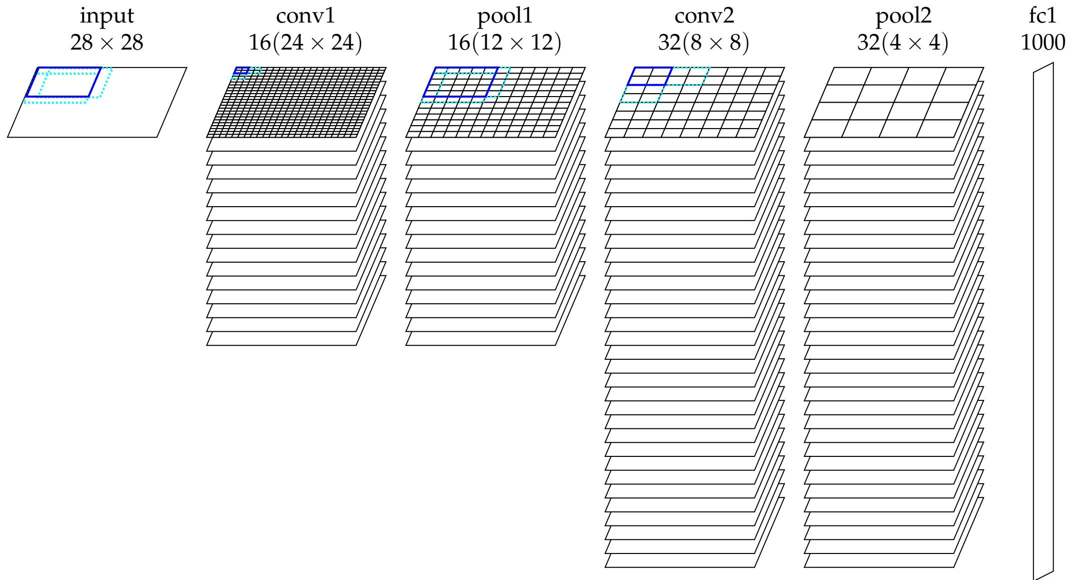

Figure 3.3: Configuration of the basic convolutional network architecture used for the experiments, showing all kernel sizes, strides and number of feature maps for each layer.

Experimental setup

Training The network is trained on a *single* randomly ordered presentation of the full MNIST training dataset of 60000 digits (i.e. no digit was shown twice to the network). No preprocessing is performed.

Simulation All simulation results for the MNIST dataset are obtained with a modified version of the N2D2 open source deep learning library by Bichler et al. [2019], using an extension of its event-based spiking neural network simulator.

Input spike encoding To become a suitable input for a spiking neural network, the MNIST images have to be converted into spike signals. Each image is converted into noisy periodic spike trains with mean firing rates proportional to the absolute value of the pixel values, which are converted to lie in the range $[0, 1]$. Each spike train is randomized by drawing the mean firing rate from a Gaussian distribution centered around the pixel value, and additionally multiplying the constant inter-spike interval length with a random number in the range $[0, 1]$ every time an event is created. However, our experiments show that the feature learning does not depend significantly on this particular conversion of the images to firing rates, as long as the firing rates grow approximately with pixel intensity. In the standard experiment, all images are presented for a fixed time to the network. Note that the timescale

Layer	Description	Parameter	Value
Conv1	Threshold STDP	$\Theta_{\text{inh,STDP}}$	8
	Threshold propagation	$\Theta_{\text{inh,prop}}$	8
	Inter-map inhibition radius STDP (in nearest neighbors)	$r_{\text{inh,STDP}}$	2
	Inter-map inhibition radius propagation (in nearest neighbors)	$r_{\text{inh,prop}}$	0
	Ration LTP vs. LTD	α_+/α_-	-8
	Filter size	K	5
	Stride	S	1
	Initial weights with STD (normally distributed)	w_{init}	$\mathcal{N}(0.8, 0.1)$
Conv2	Threshold STDP	$\Theta_{\text{inh,STDP}}$	30
	Threshold propagation	$\Theta_{\text{inh,prop}}$	30
	Inter-map inhibition radius STDP (in nearest neighbors)	$r_{\text{inh,STDP}}$	2
	Inter-map inhibition radius propagation (in nearest neighbors)	$r_{\text{inh,prop}}$	0
	Ration LTP vs. LTD	α_+/α_-	-8
	Filter size	K	5
	Stride	S	1
	Initial weights with STD (normally distributed)	w_{init}	$\mathcal{N}(0.8, 0.1)$
Fc	Threshold STDP	$\Theta_{\text{inh,STDP}}$	30
	Threshold propagation	$\Theta_{\text{inh,prop}}$	30
	Ration LTP vs. LTD	α_+/α_-	-8
	Initial weights with STD (normally distributed)	w_{init}	$\mathcal{N}(0.67, 0.1)$

Table 3.1: Network parameters used for the simulations on MNIST. An inhibition radius of 0 indicates that only neurons at exactly the same position in other maps are inhibited

here is only necessary for the spike generator and therefore only influences the number and dynamics of the spikes emitted by each training example. The processing of the network only requires the relative timing of spikes. If not stated otherwise, the presentation time is set such that each training example emits approximately between 1400 and 3500 spikes in total, depending on the average value of all pixels in the image.

Testing procedure After training, the 60000 digits of the training set are used to label the neurons in the final layer and in a second pass, the test performance is evaluated on all 10000 test images. Since the learning mechanism is unsupervised, we still need a simple classifier to assign to each neuron in the final layer the label of its preferred class. This is simply done by presenting each training image to the network and assigning to each neuron the corresponding label if it is the neuron with the highest response for this image. The preferred label of a neuron is the label which was most often assigned to it via this process. For inference, we only check if the preferred label of the neuron which fired the most during presentation of the test sample corresponds to the label of the presented image. If this is the case, the classification is considered as correct. This mechanism can be seen as a minimal classifier that only uses the prediction of the most active neuron for classification. It is therefore a valid measure of the class specificity of the neurons in the last layer. In particular, it does not influence the learned features themselves, but only

their interpretation. This distinguishes it for example from a more complex classifier such as a Support Vector Machine, which performs a classification based on a weighted combination of neuron outputs of the last convolutional layer, with parameters trained in a supervised fashion. Our approach corresponds to the more realistic unsupervised learning scenario where generally much fewer labels than training stimuli are available and therefore the labels can only be used to assess the network performance, but cannot be used for feature learning. An analysis of how the number of labels affects the classification performance can be seen in figure 3.7. We can see that a few labels are sufficient for approximate classification, but using more labels of the training set further improves the performance. This makes it possible to train the network without providing labels but still use them to improve the classification in the final layer if they become available. We restricted ourselves here to a simple classifier to obtain a meaningful measure of the quality of the top level features. Depending on the complexity of the hardware implementation, a more complex supervised classifier could be used for the top layer to improve inference performance (see for example Stromatias et al. [2017] for a supervised classifier trained with gradient descent on spike histograms).

Results

In this section, we present the main results of our simulations and investigate several properties of the architecture that could be of interest for an unsupervised online learning application.

Feature learning and inference performance Our first experiment looks at feature learning and classification on a single run of the network over the full MNIST data set. The network configuration can be seen in figure 3.3. This is what we refer to in the rest of the chapter as the reference simulation.

A visualization of the final features can be seen in figure 3.4. An interesting property of the final weight matrices (figure 3.5) is their sparsity and basically binary weight configuration. The sparsity of the weight matrices is caused by the sparse input and sparse responses of each layer due to the WTA mechanism. The binarization is caused by the STDP learning mechanism which enforces correlations. Synapses which receive consistently input that causes postsynaptic spikes will quickly saturate to the maximal weight value. During learning, this effect is attenuated by the exponential weight damping, which prevents overly fast convergence of the weights to their maximal value. The fact that the final weights are almost binary could leave the possibility to binarize the weights fully after the learning phase without changing the representations significantly. Depending on the concrete hardware implementation, this would fully binarize the network's computations and may yield additional processing efficiency.

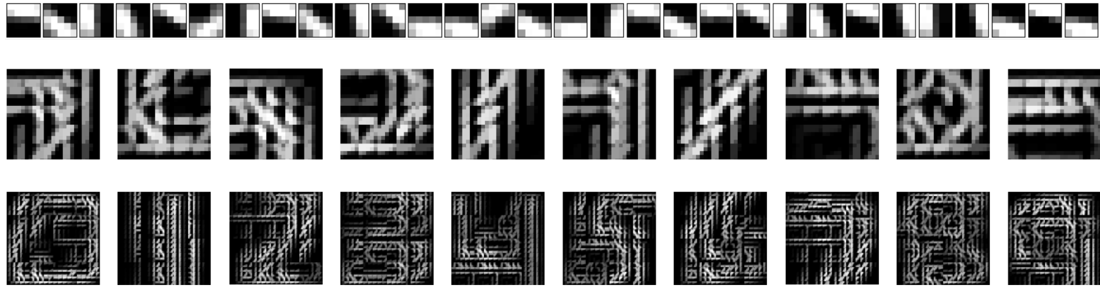

Figure 3.4: Visualization of the preferred features learned in the different layers of the network. For the first layer, the preferred feature corresponds simply to the weight kernels. We can see that this layer learns filter patches which detect local contrast differences. The higher layer features are constructed by choosing for each neuron in the feature map the feature in the lower layer to which it has the maximum average connection strength. Note that due to the overlapping nature of the weight kernel of each position, the features have a somewhat blurred appearance. As we can see for the second convolutional layer, the neurons become sensitive to parts of digits. Finally, in the fully connected layer, each neuron has learned a highly class specific version of a particular digit prototype (digits 0 to 9 from left to right).

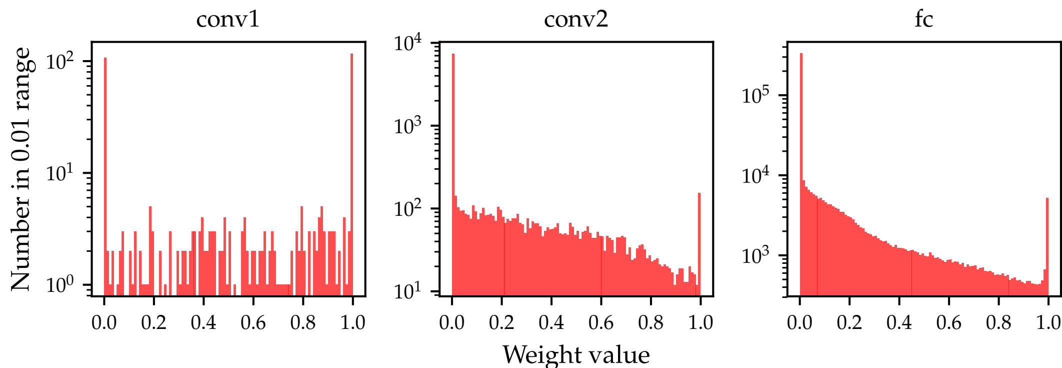

Figure 3.5: Weight distributions for the different layers of the network. A large number of weights converges to 1 or 0. In the higher layers, the weights become increasingly sparse.

Even without a homeostatic mechanism and despite the similar learning rule, the maximal performance of our network (figure 3.8) is higher than for the network of Querlioz et al. [2015] for all neuron numbers in the fully connected layer. By increasing the number of features in the convolutional layers, the network is able to yield $(96.58 \pm 0.16)\%$ accuracy on the test set in the configuration with 16 and 256 maps in the convolutional layers. It is also better than the architecture of Diehl and Cook [2015], which uses besides a homeostatic mechanism several other mechanisms to stabilize learning and improve performance (such as divisive weight normalization, tuning of the input firing

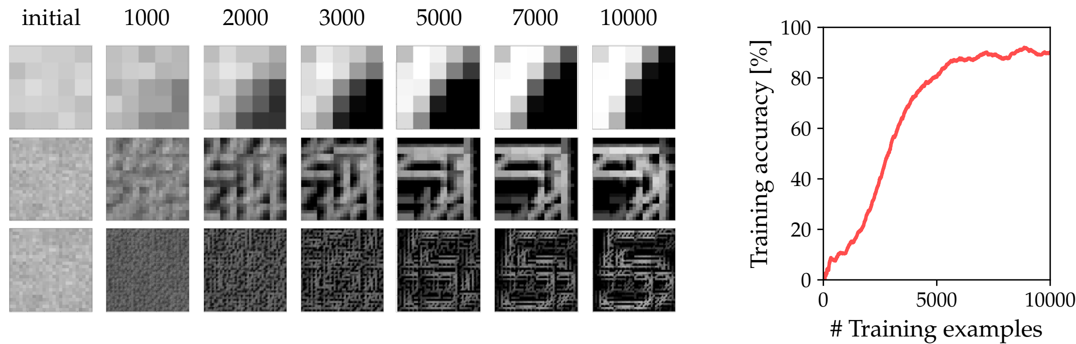

Figure 3.6: Demonstration of the simultaneous emergence of features in the different layers of the network. We can see that even if the features in the lower layers are not fully converged yet, the higher layer is able to assemble them to a more complex feature. The error plot shows the development of the running average error on the training set (averaged over a sliding window of 1000 examples). We can see that even with the fuzzy features the top layer is able to perform an approximate inference, which continuously improves with the quality of the features.

rates and reset of neuron values for each example) as well as more synapses and iterations over the training set (4.2 Mio. adjustable synapses vs. 5 Mio. and 1 presentation vs. 15 presentations of the full MNIST data set). Another main advantage of a convolutional architecture, as we have chosen it for this work, is that in contrast to these shallow networks, we can exploit translational invariance. Networks consisting only of a fully connected layer will fail to recognize a class if it is presented slightly shifted compared to the training set.

We also tested how strongly the online learning constraints affects the result. In the standard configuration, shown in figure 3.3, the performance of the network is $(95.2 \pm 0.2)\%$. If we reset the neurons after each example presentation, the performance increases slightly but insignificantly to $(95.24 \pm 0.26)\%$. The same is true if we use a layer-wise training approach, where every layer is only trained on the full training set after the lower layers have been trained, which yields $(95.27 \pm 0.23)\%$ test set accuracy. We can thus conclude that our network does not seem to be significantly affected by these constraints relating to a database framework.

Robustness to input variation and sparsity In a follow-up experiment, we tested the robustness of learning with respect to input presentation time variations. This feature is important for a real-world application, where we cannot be sure that all classes and objects will be presented to the network for the same fixed time. We therefore varied the presentation time of each digit randomly by a factor between 0.1 and 1.9, such that the total presentation time is on average equivalent to our other simulations. We observe that the final

classification performance of the network seems to be insignificantly affected by these variations, yielding an accuracy of 95.13% compared to 95.33% with the same parameters and constant stimulus duration.

Additionally, we learned the whole training set with a constantly different presentation time and evaluated the performance (while adjusting the learning rates to account for the smaller number of spike events). We can see in figure 3.7 that, as long as the presentation time stays within a certain range, the result is largely unaffected by the presentation time. This is in particular true for feature learning, which seems to be mostly unaffected by the presentation time of a single image. We noted however that the presentation time is important for the inference phase. The performance of our spike count classifier drops significantly if the number of spikes triggered in the top layer becomes only a tenth of the original number. However, if we use the standard presentation time for the labeling and testing phase, the classification accuracy is still around 93.72%. It therefore seems that the drop in inference performance is mainly due to a failure of the classifier, which requires a certain number of spikes to label the neurons and classify correctly. This indicates that the network is able to learn features even with a presentation time per image that is one order of magnitude lower than the presentation time we chose for the reference simulation. For this presentation time, there will be only approximately 100 to 500 input spikes for a single training example and the spikes triggered in the top layer are only in the order of 10. The total number of spikes triggered in the full network will be in the order of $1000 - 2000$ depending on the image (if pooling neurons, input spikes and "pseudo-spikes" by the learning accumulators are discounted). For an image size of $28 \times 28 = 784$, this means that most pixels with a value significantly higher than 0 will spike only once or twice per examples and potentially very unregularly. If we half the presentation time again to 5% of the original time, inference performance finally begins to drop strongly, although the feature learning mechanism still extracts meaningful features. For this presentation time, every image will be represented by only 40 to 100 spike events, which seems barely enough to give a meaningful representation of the 784 pixel digit. The drop in performance is therefore probably also caused by this discretization limit of the digit. Our results demonstrate that even with this noisy and sparse input, our architecture is able to extract useful features.

Scaling We also analyzed the scaling behavior of our network, which is an important property for the potential of the architecture to be extended to more complex data. In a first experiment, we investigated how the size of the fully connected top layer affects the classification performance. Similar as Querlioz et al. [2011] and Diehl and Cook [2015], we could observe that an increase in the number of neurons in this layer leads to a higher classification accuracy. However, for our network, the performance increases substantially faster than

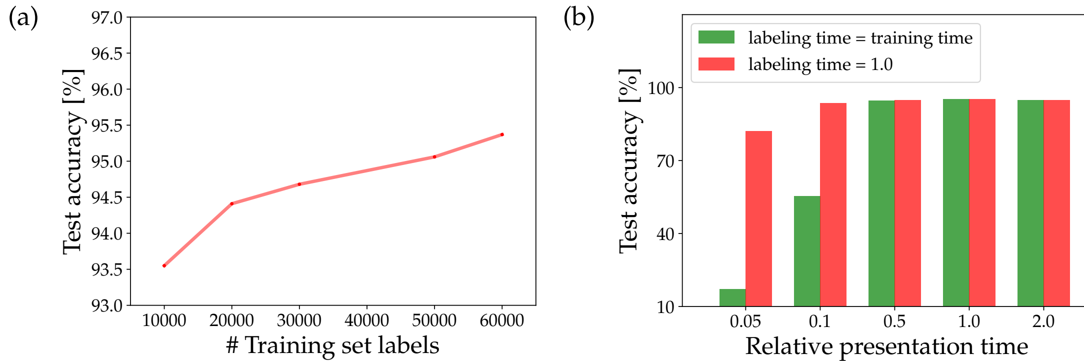

Figure 3.7: (a) Increase of test set performance as a function of the number of training set labels used to label the top layer neurons. (b) Influence of presentation time on training and labeling phase. Times are given relative to the presentation and labeling time we used for the reference simulation. *green:* test performance if presentation time during labeling and testing phase is same as during training. *red:* performance if labeling and testing time is independent of the training phase presentation time and equal to the time represented by 1.0.

for their architectures and our network yields higher or equal scores for all layer sizes (see figure 3.8). This indicates that the preprocessing done by the convolutional layers indeed helps the fully connected layer to extract more general and useful digit prototypes, as compared to the case where the layer is directly connected to the input layer. Furthermore we trained on far less iterations over the MNIST set and did not use any homeostatic mechanisms in our fully connected layer. Querlioz et al. [2011] observed in their work that homeostasis substantially improves the performance of their network by balancing the competition between neurons. The fact that our architecture performs better even without this mechanism and less iterations could indicate that the preprocessing of the convolutional layers produces a spiking output which is easier to process for the fully connected layer and increases the stability of the learning process.

In a next step, we also investigated the scaling properties of the convolutional layers by increasing the number of feature maps (see figure 3.8), while leaving all other neuron variables untouched (in particular also the thresholds of higher layers). Our results show that an increase in the number of maps in the second layer consistently leads to an increase in classification performance. This is not the case for the first convolutional layer. We suspect this could be caused by the relatively high redundancy between the maps of the first convolutional layer (see figure 3.4). Since the competitive mechanism for the inference accumulators is rather weak, many similar maps can release a spike for the same input position and trigger a spike in the layer above (whose thresholds were not changed in the scaling process). This reduces the complexity of the features that can be learned in the higher layers and could

therefore be responsible for the slight decrease in classification performance. Since the maps in the second convolutional layer are more complex, redundancy is lower and similar features dominate less the competition, which is why scaling seems to be more beneficial here.

It seems that our architecture can consistently profit from scaling, in particular in higher layers, where feature complexity is high. Note that for both the fully connected and the convolutional layers we had to increase learning rates since the training time scales approximately linearly with the number of inhibited entities (i.e. maps for the convolutional layers and neurons for the fully connected ones), which is a consequence of the winner-takes-all dynamics. This is only necessary to achieve convergence on a single presentation of the MNIST dataset and would not be a problem in an online learning setting, where unlabeled training data is abundant.

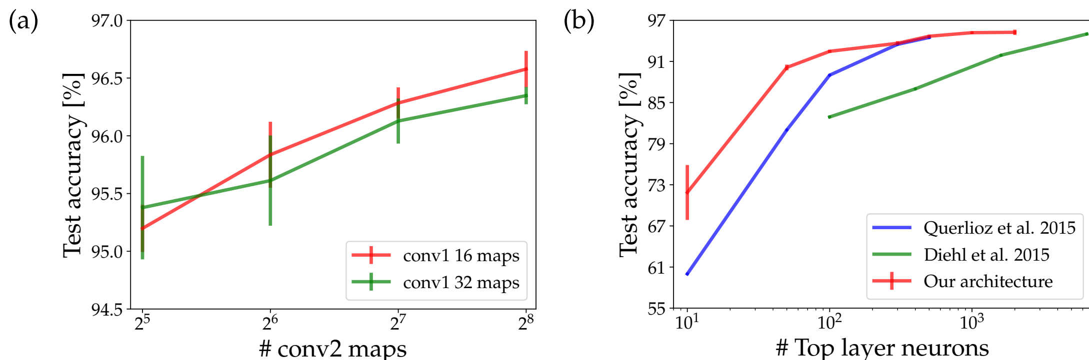

Figure 3.8: Scaling behavior of different layers of the network. (a) Scaling of the network performance with fixed number of top layer neurons and variable number of feature maps in the convolutional layers. (b) Increase in test set performance as a function of the number of neurons in the fully connected top layer (with number of feature maps fixed to 16 and 32 in the convolutional layers). The last point of Querlioz et al. [2015] is not exactly reported, but likely lies between 94% and 95%.

Robustness to learning rate variation While adjusting the learning rate for the scaling experiments, we observed that our architecture seems to be very robust against a change in the absolute values of the learning rates (while leaving the ratio between LTP and LTD constant). Figure 3.9 shows the inference performance on a sliding window of 1000 examples of the training set and the test set performance as a function of the learning rate variation. Our experiments show that the network performance is remarkably stable with respect to the absolute value of the learning rates. In a learning rate range spanning an order of magnitude, we can observe stable online training error convergence. A learning rate which is too low does not converge on a single presentation of the MNIST dataset (however if it is presented multiple

times or if there would be more images). If the learning rate is very high, the online classification performance becomes unstable after an initial steep increase. This is probably mainly due to the online labeling mechanism which is used for classification. A high learning rate will alter the learned features of the neuron continuously and therefore the labeling mechanism fails. This could explain why the test performance is only affected to a comparably small extent, since before testing, neurons are labeled while learning is disabled.

In contrast to many ANN implementations that are optimized for classification performance, we did not implement a learning rate decay policy. Such a policy would have to be defined over a fixed number of iterations over the dataset and thus would violate the online learning paradigm. Our results show that even a constantly high learning rate can lead to robust convergence. Depending on the specific application, the learning rate could be very high, allowing a fast adaption to changing input stimuli, or very low, which allows to include more information from training examples into the weights and therefore might lead to better generalization. We could also imagine a mechanism that changes the learning rate depending on the online classification performance. If the performance drops suddenly, the learning rate can be set to a higher value to enable the network to adapt to possibly unseen inputs.

Both the stability of the architecture for high learning rates and the promising scaling behavior have beneficial consequences for the parameter tuning process in a practical application. This is true in particular if the optimal architecture cannot be found easily by an optimization process over a fixed training and test set. Initially, we could set the learning rates very high and use only a small number of feature maps to check if these converge quickly to a meaningful solution. This requires only a small unlabeled dataset with approximately the same properties as the online learning data. Such a solution could be easily identified given the intuitive local and hierarchical representations of the network and the ability to assess classification performance with only a few labels necessary for the spike-count classifier. If the features seem to converge or the online inference error decreases, the learning rates can be set to a low value and the network scaled up for the high precision online learning task.

3.3.2 Extension to dynamic data

The results on the static MNIST dataset demonstrate the capability of the network to learn in an online learning scenario. However, converting the MNIST dataset into spikes is a very artificial procedure, and might not necessarily reflect the input that the network would receive from an event-based vision sensor (such as Lichtsteiner et al. [2008] or Posch and Wohlgenannt [2008]). In the following part, we show how to extend the architecture to extract hi-

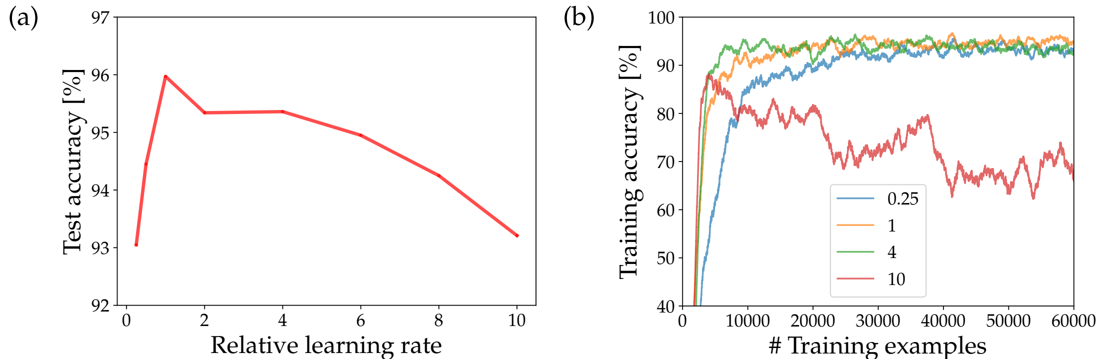

Figure 3.9: (a) Influence of the variation of the learning rate on test performance in the fully connected layer, measured relative to the learning rate that enables stable convergence over a single dataset iteration. (b) Online training performance (averaged over a sliding window of 1000 examples). Neurons are labeled online during the learning process. A learning rate that is too low will not converge over a single presentation of the MNIST dataset, while a very high learning rate leads to instabilities.

erarchical features from such a stream of event-based vision data in an unsupervised fashion and online. It is demonstrated how our network is able to learn translational invariant features, that preserve dynamic information in the data, from the event-based N-MNIST dataset of Orchard et al. [2015]. Despite the dynamic nature of the input data, the architecture remains solely accumulation-based and is able to learn without having to be adapted to the input time-scale. We show how the extracted features preserve dynamic information up to the highest hierarchical level, where the full object prototypes can be used for online inference or movement analysis based on the moving digit prototypes.

Adaptation of the topology

Since the dataset has changed to N-MNIST, we have to adapt the parameters 3.1 to the new learning task. The principal changes that are necessary are an adaptation of the thresholds to account for a different number of input spikes and adding a classification layer on top of the network. The threshold values are changed to 8 (first convolutional layer), 50 (second convolutional layer) and 60 (fully connected layer) respectively. We observe that all other parameters can be left unchanged. This is probably because the N-MNIST dataset was designed such that each moving digit has the same size (in pixel) as in the original MNIST dataset. The enlargement of the input volume from 28×28 to 34×34 is necessary because of the movement of the digit in the volume, but does not change the absolute size of the digit. Being a convolutional architecture, our network is invariant to a translation of the object and therefore

the parameters of the network do not have to be changed. In addition to the changes of parameters on existing network elements, we introduce a second fully connected layer on top of the first fully connected layer, which is trained with a supervised classifier. We demonstrate that the first layer allows us to obtain neurons that respond to digit prototypes with a specific movement direction. The second fully connected layer has a number of neurons equivalent to the number of classes in the training set and learns to merge several moving digit prototypes from the fully-connected layer below.

Event-based supervised classifier

Since the STDP feature extraction mechanism is purely unsupervised, we need to place a simple classifier on the top level features to perform inference. For training on the static MNIST dataset, a simple spike count classifier was used, since the top level neurons represent full digit prototypes. Due to the different movements direction of each digit in the N-MNIST dataset (see feature visualization in figure 3.11), we now require a classifier that is able to merge the neural responses that occur for the different saccades during an example presentation. We decided to use a simple supervised classifier that operates on spikes only and requires only the storage of an additional floating point variable for each class, which represents the current error estimate. In this way, we obtain an event-based classifier that uses an approximate supervised error estimate, while still being compatible with the local, neuromorphic processing paradigm. The learning rule with error term $E_i(t)$ is given by:

$$\Delta w_{ij}(t) = \eta \cdot E_i(t) \cdot S_j^{\text{pre}}(t), \tag{3.5}$$

where η is the learning rate. The presynaptic term $S_j^{\text{pre}}(t) = 1$ if synapse j transferred a spike at time t and $S_j^{\text{pre}}(t) = 0$ otherwise. We thus obtain the simple presynaptic event-based update rule:

$$\Delta w_{ij} = \eta \cdot E_i(t), \tag{3.6}$$

which is applied every time a spike arrives at neuron i through synapse j. E_i is updated for all neurons every time a neuron in the classification layer fires:

$$\Delta E_i = (m-1) \cdot E_i + \begin{cases} R_i - 1 & \text{if } i \text{ fired} \\ R_i & \text{otherwise} \end{cases}. \tag{3.7}$$

The variables R_i represent the targets of the network. They are initialized for each training example in a one-hot fashion, such that $R_i = 1$ if i is the class of the training example and $R_i = 0$ otherwise. This rule ensures that $E_i(t)$ moves towards zero if the neurons respond correctly. The error term will become negative if a neuron spikes that should have remained silent and

positive if a neuron that should have spiked did not. The term $m \in [0, 1]$ ensures that information from previous error updates is not lost too quickly.

This learning rule calculates an approximation of the error based on the event-driven response of the network, without the need to set an explicit target firing rate. Learning rule 3.6 will only stop weight updates if the gating term $E_i(t)$ is exactly 0. This can only happen if $E_i(t)$ is driven to zero by its update rule 3.7. For $m \neq 0$ this will require several updates, which are themselves driven by postsynaptic spikes. A low postsynaptic firing rate will therefore also slow down the error updates and therefore prevent a situation where learning stops. A large postsynaptic firing rate will lead to more updates of $E_i(t)$ and a better estimation of the current error. The precision of $E_i(t)$ therefore scales dynamically with the postsynaptic response of the network, which is directly related to the timescale of the input signal.

Like the unsupervised feature extractor, the supervised classifier is trained online while the network is receiving the stream of inputs. The difference is that the classifier is provided with external label information. Every time a new class is presented, the R_i are updated and all E_i and integration variables of the classifier neurons are reset to 0. During the testing phase, no database information is transmitted to the network, and classification is solely based on the strongest response of the neuron during the time window in which the example is presented (also no resets of classifier integration variables take place). This way, we obtain a system that can use label information to train the classifier whenever it is available. At the same time, it delivers a spiking response that can be used for approximate inference, without having to be informed explicitly that a new classification phase starts.

Experimental setup

Simulation All simulations on the event-based dataset are performed with the N2D2 open source deep learning framework by Bichler et al. [2019]. The library was extended for this purpose with a CUDA accelerated spiking neural network simulator. Training is performed on *Nvidia TitanX* graphics cards. Although the clock-based simulation requires the definition of a timescale and a corresponding time step, all mechanisms were designed to not depend on their absolute values.

Event-based input The network was trained on a *single* randomly ordered presentation of the full N-MNIST dataset by Orchard et al. [2015] of 60000 digits. The N-MNIST dataset consists of event-based recordings of the MNIST benchmark. The training examples consist of AER recordings of the ATIS (Posch and Wohlgenannt [2008]) sensor for digits moving on a screen. Each example digit is presented in 3 saccades of 100 milliseconds. During each of

these saccades, the digit moves slightly in a fixed direction and causes the ATIS sensor to emit on and off events.

Testing procedure After training on the 60000 training examples of the N-MNIST dataset, the classification performance of the network is evaluated on the 10000 test examples. As before, all simulations were performed on a *single* iteration over the dataset in order to simulate an online learning scenario where no example is encountered twice.

Results

Feature extraction Our first experiment tries to optimize the network for maximum performance on a single run of the full MNIST data set. The network configuration can be seen in figure 3.10 and a visualization of the final features can be seen in figure 3.11. We can see that the first fully connected layer of the network extracts digit prototypes with their corresponding typical pattern of on and off events. Due to the convolutional structure of the network, these prototypes can be recognized in a translational invariant fashion, while maintaining the information of their movement direction.

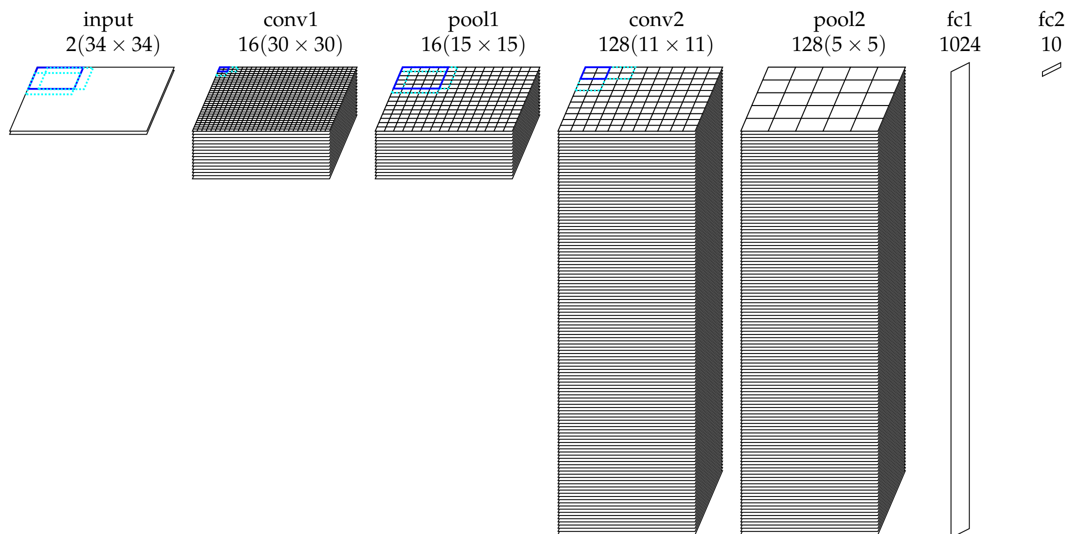

Figure 3.10: Configuration of the convolutional network architecture used for the experiments, showing all kernel sizes, strides and number of feature maps for each layer (in short notation: 34x34x2-16C5-P2-128C5-P2-1024-10).

Full example and single saccade inference One advantage of the type of features learned by the network is that we can train different classifiers on the top level, depending on the application in mind. For example, we could

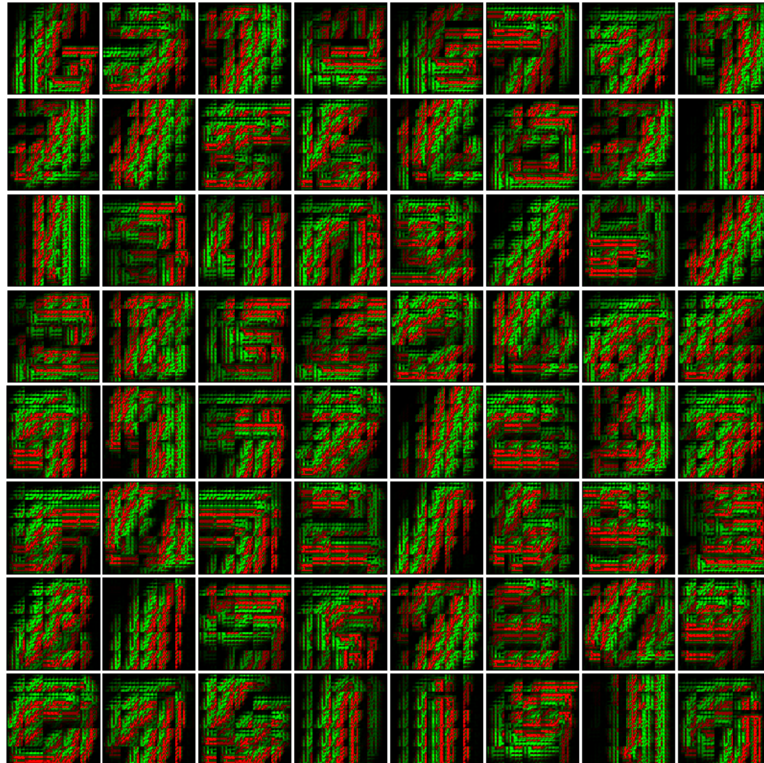

Figure 3.11: Visualization of the preferred features learned in the first fully connected layer of the network. Preferred features are constructed by recursively patching the lower level feature with maximal weight into the corresponding positions of a filter kernel. The kernel overlaps give these features a blurry appearance, which reflects their translational invariance.

imagine a classifier which only recognizes a digit if it passes the camera in a certain movement direction. That means it responds if the corresponding prototype activates. Alternatively, we can train a classifier to merge several prototypes and their corresponding movement directions. This way the typical movements of an object can be used to improve classification.

The most straightforward way to use the classifier is to test it in the same way as it was trained, i.e. on all three saccades of a test example. We present each test example and count the number of spikes that is emitted by each classifier neuron. If the neuron with the highest number of spikes corresponds to the presented class (equality is decided via the maximal integration value), the classification is considered as correct. With the configuration shown in figure 3.10, we reach a maximal classification accuracy of 95.77%. Table 3.2 compares several results on the N-MNIST benchmark and their corresponding architectures. It can be seen that our architecture is one of the few approaches that is able to perform STDP-based unsupervised feature extraction on the N-MNIST dataset.

Architecture	Training algorithm	Performance
Shallow SNN Orchard et al. [2015]	SKIM	83.44%
Shallow SNN Cohen et al. [2016]	SKIM	92.87%
Converted CNN Neil and Liu [2016]	ANN converted to SNN	95.72%
Spiking CNN Lee et al. [2016]	BP	98.74%
Spiking CNN Yin et al. [2017]	BP	96.33%
Spiking CNN Wu et al. [2018a]	BP	98.78%
Spiking CNN Jin et al. [2018]	BP	98.88%
Event-histograms Sironi et al. [2018]	HATS	99.10%
Spiking CNN Shrestha and Orchard [2018]	BP	99.20%
Frame ANN Iyer et al. [2018]	BP on acc. events	99.23%
Spiking CNN Iyer et al. [2018]	STDP	91.78%
Spiking CNN (**this work**)	STDP/event-based classifier	95.77%

Table 3.2: Comparison of different architectures used for classification on the N-MNIST benchmark.

We now investigate the possibility to use the network also on single saccades after it was trained on the N-MNIST dataset. In the dataset, each training example contains three saccades, which represent different movement directions of the digit. When training the network on a stream of real-world data, we can however not be sure that each object will always be represented by exactly three saccades and their particular movement directions. We therefore want to test if the trained network is also able to classify a digit if it is only confronted with single saccades. For this purpose, we split each training example in its three saccades and test the response of the trained network under different conditions (see figure 3.12). In a first experiment, we train features and the classifier on all 3 saccades and only performed the inference on a single one. Our results show that the network is still able to correctly classify most digits based on only one of the saccades. We however observe a drop in precision. This is to be expected, since three saccades provide the network with more information about the digit. In the second experiment, we use the same saccade 3 times for each inference. As expected, this increases the classification accuracy, since a higher number of spikes allows more accurate inference. The performance is however still lower than inference with three different saccades. In the last experiment we train the classifier on three times the same saccade and therefore optimize it for recognition of this particular saccade. As we can see, for saccade 1 and 2, this improves the accuracy, while for saccade 3, the result is slightly worse. For all three saccades, the change is however insignificant in the range of statistical error. It seems therefore that the better results when using all three saccades is indeed due to the higher information content of 3 saccades. For saccade 3, the classification results are generally worse than for saccade 1 and 2. This is possibly caused by the particular horizontal movement direction of saccade 3 (see figure 3.13),

which contains less information than the diagonal movements of saccade 1 and 2.

This demonstrates that our network can use a flexible number of saccades for classification, independent of the number of saccades that were used for training. It also shows that, independently of feature extraction, the classifier can be optimized for certain inference objectives.

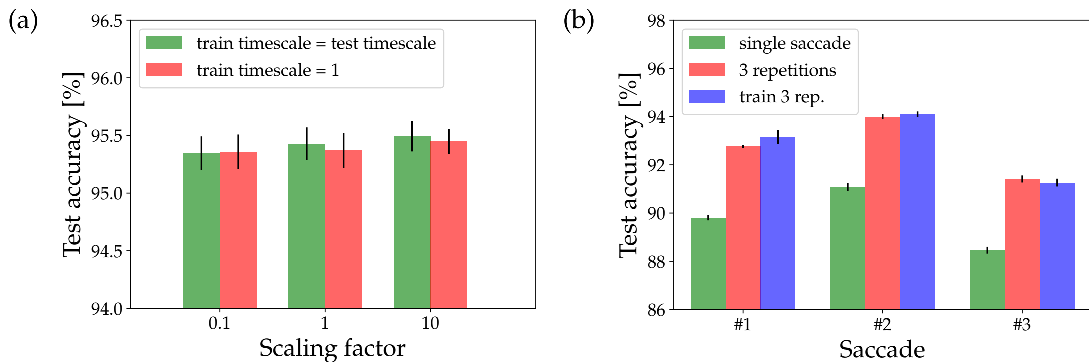

Figure 3.12: (a) Demonstration of the timescale invariance of the network. Each example was scaled such that it fits exactly into a time window which is a multiple of the standard time window we use for the other simulations. *green:* Test accuracy if training and testing are performed on the same timescale. *red:* Test accuracy if training is performed on the default timescale 1 and only the testing timescale is changed. (b) Single saccade classification performance. The horizontal axis labels the single saccade which was used. *green:* Performance on the test set if only one saccade is used for inference. *red:* Performance of the network if one saccade is used 3 times for inference. *blue:* Performance of the network if training of the classifier is also performed on 3 times the same single saccade.

Timescale invariance Due to the accumulation-based nature of the network, it should be able to recognize a digit independently of the timescale of its spike representation. To test this, we train the network for different timescales and then perform the testing in the training timescale as well as in a several different timescales. Figure 3.12 shows that the results are basically the same in the overall range of fluctuations caused by slightly different initial values of the simulation. We also test if the network and classifier trained on a particular timescale are able to classify correctly if the patterns are presented in a different timescale than the one which was used for training. Our results show that this is indeed the case, in the range of typical fluctuations caused by different initial conditions.

Stimulus response patterns Figure 3.13 shows the typical response of the trained network for the presentation of a single training example. We can

easily identify the saccadic movements in the spiking input to the network. Up to the highest layer, the whole network responds rapidly to the input while remaining quiet if no input is presented to the network. Note that the network has no explicit knowledge when a saccade or image begins or ends, i.e. no resets of the integration variables are performed between these bursts of input activity. Even without any leakage currents or explicit resets, the network closely follows the dynamics of the input activity and quickly returns to a silent mode if the input activity stops. Additionally, activity becomes increasingly sparse in higher layers, which reflects the specificity of the high level features.

3.4 Discussion

3.4.1 Analysis of the architecture characteristics

Fully event-based multi-layer learning

The multi-layer training capabilities of our architecture make it accessible to learning mechanisms that involve multi-layer top-down feedback. In particular, the online predictions of our network could be used for a reinforcement learning scheme, which could modulate STDP learning with a reward signal which is propagated through the network. Additionally, multi-layer training is more compatible with an online learning paradigm, where it is not possible to receive a stimulus a second time. This could be problematic for a layer-wise training mechanism, since higher layers would be trained on different inputs than the layers below. Finally, in contrast to other deep architectures that perform feature extraction in the final layer with a more complex classifier to improve performance, the spiking output of the fully connected layers in our network is highly specific and presents full digit prototypes. It could therefore be used directly for a higher level spike-based processing stage (see for instance Eliasmith et al. [2012] and Diehl and Cook [2016] for functional spiking models of higher cortical processing).

The fact that little changes are necessary to adapt the architecture from static to dynamic data could make it possible to gauge most parameters of the network on a spike-converted dataset with objects similar to those that have to be learned in the online learning task. Such a preliminary simulation could reduce the risk of choosing a neuromorphic implementation that is unable to perform the online learning task.

Timescale invariance of the learning rule

Intuitively, we can understand the STDP mechanism the following way: the threshold of the neurons of a layer describes a type of implicit timescale,

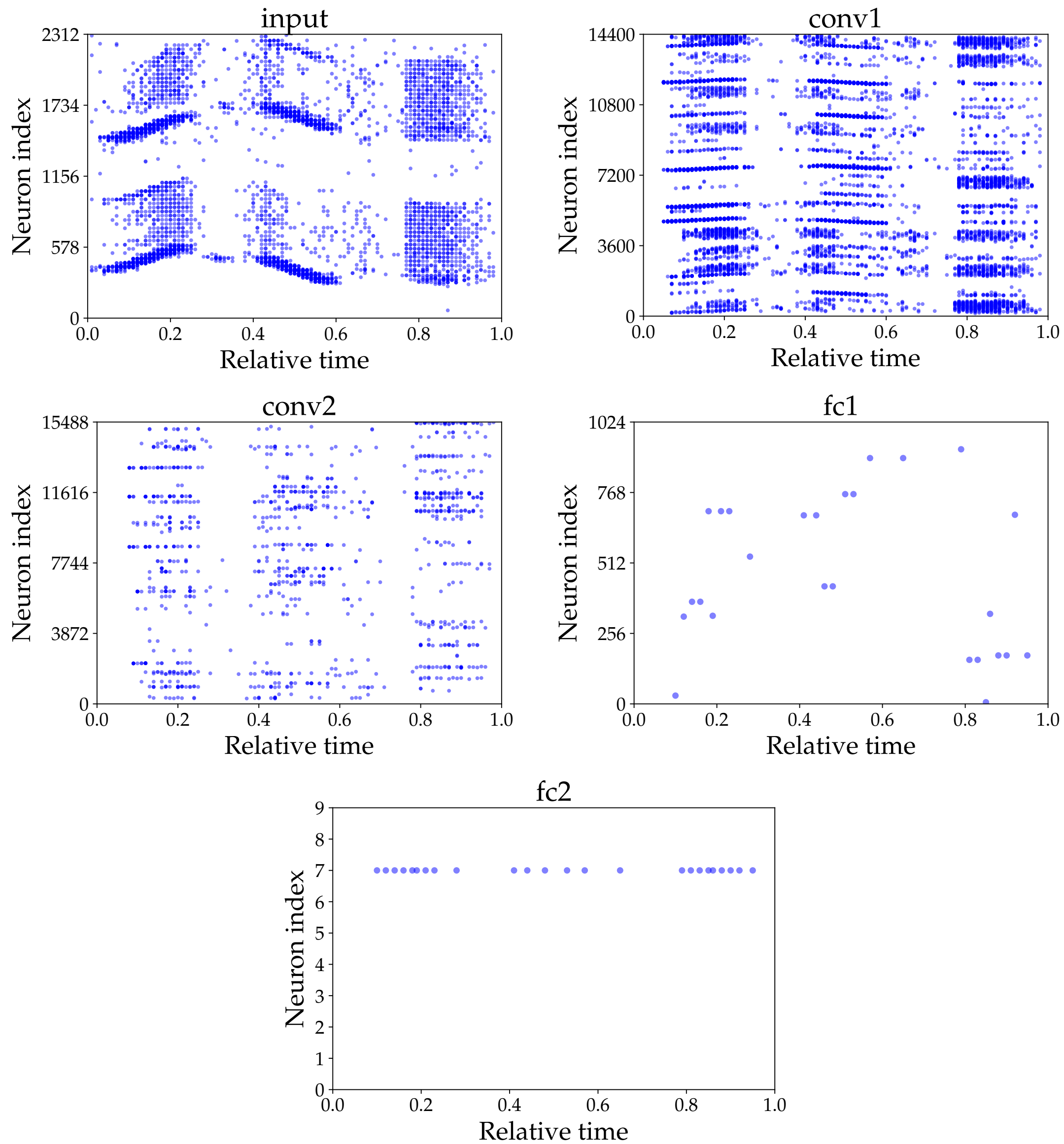

Figure 3.13: Typical firing patterns in convolutional and fully connected layers during the presentation of a single N-MNIST example. Each image is presented as 3 saccades separated by silent phases. In all layers up to the top fully connected layer, the saccades are clearly visible as bursts of spiking activity.

which is the time until one of the neurons has integrated enough information to emit a spike. All neurons that are connected to this neuron by inhibitory connections are subject to a reset of their integration variable and their relative timescale. As long as the long term input dynamics change much slower than this implicit timescale, there will be a consistent causal relationship between a certain synaptic input and the response of a neuron. Since the time until the threshold is reached depends only on the total integrated input signal, this

implicit timescale is directly set by the timescale of the input. If spikes arrive very rapidly, the threshold will be reached in a short time. If spikes arrive rather slowly, also the absolute time until the threshold is reached becomes longer. In a model that uses a leakage current or refractory times, these times would have to be adjusted to the timescale of the input signal. Since our STDP rule only depends on the relative timing of postsynaptic spikes, and therefore on this implicit timescale, we are able to use the causality reinforcement properties of STDP without defining an explicit reference time.

One-hot representations for event-based image recognition

The type of features learned by a WTA-based algorithm are so-called one-hot representations. This means that for a given input, only one or a few of the features will be activated, yielding a sparse code. On the other hand, representations learned by the backpropagation algorithm are typically distributed, which means that for a given input, a large number of neurons will be activated and represent the input through their combined activity. In the computational framework in which ANNs are typically used and with the ability to use high precision floating point numbers, distributed representations are usually superior since they use less neurons. Additionally, GPUs can process quickly the dense matrix multiplications that arise in this type of feature encoding. In a spiking network however, a high precision output of a neuron can only be obtained if we use a large number of spikes (at least as long as we use a rate code). In this case, the highly representative and sparse features of a one-hot representation enable us to encode complex information with much fewer spikes and only a few active neurons, while the other neurons remain completely silent (Kheradpisheh et al. [2017] showed that at most one spike per neuron per image can be sufficient). The price we have to pay for this sparsity is a kind of inefficient representation, which requires a large number of neurons (i.e. feature maps). We still think that it is a suitable way to encode sparse representations in event-based systems. In such a system, neurons are only activated by external input and a large number of neurons does not necessarily produce a higher total activity, since most of the neurons will remain inactive. We can thus profit from sparse activity and its computational benefits even if the number of features (and therefore neurons) becomes very large.

The features extracted from N-MNIST resemble snapshots taken from the moving objects. We believe that such an approach is well suited for image recognition if the observed object does mainly change its position, but not the relative position of the pixel values. The competitive mechanism used during learning extracts features that reflect the possible movement directions of the presented object. The same would be true for different object sizes or viewpoints. This means that the number of neurons that are necessary to represent

a class is approximately proportional to the degrees of freedom in the movement of the observed object. This could lead to a strong increase in the number of maps that are necessary to represent the data. Note that in standard deep learning architectures, these transformations also can be problematic. If a particular view of an object should be recognized, it has to be present in the training set and the capacity of the model has to be large enough to account for this transformation. One difference is that the one-hot features learned in our architecture are more specific than the distributed features typically learned in a convolutional neural network trained with gradient descent. It is therefore more difficult to reuse them for representing a large number of transformations.

One of the main advantages of convolutional architectures is that they make object recognition invariant to translational transformation of the objects. However, this also means that potential movement information is lost in the classification process. Our architecture is able to utilize the power of convolutional representations, including translational invariance, while maintaining the movement information up to the full class prototypes. The price we have to pay for this ability is an increase in the number of neurons (i.e. the number of feature maps) in our network to account for the different movement directions. As long as the number of classes is rather small, this does not represent a big problem. Additionally, the increase in the number of neurons is likely to occur mainly in the higher, class specific layers. In the lower layers, the possibility to reuse simpler features for several movement directions of an objects allows us to use less feature maps.

Classification by using movement information

The fact that we obtain representations in the fully connected layer that represent different movement directions and (possibly) viewpoints of an object opens interesting possibilities to improve classification. In this work, we simply used the supervised classifier to merge the responses of the network for different saccades. This allowed us to improve the classification by collecting more information about the presented digit. This approach however does not explicitly use the movement information in the fully connected layer. In contrast to this, we could also imagine a classifier that uses a distinct movement pattern to recognize an object. For example, our network would be able to distinguish a falling object from an ascending one. Also, it could be possible to recognize trajectories of complex objects only by a sequence of a few spikes in the fully connected layer. This would be an interesting way to extend our work with existing spike-based approaches for classifying and learning sequential information (see for instance Bichler et al. [2011]).

Hybrid supervised and unsupervised learning

The main purpose of the supervised classifier is to merge different movement directions for the N-MNIST dataset. We think that hybrid architectures, which combine unsupervised feature extraction in lower layers with a supervised classifier in the top layer, could represent a good compromise for neuromorphic systems. Low level features tend to be very similar for most objects and it may not be necessary to use a supervised feedback signal to learn useful features for higher level processing. For the final high level features, it may be desirable to use a supervised learning mechanism. Usually the high level features will be already very representative of a class and therefore training a classifier does not require many labels to yield good accuracies.

Hardware implementation aspects

Our network is fully event-driven and therefore potentially energy efficient if implemented on an event-based neuromorphic hardware platform. Differences in timescale between inputs and hardware can be a problem for neuromorphic systems if they shall operate on natural stimuli in real time, and the timescale of a neuromorphic system is often a design choice depending on the potential application (see for instance Qiao et al. [2015] and Petrovici et al. [2017] for a real time and faster than real time analog neuromorphic hardware framework). Due to the fully event-based nature of our architecture, the network circumvents this problem and is able to operate on any timescale, with computations being only driven by external input. Together with the simplicity of the neuron and synapse model, our architecture could be easily scaled up and implemented on a wide range of energy efficient neuromorphic hardware platforms. Such event-based vision systems could be interesting for resource constrained applications that have to adapt to new inputs continuously.

3.4.2 Comparison to other approaches

As other systems trained purely with STDP, our network underperforms compared to purely supervised methods trained with the backpropagation algorithm on the MNIST benchmark (see chapter 2 for a summary). For the N-MNIST dataset, almost all existing approaches are also based on backpropagation (table 3.2). Some of these approaches are trained on the N-MNIST dataset directly, while others are trained on MNIST offline with backpropagation and then converted to a spiking network, which is then tested on N-MNIST. The network of Shrestha and Orchard [2018] provides the best classification score of an SNN for the N-MNIST dataset so far, with a maximal score of 99.20%. Iyer et al. [2018] provide competitive results by training an ANN on the accumulated spikes. It is not surprising that the precision of these architectures is

superior to ours. In the supervised deep learning framework, all features of these networks are optimized to yield a high classification performance. This is not the case for an unsupervised architecture as the one we presented in this work, whose objective is merely to extract the statistically most relevant features of the training set given the constraints imposed by the architecture and the learning rules. Additionally, most of these approaches are not trained in the online learning scenario that we consider in this chapter. In addition to these conceptual differences, it is questionable how these implementations of backpropagation could be implemented in typical neuromorphic hardware platforms for on-chip learning. One exception is the implementation of Neftci et al. [2017], that demonstrates the possibility to train a deep network on-chip with an event-based version of the random backpropagation algorithm. Recent results of Kaiser et al. [2018] demonstrate the ability of this approach to learn from a DVS (Dynamic Vision Sensor) event-stream.

In contrast to these supervised approaches, our architecture does not require labels for feature learning (in the case of MNIST) or uses them only to train the classifier at the top of the network (in the case of the N-MNIST dataset). Otherwise, the training data can be treated as a continuous stream of events. In contrast to backpropagation, STDP is a local algorithm, in the sense that it can function without requiring a feedback or error signal from higher layers. This is advantageous because it enables us to perform learning massively parallel, event-based and asynchronously. These properties are beneficial for the online learning problem that we have discussed in this chapter. A similar approach for unsupervised learning on N-MNIST, which was developed in parallel with our work, is provided in Iyer et al. [2018]. They use a similar classifier in the top layer as we have used for MNIST, which is probably the reason why they achieve lower classification performance. In our experiments, we also found it difficult to achieve good performance with the spike count classifier on the N-MNIST dataset, which is why we decided to use a more complex classifier in the top layer to increase performance.

As an approach alternative to neural networks, the HATS network in Sironi et al. [2018] performs unsupervised feature extraction on the N-MNIST dataset, achieving superior performance than most spiking neural network approaches. Although a neuromorphic implementation is claimed to be possible, it is not explicitly demonstrated how efficiently their algorithm could be implemented in neuromorphic hardware, in particular for on-chip learning.

3.4.3 Future outlook

Like other recent work in the field of spike-based learning rules, our work can be seen as a proof of concept, which demonstrates that multi-layer learning of hierarchical features with STDP is possible. Our work extends current approaches by its ability to train all layers simultaneously, and other features

that make it more suitable for a systems that performs simultaneous learning and inference on a continuous stream of data.

Like most machine learning systems, the precision of the network could probably be improved by extended training time and a higher number of training examples. For simulation purposes, this could be done by dataset augmentation, for example by transforming saccades to produce additional movement directions. In an online learning setting, the training examples would simply be the sensor input, and training could be extended by exposing the system to input for a longer time.

We believe the main limit of our approach is the lack of an exact multi-layer optimization framework such as provided by the backpropagation algorithm. Despite the recent advances discussed in section 2.4.2, a high performance on-chip version of the backpropagation algorithm does not exist yet for spiking neural networks. STDP remains one of the few mechanism so far that have been shown to be able to extract sufficiently good hierarchical features in CNN topologies under neuromorphic hardware constraints. An algorithm for inter-layer feedback, compatible with large scale spiking neuromorphic systems, could however prove essential to enable extraction of more complex features and increase inference performance.

Chapter 4

On-chip Learning with Backpropagation in Event-based Neuromorphic Systems

Considering the weaknesses of STDP discussed in the previous chapter, it seems desirable to use an algorithm that is more motivated by machine learning considerations. In particular, it would be extremely useful to be able to use the backpropagation algorithm to train SNNs on-chip. In this chapter, we discuss how the backpropagation algorithm can be adapted to be more consistent with the constraints found in neuromorphic systems. As a first step, we analyze which properties of the backpropagation algorithm are problematic for a spiking neuromorphic hardware implementation.

4.1 Is backpropagation incompatible with neuromorphic hardware?

Despite recent successes for off-chip training of spiking neural networks with the backpropagation algorithm, the backpropagation algorithm has traditionally been omitted for on-chip learning in neuromorphic implementations. The main reason is that backpropagation is generally considered biologically implausible (see Baldi et al. [2017] and Baldi et al. [2018] for a more extensive analysis). Since neuromorphic hardware is inspired by the processing constraints of the human brain, BP is also considered incompatible with these kind of hardware systems. It would however be desirable to enable on-chip learning in neuromorphic chips using the power of the backpropagation algorithm, while transferring the advantages of spike-based processing to the backpropagation phase. In this section, we analyze the principal arguments that are given for the biological implausibility of backpropagation. We focus

on the most relevant points in the context of spiking neuromorphic hardware, and analyze which properties do actually impose a constraint on a potential hardware design.

4.1.1 Local vs. non-local learning algorithms

One major issue with the backpropagation algorithm is non-locality. A local learning algorithm is a learning algorithm that relies only on information immediately available at the neuron. This includes for instance information in its synapses and integration variable. Non-locality can be spatial or temporal. Spatial non-locality means in this context that the neuron requires information that is unavailable at the neuron or the synapses. Temporal non-locality means that the information could in principle be provided, but is not available at the right time. It is important to note that the principle of locality in this context is related to the physical implementation of the system, i.e. if and how information can be communicated.

From a biological perspective, STDP is spatially and temporally fully local. To see this, consider the STDP learning rule (3.4) from chapter 3, which is triggered every time a postsynaptic spike is emitted by neuron i:

$$\Delta w_{ij} = \begin{cases} \alpha_+ \cdot \exp\left(-\beta_+ \cdot w_{ij}\right) & \text{if } t_i^{\text{ref}} < t_{ij}^{\text{pre}} < t_i^{\text{post}} \\ \alpha_- & \text{otherwise} \end{cases} . \tag{4.1}$$

All variables that are necessary for learning in this context are available at the neuron and its synapses. In particular, besides lateral inhibition that can be modeled as a constant synapse, no information from other neurons or an external error signal is necessary. Learning can be triggered immediately when the neuron fires a postsynaptic spike. From an implementation perspective, this is a huge advantage since it does not require synchronization of the input and the learning signal. All neurons can learn simultaneously and asynchronously based only on the input they receive and their response. Constructing a deep spiking network with such a local learning rule was our main objective in designing the system in chapter 3.

However, locality also fundamentally restricts the representations that can be learned. In Baldi and Sadowski [2016], it is argued that this kind of local learning algorithms, where there is no possibility to obtain information from higher layers, cannot learn certain multi-layer representations. In particular, if the whole network is to be optimized for a specific task, purely local learning rules are not able to optimize all layers simultaneously. The STDP learning rule is only capable of unsupervised learning of representations that do not depend on the output of the network. As we have discussed in 2.1, this process of identifying the contribution of each weight value to the final network output is called credit assignment. Backpropagation solves the multi-layer

credit assignment problem by explicitly propagating the error signal through all layers of the network. To demonstrate this, we revisit the backpropagation learning rule 2.20:

$$\Delta w_{ij}^{l} = -\eta E_i^l a_i^{\prime l} y_j^{l-1}, \qquad E_i^l = \sum_k \delta_k^{l+1} w_{ki}^{l+1}. \tag{4.2}$$

The factors y_i^{l-1} and $a_i^{\prime l}$ describe intuitively a correlation between an input and the response of the neuron. These quantities are local and require only information directly accessible to the neuron. This correlation is however additionally weighted by the external error signal E_i^l. This error depends on other error signals δ_k^{l+1} from higher levels of the network, which are backpropagated through the synapses.

The first thing to notice is that this learning rule is temporally non-local, since it requires the simultaneous availability of the external error signal E_i^l from the backpropagation phase, and the local variables $a_i^{\prime l}$ and y_j^{l-1} from the forward propagation phase. Temporal non-locality is a problem in biology, since it requires an exact synchronization of forward and backward propagation. However, in an artificial system, temporal non-locality can be rather easily implemented if there is a clear distinction between prediction and learning phase. We therefore assume that temporal non-locality is not necessarily a problem that has to be considered for a practical neuromorphic implementation. From a spatial viewpoint, the learning rule is not necessarily non-local. Despite the fact that the error signal (4.2) depends on error information provided by other neurons, the rule is only non-local if this information is provided in a way that does not involve the synapses of the neuron, or requires using a communication method that is incompatible with the substrate. If the error signal can reach a neuron like the signals of the forward pass, it can be considered a spatially local variable. It can be seen in (4.2) that the error signal arrives in a very similar way to forward propagation, however through weights of another neuron. We could therefore say that for standard ANNs, backpropagation is spatially local, if neurons in a layer also have access to outgoing weights. In a biological systems, this would be the case if the signal can be received in reverse direction through the axon of the neuron.

4.1.2 The weight transport problem

The backpropagation of errors in (4.2) requires the propagation of a signal in the reverse direction of synapses, dendrites and axons, which is considered impossible in biological neurons. Therefore, in a system with biological constraints, backpropagation would be also a spatially non-local algorithm. This leaves the question how an error signal with sufficient information about the gradient, in particular when taking into account all weight values, can

be "transported" to a neuron. Several suggestions to solve this problem have been proposed in the literature, and there is now a huge body of work related to the biological plausibility of backpropagation (see for instance Baldi and Sadowski [2016], Baldi et al. [2017] or Sacramento et al. [2018]). One of the earliest solutions that has been proposed is *feedback alignment* by Lillicrap et al. [2016]. Feedback alignment relaxes the weight symmetry condition by propagating errors through fixed random synapses during the backward propagation phase. The approximate error $\hat{E}_i$ is obtained by multiplying the backpropagated error δ_k^{l+1} with a layer-specific, fixed random number g_{ki}^{l+1}:

$$\hat{E}_i = \sum_k \delta_k^{l+1} g_{ki}^{l+1}. \tag{4.3}$$

An even more radical extension is *direct feedback alignment* by Nøkland [2016], which propagates errors from the top layer directly to lower layers through fixed random weights g_{ik}^l:

$$\hat{E}_i = \sum_k \delta_k^L g_{ki}^l. \tag{4.4}$$

These approaches, which are sometimes summarized as *random backpropagation*, are more compatible with biology and are therefore considered more suitable for neuromorphic hardware. However, they sacrifice the exactness of the backpropagated gradients. Classification results for these algorithms show that at least for simple benchmarks, deep networks can be trained without a large degradation in performance. However, according to Baldi and Sadowski [2016], backpropagation represents from an algorithmic perspective the optimal way to communicate gradient information to all neurons in the network. The exact impact of these approximations on learning performance is subject of current research efforts (see Bartunov et al. [2018] and Xiao et al. [2019] for two recent experimental studies that come to different conclusions). It however seems that it may be difficult with these approaches to reach the performance of networks trained with the exact backpropagation algorithm that propagates through symmetric weights.

Since we are mostly interested in improving classification performance, we therefore decide to regard the avoidance of weight transport as a problem that is mostly relevant from a viewpoint of biological plausibility. Based on our analysis, we however consider that using symmetric weights may be required from an optimality perspective. Neuromorphic hardware does not necessarily distinguish as strictly as biological neurons between dendrites and axon. Symmetric synapses, that are accessible from neurons in both sending and receiving layer, may therefore be potentially unproblematic to implement. This is why we decide to use symmetric weight propagation in the algorithm developed in this chapter.

Method	spat. local	temp. local	bio. plausible	exact obj.	multi-layer opt.
STDP	✓	✓	✓	(✓)	
Random spiking BP	✓	(✓)	✓	(✓)	✓
Single layer GD	✓	(✓)	✓	✓	
Exact float BP				✓	✓
This work (*SpikeGrad*)	✓	(✓)		✓	✓

Table 4.1: Comparison of different learning algorithms in SNNs. Locality in this context refers to neuromorphic constraints. Parentheses indicate limited viability, or viability under certain conditions.

4.1.3 Coding the gradient into spikes

The main requirement of an implementation of backpropagation in an SNN is therefore that the error has to be backpropagated in the form of spikes. This can be done for instance by the gradient-based on-chip methods of Neftci et al. [2017], Samadi et al. [2017] and Kaiser et al. [2018] that we discussed in section 2.4.2, which propagate the gradient directly to each layer. Besides establishing spatial locality, these methods also try to produce temporal locality by propagating an approximate error already during forward propagation. As a solution that uses symmetric weights, O'Connor and Welling [2016] demonstrate that a small multi-layer fully connected network can be trained using gradients discretized into spikes. All these solutions are therefore compatible with the event-based communication infrastructure of SNNs.

4.1.4 Method comparison and implementation choice

Table 4.1 compares several learning methods for SNNs from a viewpoint of locality, biological plausibility and their capacity to optimize an objective function in a multi-layer structure. All algorithms that either do not communicate an error (STDP), or communicate it in the form of spikes, are spatially local from a spiking (not bio-mimetic) neuromorphic hardware perspective. Most of these algorithms are also temporally local, if an approximate error is already (partially) propagated during forward propagation. Biological plausibility is given for STDP, or for methods where the error does not have to be propagated through symmetric weights. Some implementations of STDP optimize an exact objective (for instance Nessler et al. [2013]), but most of them are rather heuristic. Random BP propagates an error signal based on an exact objective function, which is however distorted by random values. Only backpropagation based methods are able to globally optimize a multi-layer network.

We believe that for a practical application, being able to exactly optimize a multi-layer structure is the most crucial capability of an algorithm. Spatial locality is necessary for an efficient hardware implementation that uses the same communication infrastructure for forward and backward propagation.

The main objective of the approach presented in this chapter is to compromise some aspects of biological plausibility (temporal locality, weight symmetry) in order to bring the performance of spiking neural networks closer to their traditional counterparts. This is done while maintaining event-based communication properties during error propagation, which makes the approach suitable for an on-chip implementation.

4.2 The SpikeGrad computation model

We propose that backpropagation can be made coherent with typical neuromorphic constraints by discretizing the propagated error into signed spike signals. Discretization is done by an additional integration compartment U in each neuron, which integrates errors from higher layers. If a threshold Θ_{bp} is reached, the error signal is propagated in the form of a negative or positive "spike" signal to the layers below. In comparison to O'Connor and Welling [2016], the forward and backward compartments of our network use independent threshold values. This allows us to rescale activations and gradients, which can be essential in tackling the vanishing gradient problem in deep networks with many layers. Using this novel framework, which we call *SpikeGrad*, we are able to demonstrate that even for deep networks, the gradients can be discretized sufficiently well into spikes if the gradient is properly rescaled. As for the forward pass, this allows us to exploit the dynamic precision and sparsity provided by the discretization of all operations into asynchronous spike events. We first demonstrate how this framework can be used to train a network for multi-directional inference of relations. Subsequently, we show that this form of spike-based backpropagation enables us to achieve equivalent or better accuracies on the MNIST and CIFAR10 dataset than comparable state-of-the-art spiking neural networks trained with full precision gradients and comparable to the precision of standard ANNs with the same architecture. The framework is therefore particularly well adapted to neuromorphic implementations in spiking neural networks, since it is possible to use a similar communication infrastructure for forward and backward propagation. Since the algorithm is only based on accumulation and comparison operations, it is in particular suitable for digital neuromorphic platforms (as discussed in chapter 2). Based on our review of the literature, our work provides for the first time an analysis of how the sparsity of the gradient during backpropagation can be exploited within a large-scale spiking CNN processing structure. Additionally, this is the first time competitive classification performances are reported on a large-scale spiking CNN where training and inference are fully implemented with spike operations.

Besides its potential to train an SNN more effectively than previous on-chip learning algorithms, *SpikeGrad* has another advantage. Recent works of

Binas et al. [2016] and Wu et al. [2019a] have discussed how forward processing in an SNN could be mapped to an ANN. Our work extends this analysis to the backward propagation pass. We show that using a special implementation of the integrate-and-fire neuron in *SpikeGrad* allows us to describe the accumulated activations and errors of the spiking neural network in terms of an equivalent artificial neural network. This allows us to largely speed up training compared to an explicit simulation of all spike events, since we can simply simulate the equivalent ANN on high-performance GPUs.

The content of this chapter and preliminary results have been published in Thiele et al. [2018c], Thiele et al. [2019a] and Thiele et al. [2019b].

4.2.1 Mathematical description

Neuron model

We use the following notation for integration times: for each training example or mini-batch, integration is performed from $t = 0$ to $t = T$ for the forward pass and from $t = T + \Delta t$ to $t = \mathcal{T}$ in the backward pass. Since no explicit time is used in the algorithm, Δt symbolically represents the (very short) time between the arrival of an incoming spike and the response of the neuron, which is only used here to describe causality.

The architecture consists of multiple layers (labeled by $l \in [0, L]$) of integrate-and-fire (IF) neurons with integration variable $V_i^l(t)$ and threshold Θ_{ff}:

$$V_i^l(t + \Delta t) = V_i^l(t) - \Theta_{\text{ff}} s_i^l(t) + \sum_j w_{ij}^l s_j^{l-1}(t), \quad V_i^l(0) = b_i^l. \tag{4.5}$$

The variable w_{ij}^l is the weight and b_i^l a bias value. The spike activation function $s_i^l(t) \in \{-1, 0, 1\}$ is a function that triggers a signed spike event depending on the internal variables of the neuron. It will be shown later that the specific choice of the activation function is fundamental for the mapping to an equivalent ANN. After a neuron has fired, its integration variable is decremented or incremented by the threshold value Θ_{ff}, which is represented by the second term on the r.h.s. of (4.5).

As a representation of the neuron activity, we use a trace $x_i^l(t)$ that accumulates spike information over an example presentation:

$$x_i^l(t + \Delta t) = x_i^l(t) + \eta s_i^l(t). \tag{4.6}$$

The trace is updated every time a postsynaptic spike is triggered in a neuron. Since $s_i^l(t)$ is a ternary function, the trace is simply a weighted activity counter. This trace will also play a role in the weight update rule (4.13) that is derived later, where it will serve as a representation of the total received input by a synapse. By weighting the activity with the learning rate η, we

avoid performing a multiplication when weighting the total activity with the learning rate. This ensures that integration and neuron dynamics are completely multiplication free, and only based on temporal accumulations and comparisons of state variables.

Implementation of implicit ReLU and surrogate activation function derivative

It is possible to define an implicit activation function based on how the neuron variables affect the spike activation function $s_i^l(t)$. In our implementation, we use the following fully symmetric function to represent linear activation functions (used for instance in pooling layers):

$$s_i^{l,\text{lin}}(t) := \begin{cases} 1 & \text{if } V_i^l(t) \geq \Theta_{\text{ff}} \\ -1 & \text{if } V_i^l(t) \leq -\Theta_{\text{ff}} \\ 0 & \text{otherwise} \end{cases} \tag{4.7}$$

The following function corresponds to the rectified linear unit (ReLU) activation function:

$$s_i^{l,\text{ReLU}}(t) := \begin{cases} 1 & \text{if } V_i^l(t) \geq \Theta_{\text{ff}} \\ -1 & \text{if } V_i^l(t) \leq -\Theta_{\text{ff}} \text{ and } x_i^l(t) > 0 \\ 0 & \text{otherwise} \end{cases} . \tag{4.8}$$

This represents another utilization of the trace $x_i^l(t)$. If $x_i^l(t) \leq 0$, that means if the total propagated output is negative, no negative spike can be triggered. This ensures that the total propagated output is always positive. In contrast to most implementations of spiking neural networks, we however allow for the propagation of negative spikes, even in the case of the ReLU activation function. If we restrict the model to positive spikes, the output of the neuron has a strong dependence on the order of spike arrival. To give an example, assume that weight values can be positive or negative. If there are two spikes which arrive at the neuron, one of them increasing $V_i^l(t)$ by 1 and the other one decreasing $V_i^l(t)$ by -1, the net change of the integration variable will be 0. However, if the positive contribution is integrated first and the integration passes the threshold value, the neuron will emit a spike, which would not be the case if the negative contribution is integrated first. By allowing negative spikes, spikes propagated in excess can be corrected by a subsequent negative spike if the integration drops below the negative threshold $-\Theta_{\text{ff}}$ (and the total integration becomes smaller than 0). The response of the neuron is therefore similar to a dynamically discretized version of the rectified linear unit (ReLU). Negative input will trigger no activation (spikes), while positive input will increase the firing rate linearly. In a discrete time simulation or a physical

implementation, there can be an additional limit imposed on the achievable firing rate.

The pseudo-derivative of the activation function is denoted symbolically by $S_i'^{l}$. Several solutions have been proposed to approximate the derivative of the discontinuous spike activation function, that have been discussed in chapter 2. We use $S_i'^{l,\mathrm{lin}}(t) := 1$ for the linear case. For the ReLU, we use a surrogate of the form:

$$S_i'^{l,\mathrm{ReLU}}(t) := \begin{cases} 1 & \text{if } V_i^l(t) > 0 \text{ or } x_i^l(t) > 0 \\ 0 & \text{otherwise} \end{cases}. \tag{4.9}$$

These choices are based on the accumulated activity of the neuron and will be motivated in the following sections. We choose here that the derivative is also 1 if $x_i^l(t) = 0$, but $V_i^l(t) > 0$. In this way learning can also take place even if no spikes are triggered as long as the membrane potential is larger than 0. We observed additionally that setting the derivative to 0 for activations above the maximal firing rate of the network can be important for stable convergence. This is because otherwise the optimization algorithm will push the firing rate higher and higher without the network being able to represent the number, which results in a potential infinite increase of the weights. In principle the derivatives are defined for all t. To be able to exactly approximate the backpropagation algorithm, we use the derivatives defined on the final states of the forward pass at time T.

Discretization of gradient into spikes

We just described how the IF neuron model can be seen, under certain conditions, as a dynamic discretization of a standard frame-based neuron. Given the rather symmetric propagation of feedforward and backpropagated signals in ANNs, this raises the questions if the same principle can be applied to error propagation in SNNs. In particular, if it is possible to propagated the error and perform learning event-based and without the need to perform multiplications.

For this purpose, it may be helpful to revisit the equations of the backpropagation algorithm. It is easy to see that the feedforward input in (2.9) has structural similarity with the backpropagated error (4.2). Both equations are simply weighted sums of inputs from other layers. In the spiking neuron model (4.5), the multiply-accumulate operations are discretized into several accumulation operations over time. We apply now the same idea to the integration of the error signal, by introducing a second compartment in the neuron with a threshold Θ_{bp}, which integrates error signals analogously to (4.5). The process discretizes errors in the same fashion as the forward pass

discretizes an input signal into a sequence of signed spike signals:

$$U_i^l(t+\Delta t) = U_i^l(t) - \Theta_{\text{bp}} z_i^l(t) + \sum_k w_{ki}^{l+1} \delta_k^{l+1}(t). \tag{4.10}$$

To this end, we introduce a ternary *error spike activation function* $z_i^l(t) \in \{-1, 0, 1\}$ that is defined in analogy to (4.7), using the error integration variable $U_i^l(t)$ and the backpropagation threshold Θ_{bp}:

$$z_i^l(t) = \begin{cases} 1 & \text{if } U_i^l(t) \geq \Theta_{\text{bp}} \\ -1 & \text{if } U_i^l(t) \leq -\Theta_{\text{bp}} \\ 0 & \text{otherwise} \end{cases}. \tag{4.11}$$

In the same way as $V_i^l(t)$ represents a spike discretization of the feedforward input, $U_i^l(t)$ represents a spike discretization of the error (4.2). Forward and backward propagation are therefore highly symmetric in their corresponding discretization mechanisms.

The local error is then obtained by gating this ternarized function $z_i^l(t)$ with one of the surrogate activation function derivatives of the previous section (linear or ReLU):

$$\delta_i^l(t) = z_i^l(t) S_i'^l(t). \tag{4.12}$$

This is also a ternarized function if $S_i'^l(t)$ is ternary.

Weight update rule

This ternary spike signal is backpropagated through the weights to the lower layers and also applied in the update rule of the *weight increment accumulator* ω_{ij}^l:

$$\omega_{ij}^l(t+\Delta t) = \omega_{ij}^l(t) - \delta_i^l(t) x_j^{l-1}(t), \tag{4.13}$$

which is triggered every time an error spike signal $\delta_i^l(t)$ is backpropagated. Since $\delta_i^l(t)$ is discretized into spikes signals, this can also be written as:

$$\Delta\omega_{ij}^l(t+\Delta t) = \begin{cases} -x_j^{l-1}(t) & \text{if } \delta_i^l(t) = 1 \\ x_j^{l-1}(t) & \text{if } \delta_i^l(t) = -1 \\ 0 & \text{otherwise} \end{cases}. \tag{4.14}$$

For an implementation that corresponds to standard backpropagation, all weight updates are accumulated in ω_{ij}^l during backpropagation and only applied to update the weight after the backpropagation phase has finished. It

would however also be possible to apply the weight increments $\Delta\omega_{ij}^{l}$ directly every time a spike passes the synapse. This would reduce the memory requirements of the algorithm since we would not have to store ω_{ij}^{l}. On the other hand, it introduces a systematic error in the backpropagation process, since backpropagated errors have to be calculated using the same weight values as during the forward pass.

To obtain the exact implementation of backpropagation, we also have to use the final values $x_j^{l-1}(T)$, $V_i^l(T)$ and $S_i'^l(T)$ during the error propagation phase. In another variant of the algorithm, errors could already be propagated during the forward pass and weights would be updated based on the current values of $x_j^{l-1}(t)$, $V_i^l(t)$ and $S_i'^l(t)$. This would be more biologically plausible, since no strict synchronization is required, and would make the algorithm temporally local. It however only approximates the true value of the gradient and requires that the firing statistics of the neurons in the network are approximately constant during propagation. O'Connor and Welling [2016] tested a similar mechanism in a small network and found that the error seems to be rather small compared to standard BP. Since our aim was to implement backpropagation exactly, and additionally not to depend on constant firing rates, we did not test how this variation affects the performance of our algorithm.

The initial error signal is generated in the output layer and propagated continuously over a certain time span, just like during forward propagation. This enables us to obtain an error signal with dynamic precision. By encoding the error into more spikes and performing error propagation for a longer time, the precision of the propagated error can be increased.

Additional learning mechanisms

Learning rate decay In our simulations, we found it useful to implement learning rate decay, which can easily be done by changing the value of the increment η in (4.6).

Momentum To smooth the gradient and speed up learning, *momentum* is a popular method used for gradient descent based trained of DNNs (Sutskever et al. [2013]). Momentum modifies the gradient descent update rule (2.6) by including information from the last weight increment v in the weight update:

$$\Delta w = \mu \Delta w^{\text{prev}} - \eta \frac{\partial \mathcal{C}}{\partial w}, \tag{4.15}$$

where $\mu \in [0, 1]$ is used to choose the fraction of the previous update that should be included. In the framework of the *SpikeGrad* algorithm, one possible implementation is to introduce a second trace $\hat{x}_i^l$, which accumulates gradient

weight updates that are additionally weighted by μ:

$$\hat{x}_i^l(t + \Delta t) = \hat{x}_i^l(t) + \mu\eta s_i^l(t). \tag{4.16}$$

The corresponding weight accumulator is given by:

$$\hat{\omega}_{ij}^l(t + \Delta t) = \hat{\omega}_{ij}^l(t) - \delta_i^l(t)\hat{x}_j^{l-1}(t). \tag{4.17}$$

Using momentum thus requires one additional accumulator $\hat{x}_j^{l-1}$ per neuron if multiplications are to be avoided. In both the case of *SpikeGrad* and a normal ANN, momentum requires us to additionally save the (accumulated) weight increments from the last gradient descent update. In *SpikeGrad*, the variable $\hat{\omega}_{ij}^l(\hat{\mathcal{T}})$ (were $\hat{\mathcal{T}}$ presents the time when the previous BP phase has terminated) is added to the weights when the regular weight update is performed at time $\mathcal{T}$.

Weight initialization Due to the similarity of the IF neuron with the ReLU activation function, we use an initialization method proposed for deep networks using this type of activation function by He et al. [2015]:

$$w \sim \mathcal{N}(0, \sqrt{2/n^{\text{in}}}), \tag{4.18}$$

where n^{in} is the number of incoming connections of the neuron.

4.2.2 Event-based formulation

In the previous section, all equations were described in the context of a time-stepped simulation. The *SpikeGrad* algorithm can also be expressed in a purely event-based formulation, described in algorithms 1, 2 and 3. This formulation is closer to how the algorithm would be implemented in an actual SNN hardware system. Figure 4.1 shows the algorithm as an abstract computation graph.

4.2.3 Reformulation as integer activation ANN

As discussed in section 2.3, the computational complexity of the simulation of the temporal dynamics of spikes increases with the number of events. In particular, assuming integer activation values, the required number of time-steps in a clock-based simulation scales linearly with the absolute value of an activation. It would therefore be extremely beneficial if we were able to map the SNN to an equivalent ANN that does not simulate explicitly the spike dynamics and can therefore be trained much faster on standard DL hardware. In this section, we demonstrate that it is possible to find such an ANN using the forward and backward propagation dynamics described in the previous section.

Algorithm 1 Forward

```
function PROPAGATE([l, i, j], s)
    V_i^l ← V_i^l + s · w_ij^l
    s_i^l ← s_i^l(V_i^l, x_i^l)                         ▷ spike activation function
    if s_i^l ≠ 0 then
        V_i^l ← V_i^l − s_i^l · Θ_ff
        x_i^l ← x_i^l + η s_i^l                          ▷ update output trace
        for neuron k in layer l + 1 connected to neuron i do
            PROPAGATE([l + 1, k, i], s_i^l)
```

Algorithm 2 Backward

```
function BACKPROPAGATE([l, i, k], δ)
    U_i^l ← U_i^l + δ · w_ki^{l+1}
    z_i^l ← z_i^l(U_i^l)                                ▷ error activation function
    δ_i^l ← z_i^l · S'_i^l
    if z_i^l ≠ 0 then
        U_i^l ← U_i^l − z_i^l · Θ_bp
        for neuron j in layer l − 1 connected to neuron i do
            BACKPROPAGATE([l − 1, j, i], δ_i^l)
            ω_ij^l ← ω_ij^l − δ_i^l · x_j^{l−1}          ▷ weight update accumulator
```

Algorithm 3 Training of single example/batch

```
init: V ← b, U ← 0, x ← 0, ω ← 0  ▷ variables in bold describe all neurons
in network/layer
while input spikes s_i^in do
    for neuron k in l = 0 receiving s_i^in do        ▷ spikes of training input
        PROPAGATE([0, k, i], s_i^in)
S' ← S'(V, x)                                       ▷ calculate surrogate derivatives
U^L ← α · ∂L/∂V^L                                   ▷ calculate classification error
for neuron i in l = L do
    while |U_i^L| ≥ Θ_bp do                          ▷ backpropagate error spikes
        BACKPROPAGATE([L, i, −], 0)                  ▷ last layer receives no error
w ← w + ω                    ▷ update weights with weight update accumulator
```

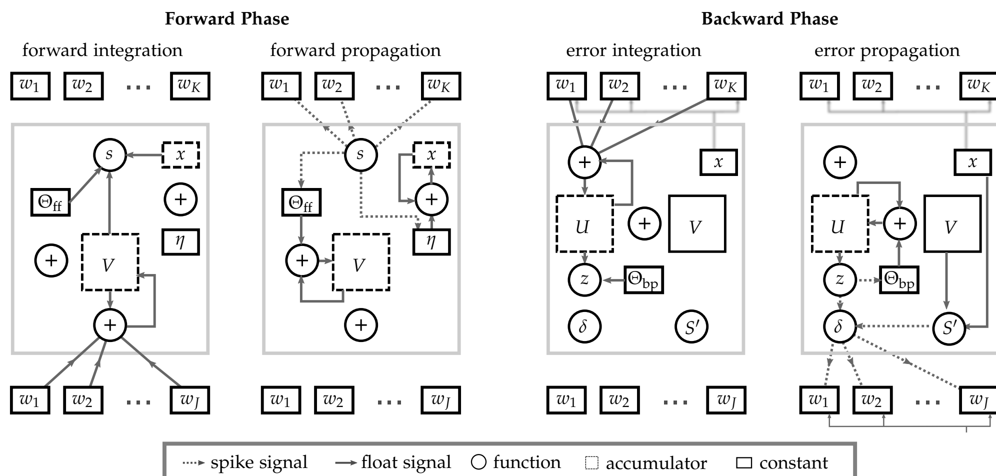

Figure 4.1: Forward and backward phase in a single neuron in the *SpikeGrad* algorithm. **Forward integration:** Every time a spike signal arrives at one of the synapses $\{w_1, ..., w_J\}$, the value of the weight is added to the integration variable V (4.5), weighted by the sign of the spike. After each such event the integration variable is compared to the threshold value $\pm\Theta_{\text{ff}}$ and the synaptic trace x by the function s, which decides if a spike is triggered. **Forward propagation:** If the conditions imposed by s are satisfied, a spike is triggered. This spike increases the trace x (4.6) by the learning rate $\pm\eta$. Additionally, it is sent to the outgoing connections. The signal also applies $\pm\Theta_{\text{ff}}$ to V depending on the sign of s (4.5). **Error integration:** Signed error spikes are received through the synapses $\{w_1, ..., w_K\}$ of the outgoing connections. The value of the weight is added to the error integration variable U (4.10). After each such signal, U is compared to $\pm\Theta_{\text{bp}}$ by the function z (4.11). **Error propagation:** If the threshold is crossed, a signed error spike is emitted. U is incremented by $\pm\Theta_{\text{bp}}$. The error spike signal is gated by the surrogate activation function derivative (4.9) (which is calculated based on V and x) and backpropagated through the incoming connections. The weights of the neuron are updated (4.13) using this error signal, and the traces (4.6) from the neurons in the layer below.

Spike discretization error We start our analysis with equation (4.5). We reorder the terms and sum over the increments $\Delta V_i^l(t) = V_i^l(t + \Delta t) - V_i^l(t)$ every time the integration variable is changed either by a spike that arrives at time $t_j^s \in [0, T]$ via connection j, or by a spike that is triggered at time $t_i^s \in [0, T]$. With the initial conditions $V_i^l(0) = b_i^l$, $s_i^l(0) = 0$, we obtain the final value $V_i^l(T)$:

$$V_i^l(T) = \sum_{t_j^s, t_i^s} \Delta V_i^l = -\Theta_{\text{ff}} \sum_{t_i^s} s_i^l(t_i^s) + \sum_j w_{ij}^l \sum_{t_j^s} s_j^{l-1}(t_j^s) + b_i^l \tag{4.19}$$

By defining the total transmitted output of a neuron as $S_i^l := \sum_{t_i^s} s_i^l(t_i^s)$ we obtain:

$$\frac{1}{\Theta_{\text{ff}}} V_i^l(T) = \mathbb{S}_i^l - S_i^l, \quad \mathbb{S}_i^l := \frac{1}{\Theta_{\text{ff}}} \left(\sum_j w_{ij}^l S_j^{l-1} + b_i^l \right) \tag{4.20}$$

The same reasoning can be applied to backpropagation of the gradient. We define the summed responses over error spike times $\tau_j^s \in [T + \Delta t, \mathcal{T}]$ as $Z_i^l := \sum_{\tau_i^s} z_i^l(\tau_i^s)$ to obtain:

$$\frac{1}{\Theta_{\text{bp}}} U_i^l(\mathcal{T}) = \mathbb{Z}_i^l - Z_i^l, \quad \mathbb{Z}_i^l := \frac{1}{\Theta_{\text{bp}}} \left(\sum_k w_{ki}^{l+1} E_k^{l+1} \right) \tag{4.21}$$

$$E_k^{l+1} = \sum_{\tau_k^s} \delta_k^{l+1}(\tau_k^s) = \sum_{\tau_k^s} S_k^{\prime l+1}(T) z_k^{l+1}(\tau_k^s) = S_k^{\prime l+1}(T) Z_k^{l+1}. \tag{4.22}$$

In both equations (4.20) and (4.21), the terms $\mathbb{S}_i^l$ and $\mathbb{Z}_i^l$ are equivalent to the output of an ANN with signed integer inputs S_j^{l-1} and E_k^{l+1}. $1/\Theta_{\text{ff}}$ and $1/\Theta_{\text{bp}}$ can be interpreted as scaling factors of activation and gradient. If gradients are not to be explicitly rescaled, backpropagation requires $\Theta_{\text{bp}} = \Theta_{\text{ff}}$. The values of the residual integrations $1/\Theta_{\text{ff}} V_i^l(T)$ and $1/\Theta_{\text{bp}} U_i^l(\mathcal{T})$ therefore represent the *spike discretization error* $\text{SDE}_{\text{ff}} := \mathbb{S}_i^l - S_i^l$ or $\text{SDE}_{\text{bp}} := \mathbb{Z}_i^l - Z_i^l$ between the ANN outputs $\mathbb{S}_i^l$ and $\mathbb{Z}_i^l$ and the accumulated SNN outputs S_i^l and Z_i^l. Since we know that $V_i^l(T) \in (-\Theta_{\text{ff}}, \Theta_{\text{ff}})$ and $U_i^l(\mathcal{T}) \in (-\Theta_{\text{bp}}, \Theta_{\text{bp}})$, this gives bounds of $|\text{SDE}_{\text{ff}}| < 1$ and $|\text{SDE}_{\text{bp}}| < 1$.

So far we can only represent linear functions. We now consider an implementation where the ANN applies a ReLU activation function instead. The SDE in this case is:

$$\text{SDE}_{\text{ff}}^{\text{ReLU}} := \text{ReLU}\left(\mathbb{S}_i^l\right) - S_i^l. \tag{4.23}$$

We can calculate the error by considering that (4.8) forces the neuron in one of two regimes (note that $x_i^l > 0 \Leftrightarrow S_i^l > 0$): In one case, $S_i^l = 0$, $V_i^l(T) < \Theta_{\mathrm{ff}}$ (this includes $V_i^l(T) \leq -\Theta_{\mathrm{ff}}$). This implies $\mathbb{S}_i^l = {}^1/_{\Theta_{\mathrm{ff}}} V_i^l(T)$ and therefore $|\mathrm{SDE}_{\mathrm{ff}}^{\mathrm{ReLU}}| < 1$ (or even $|\mathrm{SDE}_{\mathrm{ff}}^{\mathrm{ReLU}}| = 0$ if $V_i^l(T) \leq 0$). In the other case, $S_i^l > 0$, $V_i^l(t) \in (-\Theta_{\mathrm{ff}}, \Theta_{\mathrm{ff}})$, where (4.8) is equivalent to (4.7).

This equivalence motivates the choice of (4.9) as a surrogate derivative for the SNN: the condition $(V_i^l(T) > 0 \text{ or } x_i^l(T) > 0)$ can be seen to be equivalent to $\mathbb{S}_i^l(T) > 0$, which is the condition for a non-zero value of a ReLU. Finally, for the total weight increment Δw_{ij}^l, it can be seen from (4.6) and (4.13) that:

$$x_i^l(T) = \sum_{t_i^s} \Delta x_i^l(t_i^s) = \eta S_i^l, \quad \Rightarrow \quad \Delta w_{ij}^l(\mathcal{T}) = \sum_{\tau_i^s} \Delta \omega_{ij}^l(\tau_i^s) = -\eta S_j^{l-1} E_i^l, \tag{4.24}$$

which is exactly the weight update formula of an ANN defined on the accumulated variables. We have therefore demonstrated that the SNN can be represented by an ANN by replacing recursively all S and Z by $\mathbb{S}$ and $\mathbb{Z}$ and applying the corresponding activation function directly on these variables. The error that will be caused by this substitution compared to using the accumulated variables S and Z of an SNN is described by the SDE. This ANN can now be used for training of the SNN on GPUs. The *SpikeGrad* algorithm formulated on the variables s, z, δ and x represents the algorithm that would be implemented on an event-based *spiking* neural network hardware platform. We will now demonstrate how the SDE can be further reduced to obtain an ANN and SNN that are exactly equivalent.

ANN response equivalence For a large number of spikes, the SDE may be negligible compared to the activation of the ANN. However, in a framework whose objective it is to minimize the number of spikes emitted by each neuron, this error can have a potentially large impact.

One option to reduce the error between the ANN and the SNN output is to constrain the ANN during training to integer values. One possibility is to round the ANN outputs:

$$\mathbb{S}_i^{l,\mathrm{round}} := \mathrm{round}[\mathbb{S}_i^l] = \mathrm{round}\left[\frac{1}{\Theta_{\mathrm{ff}}}\left(\sum_j w_{ij}^l \mathbb{S}_j^{l-1} + b_i^l\right)\right], \tag{4.25}$$

The round function here rounds to the next integer value, with boundary cases rounded *away* from zero. This behavior can be implemented in the SNN by a modified spike activation function which is applied after the full stimulus has been propagated. To obtain the exact response as the ANN, we have to

take into account the current value of S_i^l and modify the threshold values:

$$s_i^{l,\text{res}}(T) := \begin{cases} 1 & \text{if } V_i^l(T) > {}^{\Theta_{\text{ff}}}/2 \text{ or } (S_i^l \geq 0,\ V_i^l(T) = {}^{\Theta_{\text{ff}}}/2) \\ -1 & \text{if } V_i^l(T) < -{}^{\Theta_{\text{ff}}}/2 \text{ or } (S_i^l \leq 0,\ V_i^l(T) = -{}^{\Theta_{\text{ff}}}/2) \\ 0 & \text{otherwise} \end{cases}. \tag{4.26}$$

Because this spike activation function is applied only to the residual values, we call it the *residual spike activation function*. The function is applied to a layer after all spikes have been propagated with the standard spike activation function (4.7) or (4.8). We start with the lowest layer and propagate all residual spikes to the higher layers, which use the standard activation function. We then proceed by setting the next layer to residual mode and propagating the residual spikes. This is continued until we arrive at the last layer of the network. By considering all possible rounding scenarios, it can be seen that (4.26) indeed implies:

$$S_i^l + s_i^{l,\text{res}}(T) = \text{round}[S_i^l + {}^1/_{\Theta_{\text{ff}}} V_i^l(T)] = \text{round}[\mathbb{S}_i^l]. \tag{4.27}$$

In particular, the boundary cases are rounded correctly: for $V_i = \Theta_{\text{ff}}$, we obtain $\mathbb{S} = S_i + 0.5$. For $S_i \geq 0$, this should be rounded to $\text{round}[\mathbb{S}] = S_i + 1$ and for $S_i < 0$, we should obtain $\text{round}[\mathbb{S}] = S_i$. Similarly, for $V_i = -\Theta_{\text{ff}}$, we obtain $\mathbb{S} = S_i - 0.5$. For $S_i \leq 0$, this should be rounded to $\text{round}[\mathbb{S}] = S_i - 1$ and for $S_i > 0$, we should obtain $\text{round}[\mathbb{S}] = S_i$. The same principle can be applied to obtain integer-rounded error propagation:

$$\mathbb{Z}_i^{l,\text{round}} := \text{round}\left[\mathbb{Z}_i^l\right] = \text{round}\left[\frac{1}{\Theta_{\text{bp}}}\left(\sum_k w_{ki}^{l+1} E_k^{l+1}\right)\right]. \tag{4.28}$$

We have to apply the following modified spike activation function in the SNN after the full error has been propagated by the standard error spike activation function:

$$z_i^{l,\text{res}}(\mathcal{T}) := \begin{cases} 1 & \text{if } U_i^l(\mathcal{T})) > {}^{\Theta_{\text{bp}}}/2 \text{ or } (Z_i^l \geq 0,\ U_i^l(\mathcal{T}) = {}^{\Theta_{\text{bp}}}/2) \\ -1 & \text{if } U_i^l(\mathcal{T}) < -{}^{\Theta_{\text{bp}}}/2 \text{ or } (Z_i^l \leq 0,\ U_i^l(\mathcal{T}) = -{}^{\Theta_{\text{bp}}}/2) \\ 0 & \text{otherwise} \end{cases}, \tag{4.29}$$

which implies:

$$Z_i^l + z_i^{l,\text{res}}(\mathcal{T}) = \text{round}[Z_i^l + {}^1/_{\Theta_{\text{bp}}} U_i^l(\mathcal{T})] = \text{round}[\mathbb{Z}_i^l]. \tag{4.30}$$

Therefore the SNN will, after each propagation phase, have exactly the same accumulated responses as the corresponding ANN. The same principle can be applied to obtain other forms of rounding (e.g. floor and ceil), if (4.26) and (4.29) are modified accordingly.

4.2.4 Compatible layer types

Our proposed backpropagation scheme and neuron model can be applied to most layer types that are commonly used in deep neural networks, since it operates on the fundamental elements every artificial neural network using backpropagation needs to possess. In particular, our scheme is formulated on the local level of a single neuron, and is therefore agnostic of the connectivity structure. We discuss here briefly the implementation in some common layer types.

Convolutional layers

In the case of convolutional layers, the inputs to a neuron represent the (partial) inputs of a neuron in a feature map. The error signals represent the (partial) errors that arrive from the layer above. The assignment of input and backpropagated signals to neurons is done in exactly the same fashion as in a non-spiking convolutional neural network. As typical for convolutional architectures, weights between all neurons in one feature map are shared, and therefore the weight update of a neuron affects the weights of all other neurons in the same feature map. Note that the fully connected layer is simply a subtype of the convolutional layer, where each neuron presents a feature map whose kernel covers the full input volume.

Pooling layers

Max pooling During the forward pass, a max pooling neuron only transmits a spike if it is emitted by the most active neuron in the pooling window, and its firing count is constrained to be lower or equal to the activity of this neuron. During the backward pass, the error is only propagated to the neuron in the pooling window that had the maximal activity during the forward pass. We therefore have to perform a dynamic approximation to select the neuron in the pooling window with highest activity. For this purpose, we use the following activation function (assuming that η is the same for all layers):

$$s_i^l(t) = \begin{cases} 1 & \text{if } x_i^l(t) < x_i^{l,\max}(t) \\ -1 & \text{if } x_i^l(t) > x_i^{l,\max}(t) \\ 0 & \text{otherwise} \end{cases} . \quad (4.31)$$

The variable $x_i^{l,\max}(t)$ is updated dynamically to represent the current maximal $x_j^{l-1}(t)$ of all incoming neurons in the pooling window $\mathcal{P}_i^l$:

$$x_i^{l,\max}(t) = \max\left(x_j^{l-1}(t)\right), \; j \in \mathcal{P}_i^l. \quad (4.32)$$

For backpropagation, we treat the pooling neuron as in a standard ANN. The error is only propagated to the input neuron in the pooling window with maximal activity.

Average pooling Forward propagation in an average pooling layer outputs the average of all neurons in the pooling window. For backpropagation, the error is divided by the size of the pooling window and propagated to all neurons in the pooling window. Both operations can simply be implemented as an IF neuron with linear activation function (4.8) and constant weights:

$$w_{ij}^l = \frac{1}{(P^l)^2}, \tag{4.33}$$

where P^l is the size of the pooling window. Because average pooling does not require dynamic estimation of the maximal firing rate, it is easier to implement in SNNs. This is why we use average pooling in all simulations.

Dropout

Dropout (Srivastava et al. [2014]) can be implemented by an additional gating variable which randomly removes some neurons during forward and backward propagation. During learning, the activations of the active neurons have to be rescaled by the inverse of the dropout probability $1/(1 - p_{\text{drop}})$. This can be implemented in the SNN framework by rescaling the threshold values by $(1 - p_{\text{drop}})$.

4.3 Experiments

We now demonstrate the capabilities of the *SpikeGrad* algorithm on two different machine learning tasks.

4.3.1 A network for inference of relations

Artificial neural networks are mostly known for their feedforward processing capacities, where a well-defined input X leads to a well-defined output Y. This is however different to the processing paradigm of the human brain. Even pathways that are often characterized by their feedforward structure, such as the visual system, possess a large level of recurrent connectivity between different levels of processing (Binzegger et al. [2004]). This type of recurrent connectivity allows the brain to do inference not only in a one-directional feedforward fashion, but in an associative, relational way, potentially involving several hierarchical levels (Felleman and Van Essen [1991]). Stimuli that are related with each other produce correlated activity patterns,

and presentation of any of these stimuli will also produce high activity in an area representing other, related stimuli. This enables neural networks to perform relational inference, i.e. to relate different stimuli with each other in a multi-directional way.

In this section, we present an approach to train a network of spiking neurons with the aforementioned multidirectional inference capabilities, using *SpikeGrad*. In previous work on biologically inspired implementations of relational networks (such as Deneve et al. [2001], Diehl and Cook [2016], Thiele et al. [2017b]), connections are either hardwired or were learned with biologically inspired learning rules, such as spike-timing dependent plasticity. Our work can be seen as an extension of these previous approaches for training relational networks of spiking neurons. We demonstrate that our training mechanism achieves superior performance compared to the biologically inspired approaches. We show additionally that our network is able to process more complex visual stimuli and set them in relationship with each other. This is demonstrated by learning relational inference on the visual XOR task, using images from the MNIST dataset. The use of spiking neurons in this context is particularly interesting, since the low-power properties of SNNs are predestined for the use in sensor fusion applications on mobile platforms.

The experiments on inference of relations use a simplified version of the *SpikeGrad* framework, that does not propagate residual spikes.

Relational network topology

We now describe the basic structure and properties of the relational network topology that is trained with *SpikeGrad*. The relational network allows to fuse several inputs and set them in relation with each other. Based on a subset of input stimuli, the network is able to recreate the missing ones in the form of artificial patterns that possess similar structural properties as the original ones.

To represent a relation with n variables in a relational network, we couple $n+1$ populations. n of these populations will learn to represent the n input variables, while population $n+1$ will set these representations in relationship with each other. We distinguish 3 types of populations (see figure 4.2):

- Input-output (IO) populations: These populations provide the input patterns to the network or reconstruct the missing input, depending on the inference direction.
- Peripheral populations: These populations process the input from IO-populations to find a high level representation, or use the hidden population output to generate a representation for an IO-population.
- Hidden population: The hidden population receives the processed input from the peripheral populations, processes it further and sends it to

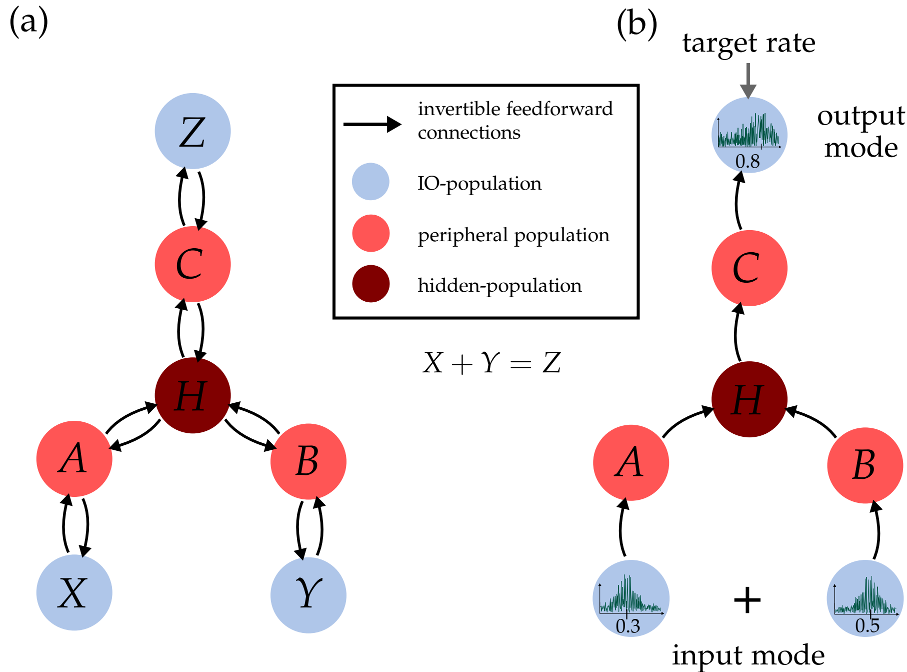

Figure 4.2: (a) Relational network architecture with three variables. IO-populations are labeled X, Y, Z. Peripheral populations are labeled A, B, C and the hidden population H. (b) Training of the network. During learning, two populations serve as input populations and provide a firing pattern. The third population is trained to reproduce a target firing pattern based on this input. The roles of the populations are interchanged during training to enable inference for all variable combinations.

other peripheral populations. It therefore sets the different inputs from the peripheral populations in relation with each other.

The basic relational network of 3 variables can in principle be extended to arbitrarily complex structures by coupling several of these basic networks, since any of the populations can simply be replaced by another relational network.

Our network has the same inter-population connectivity structure as the relational network in Diehl and Cook [2016], with the difference that additionally the IO-populations have feedforward input from the peripheral populations. The peripheral populations receive connections from their corresponding IO-populations and the hidden population. The hidden population receives feedforward input from all peripheral populations. In contrast to the network in Diehl and Cook [2016], all connections are bidirectional in the sense that they allow the backpropagation of an error signal. The overall topology however remains equivalent regarding which population is connected with each other. Our architecture requires no recurrent connections between neurons in a population.

Symbol	Description	Value
Θ_{ff}	Threshold for forward propagation	1.0
Θ_{bp}	Threshold for backpropagation	1.0
η	Learning Rate	0.00005
r_{max}	Maximal input firing rate	0.12
$t_{\mathrm{expl.}}$	Example presentation time	100
t_{BP}	BP presentation time	10
Δt	Simulation time step	1
N_{train}	# relation samples used for training	10000
$N^{A,B,C}$	# neurons in peripheral populations	256
N^{H}	# neurons in hidden population	128

Table 4.2: Parameters used for relational network training

Experimental setup

We now train the relational network topology on two relational inference tasks: a periodic addition and a visual XOR task. If not stated otherwise, we use the parameters given in table 4.2. Since our simulation does not require the definition of an explicit time scale, all time related variables are given in relative units.

All simulations are performed with a custom clock-based simulator based on the N2D2 open source machine learning library by Bichler et al. [2019]. Training is performed on *Nvidia TitanX* graphics cards.

Loss function For our experiments, each branch of the relational network is trained to predict a desired spike output pattern of a variable, given as an input the two other. We use the simple L_2 loss function:

$$\mathcal{L} = \frac{1}{2}\sum_{i}(y_i - t_i)^2 \tag{4.34}$$

where y_i is the accumulated activity of the inferring population and t_i the target spike pattern activity. We therefore encode information in accumulated spike activity. In our implementation, we use $y_i = S_i^L + V_i^L$ to enable learning even if there is no spike in the final layer. This loss yields the following error for the final layer at the beginning of the backpropagation phase:

$$\frac{d\mathcal{L}}{dy_i} = y_i - t_i. \tag{4.35}$$

In the inferring layer, it is directly transferred to the error integration variable U_i^L of each neuron in the final layer. Note that this way it is possible that $|U_i^L| > \Theta_{\mathrm{bp}}$. In our implementation, the neuron will produce a spike at every

time step and decrease U_i^L by Θ_{bp} according to (4.10) as long as the integration variable exceeds the threshold.

Training procedure The network is trained in a supervised fashion, in the sense that a subset of the variables of the relation are provided as an input, while all other variables serve as targets. $n-1$ of the IO-populations provide input spike trains, while the remaining population is trained to reproduce the spike pattern of the missing variable (figure 4.2). A subset of all possible connections is enabled so that for each variable the network functions as a feedforward network. This mechanism is rotated so that each input population serves as output populations equally often during training. This way, all relations are simultaneously represented in the network weights. All feedforward connections leading to the hidden population are simultaneously optimized for all inference directions of the relation (for instance in figure 4.2, the weights $X \to A$, $A \to H$ are optimized for inference of Y and Z, $Y \to B$, $B \to H$ for inference of X and Z, and $Z \to C$, $C \to H$ for inference of X and Y).

Input encoding The input values are converted into deterministic spike trains with equally sized inter-spike intervals. The set of input spike times for a neuron in the interval $[t_{\mathrm{start}}, t_{\mathrm{stop}}]$ during which an example is presented is defined by:

$$\mathcal{T}_i = \{t_{\mathrm{in}} : \; t_{\mathrm{in}} = t_{\mathrm{start}} + a\frac{1}{r_i}, \; t_{\mathrm{in}} \leq t_{\mathrm{stop}}, \; a \in \mathbb{N}\}. \tag{4.36}$$

For the neurons in the IO-populations, the spike behavior in a time stepped simulation with time step Δt is therefore imposed as:

$$s_i(t) = \begin{cases} 1 & \text{if } \exists \; t_{\mathrm{in}} \in \mathcal{T}_i : \; t - \Delta t < t_{\mathrm{in}} \leq t \\ 0 & \text{otherwise} \end{cases} \tag{4.37}$$

This deterministic coding is used to facilitate the representation of the numbers, but our experiments show that the scheme also works well with a more stochastic type of coding, as long as the firing rate is representative of the pixel value. A typical firing rate pattern induced in a neuron population by this kind of coding can be seen in figure 4.4.

Note that we used an explicit time in the definition of the spike input pattern. For a static pattern, which is used for the demonstrations in this work, this discretization seems kind of artificial. It is used here to reflect the fact that in the general case, the input will be arriving from a sensor which produces a dynamic number of spike events which arrive at different points in time. Additionally, as argued in 4.4.1, this type of regular encoding leads to a lower number of spikes in the network.

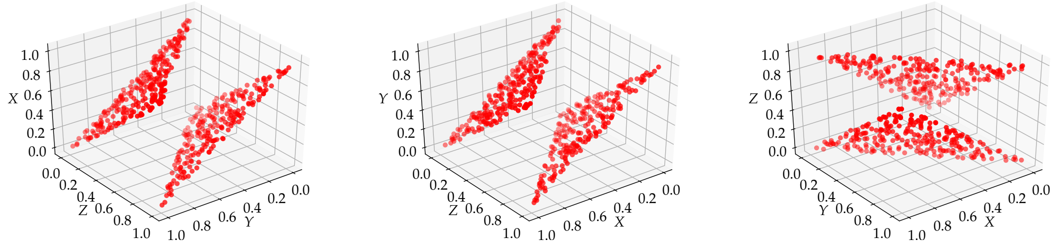

Figure 4.3: Relational inference for all three output variables. The two bottom axes describe the two input variables, while the vertical axis is the value inferred using (4.40). We can observe that the network has learned to represent all possible directions of the relation accurately. The root-mean-squared error (RMSE) averaged over the three populations is 0.0154

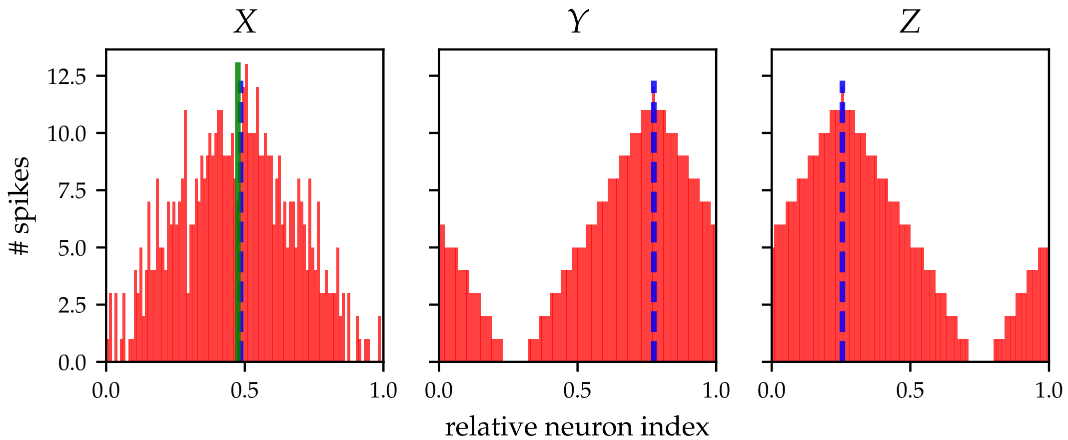

Figure 4.4: Activity profiles produced by the different populations during inference. Y and Z have activity profiles imposed to represent the two input variables, while X produces an activity profile based on this input, which presents the third number of the relation. The blue vertical lines represent the target numbers, the green line the number inferred by the population (based on (4.40)). The inferred profile of X is slightly noisier than the input firing rate profiles, but it presents accurately the desired profile (correct inferred value and overall shape).

If the IO-population is used as an output population during inference, this input encoding mechanism is disabled and the population receives only input from the corresponding peripheral population via the incoming synapses.

Implementing periodic addition

As a first test of the architecture, we implement an addition with periodic boundaries:

$$\gamma = \alpha + \beta - \lfloor \alpha + \beta \rfloor . \tag{4.38}$$

To implement relations between numbers, we use a similar approach as Diehl and Cook [2016] and encode each number as a firing rate profile of the input population. The numbers $\alpha, \beta, \gamma \in [0, 1]$ represent the numbers encoded by the IO-populations X, Y and Z. Each variable $\xi \in \{\alpha, \beta, \gamma\}$ is converted into a spike profile by assigning each of the N neurons of a IO-populations a constant firing rate based on its index i:

$$r_i(\xi) = \frac{r_{\max}}{N} \left| N - 2|N\xi - i| \right| . \tag{4.39}$$

These rates are converted into spike trains using the method outlined in section 4.3.1. We use an IO-population size of 100 neurons, which allows us to represent ξ up to a theoretical precision of ± 0.005.

Inference of the estimated value $\hat{\xi}$ is performed based on the firing pattern of the inferring population. For the visual inference task, the firing rate pattern of the inferring IO-population is directly taken (after a rescaling) as pixel values of the inferred image. For the mathematical relation, the inferred number $\hat{\xi} \in [0, 1]$ is derived from the firing rate pattern of the N neurons by finding the index of the neuron that minimizes the activity weighted distance to all other neurons in the same population:

$$\hat{\xi} = \frac{1}{N} \arg\min_i \sum_j x_j d_N[i, j] \tag{4.40}$$

using the periodic distance function:

$$d_N[i, j] := \begin{cases} |i - j| & \text{if } |i - j| \leq N/2 \\ N - |i - j| & \text{if } |i - j| > N/2 \end{cases} . \tag{4.41}$$

The response plots in figure 4.3 show that the network learns to accurately represent the relation. Each of the two input populations can be used to infer the value of the variable represented by the third population. The root-means-squared error (RMSE) is with 0.0154 considerably lower than the errors obtained by Diehl and Cook [2016] and Thiele et al. [2018a]. Additionally, the network can reproduce the approximate firing rate profile of the encoded input, since each IO-population was explicitly trained to do so (4.4). This means the network can in particular reproduce an output that has the same scale as the original input.

Visual XOR

We now apply the network to a more challenging task, which requires the network to find abstract representations of the input data before setting them in relation with each other. For this purpose, we let the network learn the

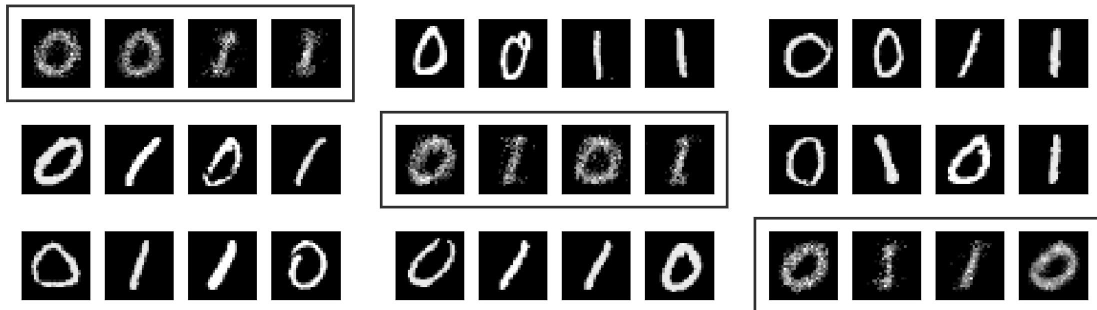

Figure 4.5: Visual relational inference for all three variables of relation (4.42). The inferred stimuli are marked by boxes. We can observe that the network has learned to represent all possible directions of relation (4.42) accurately and it is able to produce artificial stimuli that are similar to other stimuli of the dataset.

visual XOR task. The inputs X, Y, Z are now spike-encoded examples of the MNIST dataset representing the handwritten digits 0 and 1:

$$l(Z) = l(X) \oplus l(Y). \tag{4.42}$$

The function $l \in \{0, 1\}$ is the function that maps the images encoded by the IO-populations X, Y and Z to their corresponding labels. We chose in this case only examples of the dataset that represent the numbers 1 and 0, which will represent the boolean values true and false respectively. Note that this can also be seen as an addition mod 2 of integers with 0 and 1 as possible values and is therefore in a sense similar to the previous task.

The samples used for training are randomly selected images from the MNIST training set of 60000 digits (number 0 to 9), with the condition that they are consistent with the relation. This means that the number of possible samples is extremely large (in the order of $4 \cdot 6000^3$) and it is almost impossible for the network to memorize the training set. The inferences shown in figure 4.5 are still performed on images from the test set, and therefore demonstrate true generalization of the task.

For spike encoding of the images, each pixel will be assigned to a neuron of the IO-population and its firing rate is related to the assigned pixel value $p_i \in [0, 1]$ via a factor $r_{\max}$ by $r_i = p_i r_{\max}$. For images of size 28×28, this gives an IO-population size of 784.

In figure 4.5, it can be seen that the network has learned to replace all missing parts of the relation by an output that resembles an artificial stimulus. The network has therefore learned to set the abstract meaning of the images in relationship with each other.

Comparison to previous implementations of the relational network

The network of Diehl and Cook [2016] uses bio-inspired learning algorithms (different variants of STDP), which are applied on populations of inhibitory and excitatory leaky integrate-and-fire neurons. Learning in their implementation is not split into several phases where the roles of the IO-populations change, but all populations are treated equivalently during example presentation. This is possible because learning is solely based on correlated activity patterns of neurons. This leads however to a problem during inference: activity in the network tends to attenuate strongly, since the network was not trained on patterns where no input is provided to one of the populations. While this is not a problem for the rather simple inference of relations of numbers represented by the weighted mean of the output pattern, it might become a problem if the scale of the inferred pattern is relevant for inference, or if the network is very deep and activity dies out completely before it arrives at the inferring population. This problem is partially avoided by using a high number of neurons with self-regulating recurrent connectivity. The high level of recurrent connectivity can however lead to attractor states that are detrimental for learning. This problem can be solved with a wake-sleep type algorithm (Thiele et al. [2017b]). Nevertheless, the architecture still requires a high number of neurons and careful parameter tuning for good performance. Additionally, although the approach is bio-inspired, it is not necessarily easy to implement in neuromorphic hardware due to the complex nature of the STDP rules and several tricks that are used to stabilize learning (i.e. regular weight normalization).

Our algorithm is much simpler than the more biologically inspired approaches in the sense that it requires less parameters tuning. As it can be seen in table 4.2, the only network parameters that have to be tuned additionally compared to a standard ANN are the threshold values (which in our case are set to 1). In contrast to the STDP based approaches, our algorithm optimizes the network using gradient descent on an exact objective function (4.34). One limitation of our approach is that for training and inference, the network has to decide in advance which variables it wants to infer and disable the corresponding synapses (as visualized in figure 4.2). It therefore requires a kind of attention mechanism. This attention mechanism could for example be implemented by observing which subset of the populations receives the largest number of input spikes, and enabling all connections that belong to the corresponding inference direction.

As all implementations of backpropagation, our learning rule is non-local, in the sense that learning requires the presence of a feedback signal external to the neuron. However, this non-locality exists in any other architecture where the neural ensembles have to process external information which does not directly arrive at the neuron. This includes the STDP-based architecture

of Diehl and Cook [2016], where this information is implicitly communicated by the inter-population connections. Also in our approach, errors are communicated as spikes between populations, which allows us to see this external information simply as another form of special synaptic input that arrives at the neuron at a different time. As we have explained before, the temporal non-locality of *SpikeGrad*, which arises from the synchronization of forward and backward propagation, could be potentially lifted if approximate errors were allowed to propagate through the network during the forward pass. The general advantage of our approach is that spikes have a clear interpretation: they encode an approximation of the backpropagated error. This allows us to use the power of backpropagation while maintaining spike-based communication between populations. The main difference to biologically inspired architectures such as Diehl and Cook [2016] is that we require bidirectional and symmetric synapses if we want to represent the gradient accurately. However, as for other ANN implementations using the backpropagation algorithm, this condition might be lifted by using an approximation based on randomized weights, such as (direct) feedback alignment (Lillicrap et al. [2016]).

4.3.2 MNIST and CIFAR10 image classification

In this section, we apply the *SpikeGrad* algorithm on the standard machine learning task of image classification, similar as is chapter 3. For all experiments, the means, errors and maximal values are calculated over 20 simulation runs.

Experimental setup

Simulation All experiments are performed with custom CUDA/cuDNN accelerated C++ code based on the open source N2D2 deep learning framework. Training is performed on *Nvidia RTX 2080 Ti* graphic cards.

Training No preprocessing is used on the MNIST dataset. We separate the training set of size 60000 into 50000 training and 10000 validation examples, which are used to monitor convergence. Testing is performed on the test set of 10000 examples. For CIFAR10, the values of all color channels are divided by 255 and then rescaled by a factor of 20 to trigger sufficient activation in the network. The usual preprocessing and data augmentation is applied: images are padded with the image mean value by two pixels on each side and random slices of 32×32 are extracted. Additionally, images are flipped randomly along the vertical axis. We separate the training set of size 50000 into 40000 training and 10000 validation examples, which are used to monitor convergence. Testing is performed on the test set of 10000 examples. The hyperparameters for training can be seen in tables 4.3 and 4.4. The maximal

Parameter	Value
Epochs	60
Batch size	128
Θ_{ff}	1.0
Θ_{bp}	1.0
Base learning rate η	0.1
Momentum	0.9
Decay policy	mutliply by 0.1 every 20 epochs
Dropout (fc1 only)	0.5

Table 4.3: Parameters used for training on MNIST.

Parameter	Value
Epochs	300
Batch size	16
Θ_{ff}	1.0
Θ_{bp}	1.0
Base learning rate η	0.001
Momentum	0.9
Decay policy	multiply by 0.1 after 150 epochs
Dropout (all except pool and top)	0.2

Table 4.4: Parameters used for training on CIFAR10.

inference performances in the results were achieved with $\alpha = 100$ for MNIST and $\alpha = 400$ for CIFAR10. Final scores were obtained without retraining on the validation set.

Loss function and error scale We use the cross entropy loss function in the final layer applied to the Softmax of the total integrated signal $V_i^L(T)$ (no spikes are triggered in the top layer during inference). This requires more complex operations than accumulations, but is negligible if the number of classes is small. To make sure that sufficient error spikes are triggered in the top layer, and that error spikes arrive even in the lowest layer of the network, we apply a scaling factor α to the error values before transferring them to U_i^L. This scaling factor also implicitly sets the precision of the gradient, since a higher number of spikes means that a large range of values can be represented. To counteract the relative increase of the gradient scale, the learning rates have to be rescaled by a factor $1/\alpha$.

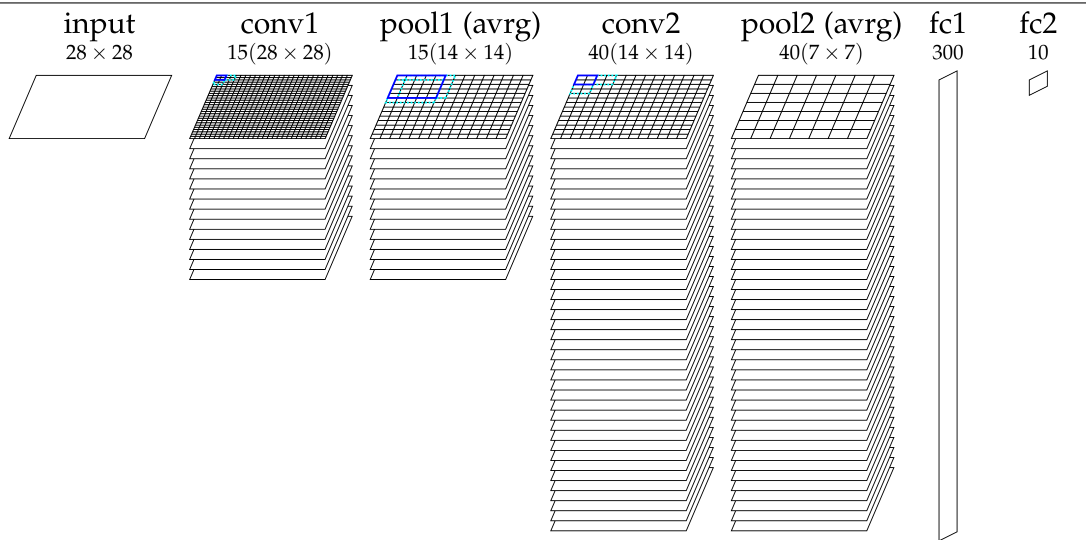

Figure 4.6: Network topology used for the MNIST dataset (28x28-15C5-P2-40C5-P2-300-10 in short notation).

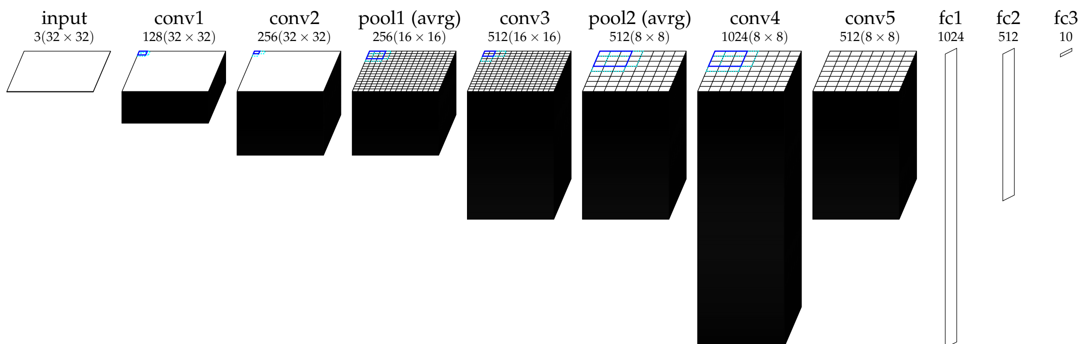

Figure 4.7: Network topology used for the CIFAR10 dataset (32x32-128C3-256C3-P2-512C3-P2-1024C3-512C3-1024-512-10 in short notation).

Input encoding As pointed out by Rueckauer et al. [2017] and Wu et al. [2018b], it is crucial to maintain the full precision of the input image to obtain good performances on complex standard benchmarks with SNNs. One possibility is to encode the input in a large number of spikes (such as Sengupta et al. [2019]). Another possibility, which has been shown to require a much lower number of spikes in the network, is to multiply the input values directly with the weights of the first layer (just like in a standard ANN). The drawback is that the first layer then requires multiplication operations. The additional cost of this procedure may however be negligible if all other layers can profit from spike-based computation. This problem does not exist for stimuli that are natively encoded in spikes.

Architecture	Method	Rec. Rate (**max**[mean±std])
Rueckauer et al. [2017]	CNN converted to SNN	99.44%
Wu et al. [2018a]*	SNN training float BP	99.42%
Jin et al. [2018]*	Macro/Micro BP	99.49%
This work*	SNN training float BP	**99.48**[99.36 ± 0.06]%
This work*	SNN training spike BP	**99.52**[99.38 ± 0.06]%

Table 4.5: Comparison of different state-of-the-art spiking CNN architectures on MNIST. * indicates that the same topology (28x28-15C5-P2-40C5-P2-300-10) was used.

Architecture	Method	Rec. Rate (**max**[mean±std])
Rueckauer et al. [2017]	CNN converted SNN (with BatchNorm)	90.85%
Wu et al. [2018b]*	SNN training float BP (no NeuNorm)	89.32%
Sengupta et al. [2019]	VGG-16 converted to SNN	91.55%
This work*	SNN training float BP	**89.72**[89.38 ± 0.25]%
This work*	SNN training spike BP	**89.99**[89.49 ± 0.28]%

Table 4.6: Comparison of different state-of-the-art spiking CNN architectures on CIFAR10. * indicates that the same topology (32x32-128C3-256C3-P2-512C3-P2-1024C3-512C3-1024-512-10) was used.

Classification performance

Tables 4.5 and 4.6 compare the state-of-the-art results for SNNs on the MNIST and CIFAR10 datasets. It can be seen that in both cases our results are competitive with respect to the state-of-the-art results of other SNNs trained with high precision gradients. Compared to results using the same topology, our algorithm performs at least equivalently.

The final classification performance of the network as a function of the error scaling term α in the final layer can be seen in figure 4.8. Previous work on low bit-width gradients by of Zhou et al. [2018] found that gradients usually require a higher precision than both weights and activations. Our results also seem to indicate that a certain minimum number of error spikes is necessary to achieve convergence. This strongly depends on the depth of the network and if enough spikes are triggered to provide sufficient gradient signal in the bottom layers. For the CIFAR10 network, convergence becomes unstable for approximately $\alpha < 300$. As soon as the number of operations is large enough for convergence, the required precision for the gradient does not seem to be extremely high. On the MNIST task, the difference in test performance between a gradient rescaled by a factor of 50 and a gradient rescaled by a factor of 100 becomes insignificant. In the CIFAR10 task, this is true for a rescaling by 400 or 500. Also the results obtained with the float precision gradients in tables 4.5 and 4.6 demonstrate the same performance, given the range of the error.

Sparsity in backpropagated gradients

To evaluate the potential efficiency of the spike coding scheme relative to an ANN, we use the metric of *relative synaptic operations*. A synaptic operation corresponds to a multiply-accumulate (MAC) in the case of an ANN, and a simple accumulation (ACC) in the case of an SNN. This metric allows us to compare networks based on their fundamental operation. The advantage of this metric is the fact that it does not depend on the exact implementation of the operations (for instance the number of bits used to represent each number). Since an ACC is however generally cheaper and easier to implement than a MAC, we can be sure that an SNN is more efficient in terms of its operations than the corresponding ANN if the number of ACCs is smaller than the number of MACs.

In figure 4.8 it can be seen that the number of operations (which is indicative of the number of spikes) decreases with increasing inference precision of the network. This is a result of the decrease of error in the classification layer, which leads to the emission of a smaller number of error spikes. Numbers were obtained with the integer activations of the equivalent ANN to keep simulation times tractable. As explained in 4.4.1, the integer response of the equivalent ANN represents the best case scenario for the SNN, with the lowest number of spikes. The number of events and synaptic operations in an actual SNN may therefore slightly deviate from these numbers. Figure 4.9 demonstrates how this minimal number of operations during the backpropagation phase is distributed in the layers of the network (float precision input layer and average pooling layers were omitted). While propagating deeper into the network, the relative number of operations decreases and the error becomes increasingly sparse. This tendency is consistent during the whole training process for different epochs.

4.4 Discussion

4.4.1 Computational complexity estimation

Note that we have only demonstrated the equivalence of the accumulated neurons responses. However, for each of the response values, there is a large number of possible combinations of 1 and -1 values that lead to the same response. The computational complexity of the event-based algorithm depends therefore on the total number n of these events. The best possible case is when the accumulated response value S_i^l is represented by exactly $|S_i^l|$ spikes. In the worst case, a large number of additional redundant spikes is emitted which sum up to 0. The maximal number of spikes in each layer is bounded by the largest possible integration value that can be obtained. This depends on the maximal weight value w_{max}^l, the number of connections N_{in}^l and the number

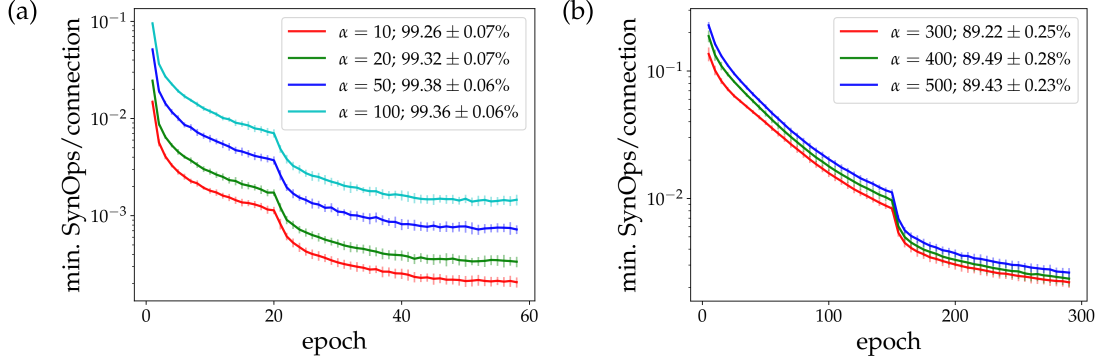

Figure 4.8: Minimal number of relative synaptic operations during backpropagation for different error scaling factors α as a function of the epoch. Numbers are based on activation values of the equivalent ANN. Test performance with error is given for each α. (a) MNIST (learning rate decay at epoch 20). (b) CIFAR10 (learning rate decay at epoch 150).

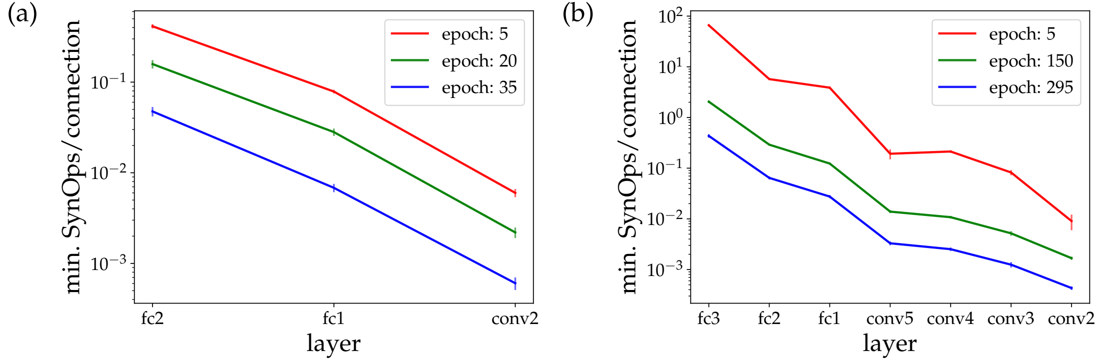

Figure 4.9: Minimal number of relative synaptic operations during backpropagation in each layer (connections in direction of backpropagation) for different epochs. (a) MNIST with $\alpha = 100$. (b) CIFAR10 with $\alpha = 500$.

of spike events n^{l-1} each connection receives, which is given by the maximal value of the previous layer (or the input in the first layer):

$$n^l_{\text{min}} = |S^l_i|, \quad n^l_{\text{max}} = \left\lfloor \frac{1}{\Theta_{\text{ff}}} N^l_{\text{in}} w^l_{\text{max}} n^{l-1}_{\text{max}} \right\rfloor . \tag{4.43}$$

The same reasoning applies to backpropagation. Our experiments show that for input encodings where the input is provided as a constant rate, and weight values that are on average much smaller than the threshold value, the deviation from the best case scenario is rather small. This is because in this case the sub-threshold integration allows to average out the fluctuations in the signal. This way, the firing rate stays rather close to its long term average and few redundant spikes are emitted. For the total number of spikes n in the full network on the CIFAR10 test set, we obtain empirically $(n - n_{\text{min}})/n_{\text{min}} < 0.035$.

4.4.2 Hardware considerations

Several implementation possibilities exist depending on the desired substrate and the hardware constraints. For both forward and backward propagation, *SpikeGrad* requires a similar communication infrastructure between neurons, which facilitates a spiking hardware implementation. Another major advantage of our implementation is that no explicit spike time has to be communicated. This removes the need to propagate and save timestamps. The spike signals can directly be propagated as events that encode the sign in a 1 bit variable. A digital implementation can additionally profit from the fact that all required operations are either accumulations or comparisons. Since the only information required about V during backpropagation is included in the spike activation function derivative, the two integrators V and U used in the description of the backpropagation algorithm could even be represented by the same integration variable. In the case of an analog implementation, the neuron and synapse models probably have to be adapted to reflect more accurately the dynamics of analog neurons. The main restriction of *SpikeGrad* is the need for negative spikes, which could be problematic depending on the particular hardware that is used.

4.4.3 Conclusion and future outlook

In this chapter, we introduced *SpikeGrad*, a neuromorphic version of the backpropagation algorithm. As a first application scenario, we showed that it can be used to train a network for relational inference. Our results show that this implementation can be advantageous compared to previous STDP-based implementations in several aspects. Additionally, we showed that our network is able to learn a visual XOR task based on images of handwritten digits. This could make our approach promising for low power mobile platforms, where several sensor outputs have to be processed and set into relationship with each other. In the work presented here, we focused on relations of stimuli of the same type, but in principle our network could be extended to merge stimuli of different nature, such as visual, audio or numeric stimuli. In future work, we would like to investigate if our approach can be scaled to more complex relationships between stimuli. Additionally, it would be interesting to adapt the algorithm to analog or mixed signal neuromorphic implementations (such as Moradi et al. [2018]), which would allow processing even closer to the sensor.

Using an image classification task, we demonstrated that competitive inference performance can be achieved with spiking networks trained with the *SpikeGrad* algorithm. Additionally, gradient backpropagation seems particularly suitable to leverage spike-based processing by exploiting high signal sparsity. In particular, the topology used for CIFAR10 classification is how-

ever rather large for the given task. We decided to use the same topologies as the state-of-the-art to allow for better comparison, but the same task could probably also be performed by a smaller network. In an ANN implementation, it is in general undesirable to use a network with a large number of parameters, since it increases the need for memory and computation. The relatively large number of parameters of the model may, to a certain extent, explain the very low number of relative synaptic operations that we observed during backpropagation. In an SNN, a large number of parameters is less problematic from a computational perspective, since only the neurons which are activated by input spikes will trigger computations. A large portion of the network will therefore remain inactive. It would still be interesting to investigate signal sparsity and performance of *SpikeGrad* in ANN topologies that were explicitly designed for minimal computation and memory requirements.

Chapter 5

Summary and Future Outlook

This chapter summarizes the results of our experiments and discusses possible future developments in the field of deep spiking networks.

5.1 Summary of our results

In this thesis, we presented two principal solutions for on-chip learning in spiking neural networks. Both were designed considering the same basic constraints of neuromorphic systems. However, due to the different application scenarios for which they were envisioned, different choices were made regarding the learning algorithm. The STDP-based system was designed from the viewpoint of full spatial and temporal locality. Additionally, feature learning should be performed only based on the event-based input, without the need for a labeled dataset. This can only be realized by the use of an unsupervised algorithm. By using a simplified STDP rule, we were able to satisfy these constraints. Our approach is in particular suitable for data that is encoded in relative firing rate values (e.g. spatial pixel correlations), but whose information content is independent of the timescale. However, our work also demonstrated first-hand the typical problems and limitations that are encountered using this type of bio-inspired learning rules. Due to its unsupervised and local nature, STDP is not able to jointly optimize all layers of the network for an explicit global learning objective, which eventually negatively affects classification performance.

We therefore shifted our focus to optimization methods that are closer to training algorithms for deep networks that are used in machine learning. In this context, we presented *SpikeGrad*, an event-based version of the backpropagation algorithm, which is adapted to be compatible with the typical hardware constraints of neuromorphic systems. Our results demonstrate that this mechanism is capable of exploiting the high sparsity of the gradient during backpropagation by using the dynamic precision properties of spiking

neurons. A spike coding solution therefore seems particularly suitable for the gradient. The algorithm was tested in two different application scenarios: multi-directional inference of numeric and visual relations, and a typical image classification task. The latter case demonstrated that the algorithm is able to scale to large-scale convolutional network structures, yielding classification accuracy as good as methods using real valued gradients (in the range of statistical error). *SpikeGrad* delivers two main contributions to the research community: firstly, it shows that spikes can be used as an effective means to train large convolutional neural networks. Secondly, it greatly facilitates this training by the introduction of an equivalent ANN, that can be trained much faster than an SNN whose spike dynamics are simulated explicitly. These contributions enable us to investigate the scaling of SNN performance on large scale problems, which is crucial for the transfer of SNNs from research to practical applications.

Due to the extensive computational resources that are necessary to train large scale SNNs, we were not yet able to test the performance of our learning mechanisms on more complex datasets than CIFAR10. Even *SpikeGrad*, which has in principle the same computational complexity for training as the equivalent ANN, would potentially take over a week for training on the ImageNet benchmark. Given that we cannot use the same hyperparameters that are used in standard ANNs, optimizing performance on such a benchmark task would potentially take several weeks or months to yield sufficient accuracy. Performing such an extensive optimization procedure was therefore out of the scope of this work. Investigating the scaling properties of *SpikeGrad* on this type of datasets represents however a crucial future step in verifying the viability of our approach for solving practical machine learning problems. The fact that *SpikeGrad* can train an SNN in equivalent time as a comparable ANN makes such a simulation at least possible in principle, which is not the case for most other fully event-based on-chip algorithms.

5.2 The future of SNNs

5.2.1 Training by mapping SNNs to ANNs

This paradigm shift from bio-inspired learning rules towards adaptation of machine learning algorithms to SNNs seems to be indeed the general tendency that could be observed in the field in recent years (see figure 5.1). Also our work demonstrates that in this way it is possible to solve more complex problems than by using rather heuristic STDP rules. We believe that the best way to achieve high performance on SNNs is to try to map them to structures that can be optimized with mathematically well-defined optimization algorithms. This can either be done by explicitly simulating the SNN dynamics, or by optimizing an ANN that represents the SNN.

Our work is in line with previous research that analyses how an exact ANN equivalent can be found for specific types of SNN models (Binas et al. [2016] and Wu et al. [2019a]). This presents an extension to the traditional approach that converts ANNs to SNNs. While the traditional conversion approach uses an ANN that was trained without knowledge of the spike coding, these novel approaches allow optimization of an ANN that explicitly takes into account the constrained representation capacities of the SNN (e.g. integer activation values). This way, a performance loss can be avoided when the parameters of the equivalent ANN are used in an SNN implementation.

The problems considered in this thesis did not possess temporal dynamics, or these were not explicitly exploited by our algorithms. We however believe that also for sequential, temporal data, finding a mapping between an RNN and an SNN (as described in Wu et al. [2018a], Neftci et al. [2019] or Bellec et al. [2018]) is probably the most promising approach for effective training.

In cases where an exact mapping between SNNs and ANNs is not possible, approximations may still be necessary. We think however that the most promising approach is to start from a model that is as close as possible to a known ANN description and then analyze possible approximations that are necessary to make the mechanism compatible with SNN constraints. This way it can be easier to quantify the approximation errors that are introduced by the SNN realization.

5.2.2 On-chip vs. off-chip optimization

While this thesis has a focus on on-chip learning algorithms for SNNs, the future development of on-chip learning applications is difficult to anticipate. In a cluster-based, high-performance computing environment, an energy efficient implementation of the backpropagation algorithm is in principle a desirable objective, since the energy cost for training large deep learning models can be tremendous (Strubell et al. [2019]). In this situation, SNNs have the advantage that activations and errors in deep networks become increasingly sparse with the number of (back)propagation levels. Since the systems used for training are usually designed for very high data loads and continuous use, SNNs might however not be able to profit fully from temporal sparsity. It is questionable if, in these cases, activation sparsity is high enough to set off the disadvantages which arise from the higher hardware complexity of SNNs.

Due to their event-based nature, SNNs are more suitable for applications where relevant inputs are observed rarely and the system remains inactive most of the time. This makes them predestined candidates for autonomous systems. Online learning, that can be performed on-chip, seems indeed very interesting from an artificial intelligence perspective, since most natural systems learn by continuous observation of their environment. However, we think that there are many potential caveats that currently prevent online learn-

ing from being used in practical industry applications. The additional degree of autonomy provided by online learning is not actually desired in most modern technology. Even most "autonomous" systems require a rather predictable response. The response of an intelligent system depends however strongly on the training process and the data that is used. Online learning could lead to unexpected responses that can represent a security risk that is difficult to anticipate. It is therefore more practical for most applications to disable learning in the system during operation, and update the system occasionally and under strictly controlled conditions.

These are the reasons why we believe that the short-term future of SNNs lies more likely in off-chip optimization with subsequent hardware deployment. ANN topologies that are optimized for resource constrained environments, such as MobileNets by Howard et al. [2017], already play a huge role in designing AI applications for low power chips, such as the *Google Edge TPU* or the *Nvidia Jetson Nano*. SNNs are likely to become a part of these specialized solutions. While SNNs will probably not generally replace ANNs in all application scenarios, they could be interesting for problems that have high temporal and spatial sparsity. This could be for instance the detection of rare events with an event-based vision systems, or event-based speech recognition. Although the main objective of the *SpikeGrad* algorithm was to allow efficient training of SNNs, it would be also possible to only code the forward pass in spikes and optimize the network off-chip with a full precision gradient. This would pose less constraints on the layer types and cost functions that can be used, and therefore potentially allow better optimization of the SNN for efficient inference.

5.2.3 Hardware complexity as a major bottleneck of SNNs

The main efficiency argument for digital SNNs is that the complex multiplication operation can be replaced by a sequence of event-based additions. This permits to scale computation in the network dependent on spatial and temporal sparsity, and allocate computing resources dynamically to active parts of the network. As mentioned in 2.2.3, this relies however on the assumption that certain hardware operations and the allocation mechanism can be implemented efficiently. A particular problem of SNNs is the large quantity of memory that is required to store the state variables, and the large number of memory accesses that are necessary to update these variables with each event. This is particularly true for spike coding mechanism that are based on the number of spikes. Also the dynamic allocation of computing resources could be a problem in digital hardware if events arrive in a highly irregular fashion.

We believe that the main bottleneck in bringing SNNs to practical applications is the lack of appropriate hardware. In principle, the field now knows

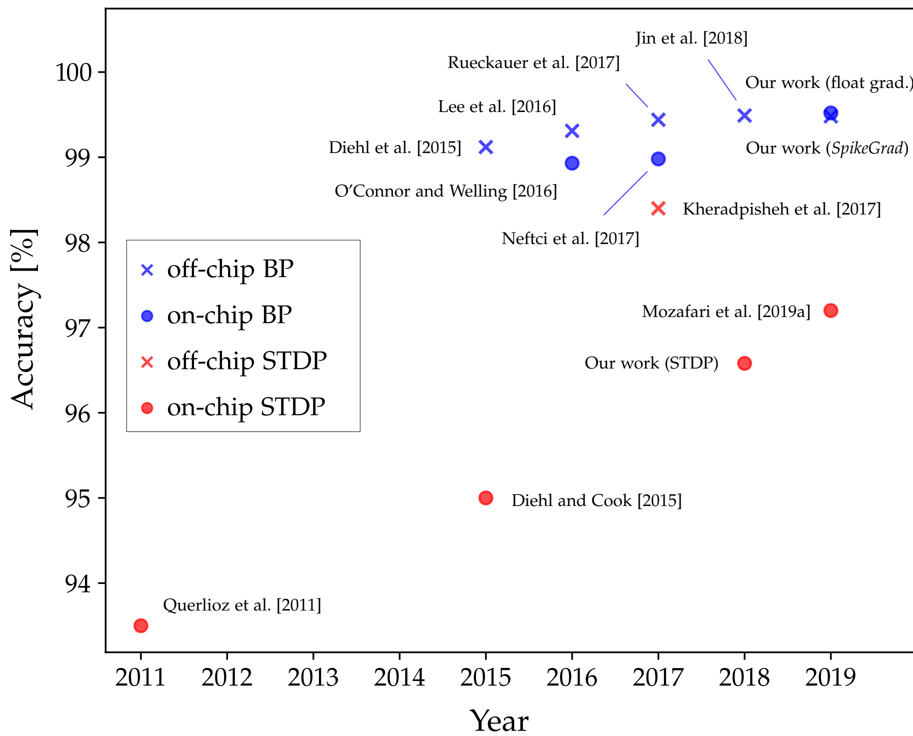

Figure 5.1: Development of the state of the art inference accuracies on the MNIST dataset. It can be seen that early works mostly used STDP. While there is still research on improving the performance of SNNs trained with STDP, the best results in recent years are provided by methods that use approximations of the backpropagation algorithm. Our work (*SpikeGrad*) represents an on-chip learning method that yields accuracies comparable to off-chip training methods. Please note that this presentation is simplified and only shows the general tendencies that are observable in field. In particular, we do not take into account energy efficiency or slightly varying hardware constraints, objectives and methods. Kheradpisheh et al. [2017] is counted as an off-chip method due to the use of an SVM classifier in the final layer.

a sufficient number of algorithms that are able to train SNNs effectively for simple applications. It remains to show that a dedicated hardware can be built for such an application that allows to be more efficient than a standard ANN solution, while maintaining the same performance. This requires an efficient spike coding mechanism and a specialized hardware that is optimized exactly for this type of encoding. Most SNN hardware is still in a research or prototype stage and generally not as optimized as hardware for traditional ANNs. Additionally, the necessity to co-design hardware and algorithms requires a large amount of specialized knowledge, which increases the difficulty of building systems for productive industrial environments. We believe that this complexity is the main obstacle that currently prevents SNNs from being used in practical applications.

Bibliography

Alessandro Aimar, Hesham Mostafa, Enrico Calabrese, Antonio Rios-Navarro, Ricardo Tapiador-Morales, Iulia-Alexandra Lungu, Moritz B. Milde, Federico Corradi, Alejandro Linares-Barranco, Shih-Chii Liu, and Tobi Delbrück. Nullhop: A flexible convolutional neural network accelerator based on sparse representations of feature maps. *IEEE Trans. Neural Netw. Learning Syst.*, 30(3):644–656, 2019.

Pierre Baldi and Peter Sadowski. A theory of local learning, the learning channel, and the optimality of backpropagation. *Neural Networks*, 38:51–74, 2016.

Pierre Baldi, Peter Sadowski, and Zhiqin Lu. Learning in the machine: The symmetries of the deep learning channel. *Neural Networks*, 95:110 – 133, 2017.

Pierre Baldi, Peter Sadowski, and Zhiqin Lu. Learning in the machine: Random backpropagation and the deep learning channel. *Artificial Intelligence*, 260:1 – 35, 2018.

Sergey Bartunov, Adam Santoro, Blake A. Richards, Geoffrey E. Hinton, and Timothy P. Lillicrap. Assessing the Scalability of Biologically-Motivated Deep Learning Algorithms and Architectures. In *Advances in Neural Information Processing Systems (NIPS)*, 2018.

Mark F. Bear, Barry W. Connors, and Michael A. Paradiso. *Neuroscience: exploring the brain*. Lippincott Williams & Wilkins, 3 edition, 2007.

Guillaume Bellec, Darjan Salaj, Anand Subramoney, Robert Legenstein, and Wolfgang Maass. Long short-term memory and learning-to-learn in networks of spiking neurons. In *Advances in Neural Information Processing Systems (NIPS)*, 2018.

Yoshua Bengio. Estimating or propagating gradients through stochastic neurons. *arXiv:1308.3432v1*, 2013.

Guo-qiang Bi and Mu-ming Poo. Synaptic Modifications in Cultured Hippocampal Neurons : Dependence on Spike Timing , Synaptic Strength , and Postsynaptic Cell Type. *The Journal of Neuroscience*, 18(24):10464–10472, 1998.

Olivier Bichler, Damien Querlioz, Simon J Thorpe, Jean-philippe Bourgoin, and Christian Gamrat. Unsupervised Features Extraction from Asynchronous Silicon Retina through Spike-Timing-Dependent Plasticity. In *International Joint Conference on Neural Networks (IJCNN)*, pages 859–866, 2011.

Olivier Bichler, David Briand, Victor Gacoin, Benjamin Bertelone, Thibault Allenet, and Johannes C. Thiele. N2D2 - Neural Network Design & Deployment. *Manual available on Github*, 2019.

Jonathan Binas, Giacomo Indiveri, and Michael Pfeiffer. Deep counter networks for asynchronous event-based processing. *arXiv:1611.00710v1, NIPS 2016 workshop "Computing with Spikes"*, 2016.

Tom Binzegger, Rodney J. Douglas, and Kevan A. C. Martin. A Quantitative Map of the Circuit of Cat Primary Visual Cortex. *The Journal of Neuroscience*, 24(39):8441–8453, 2004.

Sander Bohte, Joost N. Kok, and Johannes A. La Poutré. Spikeprop: backpropagation for networks of spiking neurons. In *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*, volume 48, pages 419–424, 01 2000.

Léon Bottou and Yann LeCun. Large scale online learning. In *Advances in Neural Information Processing Systems (NIPS)*, pages 217–224, 2004.

Maxence Bouvier, Alexandre Valentian, Thomas Mesquida, Francois Rummens, Marina Reyboz, Elisa Vianello, and Edith Beigne. Spiking neural networks hardware implementations and challenges: A survey. *J. Emerg. Technol. Comput. Syst.*, 15(2):22:1–22:35, April 2019.

Yongqiang Cao, Yang Chen, and Deepak Khosla. Spiking deep convolutional neural networks for energy-efficient object recognition. *International Journal of Computer Vision*, 113(1):54–66, May 2015.

Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cudnn: Efficient primitives for deep learning. *CoRR*, abs/1410.0759, 2014.

Elisabetta Chicca, Fabio Stefanini, Chiara Bartolozzi, and Giacomo Indiveri. Neuromorphic Electronic Circuits for Building Autonomous Cognitive Systems. *Proceedings of the IEEE*, 102(9):1367–1388, 2014.

Gregory K. Cohen, Garrick Orchard, Sio-Hoi Leng, Jonathan Tapson, Ryad B. Benosman, and André van Schaik. Skimming Digits: Neuromorphic Classification of Spike-Encoded Images. *Frontiers in Neuroscience*, 28 April 2016.

cuBLAS. https://docs.nvidia.com/cuda/cublas/index.html. version 10.1.168.

cuDNN. https://docs.nvidia.com/deeplearning/sdk/cudnn-developer-guide/index.html. version 7.6.0.

M. Davies, N. Srinivasa, T. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y. Weng, A. Wild, Y. Yang, and H. Wang. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, 38(1):82–99, January 2018.

Peter Dayan and L. F. Abbott. *Theoretical Neuroscience*. MIT Press, 2001.

S. Deneve, P.E. Latham, and A. Pouget. Efficient computation and cue integration with noisy population codes. *Nature Neuroscience*, 4(8):826–831, 2001.

Peter U. Diehl and Matthew Cook. Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Frontiers in Computational Neuroscience*, 9:Article 99, August 2015.

Peter U. Diehl and Matthew Cook. Learning and Inferring Relations in Cortical Networks. *arXiv:1608.08267v1*, 2016.

Peter U. Diehl, Daniel Neil, Jonathan Binas, Matthew Cook, Shih-Chii Liu, and Michael Pfeiffer. Fast-Classifying, High-Accuracy Spiking Deep Networks Through Weight and Threshold Balancing. In *International Joint Conference on Neural Networks (IJCNN)*, 2015.

Chris Eliasmith, Terrence C. Stewart, Xuan Choo, Trevor Bekolay, Travis DeWolf, Yichuan Tang, and Daniel Rasmussen. A Large-Scale Model of the Functioning Brain. *Science*, 338(6111):1202–1205, 2012.

Steven K. Esser, Paul A. Merolla, John V. Arthur, Andrew S. Cassidy, Rathinakumar Appuswamy, Alexander Andreopoulos, David J. Berg, Jeffrey L. McKinstry, Timothy Melano, Davis R. Barch, Carmelo di Nolfo, Pallab Datta, Arnon Amir, Brian Taba, Myron D. Flickner, and Dharmendra S. Modha. Convolutional networks for fast, energy-efficient neuromorphic computing. *Proceedings of the National Academy of Sciences*, 113(41):11441–11446, 2016.

Daniel J. Felleman and David C. Van Essen. Distributed Hierarchical Processing in the Primate Cerebral Cortex. *Cerebral Cortex*, 91(1047-3211):1:1–47, 1991.

Charlotte Frenkel, Martin Lefebvre, Jean-Didier Legat, and David Bol. A 0.086-mm^2 12.7-pj/sop 64k-synapse 256-neuron online-learning digital spiking neuromorphic processor in 28nm cmos. *IEEE Transactions on Biomedical Circuits and Systems*, 2018. doi: 10.1109/TBCAS.2018.2880425.

Stefan Habenschuss, Johannes Bill, and Bernhard Nessler. Homeostatic plasticity in bayesian spiking networks as expectation maximization with posterior constraints. In *Advances in Neural Information Processing Systems (NIPS)*, pages 773–781. 2012.

Stefan Habenschuss, Stefan Puhr, and Wolfgang Maass. Emergence of optimal decoding of population codes through stdp. *Neural Computation*, 25:1371–1407, 2013.

Hananel Hazan, Daniel J. Saunders, Hassaan Khan, Devdhar Patel, Darpan T. Sanghavi, Hava T. Siegelmann, and Robert Kozma. Bindsnet: A machine learning-oriented spiking neural networks library in python. *Frontiers in Neuroinformatics*, 12, 2018.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *Proceedings of the International Conference on Computer Vision*, pages 1026–1034, 2015.

Donald Olding Hebb. *The organization of behavior; a neuropsychological theory*. Wiley, Oxford, England, 1949.

M. Horowitz. 1.1 computing's energy problem (and what we can do about it). In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pages 10–14, Feb 2014.

Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv:1704.04861*, 2017.

Yanping Huang, Youlong Cheng, Ankur Bapna, Firat Orhan, Mia Xu Chen, Dehao Chen, Hyouk Joong Lee, Jiquan Ngiam, Quoc V. Le, Yonghui Wu, and Zhifeng Chen. GPipe: Easy Scaling with Micro-Batch Pipeline Parallelism. *arXiv:1811.06965v5*, 2019.

Dongsung Huh and Terrence J Sejnowski. Gradient descent for spiking neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1433–1443, 2018.

Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv:1502.03167v3*, 2015.

Laxmi R. Iyer, Yansong Chua, and Haizhou Li. Is Neuromorphic MNIST neuromorphic? Analyzing the discriminative power of neuromorphic datasets in the time domain. *arXiv:1807.01013v1*, 2018.

Benoit Jacob, Kligys Skirmantas, Chen Bo, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. *arXiv:1712.05877v1*, 2017.

Yingyezhe Jin and Peng Li. AP-STDP: A Novel Self-Organizing Mechanism for Efficient Reservoir Computing. In *International Joint Conference on Neural Networks (IJCNN)*, pages 1158–1165, 2016.

Yingyezhe Jin, Yu Liu, and Peng Li. SSO-LSM: A Sparse and Self-Organizing Architecture for Liquid State Machine based Neural Processors. In *IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, pages 55–60, 2016.

Yingyezhe Jin, Peng Li, and Wenrui Zhang. Hybrid Macro/Micro Level Backpropagation for Training Deep Spiking Neural Networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 7005–7015, 2018.

Jaques Kaiser, Hesham Mostafa, and Emre O. Neftci. Synaptic Plasticity Dynamics for Deep Continuous Local Learning. *arXiv:1811.10766v1*, 2018.

Saeed Reza Kheradpisheh, Mohammad Ganjtabesh, and Timothée Masquelier. Bio-inspired unsupervised learning of visual features leads to robust invariant object recognition. *Neurocomputing*, 205:382–392, 2016.

Saeed Reza Kheradpisheh, Mohammad Ganjtabesh, Simon J. Thorpe, and Timothée Masquelier. STDP-based spiking deep convolutional neural networks for object recognition. *Neural Networks*, 2017.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, November 1998.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521:436–444, 28 May 2015.

Jun Haeng Lee, Tobi Delbruck, and Michael Pfeiffer. Training Deep Spiking Neural Networks Using Backpropagation. *Frontiers in Neuroscience*, (10:508), 2016.

Patrick Lichtsteiner, Christoph Posch, and Tobi Delbruck. A 128 $\times$ 128 120 dB 15 μs Latency Asynchronous Temporal Contrast Vision Sensor. In *IEEE Journal of Solid-State Circuits, Seattle, WA, USA*, volume 43, pages 566–576, 2008.

Timothy P. Lillicrap, Daniel Cownden, Douglas B. Tweed, and Colin J. Akerman. Random synaptic feedback weights support error backpropagation for deep learning. *Nature Communications*, 2016.

Tao Liu, Liu Zihao, Lin Fuhong, Yier Jin, Gang Quan, and Wujie Wen. MT-Spike: A Multilayer Time-based Spiking Neuromorphic Architecture with Temporal Error Backpropagation. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD), Irvine, CA, USA*, pages 450–457, 2017.

Dominik Marek Loroch, Franz-Josef Pfreundt, Norbert Wehn, and Janis Keuper. Sparsity in Deep Neural Networks - An Empirical Investigation with TensorQuant. *arXiv:1808.08784v1*, 2018.

Vincent Lorrain. *Study and design of an innovative chip leveraging the characteristics of resistive memory technologies*. PhD thesis, Université Paris-Saclay, January 2018.

M. Lukoševičius and H. Jaeger. Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, 3(3):127–149, 2009.

Wolfgang Maass. Networks of Spiking Neurons: The Third Generation of Neural Network Models. *Electronic Colloquium on Computational Complexity*, (9):1659–1671, 1997.

Carver Mead. Neuromorphic electronic systems. *Proceedings of the IEEE*, 78 (10):1629–1636, Oct 1990.

Paul A. Merolla, John V. Arthur, Rodrigo Alvarez-Icaza, Andrew S. Cassidy, Jun Sawada, Filipp Akopyan, Bryan L. Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, Bernard Brezzo, Ivan Vo, Steven K. Esser, Rathinakumar Appuswamy, Brian Taba, Arnon Amir, Myron D. Flickner, William P. Risk, Rajit Manohar, and Dharmendra S. Modha. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673, 2014.

Saber Moradi, Ning Qiao, Fabio Stefanini, and Giacomo Indiveri. A scalable multi-core architecture with heterogeneous memory structures for Dynamic

Neuromorphic Asynchronous Processors (DYNAPs). In *IEEE Transactions on Biomedical Circuits and Systems*, volume 12, pages 1–15, 2018.

Hesham Mostafa. Supervised Learning Based on Temporal Coding in Spiking Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*, (99):1–9, 2017.

Hesham Mostafa, Bruno U. Pedroni, Sadique Sheik, and Gert Cauwenberghs. Fast Classification Using Sparsely Active Spiking Networks. In *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2017.

Milad Mozafari, Mohammad Ganjtabesh, Abbas Nowzari-Dalini, Simon J. Thorpe, and Timothée Masquelier. Bio-inspired digit recognition using reward-modulated spike-timing-dependent plasticity in deep convolutional networks. *to appear in: Pattern Recognition*, 2019a.

Milad Mozafari, Saeed Reza Kheradpisheh, Timothée Masquelier, Abbas Nowzari-Dalini, and Mohammad Ganjtabesh. SpykeTorch: Efficient Simulation of Convolutional Spiking Neural Networks with at most one Spike per Neuron. 2019b.

Manu V. Nair, Lorenz K. Muller, and Giacomo Indiveri. A differential memristive synapse circuit for on-line learning in neuromorphic computing systems. *arXiv:1709.05484v1*, 2017.

A. Neckar, S. Fok, B. V. Benjamin, T. C. Stewart, N. N. Oza, A. R. Voelker, C. Eliasmith, R. Manohar, and K. Boahen. Braindrop: A mixed-signal neuromorphic architecture with a dynamical systems-based programming model. *Proceedings of the IEEE*, 107(1):144–164, Jan 2019.

Emre O. Neftci, Srinjoy Das, Bruno Pedroni, Kenneth Kreutz-Delgado, and Gert Cauwenberghs. Event-driven contrastive divergence for spiking neuromorphic systems. *Frontiers in Neuroscience*, 7:272, 2014.

Emre O. Neftci, Charles Augustine, Paul Somnath, and Georgios Detorakis. Event-Driven Random Backpropagation: Enabling Neuromorphic Deep Learning Machines. *Frontiers in Neuroscience*, 11(324), 2017.

Emre O. Neftci, Charles Augustine, Paul Somnath, and Georgios Detorakis. Surrogate Gradient Learning in Spiking Neural Networks. *arXiv:1901.09948v1*, 2019.

Daniel Neil and Shih-Chii Liu. Effective sensor fusion with event-based sensors and deep network architectures. In *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 2282–2285, May 2016.

Bernhard Nessler, Michael Pfeiffer, Lars Buesing, and Wolfgang Maass. Bayesian Computation Emerges in Generic Cortical Microcircuits through Spike-Timing-Dependent Plasticity. *PLoS Computational Biol*, 9(4), 2013.

Arild Nøkland. Direct Feedback Alignment Provides Learning in Deep Neural Networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2016.

Peter O'Connor and Max Welling. Deep Spiking Networks. *arXiv:1602.08323v2, NIPS 2016 workshop "Computing with Spikes"*, 2016.

Garrick Orchard, Ajinkya Jayawant, Gregory K. Cohen, and Nitish Thakor. Converting Static Image Datasets to Spiking Neuromorphic Datasets Using Saccades. *Frontiers in Neuroscience*, (9:437), 2015.

Priyadarshini Panda and Kaushik Roy. Unsupervised Regenerative Learning of Hierarchical Features in Spiking Deep Networks for Object Recognition. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 299–306, 2016.

Priyadarshini Panda, Gopalakrishnan Srinivasan, and Kaushik Roy. Convolutional Spike Timing Dependent Plasticity based Feature Learning in Spiking Neural Networks. *arXiv:1703.03854v2*, 2017.

Mihai A. Petrovici, Johannes Bill, Ilya Bytschok, Johannes Schemmel, and Karlheinz Meier. Stochastic inference with deterministic spiking neurons. *arXiv:1311.3211v1*, 2013.

Mihai A. Petrovici, Sebastian Schmitt, Johann Klähn, David Stöckel, Anna Schroeder, Guillaume Bellec, and Johannes Bill. Pattern representation and recognition with accelerated analog neuromorphic systems. *arXiv:1703.06043*, 2017.

Michael Pfeiffer and Thomas Pfeil. Deep Learning With Spiking Neurons: Opportunities and Challenges. *Frontiers in Compututational Neuroscience*, 12 (774), 2018.

Christoph Posch and Rainer Wohlgenannt. An Asynchronous Time-based Image Sensor. In *IEEE International Symposium on Circuits and Systems (ISCAS) 2008*, pages 2130–2133, 2008.

Dimitri Probst, Mihai A. Petrovici, Ilja Bytschok, Johannes Bill, Dejan Pecevski, Johannes Schemmel, and Karlheinz Meier. Probabilistic inference in discrete spaces can be implemented into networks of LIF neurons. *Frontiers in Computational Neuroscience*, 9(13), 2015.

Ning Qiao, Hesham Mostafa, Federico Corradi, Marc Osswald, Fabio Stefanini, Dora Sumislawska, and Giacomo Indiveri. A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128K synapses. *Frontiers in Neuromorphic Engineering*, 2015.

Damien Querlioz, Olivier Bichler, and Christian Gamrat. Simulation of a Memristor-Based Spiking Neural Network Immune to Device Variations. In *International Joint Conference on Neural Networks (IJCNN)*, number 8, pages 1775–1781, 2011.

Damien Querlioz, Olivier Bichler, Adrien Francis Vincent, and Christian Gamrat. Bioinspired Programming of Memory Devices for Implementing an Inference Engine. In *Proceedings of the IEEE*, volume 103, pages 1398–1416, 2015.

M. Rhu, M. O'Connor, N. Chatterjee, J. Pool, Y. Kwon, and S. W. Keckler. Compressing dma engine: Leveraging activation sparsity for training deep neural networks. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 78–91, Feb 2018.

B. Rueckauer and S. Liu. Conversion of analog to spiking neural networks using sparse temporal coding. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, May 2018.

Bodo Rueckauer, Iulia-Alexandra Lungu, Yuhuang Hu, Michael Pfeiffer, and Shih-Chii Liu. Conversion of Continuous-Valued Deep Networks to Efficient Event-Driven Networks for Image Classification. *Frontiers in Neuroscience*, 11(682), 2017.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323, October 1986.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

João Sacramento, Rui Costa, Y Bengio, and Walter Senn. Dendritic cortical microcircuits approximate the backpropagation algorithm. In *Advances in Neural Information Processing Systems (NIPS)*, 2018.

Arash Samadi, Timothy P. Lillicrap, and Douglas B. Tweed. Deep Learning with Dynamic Spiking Neurons and Fixed Feedback Weights. *Neural Computation*, (29):578–602, 2017.

Jüergen Schmidhuber. Deep Learning in Neural Networks: An Overview. *arXiv:1404.7828v4*, 2014.

Sebastian Schmitt, Johann Klahn, Guillaume Bellec, Andreas Grubl, Maurice Guttler, Andreas Hartel, Stephan Hartmann, Dan Husmann, Kai Husmann, Sebastian Jeltsch, Vitali Karasenko, Mitja Kleider, Christoph Koke, Alexander Kononov, Christian Mauch, Eric Müller, Paul Muller, Johannes Partzsch, Mihai A. Petrovici, and Karlheinz Meier. Neuromorphic hardware in the loop: Training a deep spiking network on the brainscales wafer-scale system. In *International Joint Conference on Neural Networks (IJCNN)*, pages 2227–2234, May 2017.

Catherine Schuman, Thomas Potok, Robert Patton, J Birdwell, Mark Dean, Garrett Rose, and James Plank. A survey of neuromorphic computing and neural networks in hardware. *arXiv:1705.06963v1*, 2017.

Abhronil Sengupta, Yuting Ye, Robert Wang, Chiao Liu, and Kaushik Roy. Going Deeper in Spiking Neural Networks: VGG and Residual Architectures. *Frontiers in Neuroscience*, 13:95, 2019.

William Severa, Craig M. Vineyard, Ryan Dellana, Stephen J. Verzi, and James B. Aimone. Training deep neural networks for binary communication with the Whetstone method. *Nature Machine Intelligence*, (1):86–94, 2019.

Sumit Bam Shrestha and Garrick Orchard. Slayer: Spike layer error reassignment in time. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1412–1421, 2018.

Amos Sironi, Manuele Brambilla, Nicolas Bourdis, Xavier Lagorce, and Ryad Benosman. Hats: Histograms of averaged time surfaces for robust event-based object classification. In *Conference on Computer Vision and Pattern Recognition (CVPR) 2018, Salt Lake City, UT, USA*, pages 1731–1740, 06 2018.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

Evangelos Stromatias, Miguel Soto, Teresa Serrano-Gotarredona, and Linares-Barranco Bernabé. An Event-Driven Classifier for Spiking Neural Networks Fed with Synthetic or Dynamic Vision Sensor Data. *Frontiers in Neuroscience*, 11(350):1–15, 2017.

Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and Policy Considerations for Deep Learning in NLP. *arXiv:1906.02243v1*, 2019.

Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on Machine Learning*, volume 28, pages 1139–1147, 17–19 Jun 2013.

Amirhossein Tavanaei and Anthony Maida. Bp-stdp: Approximating backpropagation using spike timing dependent plasticity. *Neurocomputing*, 330: 39 – 47, 2019.

Amirhossein Tavanaei and Anthony S. Maida. Bio-inspired spiking convolutional network using layer-wise sparse coding and STDP learning. *arXiv:1611.03000v3*, 2017.

Amirhossein Tavanaei, Timothée Masquelier, and Anthony S. Maida. Acquisition of visual features through probabilistic spike-timing-dependent plasticity . In *International Joint Conference on Neural Networks (IJCNN), Vancouver, BC, Canada*, pages 307–314, 2016.

Amirhossein Tavanaei, Masoud Ghodrati, Saeed Reza Kheradpisheh, Timothée Masquelier, and Anthony" Maida. Deep learning in spiking neural networks. *Neural Networks*, 111:47 – 63, 2019.

Johannes C. Thiele, Olivier Bichler, and Antoine Dupret. Using STDP for unsupervised, event-based online learning. *NIPS 2017 workshop "Cognitively Informed Artificial Intelligence: Insights from Natural Intelligence", Long Beach, CA, USA*, 2017a.

Johannes C. Thiele, Peter U. Diehl, and Matthew Cook. A wake-sleep algorithm for recurrent, spiking neural networks. *NIPS 2016 workshop "Computing with Spikes"*, 2017b.

Johannes C. Thiele, Olivier Bichler, and Antoine Dupret. Event-Based, Timescale Invariant Unsupervised Online Deep Learning With STDP. *Frontiers in Computational Neuroscience*, 12:46, 2018a.

Johannes C. Thiele, Olivier Bichler, and Antoine Dupret. A Timescale Invariant STDP-Based Spiking Deep Network for Unsupervised Online Feature Extraction from Event-Based Sensor Data. In *International Joint Conference on Neural Networks (IJCNN), Rio de Janeiro, Brazil*, pages 1666–1673, 2018b.

Johannes C. Thiele, Olivier Bichler, and Antoine Dupret. Ternarized gradients for efficient on-chip training of spiking neural networks. In *Cognitive Computing - Merging Concepts with Hardware (Extended Abstract), Hannover, Germany*, 2018c.

Johannes C. Thiele, Olivier Bichler, and Antoine Dupret. SpikeGrad: An ANN-equivalent Computation Model for Implementing Backpropagation with Spikes. *arXiv:1906.00851*, 2019a.

Johannes C. Thiele, Olivier Bichler, Antoine Dupret, Sergio Solinas, and Giacomo Indiveri. A Spiking Network for Inference of Relations Trained with Neuromorphic Backpropagation. In *(to appear) International Joint Conference on Neural Networks (IJCNN), arXiv:1903.04341*, 2019b.

Simon Thorpe, Denis Fize, and Catherine Marlot. Speed of processing in the human visual system. *Nature*, 381:520–522, 1996.

Simon Thorpe, Arnaud Delorme, and Rufin Van Rullen. Spike-based strategies for rapid processing. *Neural Networks*, 14:715–725, 2001.

Jibin Wu, Yansong Chua, Malu Zhang, Qu Yang, Guoqi Li, and Haizhou Li. Deep Spiking Neural Network with Spike Count based Learning Rule. *arXiv:1902.05705v1*, 2019a.

Xi Wu, Yixuan Wang, Huajin Tang, and Rui Yan. A structure–time parallel implementation of spike-based deep learning. *Neural Networks*, 113:72 – 78, 2019b.

Yujei Wu, Lei Deng, Guoqi Li, Jun Zhu, and Luping Shi. Spatio-Temporal Backpropagation for Training High-Performance Spiking Neural Networks. *Frontiers in Neuroscience*, (12:331), 2018a.

Yujei Wu, Lei Deng, Guoqi Li, Jun Zhu, and Luping Shi. Direct Training for Spiking Neural Networks: Faster, Larger, Better. *arXiv:1809.05793v1*, 2018b.

Will Xiao, Honglin Chen, Qianli Liao, and Tomaso Poggio. Biologically-plausible learning algorithms can scale to large datasets. In *International Conference on Learning Representations*, 2019.

Penghan Yin, Jiancheng Lyu, Shuai Zhang, Stanley Osher, Yingyong Qi, , and Jack Xin. Understanding Straight-Through Estimator in Training Activation Quantized Neural Nets. In *International Conference on Learning Representations 2019*, 2019.

Shihui Yin, Shreyas K. Venkataramanaiah, Gregory K. Chen, Ram Krishnamurthy, Yu Cao, Chaitali Chakrabarti, and Jae-sun Seo. Algorithm and Hardware Design of Discrete-Time Spiking Neural Networks Based on Back Propagation with Binary Activations. *arXiv:1709.06206v1*, 2017.

Amirreza Yousefzadeh, Timothée Masquelier, and Teresa Serrano-Gotarredona. Hardware Implementation of Convolutional STDP for On-line Visual Feature Learning. In *2017 IEEE International Symposium on Circuits and Systems (ISCAS), Baltimore, MD, US*, 2017.

Friedemann Zenke and Surya Ganguli. Superspike: Supervised learning in multilayer spiking neural networks. *Neural Computation*, (30):1514–1541, 2018.

Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv:1606.06160v3*, 2018.

ÉCOLE DOCTORALE

Physique et ingénierie : électrons, photons, sciences du vivant (EOBE)

université PARIS-SACLAY

Titre : L'apprentissage profond dans les systèmes évènementiels, bio-inspirés

Mots clés : réseaux de neurones évènementiels ; apprentissage bio-inspiré ; algorithme de rétro-propagation ; matériel bio-inspiré ; apprentissage profond

Résumé :
Inférence et apprentissage dans les réseaux de neurones profonds nécessitent une grande quantité de calculs qui, dans beaucoup de cas, limite leur intégration dans les environnements limités en ressources. Les réseaux de neurones évènementiels de type ≪spike≫ présentent une alternative aux réseaux de neurones artificiels classiques, et promettent une meilleure efficacité énergétique. Cependant, entraîner les réseaux spike demeure un défi important, particulièrement dans le cas où l'apprentissage doit être exécuté sur du matériel de calcul bio-inspiré, dit matériel neuromorphique. Cette thèse constitue une étude sur les algorithmes d'apprentissage et le codage de l'information dans les réseaux de neurones spike.
A partir d'une règle d'apprentissage bio-inspirée, nous analysons quelles propriétés sont nécessaires dans les réseaux spike pour rendre possible un apprentissage embarqué dans un scénario d'apprentissage continu. Nous montrons qu'une règles basée sur le temps de déclenchement des neurones (type ≪spike-timing dependent plasticity≫) est capable d'extraire des caractéristiques pertinentes pour permettre une classification d'objets simples comme ceux des bases de données MNIST et N-MNIST.
Pour dépasser certaines limites de cette approche, nous élaborons un nouvel outil pour l'apprentissage dans les réseaux spike, *SpikeGrad*, qui représente une implémentation entièrement évènementielle de la rétro-propagation du gradient. Nous montrons comment cette approche peut être utilisée pour l'entrainement d'un réseau spike qui est capable d'inférer des relations entre valeurs numériques et des images MNIST. Nous démontrons que cet outil est capable d'entrainer un réseau convolutif profond, qui donne des taux de reconnaissance d'image compétitifs avec l'état de l'art sur les bases de données MNIST et CIFAR10. De plus, *SpikeGrad* permet de formaliser la réponse d'un réseau spike comme celle d'un réseau de neurones artificiels classique, permettant un entraînement plus rapide.
Nos travaux introduisent ainsi plusieurs mécanismes d'apprentissage puissants pour les réseaux évènementiels, contribuant à rendre l'apprentissage des réseaux spike plus adaptés à des problèmes réels.

Title : Deep learning in event-based neuromorphic systems

Keywords : spiking neural network ; spike-timing dependent plasticity ; backpropagation algorithm ; neuromorphic hardware ; deep learning

Abstract : Inference and training in deep neural networks require large amounts of computation, which in many cases prevents the integration of deep networks in resource constrained environments. Event-based spiking neural networks represent an alternative to standard artificial neural networks that holds the promise of being capable of more energy efficient processing. However, training spiking neural networks to achieve high inference performance is still challenging, in particular when learning is also required to be compatible with neuromorphic constraints. This thesis studies training algorithms and information encoding in such deep networks of spiking neurons.
Starting from a biologically inspired learning rule, we analyze which properties of learning rules are necessary in deep spiking neural networks to enable embedded learning in a continuous learning scenario. We show that a time scale invariant learning rule based on spike-timing dependent plasticity is able to perform hierarchical feature extraction and classification of simple objects of the MNIST and N-MNIST dataset.
To overcome certain limitations of this approach we design a novel framework for spike-based learning, *SpikeGrad*, which represents a fully event-based implementation of the gradient backpropagation algorithm. We show how this algorithm can be used to train a spiking network that performs inference of relations between numbers and MNIST images. Additionally, we demonstrate that the framework is able to train large-scale convolutional spiking networks to competitive recognition rates on the MNIST and CIFAR10 datasets. In addition to being an effective and precise learning mechanism, *SpikeGrad* allows the description of the response of the spiking neural network in terms of a standard artificial neural network, which allows a faster simulation of spiking neural network training.
Our work therefore introduces several powerful training concepts for on-chip learning in neuromorphic devices, that could help to scale spiking neural networks to real-world problems.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France