UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

RICARDO GABRIEL HERDT

Complex-Valued Multilayer Perceptron for Object Recognition in Polarimetric SAR Images

Trabalho de graduação.

Trabalho realizado na Technische Universität Berlin dentro do acordo de dupla diplomação UFRGS - TU Berlin.

Orientador brasileiro:
Prof. Dr. Manuel Menezes de Oliveira Neto

Orientador alemão:
Prof. Dr. Olaf Hellwich

Co-orientador alemão:
Dr. Ronny Hänsch

Porto Alegre
Dezembro 2017

CIP — CATALOGAÇÃO NA PUBLICAÇÃO

Herdt, Ricardo Gabriel

Complex-Valued Multilayer Perceptron for Object Recognition in Polarimetric SAR Images / Ricardo Gabriel Herdt. – Porto Alegre: CIC da UFRGS, 2018.

70 f.: il.

Trabalho de conclusão (Bacharelado) – Universidade Federal do Rio Grande do Sul. Bacharelado em Ciência da Computação, Porto Alegre, BR–RS, 2018. Orientador brasileiro: Manuel Menezes de Oliveira Neto; Orientador alemão: Olaf Hellwich; Co-orientador alemão: Ronny Hänsch.

1. Reconhecimento de Objetos. 2. Classificação de Imagens. 3. Machine Learning. 4. Multilayer Perceptron. 5. PolSAR. I. Menezes de Oliveira Neto, Manuel. II. Título.

RESUMO

Radares de Abertura Sintética Polarimétricos (PolSAR) são capazes de prover imagens de alta resolução da Terra, independentemente da luz-do-dia e sob praticamente quaisquer condições climáticas. Tais imagens são úteis em diversas aplicações, incluindo as que têm por fim reconhecer categorias de objetos em uma superfície, por exemplo para monitorar a utilização do solo ou fornecer suporte a operações de resgate após catástrofes naturais. Estas imagens possuem propriedades geométricas e radiométricas que dificultam sua interpretação, o que é exacerbado pelo constante crescimento da quantidade de tais imagens disponíveis em alta resolução.

A maioria dos métodos de reconhecimento de objetos operam sobre imagens representadas por matrizes de números reais. O fato de dados PolSAR serem de natureza complexa leva ao questionamento se métodos especificamente desenvolvidos para trabalhar com números complexos seriam capazes de explorar melhor as propriedades destes dados. Uma das possibilidades é o uso do *Complex-Valued Multilayer Perceptron* (CV-MLP). Trabalhos prévios mostraram resultados promissores ao aplicar uma implementação básica deste algoritmo à tarefa em questão. O presente trabalho mostra que um CV-MLP usando técnicas como Inicialização de Xavier, Adagrad e unidades lineares retificadas é capaz de aprimorar tais resultados. A precisão obtida é no entanto equivalente a de um *Multilayer Perceptron* de natureza real análogo, com a condição de que uma transformação apropriada do domínio complexo para o real seja empregada sobre os dados fornecidos ao algoritmo.

Palavras-chave: Reconhecimento de Objetos. Classificação de imagens. Machine Learning. Multilayer Perceptron. Radar de abertura sintética polarimétrico.

Resumo estendido

Este é um resumo estendido em português para a Universidade Federal do Rio Grande do Sul do trabalho original que segue. O trabalho de conclusão original, em inglês, foi apresentado na Technische Universität Berlin através do programa de dupla-diplomação entre as duas universidades.

1 INTRODUÇÃO

Radares de abertura sintética polarimétricos (*Polarimetric Synthetic Aperture Radar* – PolSAR) são largamente usados para aplicações de sensoreamento remoto, uma vez que fornecem imagens de alta resolução independentemente da luz do dia e sob praticamente qualquer condição climática. Tais imagens são uma fonte valiosa de informações para diversas aplicações. Particularmente o reconhecimento de objetos em imagens PolSAR demonstrou-se útil no monitoramento de desmatamento e na coordenação de equipes de resgate em situações de desastre (HÄNSCH, 2014). A natureza e quantidade de dados disponíveis impõem no entanto diversas dificuldades à sua interpretação, gerando uma demanda por métodos automáticos eficientes e confiáveis para reconhecimento de objetos em imagens PolSAR. Hänsch e Hellwich (2009) investigaram o uso de Perceptrons Multicamada de natureza complexa (*Complex-Valued Multi-Layer Perceptrons* – CV-MLP) para a categorização destas imagens, atendo-se a uma implementação elementar do algoritmo. O propósito do presente trabalho é explorar as possibilidades de CV-MLPs para esta tarefa, adaptando e implementando diversos aperfeiçoamentos comumente empregados a *Multi-Layer Perceptrons* (MLP) de natureza real (*Real-Valued Multilayer Perceptron* –RV-MLP) ao CV-MLP.

2 CONCEITOS BÁSICOS

Imagens PolSAR

Imagens de sistemas PolSAR são geradas a partir de reflexões de ondas eletromagnéticas de diferentes polarizações transmitidas sobre a área observada. O sinais refletidos são somados coerentemente e processados de modo a permitir a distinção dos diferentes objetos. Isso é possível devido à diferente resposta de certos objetos a determinadas polarizações (HÄNSCH; HELLWICH, 2010).

A análise de imagens PolSAR, seja ela manual ou automática, é dificultada por diversos fatores. Além de diferentes propriedades geométricas quando comparadas a imagens ópticas, elas sofrem de efeitos tais como sombras (pontos negros originados pela oclusão de ondas), *layover* (junção das reflexões de diferentes objetos em um único pixel) e *speckle*, uma granularidade originada pela interferência de diversas ondas coerentes.

Reconhecimento de Objetos em imagens PolSAR

O propósito do *Reconhecimento de Objetos* é encontrar objetos em uma cena com base em um modelo do mesmo (JAIN; KASTURI; SCHUNCK, 1995). Aqui cabe distinguir *Reconhecimento de Objetos Genérico*, também conhecido por categorização, onde busca-se classificar regiões da imagem de acordo com certas categorias, do *Reconhecimento de Objetos Específico*, que atém-se a reconhecer um objeto específico. Aplicações de sensoreamento remoto com base em imagens PolSAR geralmente têm por objetivo a categorização de imagens, o que se deve a diversos fatores, tais como a falta de imagens de alta resolução até anos recentes e a ausência de extratores de características (ing. *features*) adequados (HÄNSCH; HELLWICH, 2010, p. 127). Por este motivo, este trabalho delimita a tarefa de Reconhecimento de Objetos à atribuição de categorias aos elementos da imagem em questão.

Hänsch (2014, p. 7) estende a interpretação de Reconhecimento de Objetos Genérico ao assumir que um sistema neste contexto deve ser capaz de categorizar a imagem sem utilizar conhecimento prévio de um conjunto de dados específico em seu desenvolvimento.

Perceptron Multicamada

Perceptrons Multicamada são um tipo de Rede Neural Pré-Alimentada (*Feed-Forward Neural Network*). Um MLP consiste de diversas unidades denominadas neurônios dispostas em camadas e sem retroalimentação (*feedback loop*).

3 IMAGENS DE RADAR DE ABERTURA SINTÉTICA POLARIMÉTRICO

Imageamento por Radar

Imagens de Radar são geradas a partir de faixas capturadas por um satélite em movimento ou aeronave. Estas são obtidas através da transmissão de pulsos eletromagnéticos de frequência modulada, os quais são refletidos por objetos na região observada. A distinção dos objetos emprega diferentes técnicas de acordo com a orientação de resolução almejada. Na direção de alcance (*range direction*) objetos são separados de acordo com a diferença temporal entre as reflexões obtidas independentemente da tecnologia de Radar em questão. A resolução ao longo do trajeto da aeronave (a chamada direção de azimute) por sua vez difere consideravelmente entre os sistemas existentes. Radares de Abertura Sintética fazem isso considerando o efeito Doppler ocasionado pela diferente disposição de objetos na superfície, o que requer o armazenamento da fase dos sinais recebidos. Uma explanação detalhada da técnica empregada encontra-se em Zyl (2011).

PolSAR

Radares de Abertura Sintética Polarimétricos aperfeiçoam a técnica SAR empregando ondas eletromagnéticas polarizadas. Polarização consiste na restrição da direção ao longo da qual o componente elétrico da onda oscila. Devido ao fato de certo objetos refletirem sinais diferentemente de acordo com a polarização empregada, imagens PolSAR permitem "uma inferência mais completa da superfície natural ou outros parâmetros objetivos" (ZEBKER; ZYL, 1991, p. 1583, tradução nossa). As diferentes combinações de polarização horizontal e vertical dos sinais transmitidos e recebidos são armazenados em uma matriz 2×2 de dispersão (*scattering matrix*) de elementos complexos, a qual representa um "*pixel*" da imagem.

4 PERCEPTRON MULTICAMADA DE NATUREZA COMPLEXA

Um MLP consiste de diversas unidades de processamento (neurônios) dispostas em camadas, sendo que as saídas de neurônios de uma camada são conectadas às entradas da camada seguinte (sem retroalimentação). Em cada neurônio, uma função de transferência (também denominada função de ativação) não linear é aplicada sobre uma combinação linear de suas respectivas entradas acrescida de um termo de *bias* (para lidar com vetores de entrada nulos). Como um todo um MLP pode portanto ser compreendido como uma função não linear de múltiplas variáveis sobre o vetor de entrada (BISHOP, 1995, p. 117). Um MLP de natureza complexa difere de um MLP real por possuir entradas, parâmetros (pesos individuais por entrada usados no cálculo da combinação linear) e saídas complexos. Além disso as funções de transferência têm domínio e imagem complexos.

Mecanismo

O processo de aprendizagem de um MLP dá-se em duas etapas principais, as quais são repetidas sucessivamente sobre os dados de entrada. Primeiramente a entrada (ou um conjunto delas no caso do processamento em *batch*, ver adiante) é fornecida à primeira camada de neurônios do MLP e propagada até a última camada, aplicando-se a função de transferência sobre uma combinação linear das saídas da camada anterior (incluindo o termo *bias*).

A saída da última camada corresponde à saída do MLP em si. Esta é comparada com valores objetivo (*targets*) utilizados no treinamento por meio de uma função de erro, a qual deve ser minimizada pelo algoritmo. Isso é feito na segunda etapa denominada *back-propagation*, onde os parâmetros de todas as camadas são ajustados de modo a minimizar o erro obtido, seguindo o método do gradiente:

$$w^l_{ji}(t+1) = w^l_{ji}(t) + \mu \Delta w^l_{ji}(t) \tag{4.1}$$

$$\Delta w^l_{ji}(t) = -\nabla E_{\{w_{ji}\}}(\mathbf{D}) \tag{4.2}$$

onde w_{ji} são os pesos conectando os neurônios j e i, μ é a constante de taxa de aprendizagem e 4.2 é o gradiente da função de erro respectivamente aos atuais parâmetros da rede e o conjunto dos dados de entrada.

Funções de Ativação

MLPs impõem uma série de requisitos às funções de ativação. Usualmente estas devem ser não lineares, analíticas (de modo a permitir o emprego do método do gradiente) e idealmente limitadas de modo a prevenir o crescimento das entradas e consequentemente problemas de representação numérica. Exemplos típicos no domínio real são as funções sigmoidais, tais como a tangente hiperbólica e a função logística.

Tais restrições no entanto não podem ser aplicadas a um CV-MLP, uma vez que pelo Teorema de Liouville somente funções constantes satisfazem-nas no domínio complexo. Georgiou e Koutsougeras (1992) propõem um relaxamento das restrições, particularmente exigindo somente que as derivadas parciais (respectivamente à componente real ou imaginária da variável) existam e sejam limitadas. Exemplos de funções de ativação complexas que satisfazem tais condições são a *split-tanh* e *split-logistic*, que aplicam a respectiva função real (*tanh* ou logística) separadamente à componente real e imaginária da combinação linear de entrada. Este trabalho investiga além das funções supracitadas a função retificada (*rectifier*) e sua variante complexa *split-rectifier*, além da função *tanh* no domínio complexo.

Aperfeiçoamentos

Uma série de técnicas podem ser empregadas para otimizar o processo de aprendizado de um MLP.

Weight Decay

Um técnica de regularização que consiste em adicionar um termo à função de erro de modo a penalizar altos valores nos pesos. Isso leva o algoritmo a priorizar as menores soluções para o problema de otimização em questão.

Aprendizado em Batch *e Algoritmo do Gradiente Estocástico*

O algoritmo MLP original ajusta os pesos da rede após processar todos os dados de entrada, acumulando os respectivos erros. Esta estratégia também é conhecido por aprendizado em *Batch*. Uma alternativa, denominada Algoritmo do Gradiente Estocástico, consiste em atualizar os pesos após o processamento individual de cada um dos dados de entrada, o que confere um comportamento mais "ruidoso" ao processo de aprendizado e permite à rede evitar mínimos locais do espaço de soluções (LECUN et al., 2012, p. 13). Finalmente pode-se considerar *mini-batches* (um subconjunto dos dados de entrada) para a atualização dos pesos, combinando propriedades dos métodos supracitados.

Inicialização dos parâmetros

A escolha de valores iniciais adequados para os pesos do algoritmo garante uma convergência mais rápida e evita possíveis problemas numéricos. Uma técnica comumente empregada consiste na inicialização de *Xavier*, a qual atribui valores aleatórios a cada parâmetro de modo a garantir que a variância dos pesos de uma camada satisfaça a seguinte relação:

$$Var(\mathbf{w}^l) = \frac{2}{N_l + N_{l+1}} \tag{4.3}$$

onde $\mathbf{w}^l$ é o vetor de pesos da camada l, e N_l o número de unidades da camada l. Com pequenas alterações derivamos a distribuição de valores necessária para garantir a mesma propriedade em um CV-MLP.

Ajuste da taxa de aprendizado

A taxa de aprendizado da equação 4.1 é um importante hiperparâmetro do algoritmo. Uma vez que a escolha de uma taxa adequada tem grande influência nos resultados e depende de particularidades do conjunto de dados, convém empregar alguma estratégia de ajuste deste parâmetro de modo a otimizar o aprendizado. O presente trabalho investiga dois algoritmos presentes na literatura, o *Adagrad* (*Adaptive Gradient Algorithm*) e o *Adam* (*Adaptive Moment Estimation*), que ajustam os pesos individualmente, concedendo taxas maiores a características (*features*) menos frequentes.

5 AVALIAÇÃO E RESULTADOS

Com o propósito de comparar a performance de um MLP complexo à de um real, implementamos o algoritmo completo capaz de lidar com ambas variantes. A implementação suporta as técnicas mencionadas anteriormente, tais como *Weight Decay*, *Adagrad*, *Adam*, inicialização de *Xavier* e múltiplas funções de ativação.

Os dados utilizados consistem em diversas imagens extraídas de uma imagem PolSAR fornecida pelo Centro Aeroespacial Alemão. Esta foi classificada manualmente por um grupo de especialistas, os quais atribuíram a cada pixel uma dentre cinco classes (floresta, plantação, arbustal, estrada e construção). Todos experimentos utilizam *cross-folding*, o que gera uma maior variedade entre amostras de uma mesma classe e permite o preparo de uma quantidade satisfatória de dados para treinamento e teste do algoritmo. Os experimentos foram escolhidos de modo a verificar a influência dos aperfeiçoamentos selecionados para o presente trabalho.

Como esperado, o método do gradiente estocástico e a técnica *mini-batch* levaram a uma convergência consideravelmente rápida da rede comparados ao método *batch*, com uma ligeira vantagem para o primeiro método. Dado o relativo baixo custo computacional e a rápida convergência do *mini-batch*, optamos por utilizá-lo para os demais experimentos, viabilizando uma maior quantidade e variedade de testes.

O uso de uma taxa fixa de aprendizagem leva a grandes contrastes entre as diversas funções de ativação, e constatamos uma vantagem considerável do CV-MLP comparado ao RV-MLP quando a função de ativação "correspondente" (por ex. *rectifier* e *split-rectifier*) foi empregada. Além disso a função de ativação $tanh$ levou neste caso a melhores resultados se comparada à função retificada. Já a função de ativação *tanh* complexa resultou em uma divergência da rede em alguns experimentos, o que deve-se dentre outros fatores a singularidades da função no domínio complexo.

O uso do *Adagrad* ou *Adam* reduz consideravelmente as diferenças entre o CV-MLP e o RV-MLP, levando à uma ligeira vantagem do último dependendo da transformação aplicada aos dados de entrada para o domínio real. Além disso constatamos uma ligeira vantagem da função retificada quando comparada às funções sigmoidais.

6 CONCLUSÃO

Este trabalho investigou os méritos do uso de um CV-MLP para a classificação de imagens PolSAR. Selecionamos e adaptamos ao domínio complexo diversos aperfeiçoamentos da literatura, além de diferentes funções de ativação, incluindo as funções retificadas de trabalhos recentes.

Mostramos experimentalmente que o uso de técnicas de ajuste da taxa de aprendizagem (no caso *Adagrad* ou *Adam*) influenciam mais a performance da rede do que a escolha entre um CV-MLP e RV-MLP ou da função de ativação. Além disso, tais técnicas ressaltam os benefícios do uso de uma função retificada, sobretudo quando redes com um maior número de neurônios são usadas.

Trabalhos futuros podem investigar outras funções complexas analíticas mas irrestritas, as quais, acompanhadas de técnicas de normalização adequadas, podem levar a outras constatações. O emprego de *Adagrad* ou *Adam* é aconselhável, uma vez que como constatamos o uso de uma taxa fixa de aprendizagem influencia negativamente o comportamento da rede.

Complex-Valued Multilayer Perceptron for Object Recognition in Polarimetric SAR Images

Vorgelegt von

Ricardo Gabriel HERDT

Von der Fakultät IV - Elektrotechnik und Informatik

der Technischen Universität Berlin

zur Erlangung des akademischen Grades

Bachelor in Informatik

- B. Informatik -

genehmigte Abschlussarbeit.

Gutachter :	Prof. Dr.-Ing. Olaf HELLWICH
	Prof. Dr. Ben JUURLINK
Betreuer :	Dr.-Ing. Ronny HÄNSCH

Eidesstattliche Versicherung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ohne unerlaubte fremde Hilfe und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.

Berlin, den 29. September 2017 ....

Ricardo Gabriel HERDT

Abstract

Polarimetric Synthetic Aperture Radars (PolSAR) are able to deliver high-resolution images of the Earth, independently of daylight and almost under any weather condition. Such images are useful in several applications, including those interested in recognizing object categories over a region, for instance for controlling land cover or supporting rescue efforts after natural catastrophes. These images have some geometrical and radiometric properties that make their interpretation a non-trivial task though, what gets exacerbated by the increasing amount of images and improvements of their resolution. Thus automatic methods are essential for efficiently accomplishing such tasks.

Most general object recognition methods are concerned with real-valued optical images. The fact that PolSAR measurements are inherently complex-valued raises the question if methods specially designed to work with complex numbers could better exploit their properties. One of the possibilities is to use a Complex-valued Multilayer Perceptron (CV-MLP). Early works showed promising results by applying a basic implementation of a CV-MLP to the aforementioned task. This work shows that a CV-MLP using techniques such as Xavier Initialization, Adagrad and ReLUs is able to obtain much better results. The achieved accuracy is equivalent to that of an analogous real-valued MLP though, provided a proper mapping of the complex-valued PolSAR data to the real domain is used.

Zusammenfassung

Polarimetrische Radars mit synthetischer Apertur (eng. PolSAR) sind in der Lage hoch auflösende Bilder der Erde unabhängig von Tageslicht und unter fast allen Wetterbedingungen zu liefern. Solche Bilder sind bei vielen Anwendungen nützlich, unter anderem bei solchen die an der Erkennung von Objektkategorien auf Gebieten interessiert sind, um beispielsweise Landbedeckung zu kontrollieren und Rettungsaktionen nach natürlichen Katastrophen zu unterstützen. Diese Bilder besitzen geometrische und radiometrische Eigenschaften die ihre Interpretierung zu einer nicht-trivialen Aufgabe machen, was durch die immer größer werdende Datenmenge und die Verbesserung derer Auflösung verschlimmert wird. Daher sind automatische Methoden unabdingbar um solche Aufgaben effizient zu erfüllen.

Die meisten allgemeinen Objekterkennungsmethoden beziehen sich auf reell wertige optische Bilder. Die Tatsache, dass PolSAR Messungen inhärent komplexwertig sind, wirft die Frage auf, ob Methoden die für den Umgang mit komplexen Zahlen entworfen sind die Eigenschaften dieser Daten besser ausnutzen können. Eine Möglichkeit ist ein Komplexwertiges Mehrlagiges Perceptron (eng. CV-MLP) zu benutzen. Frühere Arbeiten konnten vielversprechende Ergebnisse bei der Anwendung einer einfachen Implementation von einem CV-MLP für die Klassifizierung von PolSAR Bildern feststellen. Diese Arbeit zeigt, dass ein CV-MLP das Techniken wie Xavier Initialisierung, Adagrad und ReLUs verwendet in der Lage ist, deutlich bessere Ergebnisse zu erzielen. Die erreichte Genauigkeit ist allerdings äquivalent zu der von einem analogen reellwertigen MLP, vorausgesetzt, dass eine geeignete Abbildung der komplexwertigen PolSAR-Daten auf den reellen Bereich benutzt wird.

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1 INTRODUCTION

Polarimetric Synthetic Aperture Radars (short PolSAR) are nowadays widely used for remote sensing applications, since they provide high resolution images independent of daylight and almost under any weather condition (HÄNSCH; HELLWICH, 2010). These images are a valuable source of information for several applications. The ability to recognize objects on a radar image for instance might be used for tasks such as deforestation monitoring and disaster management (HÄNSCH, 2014). The nature and amount of data available imposes several difficulties on its manual interpretation, creating a demand for efficient and reliable automatic methods for object recognition in PolSAR images. Hänsch and Hellwich (2009) investigated the use of Complex-Valued Multi-Layer Perceptrons (short CV-MLP) for categorization of this images. The purpose of the current work is to further explore the capabilities of a CV-MLP for this task by adapting and implementing several improvements used in real-valued MLPs (short RV-MLP) to the complex-valued counterpart.

1.1 Concepts

1.1.1 PolSAR images

PolSAR systems generate images based on reflections of electromagnetic waves transmitted in different polarisations over an observed area. The back-scattered signals are coherently summed and further processed in order to distinguish scattering objects. The use of different polarisation delivers extra information that can be useful for object recognition, since "targets show different behaviours regarding polarisations" (HÄNSCH; HELLWICH, 2010).

The complexity of PolSAR data impose a lot of problems regarding its analysis though. Besides different geometrical properties of PolSAR images if compared to optical ones, they are affected by several other issues like shadows, layover and speckle effect (MAGHSOUDI, 2011), which are described in Chapter 2. This makes categorization of object in such images – be it manually done by an expert or automatically – specially difficult. Several works try to improve this task by either developing new decomposition and preprocessing techniques or tackling the problem with different approximation algorithms.

1.1.2 Object Recognition

Object Recognition deals with finding objects on a scene based on an object model (JAIN; KASTURI; SCHUNCK, 1995). The author further defines the object recognition task as follows:

> Formally, given an image containing one or more objects of interest (and background) and a set of labels corresponding to a set of models *known* to the system, the system should assign correct labels to regions, or a set of regions, in the image. (JAIN; KASTURI; SCHUNCK, 1995, p. 459)

This goal resembles what Pinz (2005, p. 258) terms *generic Object Recognition* (short *generic OR*, also known as *categorization*), which contrasts to what he defines as *specific Object Recognition* (short *specific OR*), a task concerned with the recognition of specific, individiual objects.

The generic OR concept can be further extended to the understanding that the system is designed without any previous object category knowledge (HÄNSCH, 2014, p. 7). While expert systems designed for specific categorization scenarios might achieve better results, since particularities of a specific dataset might be explored during the design phase, they usually lack the flexibility required by time-critical use cases. On the other side an artificial neural network such the MLP doesn't depend on a particular dataset during its design, and can be trained on arbitrary data.

Object Recognition is usually done in different layers of processing. In a bottom-up approach, for instance, low-level layers extract features (radiometric vectors, geometrical properties etc.) from their input data and deliver semantically meaningful information to subsequent layers (HÄNSCH; HELLWICH, 2010, p. 110). This is opposed to top-down methods, which use prior knowledge about object models in order to find them in a specific image. Powerful systems can be build that combine both approaches (HÄNSCH; HELLWICH, 2010).

1.1.2.1 Object Recognition in PolSAR data

Regarding Object Recognition in PolSAR images, applications usually are concerned with generic OR (HÄNSCH; HELLWICH, 2010, p. 125). This is related to "the difficult image characteristics, the lack of appropriate feature extractors, the high in-class variety, and just recently available high-resolution PolSAR data ..." (HÄNSCH; HELL-

WICH, 2010, p. 127). Regarding the difficulties inherent to the nature of these images, even shape and object parts might differ within samples of the same category considerably. In this case, applications usually have to rely to a larger degree on this radiometric information, as opposed to geometrical properties (HÄNSCH; HELLWICH, 2010, p. 110). The current work thus shares this understanding, and considers object recognition as the task of attributing categories to objects on an image.

1.1.3 Multilayer Perceptron

Multilayer Perceptrons (short MLP) are a kind of *feed-forward neural network*. They consist of several units called neurons arranged layerwise without feed-back loops. Due to their internal mechanisms explained in Chapter 3, MLPs are capable of approximating arbitrary non-linear functions, what makes them specially useful in tasks with the requirement of handling generic problems, such as generic object recognition.

1.2 Motivation

There are several automatic methods for tackling the task of object recognition. Several of them belong to the class of *supervised learning* methods. These consist in feeding the algorithm with a set of *training inputs* and the corresponding *targets*, which are used to learn the internal parameters of the framework in order to approximate the underlying mapping of inputs to targets. This is then used to classify new data. Since PolSAR data is commonly stored in complex-valued scattering matrices, the use of real-valued supervised learning methods require some kind of processing in order to map complex-values to real-values. An alternative approach would be to use a learning method designed to work directly with the complex-valued inputs. Previous works investigated the use of such methods for different kinds of input data, and Hänsch (2010) showed promising results using complex-valued MLPs for classification of PolSAR images.

The CV-MLP investigated by Hänsch (2010) was "an extension of the most basic form of RV-MLP", as pointed by the author. He thus suggests further exploring several techniques that improve this basic algorithm, specially those related to approximation, normalization and regularisation.

1.3 Objective and Assumptions

The goal of this work is to research several improvements to RV-MLPs suggested by previous works, adapt them for using in a CV-MLP and investigate their merits for the concrete task of general object recognition in PolSAR images. The improvements investigated include different activation functions, basic regularization techniques and adaptive learning strategies such as Adagrad and Adam.

Since object recognition here is restricted to generic OR, it is assumed that the CV-MLP should be able to categorize its inputs, and not identify an specific instance of an object. In order to achieve its goal, this works assumes that the CV-MLP has enough labeled training data.

This work wants to investigate following questions: how does a CV-MLP supporting the aforementioned techniques perform when used for bottom-up object recognition of PolSAR images? Does it achieve similar results or even outperform a RV-MLP? How does each technique mentioned influence its approximation abilities?

1.4 Methodology

In order to investigate the merits of a complex-valued MLP for this task, this work compares its performance to that of a real-valued MLP by implementing a complete MLP system supporting both approaches. This ensures that they operate under similar conditions and can be tested over the same prepared and normalized dataset. Our MLP system supports methods such as Xavier initialization, Adagrad, Adam and different activation functions, which are explained in Chapter 3.

We then use a labeled image provided by the *German Aerospace Center* (german abbreviation *DLR*) to build a proper dataset for training and testing purposes. In order to better explore the dataset and extract results that resemble real usage conditions, the experiments use cross-validation, i.e. we divide the dataset in several folds, train using a subset of it and test on the remaining ones, repeating the process with other partitionings. Several experiments were done using a variety of hyper-parameters in order to better understand the concrete influence of each method on the performance of a CV-MLP for this concrete task.

1.5 Related Work

A first rigorous analysis of the mathematical properties of feed-forward neural network was presented in 1961 by Minsky and Papert (1969), which included a critical view of its benefits and limitations. Only in 1988 though, with introduction of the Back-Propagation algorithm by Rumelhart et al. (1988), a way to learn parameters throughout a multi-layered network was shown. In contrast with a single layered perceptron, this is able to learn non-linear mappings between inputs and outputs, expanding its capabilities (BISHOP, 1995, p. 117).

Early works on complex-valued neural networks specially regarding MLPs, such as Kim and Guest (1990), derived a complex back-propagation algorithm assuming that the complex activation functions used were differentiable. As Georgiou and Koutsougeras (1992) later showed this couldn't be assumed for the activation functions used. By using the Liouville's theorem – which states that "if $f(z)$ is entire[1] and bounded on the complex plane, then $f(z)$ is a constant function"(GEORGIOU; KOUTSOUGERAS, 1992, p. 332) – he proposed a set of requirements for suitable activation functions. With the foundations of complex-valued neural networks set, several works investigated their benefits for different applications, including speech enhancement (DRUDE; RAJ; HAEB-UMBACH, 2016) and Magnetic Resonance Images (MRI) (VIRTUE; YU; LUSTIG, 2017).

Regarding Object Recognition in (Pol)SAR images, Hänsch (2014) proposed a two stage approach that uses a Random Forest classifier for dealing with low-level features, and another Random Forest classifier that computes high-level features taking the outputs from the first stage as inputs. Other works address the problem using a combination of Support Vector Machines and Markov Random Fields ((BOVOLO; BRUZZONE, 2005) and (MASJEDI; ZOEJ; MAGHSOUDI, 2016)). The concrete use of artificial neural networks for supervised SAR classification was explored among others by Antropov et al. (2014), who applied a probabilistic neural network in order to map land cover and classify soil type in boreal forests, and Serpico, Bruzzone and Roli (1996), who compared the performance of an MLP, a probabilistic neural network and a proposed structured neural network for classification of multi sensor (optical and SAR) images, and obtained better accuracy results with the first approach. Hänsch (2010) investigated the use of a complex-valued MLP for classification of PolSAR data, and showed that a simple implementation

[1]A function $f(z)$ is said to be *entire* if it is analytic at all points $z \in \mathbb{C}$(GEORGIOU; KOUTSOUGERAS, 1992)

of a CV-MLP could achieve similar results to a RV-MLP, despite extra difficulties regarding activation function choice and its differentiability.

1.6 Outline

This text is structured as follows: Chapter 2 explains the basic characteristics of PolSAR images, their benefits and particularities. Chapter 3 addresses structure and mechanisms of CV-MLPs, highlighting possible differences to RV-MLPs. Further it describes the techniques explored, including changes needed for a CV-MLP when needed. Chapter 4 first describes the experimental setup, including dataset preparation, input normalization and parameter selection. Then the results are presented and discussed. Chapter 5 summarizes the contributions of this work, followed by suggestions for further investigation paths.

2 POLARIMETRIC SYNTHETIC-APERTURE RADAR IMAGES

Radar systems are able to detect objects by illuminating them with electromagnetic waves and measuring the received echo. Since the signal is actively transmitted, radars can detect objects independently of daylight and under almost any weather condition (HÄNSCH; HELLWICH, 2009). For these reasons they are widely used in remote sensing applications.

2.1 Radar imaging

As thoroughly explained by (ZYL, 2011, p. 14), radar images are generated from several stripes (called *swaths* or *tracks*) acquired by a moving satellite or airplane at one side of the flight path. This is done by transmitting pulses of frequency-modulated electromagnetic at a Pulse Repetition Frequency (short PRF) and capturing their reflections. The received signal is then processed in order to distinguish reflections from different scatterers.

Distinguishing objects in radar images requires different approaches depending on which direction this is done. Along the *range direction* (perpendicular to the flight track) objects are separated according to the time difference between the received back-scattered signals. The *range resolution* depends thus on the duration of the transmitted pulses or the signal bandwidth. The achievable along-track resolution (called *azimuth*) depends on the radar method used. In real aperture radars, this is determined by the antenna size. Due to physical limitations, this leads to poor resolutions, what makes this kind of technique unsuitable for remote sensing applications (ZYL, 2011, 14).

Based on a technique firstly proposed by Carl Wiley in 1951, Synthetic Aperture Radars distinguish objects along the azimuth direction by considering their Doppler frequencies, which differ if they are at different positions relatively to the station's velocity vector (ZYL, 2011, p. 14). The Doppler history can be obtained by coherently summing the back-scattered signal and processing it. A detailed explanation can be found in (ZYL, 2011). This method allows the use of smaller antennas and achieves a much higher along-track resolution, which is independent of the distance to the scattering object (ZYL, 2011, p. 16). For these reasons, SAR images are extensively used nowadays (HÄNSCH, 2010).

2.2 Polarimetric Synthetic Aperture Radar

Polarimetric Synthetic Aperture Radars (short PolSAR) improve the SAR technique by taking into account the polarisation of the electromagnetic waves used. Polarisation restricts the direction in which the electrical field component of the wave oscillates. From the three types of polarisation (circular, elliptical and linear), the linear polarisation is the most commonly used in PolSAR (HÄNSCH; HELLWICH, 2010, p. 112). Polarimetry allows "a more complete inference of natural surface or other target parameters than is possible with a single channel radar system" (ZEBKER; ZYL, 1991, p. 1583). This is possible since "most targets show different behaviours regarding different polarisations", and also because "scatterers change the polarisation of the incident wave due to material or geometrical properties (HÄNSCH; HELLWICH, 2010, p. 112).

The different combinations of horizontal and vertical polarisations from the transmitted and received signals are stored in a 2x2 complex *scattering matrix* **S** defined as:

$$\mathbf{S} = \begin{bmatrix} S_{hh} & S_{hv} \\ S_{vh} & S_{vv} \end{bmatrix}$$

where S_{tr} is the acquired complex value correspondent to the polarisation t of the transmitted wave and the polarisation r of the received wave.

Assuming reciprocity of natural targets, this matrix can be reduced to a three-element complex vector **k** (LEE; GRUNES; GRANDI, 1999, p. 2363):

$$\mathbf{k} = (S_{hh}, \sqrt{2}S_{hv}, S_{vv})^T \tag{2.1}$$

where the factor $\sqrt{2}$ "is to ensure consistency in the (span) total power computation" (LEE; GRUNES; GRANDI, 1999, p. 2363).

An alternative way to store information is the covariance matrix $\mathbf{C}$ (LEE; GRUNES; GRANDI, 1999, p. 2363)

$$\mathbf{C} = (\mathbf{k} \cdot \mathbf{k}^*)^T = \begin{bmatrix} |S_{hh}|^2 & \sqrt{2}S_{hh}S_{hv}^* & S_{hh}S_{vv}^* \\ \sqrt{2}S_{hv}S_{hh}^* & 2|S_{hv}|^2 & \sqrt{2}S_{hv}S_{vv}^* \\ S_{vv}S_{hh}^* & \sqrt{2}S_{vv}S_{hv}^* & |S_{vv}|^2 \end{bmatrix}$$

which provides a compact representation of collective properties of a group of resolution elements (ZEBKER; ZYL, 1991) and simplifies some filtering techniques discussed next.

2.3 SAR-related Effects

Applications analysing SAR and PolSAR images usually have to deal with some typical effects conditioned by the nature of SAR image acquisition. For instance due to the side-looking capturing process and depending on the angle of incidence and geometrical properties of objects and terrain, waves can be occluded at some spots, resulting in black areas called *shadows* (HÄNSCH; HELLWICH, 2010, p. 117).

A second problem called *layover* also relates to geometrical characteristics of the scene. Since "SAR measures the distance between sensor and ground by usage of an electromagnetic wave with a certain extension of the wave front in range direction" (HÄNSCH; HELLWICH, 2010, p. 118), there may be several points within exactly the same distance from the sensor that get merged into a single resolution point. This is for example the case of tall buildings, which are projected onto a same pixel as objects and ground directly in front of them (with respect to the radar position). These effects have an important influence on the particular task of object recognition in (Pol)SAR images, and are some of the reasons that these tend to rely more on radiometric than geometric properties.

2.3.1 Speckle effect and filtering

Raw PolSAR signals typically have a noise-like effect called *speckle*. It is the result of "coherent interference of waves reflected from many elementary scatterers" (LEE; GRUNES; GRANDI, 1999), and can be visually observed as a granular texture over the image. As stressed by (HÄNSCH; HELLWICH, 2010, p.117), this is described as a noise-like effect because although its spatial distribution resembles a random process, it is actually a completely deterministic process.

There are several techniques to reduce the speckle effect. A common approach is the so called *multi-look* processing, which consists in averaging several *one-look* covariance matrices over a group of N image points (LEE; GRUNES; GRANDI, 1999):

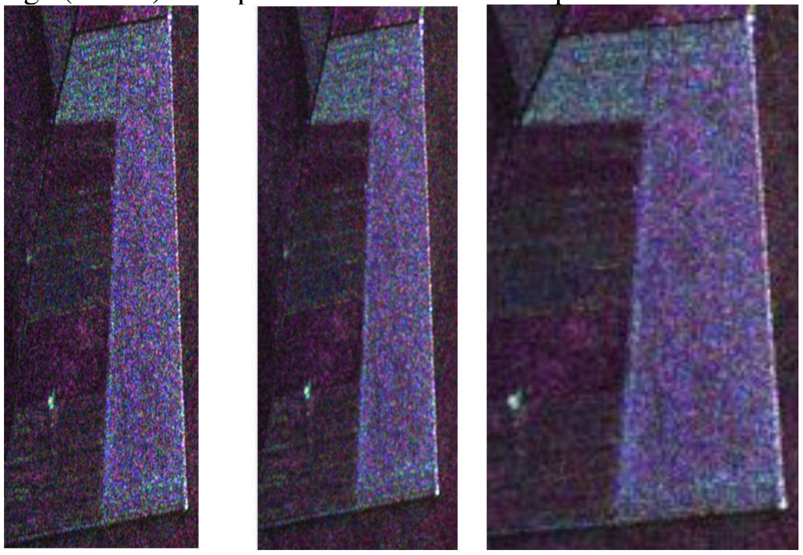
$$\mathbf{Z} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{C}(i) \tag{2.2}$$

One possible disadvantage of this method is the loss of resolution due to averaging over a region.

Other processing techniques for speckle reduction are the use of a *boxcar* filter – a "convolution filter that replaces the center pixel by the mean value of pixels in a square filtering window" (LEE; GRUNES; GRANDI, 1999) – and the *Enhanced Lee Filter*, which uses an estimation of the local variance statistics (LEE; GRUNES; GRANDI, 1999) for filtering images. Both techniques significantly simplify classification tasks, as verified by (HÄNSCH, 2010).

Figure 2.1 illustrates the speckle effect and the resulting images after applying multi-look processing and boxcar filter.

Figure 2.1: Left: original PolSAR image; center: boxcar filtered image; right: multi-look processed image (scaled). Samples extracted from Oberpfaffenhofen dataset (DLR)



3 COMPLEX-VALUED MULTI-LAYER PERCEPTRON

This chapter describes the structures and mechanisms of a Multi-Layer Perceptron (short MLP). To avoid redundancy and complexity, the notation will focus on the complex case, with possible differences to the real-case being discussed when relevant.

The chapter begins with a description of the overall characteristics of an MLP, including a derivation of the back-propagation algorithm, both based on the ones provided by Hänsch (2010). After a discussion about activation functions, including an explanation about the investigated complex-valued ones, several established improvements to the basic MLP model are shown.

3.1 Structure

As a kind of *feed-forward* neural network, Multi-Layer Perceptrons provide "a framework for representing non-linear functional mappings between a set of input variables and a set of output variables" (BISHOP, 1995, p. 117). They consist of a layered arrangement of processing units, with connections between the outputs of the units from one layer and the subsequent ones. In each processing unit a single-valued non-linear activation function (also known as *transfer function*) is applied over a weighted linear combination of its inputs. Due to this composition of several non-linear functions of a single variable, and the fact that there are no feed-back loops in the described network, an MLP as a whole can be seen as a multi-variable non-linear function over its input vector (BISHOP, 1995, p. 117).

In an MLP with L layers, the first one ($l = 1$) is called *input layer*, the last one ($l = L$) *output layer*. The others are referred to as *hidden layers*.

A complex-valued MLP differs from its real counterpart by having complex-valued inputs, weights and outputs. Also activation functions are mappings between complex values. The error function usually maps complex values to real ones, simplifying the minimization task.

3.2 Mechanisms

Given a training set

$$\mathbf{D} = \left\{ (\mathbf{z}, \mathbf{t})^{(\alpha)} \right\}_{\alpha=1,\ldots,\mathbf{P}} \tag{3.1}$$

where $\mathbf{P}$ is the number of training samples, $\mathbf{z} \in \mathbb{C}^{N_0}$ are input vectors, $\mathbf{t} \in \mathbb{C}^{N_L}$ the corresponding targets and N_l the number of neurons at layer l, the goal of an MLP is to find a set of weights $\{w\}^{optimal}$ with which the network output approximates the given targets. Using individual error functions $err(\cdot)$ for each network output $\mathbf{y_L}$ and the corresponding target, the weights of an MLP are adjusted so that the global error function

$$E_{\{w\}}(\mathbf{D}) = \frac{1}{P} \sum_{\alpha=1}^{P} err \left(\mathbf{y}_L(\mathbf{z}^{(\alpha)}), \mathbf{t}^{(\alpha)} \right) \tag{3.2}$$

gets minimized.

This is achieved in a two-way process. First the response of the network given a sample input is computed (*feed-forward*), then the resulting error is *back-propagated* and the networks weights are updated accordingly.

3.2.1 Feed-Forward

This step consists in feeding the inputs into the first layer of the network, and propagating them and the neurons' responses layer-wise towards the output layer. For each neuron i at layer l, the corresponding output y_i^l is computed as follows:

$$y_i^l = \begin{cases} z_i, & l = 1, \\ f^l(h_i^l) & l > 1 \end{cases} \tag{3.3}$$

$$h_i = \sum_{j=1}^{N_{l-1}} y_j^{l-1} \cdot w_{ji}^l \qquad l > 1 \tag{3.4}$$

where h_i is the net input fed into neuron i, $f(h) : \mathbb{C} \to \mathbb{C}$ is the corresponding activation function, $z_i \in \mathbb{C}$ the ith element from the input vector and $w_{ji}^l \in \mathbb{C}$ the weight connecting this neuron to the jth neuron from the preceding layer.

Note that MLPs also add a bias term to Equation 3.3, which allows neurons to deal with all-zero inputs. A common trick is to add to each layer a fixed input, for instance $1 + i$ or 1 when real, and the corresponding weight, what effectively adds a bias term while conserving the simplicity of the feed-forward equations shown.

3.2.2 Back-Propagation

The actual learning process happens during the *back-propagation* of the resulting error. The underlying mechanism was firstly described by Rumelhart et al. (1988). Using the *delta rule*, the algorithm updates each weight following a *gradient descent* strategy, so that the error gets minimized. More specifically a weight w_{ji} is updated at iteration t according to following equations:

$$w^l_{ji}(t+1) = w^l_{ji}(t) + \mu \Delta w^l_{ji}(t) \tag{3.5}$$

$$\Delta w^l_{ji}(t) = -\nabla E_{\{w_{ji}\}}(\mathbf{D}) \tag{3.6}$$

where $\mu \in \mathbb{R}$ is the *learning rate*.

Individual weight updates are proportional to $-\nabla E_{\{w\}}(\mathbf{D})$. In the real case, this corresponds to $-\frac{dE_{\{w\}}(\mathbf{D})}{dw^l_{ji}}$. Since $E_{\{w\}}(\mathbf{D})$ in the complex case is real-valued (with a complex argument), the following rule for gradient descent computation can be applied (PETERSEN; PEDERSEN et al., 2008, p. 19):

$$\begin{aligned}\nabla E_{\{w\}}(\mathbf{D}) &= 2\frac{dE_{\{w\}}(\mathbf{D})}{d\mathbf{w}^*} \\ &= \frac{2}{P}\sum_{\alpha=1}^{P} \frac{\partial err(\mathbf{y}^L(\mathbf{z}^{(\alpha)}, \mathbf{t}^{(\alpha)}))}{\partial {w^l_{ji}}^*}\end{aligned} \tag{3.7}$$

Because $E_{\{w\}}(\mathbf{D})$ is actually a composition of functions, this derivative is obtained with the chain rule of analysis, which, for analytic functions, also applies to complex-valued functions. For non-analytic functions (usually the case in CV-MLPs, see Section 3.2.2), the chain rule reads (PETERSEN; PEDERSEN et al., 2008):

$$\begin{aligned}\frac{\partial g(u)}{\partial x} &= \frac{\partial g}{\partial u}\frac{\partial u}{\partial x} + \frac{\partial g}{\partial u^*}\frac{\partial u^*}{\partial x} \\ &= \frac{\partial g}{\partial u}\frac{\partial u}{\partial x} + \left(\frac{\partial g^*}{\partial u}\right)^* \frac{\partial u^*}{\partial x} \end{aligned} \tag{3.8}$$

where $x \in \mathbb{C}, u = f(x)$.

Thus the derivative of the error function becomes:

$$\frac{\partial err}{\partial {w_{ji}^l}^*} = \frac{\partial err}{\partial h_i^l} \cdot \frac{\partial h_i^l}{\partial {w_{ji}^l}^*} + \frac{\partial err}{\partial {h_i^l}^*} \cdot \frac{\partial {h_i^l}^*}{\partial {w_{ji}^l}^*} \tag{3.9}$$

By deriving Eq. 3.4 we obtain:

$$\frac{\partial h_i^l}{\partial {w_{ji}^l}^*} = 0 \tag{3.10}$$

$$\frac{\partial {h_i^l}^*}{\partial {w_{ji}^l}^*} = \left(\frac{\partial h_i^l}{\partial w_{ji}^l}\right)^* = y_j^{l-1^*} \tag{3.11}$$

where in the last step Eq. A.5 was used.

For simplicity, $\frac{\partial err}{\partial h_i^{l^*}}$ is usually denoted delta term of neuron i:

$$\delta_i^{l^*} = \frac{\partial err}{\partial h_i^{l^*}} = \frac{\partial err}{\partial y_i^l} \cdot \frac{\partial y_i^l}{\partial h_i^{l^*}} + \frac{\partial err}{\partial y_i^l} \cdot \frac{\partial y_i^l}{\partial h_i^{l^*}} \tag{3.12}$$

$\frac{\partial y_i^l}{\partial h_i^l}$ and $\frac{\partial y_i^{l^*}}{\partial h_i^l}$ are computed in the same manner regardless of layer (see Section 3.3). $\frac{\partial err}{\partial y_i^l}$ and $\frac{\partial err}{\partial y_i^{l^*}}$ are determined differently though, depending if the current layer l is an output layer or not. In the first case, the derivative of the error function can be computed directly (see Section 3.4). At hidden layers they depend on delta terms from subsequent layers, so they have to be computed layer-wise towards the input layer (hence *back-propagation*):

$$\frac{\partial err}{\partial y_i^l} = \sum_{r=1}^{N_{l+1}} \left(\frac{\partial err}{\partial h_r^{l+1}} \cdot \frac{\partial h_r^{l+1}}{\partial y_i^l} + \frac{\partial err}{\partial h_r^{l+1^*}} \cdot \frac{\partial h_r^{l+1^*}}{\partial y_i^l}\right) \tag{3.13}$$

$$\frac{\partial err}{\partial y_i^{l^*}} = \sum_{r=1}^{N_{l+1}} \left(\frac{\partial err}{\partial h_r^{l+1}} \cdot \frac{\partial h_r^{l+1}}{\partial y_i^{l^*}} + \frac{\partial err}{\partial h_r^{l+1^*}} \cdot \frac{\partial h_r^{l+1^*}}{\partial y_i^{l^*}}\right) \tag{3.14}$$

The derivative of the net inputs is given by:

$$\frac{\partial h_r^{l+1}}{\partial y_i^l} = w_{ir}^{l+1} \tag{3.15}$$

$$\frac{\partial h_r^{l+1^*}}{\partial y_i^l} = 0 \tag{3.16}$$

$$\frac{\partial h_r^{l+1}}{\partial y_i^{l^*}} = 0 \tag{3.17}$$

$$\frac{\partial h_r^{l+1^*}}{\partial y_i^{l^*}} = \left(\frac{\partial h_r^{l+1}}{\partial y_i^l}\right)^* = w_{ir}^{l+1^*} \tag{3.18}$$

In summary, the delta term in the complex case is determined as follows:

$$\delta_i^l = \begin{cases} \frac{\partial err}{\partial y_i^l} \cdot \frac{\partial y_i^l}{\partial h_i^l} + \frac{\partial err}{\partial y_i^{l^*}} \cdot \frac{\partial y_i^{l^*}}{\partial,} & l = L \\ \sum_{r=1}^{N_{l+1}} \left(\delta_r^{l+1} w_{ir}^{l+1} \frac{\partial y_i^l}{\partial h_i^l} + \delta_r^{l+1^*} w_{ir}^{l+1^*} \frac{\partial y_i^{l^*}}{\partial h_i^l} \right), & l < L \end{cases} \tag{3.19}$$

(3.20)

3.3 Activation functions

The choice of activation functions has an important impact on the capabilities of an MLP. A first consideration concerns their non-linearity. If the hidden layers use linear activation functions to determine their outputs, the network loses its advantage over single-layered feed-forwards networks (BISHOP, 1995, p. 121). Further the activation function should be analytic (since the back-propagation algorithm needs its derivative, see Section 3.2.2) and ideally bounded so that the inputs to subsequent layers remain small (what can prevent numerical problems).

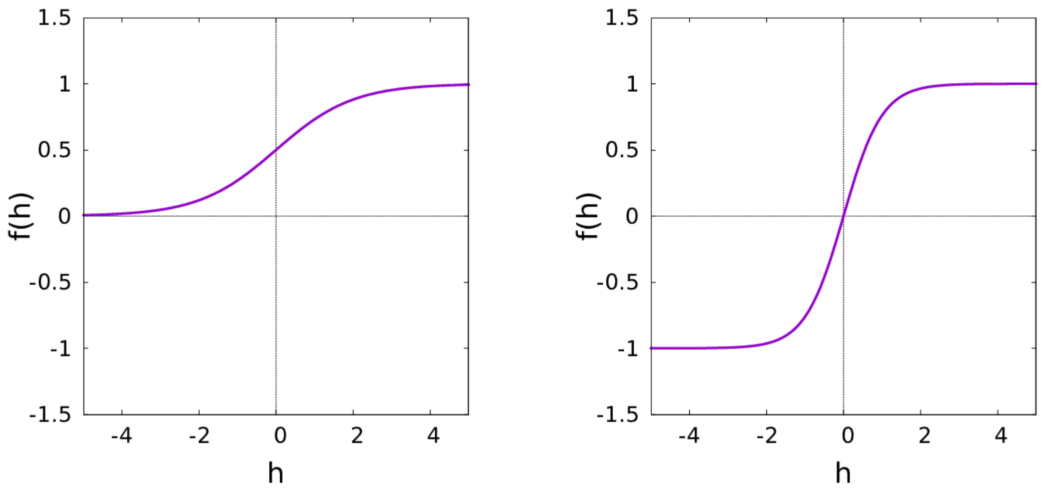
Real-valued activation functions

Typical activation functions in a real MLP are so called sigmoids. They are everywhere differentiable and saturate when their net input h tends to $\pm\infty$. Two common real-valued sigmoids are the logistic and the hyperbolic tangent functions, defined respectively as

$$f(h) = 1/(1 + e^{-h}) \tag{3.21}$$

$$f(h) = \tanh(h) = \frac{e^h - e^{-h}}{e^h + e^{-h}} \tag{3.22}$$

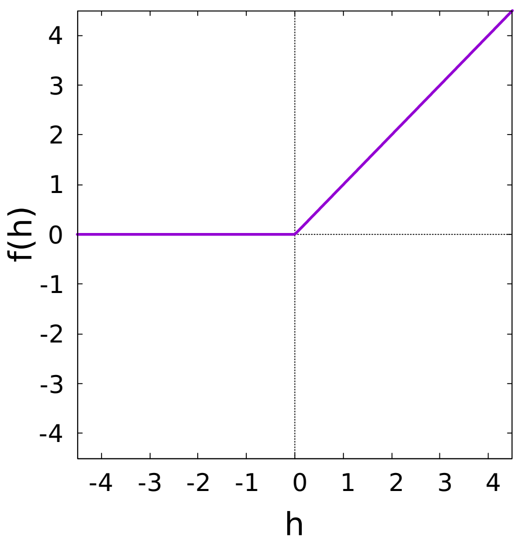
Figure 3.1: Left: logistic, right: tanh



Recent works show that activation functions that are neither analytic nor bounded may also be used, provided inputs are properly normalized and the network uses some mechanism to limit the neurons' outputs. Such a function is the rectifier function rectifier(x) = max$(0, x)$ (neurons using it are so called *Rectified Linear Units*, short *ReLU*). As can be seen in Figure 3.2, this function has a non-differentiable spot at $x = 0$, and for $x > 0$ it grows linearly. Since an input should rarely be exactly zero, and by applying some kind of regularization technique to control the neuron's output, this kind of functions can successfully be used as activation functions in hidden layers. The lack of saturation for positive inputs and the resulting sparse network – since, for zero centred inputs, approximately half of the neurons won't fire – may lead to good learning properties of the network, as shown by Glorot, Bordes and Bengio (2011).

Another advantage of the rectifier function is its simple computation at the feed-forward and back-propagation stages (since its derivative is 1 for positive inputs, 0 otherwise). The zero derivative can lead to a problem though. Namely if a unit initially doesn't activate, its weights won't be adjusted (since the corresponding derivative is 0). Maas, Hannun and Ng (2013) suggested a simple change allowing a small gradient for

Figure 3.2: Rectifier function



non-active units. The *leaky ReLU* is thus defined as:

$$f(h) = \begin{cases} h, & h > 0 \\ 0.01h, & h \leq 0 \end{cases} \tag{3.23}$$

The derivatives for the listed real-valued functions are following:

$$\frac{d\,\text{tanh}(h)}{dh} = 1 - \text{tanh}^2(h) \tag{3.24}$$

$$\frac{d\,\text{logistic}(h)}{dh} = \frac{1}{1+e^{-h}}\left(1 - \frac{1}{1+e^{-h}}\right) \tag{3.25}$$

$$\frac{d\,\text{rectifier}(h)}{dh} = \begin{cases} 1, & h > 0, \\ 0, & h \leq 0 \end{cases} \tag{3.26}$$

$$\frac{d\,\text{leaky-rectifier}(h)}{dh} = \begin{cases} 1, & h > 0, \\ 0.01, & h \leq 0 \end{cases} \tag{3.27}$$

Complex-valued activation functions

According to the Liouville theorem, an entire complex function [1] must be constant (GEORGIOU; KOUTSOUGERAS, 1992). Thus a proper activation function in the

[1]A function is said to be entire if it is at the same time holomorphic (complex differentiable) and bounded.

complex domain must fulfil other set of requirements. Georgiou and Koutsougeras (1992) propose that an activation function $f(z) = u(\Re z, \Im z) + iv(\Re z, \Im z)$, where $\Re z$ and $\Im z$ are the real and imaginary parts of z respectively, shall satisfy following conditions:

1. $f(z)$ is non-linear in $\Re z$ and $\Im z$.
2. $f(z)$ is bounded.
3. $f(z)$ has bounded partial derivatives: $\frac{\partial u}{\partial \Re z}, \frac{\partial u}{\partial \Im z}, \frac{\partial v}{\partial \Re z}, \frac{\partial v}{\partial \Im z}$.
4. $f(z)$ is not entire (due to the Liouville theorem).
5. $\frac{\partial u}{\partial \Re z}\frac{\partial v}{\partial \Im z} \neq \frac{\partial v}{\partial \Re z}\frac{\partial u}{\partial \Im z}$

One set of functions satisfying those conditions are those that apply real-valued activation functions separately to the real and imaginary parts. For instance:

- split-logistic: $f(z) = \frac{1}{1+e^{-\Re h}} + i\frac{1}{1+e^{-\Im h}}$
- split-tanh: $f(z) = \tanh(\Re z) + i\ \tanh(\Im z)$
- split-rectifier: $f(z) = \max(0, \Re z) + i\ \max(0, \Im z)$
- split-leaky-rectifier: $f(z) = \max(0.01\Re z, \Re z) + i\ \max(0.01\Im z, \Im z)$

Split-tanh is a widely used function in CV-MLPs. Split-rectifier and split-logistic are mentioned in Popa (2017, p. 820).

Figure 3.3: Split-logistic. Left: real part; right: imaginary part

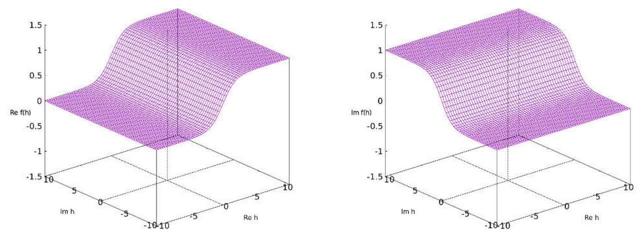
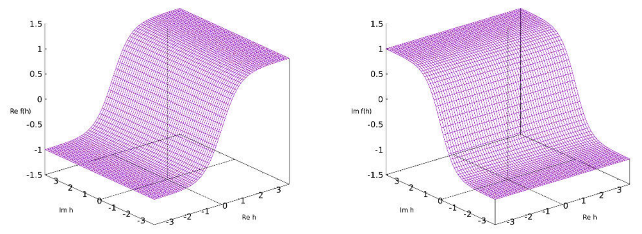


Figure 3.4: Split-tanh. Left: real part; right: imaginary part



The other approach is to use analytic but unbounded functions. Kim and Adali (2002) investigate several elementary transcendental functions for this matter and show

Figure 3.5: Split-rectifier. Left: real part; right: imaginary part

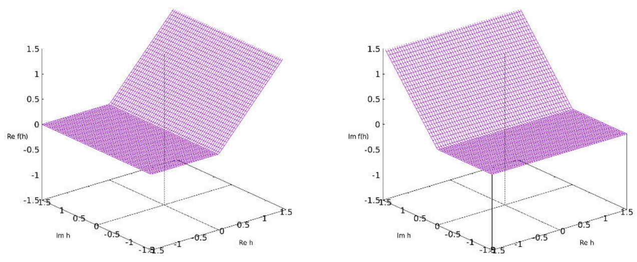
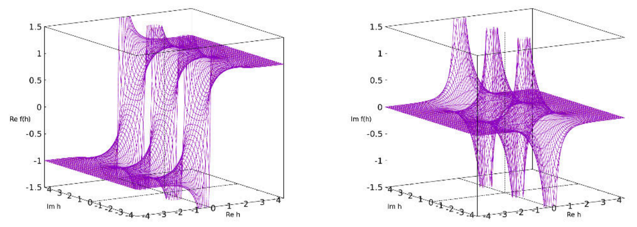


Figure 3.6: Complex tanh. Left: real part; right: imaginary part



good results by using the complex tanh function:

$$f(h) = \tanh(h) = \frac{e^h - e^{-h}}{e^h + e^{-h}}, h \in \mathbb{C} \tag{3.28}$$

As can be seen in Figures 3.6 and 3.7, this function has singularities at every $\frac{1}{2+n}\pi, n \in \mathbb{N}$. By carefully adapting the inputs (say by applying a sigmoid function like the tanh before feeding them to the MLP) these points could be avoided.

Using the rules from Wirtinger Calculus (see Appendix A.1), the derivatives of the listed activation functions are following:

- split-tanh:

$$\frac{\partial f(h)}{\partial h} = \frac{1}{2}\left(2 - \tanh^2(\Re h) - \tanh^2(\Im h)\right) \tag{3.29}$$

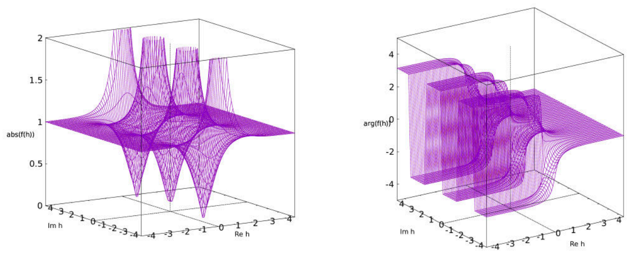
$$\frac{\partial f(h)}{\partial h^*} = \frac{1}{2}\left(1 - \tanh^2(\Re h) + \tanh^2(\Im h)\right) \tag{3.30}$$

- split-logistic:

$$\frac{\partial f(h)}{\partial h} = \frac{1}{2}\left(\Re f(z)(1 - \Re f(z)) + \Im f(z)(1 - \Im f(z))\right) \tag{3.31}$$

$$\frac{\partial f(h)}{\partial h^*} = \frac{1}{2}\left(\Re f(z)(1 - \Re f(z)) + \Im f(z)(1 - \Im f(z))\right) \tag{3.32}$$

Figure 3.7: Complex tanh. Left: magnitude; right: phase



- split-rectifier:

$$\frac{\partial f(h)}{\partial h} = \begin{cases} 1, & \Re h > 0 \text{ and } \Im h > 0, \\ 1/2, & \Re h > 0 \text{ and } \Im h \leq 0, \\ 1/2, & \Re h \leq 0 \text{ and } \Im h > 0, \\ 0, & \Re h \leq 0 \text{ and } \Im h \leq 0, \end{cases} \tag{3.33}$$

$$\frac{\partial f(h)}{\partial h^*} = \begin{cases} 0, & \Re h > 0 \text{ and } \Im h > 0, \\ 1/2, & \Re h > 0 \text{ and } \Im h \leq 0, \\ -1/2, & \Re h \leq 0 \text{ and } \Im h > 0, \\ 0, & \Re h \leq 0 \text{ and } \Im h \leq 0, \end{cases} \tag{3.34}$$

- split-leaky-rectifier:

$$\frac{\partial f(h)}{\partial h} = \begin{cases} 1, & \Re h > 0 \text{ and } \Im h > 0, \\ 1.01/2, & \Re h > 0 \text{ and } \Im h \leq 0, \\ 1.01/2, & \Re h \leq 0 \text{ and } \Im h > 0, \\ 0.02, & \Re h \leq 0 \text{ and } \Im h \leq 0, \end{cases} \tag{3.35}$$

$$\frac{\partial f(h)}{\partial h^*} = \begin{cases} 0, & \Re h > 0 \text{ and } \Im h > 0, \\ 0.99/2, & \Re h > 0 \text{ and } \Im h \leq 0, \\ -0.99/2, & \Re h \leq 0 \text{ and } \Im h > 0, \\ 0, & \Re h \leq 0 \text{ and } \Im h \leq 0, \end{cases} \tag{3.36}$$

- tanh (complex):

$$\frac{\partial f(h)}{\partial h} = 1 - \tanh^2(h) \tag{3.37}$$
$$\frac{\partial f(h)}{\partial h^*} = 0 \tag{3.38}$$
$$\tag{3.39}$$

3.4 Error function

The global error consists of a summation of several individual error functions applied to the output neurons of an MLP after feed-forwarding training inputs. Thus these should all have the same sign, otherwise they could cancel each other out. In the real case, a typical choice is the squared error function:

$$err_{\mathbb{R}}\left(\mathbf{y_L}(\mathbf{z}^{(\alpha)}, \mathbf{t}^{(\alpha)})\right) = \frac{1}{2}\sum_{k=1}^{N_L}(y_k - t_k)^2 \tag{3.40}$$

In the complex case a real-valued function (but with complex arguments) is needed in order to have a comparable objective function to minimize. Hänsch (2010) investigates four different functions and comes to the conclusion that for more complex topologies they all lead to a similar performance. Thus this work only considers following *complex quadratic error function*:

$$err\left(\mathbf{y_L}(\mathbf{z}^{(\alpha)}, \mathbf{t}^{(\alpha)})\right) = \sum_{k=1}^{N_L}(y_k - t_k)(y_k - t_k)^* \tag{3.41}$$

According to Eq. 3.19 this requires the derivatives $\frac{\partial err}{\partial y_i^l}$ and $\frac{\partial err}{\partial y_i^{l*}}$. In the case of the complex quadratic error function, those are following:

$$\begin{aligned}
\frac{\partial err}{\partial y_i^l} &= \frac{\partial \sum_{k=1}^{N_L}(y_k - t_k)(y_k - t_k)^*}{\partial y_i^l} \\
&\overset{(1)}{=} \frac{\partial (y_i^l - t_i)(y_i^l - t_i)^*}{\partial y_i^l} \\
&\overset{(2)}{=} (y_i^l - t_i)^* \frac{\partial (y_i - t_i)}{\partial y_i^l} = (y_i^l - t_i)^* && (3.42) \\
\frac{\partial err}{\partial y_i^{l*}} &= \frac{\partial \sum_{k=1}^{N_L}(y_k - t_k)(y_k - t_k)^*}{\partial y_i^{l*}} \\
&\overset{(1)}{=} \frac{\partial (y_i^l - t_i)(y_i^l - t_i)^*}{\partial y_i^{l*}} \\
&\overset{(2)}{=} (y_i^l - t_i) \frac{\partial (y_i - t_i)^*}{\partial y_i^{l*}} = (y_i^l - t_i) && (3.43)
\end{aligned}$$

where (1) uses the fact that the derivative of $(y_k^l - t_k)(y_k^l - t_k)^*$ with respect to y_i^l or y_i^l is 0 unless $i = k$, and (2) uses Eq. A.4.

3.4.1 Weight decay

One way to improve generalization is to add a *regularization term* to the error function so that large weights get penalized. For instance:

$$E'_{\{w^l\}}(\mathbf{D}) = E_{\{w^l\}}(\mathbf{D}) + \frac{1}{2}\lambda \sum_k w_k^2 \tag{3.44}$$

where λ is the *regularization* factor [2]. By taking the negative gradient of this function we obtain the *weight decay* update rule:

$$\Delta w_{ji}^l(t) = -\nabla E_{\{w\}}(\mathbf{D}) - \lambda w_{ji}^l(t-1) \tag{3.45}$$

This technique allows the network to choose the smallest set of weights that solve the optimization problem. This is desirable since it forces net inputs to get closer to zero, away from the saturation point of some activation functions (MARSLAND, 2015, p. 84). Another benefit is that it prevents the network from fitting the noise from the training data (KROGH; HERTZ, 1992, p. 954).

[2] For simplicity we slightly abuse notation by indexing all elements of the matrix with a single variable k.

3.5 Input normalization

Several transformation strategies of input data can lead to faster convergence of the network. Two common ones are to center data so that features across the training samples have zero mean, and scaling it so that the variance of features are all approximately the same. As explained by LeCun et al. (2012, p. 16), centered data improves learning since it allows weights to be adjusted in different directions in a turn. By scaling inputs to force a specific variance we also ensure that all weights can be changed at a similar rate.

3.6 Batch Learning and Stochastic Gradient Descent

The delta rule as described considers the global error function of all training samples in order to decide how to adjust the weights, what is known as *batch* learning. An alternative approach called *stochastic gradiend learning* (short SGD, also known as *online learning*) updates weights after each single training sample. Thus in the complex case Eq. 3.7 becomes:

$$\Delta w_{ji}^{l}(t) = -2\frac{\partial err(\mathbf{y}^{L}(\mathbf{z}^{(\alpha)}, \mathbf{t}^{(\alpha)}))}{\partial {w_{ji}^{l}}^{*}} \tag{3.46}$$

As explained at LeCun et al. (2012, p. 13), stochastic learning leads to a noisy estimate of the gradient. Since the investigated non-linear functions usually have multiple local minima, this property helps the network to "jump" between different local minima, whereas batch learning estimates the gradient towards the local minimum determined by the (randomly) initialized weights.

A possible trade-off is to consider groups of training samples at each update, which are called *mini-batches*. Following this strategy, Eq. 3.6 becomes:

$$\Delta w_{ji}^{l}(t) = \frac{2}{N}\sum_{\alpha=1}^{N}\frac{\partial err(\mathbf{y}^{L}(\mathbf{z}^{(\alpha)}, \mathbf{t}^{(\alpha)}))}{\partial {w_{ji}^{l}}^{*}} \tag{3.47}$$

where N is the batch size, which is smaller than the total number of available training samples P.

An important difference between these approaches is that SGD and mini-batches require shuffling the training inputs at each epoch, so that each sample can influence the network differently depending in which order it was seen. Offline learning doesn't have this requirement, since here weights get updated according to the sum of all gradients.

3.7 Weight initialization

For similar reasons that inputs are normalized so that features have zero mean and unit variance, weights can be initialized in such a way that, combined with proper activation functions, each layer receives inputs with these same overall characteristics. A simple strategy for achieving this is the *Xavier initialization* proposed by Glorot and Bengio (2010). It consists of initializing the weights in a way that following holds:

$$Var(\mathbf{w}^l) = \frac{2}{N_l + N_{l+1}} \tag{3.48}$$

Weights are chosen according to a uniform distribution. Given an interval $[a, b]$, the standard deviation from a uniform distribution is given by $\frac{b-a}{\sqrt{12}}$. Thus

$$\frac{b-a}{\sqrt{12}} = \sqrt{\frac{2}{N_l + N_{l+1}}} \tag{3.49}$$

$$\Rightarrow b - a = \sqrt{\frac{24}{N_l + N_{l+1}}} \tag{3.50}$$

By choosing $-a = b$, so that the distribution is centred around zero, we obtain:

$$2b = \sqrt{\frac{24}{N_l + N_{l+1}}} \tag{3.51}$$

$$\Rightarrow b = \frac{\sqrt{6}}{\sqrt{N_l + N_{l+1}}} \tag{3.52}$$

Thus we initialize each individual weight at layer l with a random value from following uniform distribution:

$$U\left[-\frac{\sqrt{6}}{\sqrt{N_l + N_{l+1}}}, -\frac{\sqrt{6}}{\sqrt{N_l + N_{l+1}}}\right] \tag{3.53}$$

By using the property $Var(Z) = Var(\Re Z) + Var(\Im Z)$ (see proof at Eq. A.6) and by choosing $Var(\Re Z) = Var(\Im Z)$, the Xavier initialization can be easily adapted to the complex case:

$$Var(Z) = 2Var(\Re Z) = \frac{2}{N_l + N_{l+1}} \tag{3.54}$$

$$\Rightarrow Var(\Re Z) = \frac{1}{N_l + N_{l+1}} \tag{3.55}$$

Following the same logic as in the real case, we want to choose b so that

$$2b = \sqrt{\frac{12}{N_l + N_{l+1}}} \tag{3.56}$$

$$\Rightarrow b = \frac{\sqrt{3}}{\sqrt{N_l + N_{l+1}}} \tag{3.57}$$

So the real and imaginary part of each weight are chosen according to the uniform distribution:

$$U\left[-\frac{\sqrt{3}}{\sqrt{N_l + N_{l+1}}}, -\frac{\sqrt{3}}{\sqrt{N_l + N_{l+1}}}\right] \tag{3.58}$$

3.8 Learning strategies

The chosen learning rate not only directly influences speed of convergence (as can be seen in Eq. 3.6) but also determines if the network will converge at all (see LeCun et al. (2012) for a detailed discussion about learning rate and convergence). Besides choosing different learning rates according to the layer size, several learning strategies may be used in order to improve convergence. Some of them are briefly described next.

3.8.1 Momentum

The delta rule can be changed as follows:

$$\Delta w_{ji}^l(t) = \alpha \Delta w_{ji}^l(t-1) - \nabla E_{\{w\}}(\mathbf{D}) \tag{3.59}$$

$\alpha \Delta w_{ji}^l(t-1)$ is called *momentum term*, and α the momentum constant (usually set to a value close to 0.9). This helps stabilizing training by accelerating it when the learning surface has a low curvature and slowing it down otherwise.

3.8.2 Adagrad

The *Adaptive Gradient Algorithm* (short Adagrad) (DUCHI; HAZAN; SINGER, 2011) scales the global learning rate on a per weight basis:

$$\Delta w^l_{ji}(t) = -\frac{\nabla E_{\{w_{ji}\}}(\mathbf{D}_t)}{\sqrt{\sum_i^t \nabla E_{\{w_{ji}\}}(\mathbf{D}_i)}} \tag{3.60}$$

As can be seen at Eq. 3.60, the scaling factor is inversely proportional to the accumulated l_2 norm of all gradients computed until now, practically giving "frequently occurring features very low learning rates and infrequent features high learning rates" (DUCHI; HAZAN; SINGER, 2011, p. 2122).

3.8.3 Adam

As in the case of Adagrad, *Adaptive Moment Estimation* (short Adam) (KINGMA; BA, 2014) also scales a global learning rate on a per-parameter base, in this case by using moving averages of the mean and uncentered variance of the gradient. As experimentally shown by the authors, it converges faster than Adagrad and Stochastic Gradient Descent.

The update rule is changed as follows:

$$w^l_{ji}(t+1) = w^l_{ji}(t) + \mu \frac{\widehat{\mathbb{E}}_{t+1}[g]}{\sqrt{\widehat{Var}_{t+1}[g] + \epsilon}} \tag{3.61}$$

$$\mathbb{E}_{t+1}[g] = \beta_1 \cdot \mathbb{E}_t[g] + (1-\beta_1) \cdot g_{t+1} \tag{3.62}$$

$$Var_{t+1}[g] = \beta_2 \cdot Var_t[g] + (1-\beta_2) \cdot g^2_{t+1} \tag{3.63}$$

$$\widehat{\mathbb{E}}_{t+1}[g] = \frac{\mathbb{E}_{t+1}[g]}{1-\beta_1^{t+1}} \tag{3.64}$$

$$\widehat{\text{Var}}_{t+1}[g] = \frac{\text{Var}_{t+1}[g]}{1-\beta_2^{t+1}} \tag{3.65}$$

where g_t is the gradient $\nabla E_{\{w\}}(\mathbf{D})$ measured at instant t, ϵ is a stabilization factor and β_1 and β_2 are the decay rates (typically 0.9 and 0.999 respectively). Equations 3.64 and 3.65 correct the initialization bias, since without them the moment estimates have a bias towards zero (see Kingma and Ba (2014)).

4 EVALUATION

4.1 Experiment Description

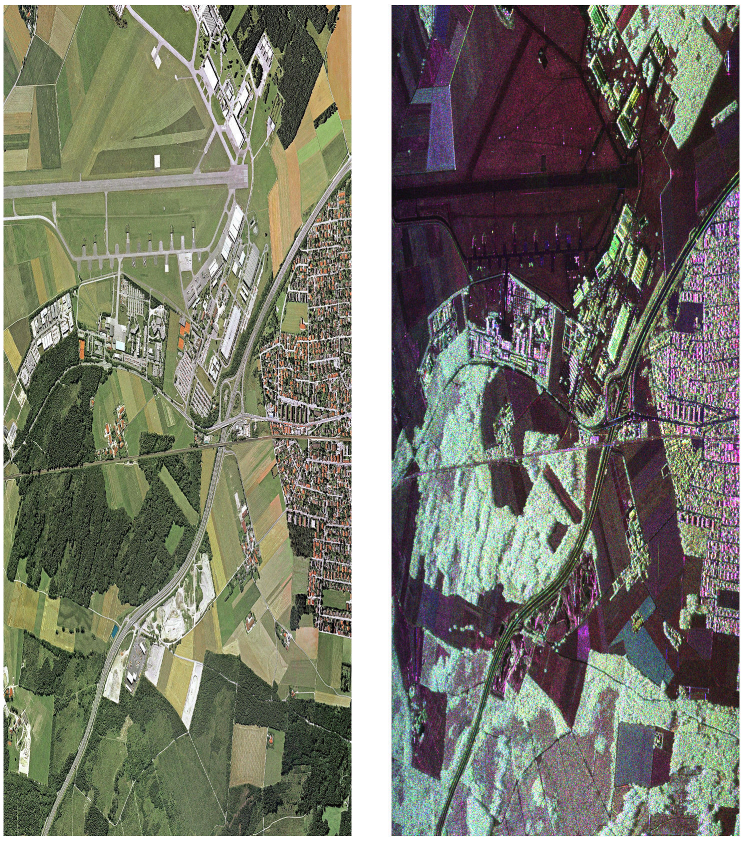
The purpose of these experiments is to investigate the performance of the CV-MLP described in Chapter 3 in the task of general object recognition from PolSAR images. We are specially interested in its approximation capabilities compared to a RV-MLP, and whether the selected improvements influence not only the overall performance, but also the relative learning behaviour to a RV-MLP. This requires an implementation of a MLP that allows switching between both variants without major changes to the underlying structure and mechanisms. Therefore a complete MLP supporting different activation functions, adaptive learning rate algorithms, different learning strategies (Batch, Mini-Batch and SGD) as well as Xavier initialization was implemented from scratch, since as far as we know no such an implementation is freely available.

4.1.1 Data

A dataset provided by DLR is used for training and evaluation. It consists of a PolSAR image of the city Oberpfaffenhofen taken by the E-SAR sensor (Figure 4.1 right) and five target images showing the classes of each pixel. These are illustrated on the left side of Figure 4.11 using following color scheme: roads in blue, fields in yellow, forests in dark green, shrubland in light green and city in red. Ideally multiple images for training and others for validation/testing should be used, but differences in acquisition conditions and processing techniques make the generation of such a dataset a difficult task. Nevertheless an image of the city Alling (also provided by DLR) is used for comparison purposes at the end of this chapter.

The manual classification of images also requires some care. These images were independently classified by several experts, and only pixels where all experts agreed where considered for training and evaluation. Assigning them a separate class would mislead the model, since they might belong to one of the known classes while not being labeled as such due to disagreements between the experts. Non-labeled pixels are represented in Figure 4.11 in black.

Figure 4.1: Oberpfaffenhofen dataset (provided by DLR). Left: optical image; right: PolSAR image



4.1.1.1 Data format and pre-processing

Before extracting feature vectors as inputs, the whole image was first converted to its covariance matrix (see Eq. 2.2) representation and underwent a multi-look processing step using a window of 4 pixels along range (horizontal) direction and 2 pixels along the azimuth (vertical) direction, both steps done with the RAT software (REIGBER; HELLWICH, 2004). As explained in Section 2.3.1 this merges several pixels in a 4x2 window, which thus requires a decision regarding which target to attribute to resulting pixels. We decided to pick the most frequent class inside the window, since it is representative for more elements contained in the respective area.

4.1.1.2 Features

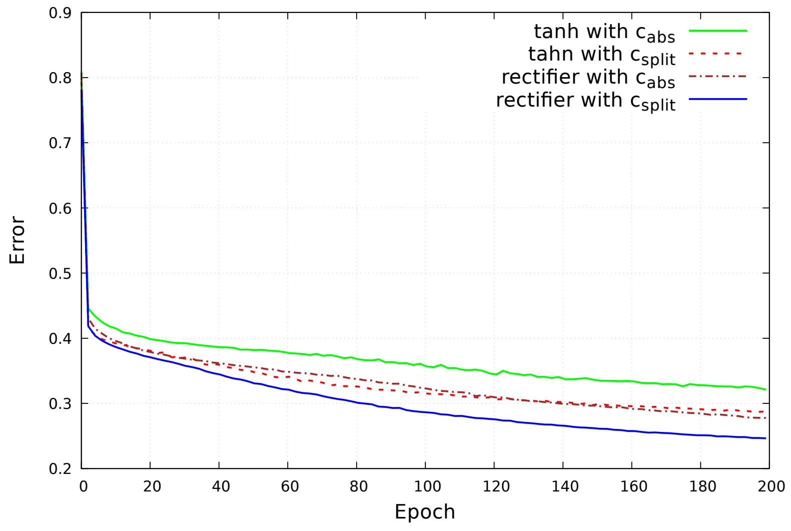
Since the covariance matrix obtained in Section 4.1.1.1 is hermitian (i.e. it's equal to its own conjugate transpose), the 3x3 matrix has redundant information. In order to reduce dimensionality of the input vectors, the values below the diagonal elements are ignored, since they are the conjugate versions of the numbers above it. Each pixel is thus described as a vector $\mathbf{c}$ containing 6 complex numbers (HÄNSCH, 2010):

$$\mathbf{c} = (c_{11}, c_{12}, c_{13}, c_{22}, c_{23}, c_{33})^T \tag{4.1}$$

where c_{ij} is the element at row i and column j from the covariance matrix $\mathbf{C}$ (see Eq. 2.2).

In order to feed this information into the RV-MLP, these features have to be mapped to some real-valued feature vector. There are several strategies for this. We tested two common approaches, namely constructing a vector $\mathbf{c}_{abs}$ with the magnitudes of the elements from $\mathbf{c}$, as done in Hänsch (2010), and by using real-valued vectors containing the real and imaginary parts concatenated $\mathbf{c}_{split}$. Our experiments show that the second approach delivered better results (see Figure 4.2 for an example), suggesting that information lost in $\mathbf{c}_{abs}$, most obviously the phase of the signal, is indeed relevant for object recognition tasks. Since the goal is to evaluate both MLPs under similar conditions, this work opted to use $\mathbf{c}_{split}$ in most of its comparisons, thus giving the RV-MLP the full complex information as separate real values. Table 4.2 summarizes the test errors obtained using each of these feature vectors.

Figure 4.2: Train error of a RV-MLP using a 50 neurons hidden-layer with Adagrad with varying transfer functions and input vectors.



To provide the network with more contextual information, 3x3 patches of these vectors are used as input images. Each pixel is thus described by its own value and its surrounding neighbours.

4.1.1.3 Sample choice

In order to train under more realistic conditions, the whole image was first divided in 5 stripes in order to apply a 5-fold cross-validation, as suggested by Hänsch (2014, p. 149). As noted by the author, these would be ideally horizontal stripes, assuring that each stripe has objects in far and near range (the satellite moved along the y-direction). Due to the class distribution though, this would make stripes with too few samples of some classes for some folds. From each stripe, 3x3 patches were extracted as described above. From these patches, a set of approximately 2400 patches from each class was randomly selected per stripe. Hence the whole data set consists of 5 sets of ca. 12000 patches each. This approach helps avoiding that the network gets biased towards dominant classes due to the distribution present in this specific dataset. This also has the benefit of limiting the variety of data, since a whole stripe can be selected for testing purposes while no sample contained in it is seen during training. This resembles real usage conditions, since we can't suppose that a MLP has access to samples from all kinds of a class. For instance, some forests may differ considerably from others, while belonging to the same class, but it is improbable that the algorithm will have samples from all kinds of forest for training. Nevertheless the model should be able to generalize and correctly handle the diversity of data.

4.1.1.4 Normalization

Before starting training, the whole training set is normalized to obtain zero mean and unit variance. This is done by subtracting from each feature its mean across all training samples and dividing by the corresponding standard deviation. Each feature mean and standard variance obtained is stored in order to normalise testing inputs later. This ensures that features present in both datasets are mapped to the same value after normalisation, while also avoiding that information contained in the test dataset is seen during training.

4.1.1.5 Targets

Our targets for the Oberpfaffenhofen dataset are one-hot encoded. They are represented thus as a five-element vector of real values or complex values, depending on which MLP variant is used. The vector contains at the index corresponding to the target class the value of $1+i$ (1 in the real case), towards which the sigmoids studied saturate. The choice of the value for the non-class indices depends on the activation function used in the output layer. If it is a (split-)logistic function, then they are set to 0. If it is a (split-)tanh function, they are set to $-1-i$ (or -1 if real), exploiting the saturation of the (split-)tanh function in the opposite direction. As discussed in Section 4.1.4.1, these experiments don't use other transfer functions in the output layer.

4.1.2 Evaluation Methodology

For evaluation of the model cross validation is used. More precisely, we reserve one of the stripes for testing and use the selected patches from the remaining stripes for training. After training, the whole test image is classified. This is done five times, each turn using a different test image. Unless stated otherwise, the performance measurements are obtained by repeating each experiment several times and averaging the results. This allows us to better understand the network behaviour regardless of the specific combination of randomly initialized parameters and train order of the images.

4.1.2.1 Metric

Since testing is done over a complete image stripe, we need a performance metric that not only deals with multi-class problems but also with extremely unbalanced data. As explained by Hänsch (2014, p. 143), the typically used overall accuracy measure consisting of the number of correctly classified pixels divided by the total of pixels could lead to overly optimistic performance results, for example by simply ignoring labels from rarely appearing categories. A more adequate metric is the *balanced accuracy* (BRODERSEN et al., 2010) defined as:

$$\mathrm{BA}(T) = \sum_{i}^{|T|} \sum_{k}^{K} \frac{v_k(T,i)}{C_k \cdot K} \tag{4.2}$$

where T is the test image, $|T|$ its number of pixels, K the number of classes, $C_k(T)$ the number of samples of class k in the test image and $v_k(T,i)$ is the indicator function defined as:

$$v_k(T,i) := \begin{cases} 1, & \text{sample } i \text{ in image } T \text{ belongs to class } k \\ 0, & \text{otherwise} \end{cases} \tag{4.3}$$

This metric attributes to each correctly classified sample a weight inversely proportional to the class frequency and reflects thus the generalization ability of the model.

We also shall derive a balanced error (BE) metric directly from the balanced accuracy: When in these experiment training or test error is mentioned, the balanced error is meant.

$$\text{BE(T)} = 1 - \text{BA}(T) \tag{4.4}$$

4.1.3 Plot characteristics

Plots in this chapter show the interpolated training error measured at the start of each epoch. The reason for showing the training error instead of the test error is that these plots are meant to illustrate the learning process and the ability of the specific algorithm to approximate the underlying function. Because the prepared training set is balanced, the BE measurement delivers the same result as the unbalanced error $1 - \sum_i^{|T|} \sum_k^K \frac{v_k(T,i)}{K}$, which is therefore used as error measure in the plots.

Since we are also interested in the generalization ability of our model, the results discussed later in this chapter use the balanced test error.

4.1.4 MLP Setup

As stated our purpose it to investigate different combinations of activation functions (complex or not) and some selected improvements. Due to time constrains we also had to restrict the hyper-parameter choice, so that longer evaluation experiments were possible. This requires several trial and error in order to find suitable hyper-parameters for this classification problem. Next the decisions made are described.

4.1.4.1 Transfer functions

Different experiments were done using several activation functions in hidden units. It should be noted though that not all function listed in Section 3.3 are adequate for output neurons. For instance the unbounded output of a ReLU unit leads to large error values which the MLP can't handle properly. The complex-tanh function has the same problem. As Figures 3.6 and 3.7 show, the function is not only unbounded, but has periodical singularities, what makes choosing proper target values difficult.

For these reasons bounded functions are used at output layers, for example the sigmoid functions and their split complex variants. Table 4.1.4.1 summarizes the combinations of hidden and output activation functions investigated:

Hidden layer	Output layer
tanh (real)	tanh (real)
rectifier	logistic $^{(1)}$
complex-tanh	split-tanh
split-tanh	split-tanh
split-rectifier	split-logistic $^{(1)}$

In (1) the (split-)logistic is used instead of (split-)tanh because preceding layers only output non-negative values. The same approach is used by Popa (2017, p. 820).

The leaky-rectifier and split-leaky-rectifier functions didn't perform significantly different than the rectifier functions. A possible reason is the fact that the topologies used were rather small, in which case the problem of dying ReLUs is rather unlikely. Therefore in these experiments only the simpler, "non-leaky" rectifier functions were considered.

4.1.4.2 Learning strategy and corresponding hyper-parameters

Each learning strategy requires a different set of hyper-parameters. In our experiments, for instance, the learning rate for the Adam strategy had to differ by two orders of magnitude from the learning rate chosen for Adagrad. Inappropriate learning rates would lead to a divergent learning behaviour.

After some experimenting, we settled on following hyper-parameter selection:

- Fixed learning rate: $\mu = 0.001$
- Fixed learning rate with momentum and weight decay: $\mu = 0.001, \alpha = 0.9, \lambda = 1^{-8}$
- Adagrad: $\mu = 0.01$
- Adam: $\mu = 0.0001, \beta_1 = 0.9, \beta = 0.999$

where recall that μ is the learning rate, α the momentum constant, λ the weight decay factor, β_1 and β_2 are the decay rates.

4.1.4.3 Topology

Our experiments use MLPs with three different topologies: a single hidden layer with 50 neurons, a single hidden layer with 10 neurons and two hidden layers with 10 neurons each. Although at first sight small sized, these topologies seemed to provide enough variability of number of neurons and layers for our concrete experimental setup, which uses feature vectors extracted from relatively small image patches as described above.

4.1.4.4 Training time and convergence

All of our tests are restricted to 200 epochs, during which no over-fitting effect was noticed (the performance over the test dataset was monitored to see whether this occurred). Although some setups clearly converge during this time, others show that longer training periods would possibly lead to better results. The performance increase towards epoch 200 is relatively small though. As stated above, each experiment was repeated several times and averaged, unless stated otherwise.

4.2 Results and Discussion

4.2.1 Momentum and weight decay

The use of momentum didn't bring significant changes to the experiments using adaptive learning rate. For a fixed learning rate of 0.001, adding a momentum term lead to worse results, what can be seen in Figure 4.3. Possible reasons for this behaviour are that the chosen learning rate is large enough to overcome local minima, and that the usually adopted moment constant of 0.9 was too large for this problem.

We also couldn't see any major influence of weight decay in our experiments.

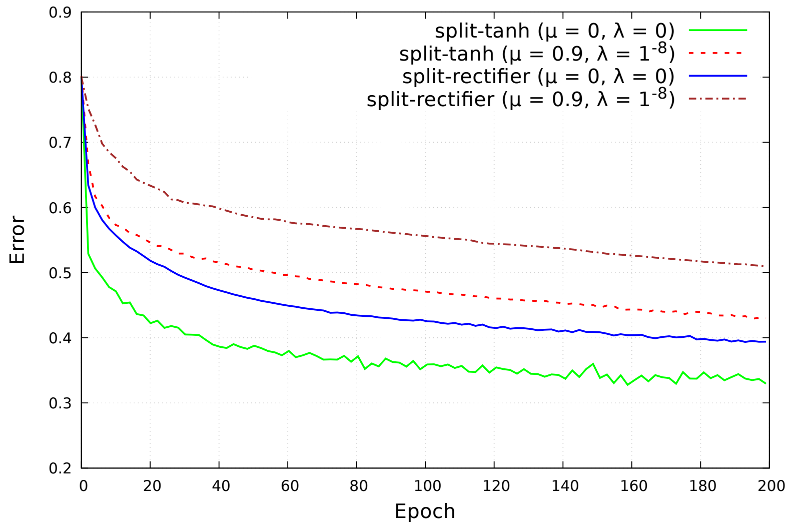
For these reasons, further experiments don't use these techniques to avoid irrelevant computations.

4.2.2 Stochastic Gradient Descent and Batch strategy

Figure 4.4 shows the train error curve of three runs using online learning, mini-batch of size 64 and offline learning. All of them use split-rectifier in 50-sized hidden layer, split-logistic in the output layer and a fixed learning rate.

Offline learning provided a extremely slow decreasing error. A possible reason for this behaviour is that gradients with different signs cancel each other out, decreasing

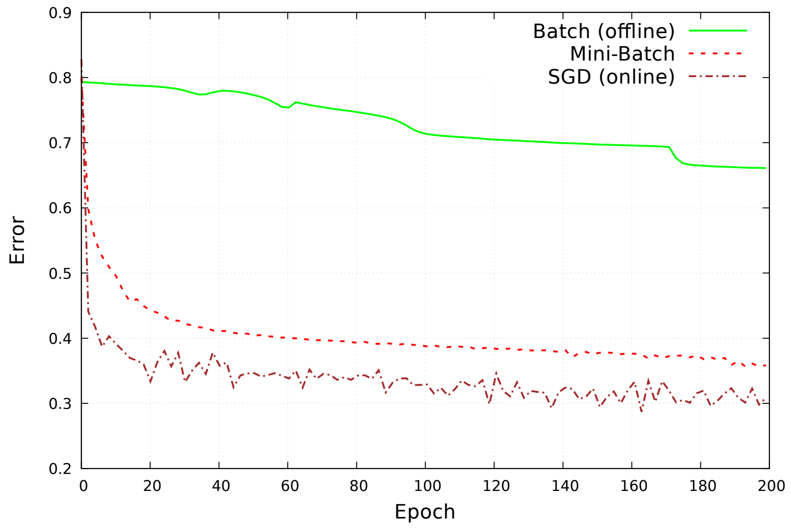
Figure 4.3: Train error using fixed learning rate with momentum and weight decay (single run)



the adjustments steps needed to achieve a faster convergence. The curve also illustrates the problem of a fixed learning rate. The small increment of the error around epoch 38 is possibly caused by a too big learning rate for that stage of learning, causing the short period of divergence.

Stochastic Gradient Descent was able to learn considerably faster than the other approaches, but showed a rather noisy error curve. We found that mini-batches provided the best trade-off between learning speed and training time. Our examples use therefore mini-batches with 64 input samples each.

Figure 4.4: Comparison of SGD, Mini-Batch and Batch Learning. All plots use split-rectifier and fixed learning rate $\mu = 0.001$. Mini-batches have 64 samples each.



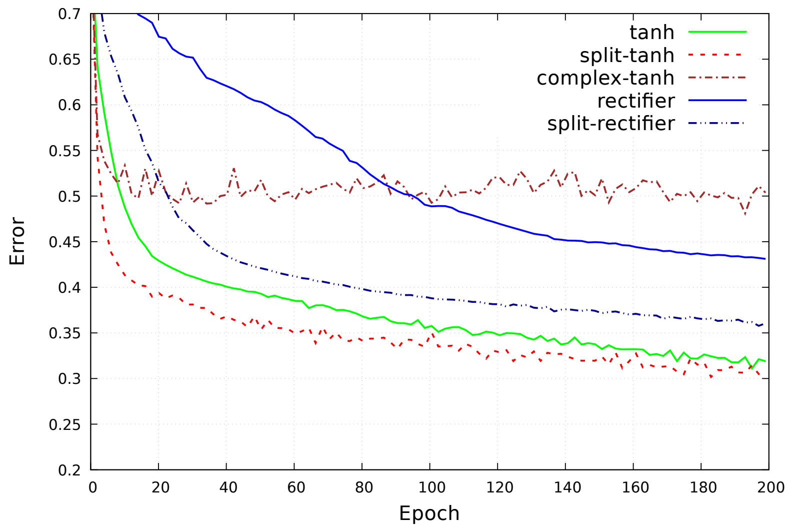
4.2.3 Learning Strategy

As seen in section 3.8, Adam and Adagrad both improve learning by using separate learning rates for each parameter and adapting them according to some strategy.

This could be verified in our experiments. By comparing Figures 4.5 to 4.7 it becomes evident that the use of adaptive learning leads to a faster convergence and a more stable learning behaviour. Table 4.1 also shows how both Adagrad and Adam clearly outperform a network with fixed learning rate. This is expected, since they adopt individual learning rates for the parameters and change them based on the relative frequency of the features.

The evolution of the error curve shows that Adam and Adagrad converge at similar rates, and after 200 epochs both achieved similar performances. We could verify a small advantage of Adagrad in our runs (see Table 4.1).

Figure 4.5: Two hidden layers with 10 and 10 neurons each. Fixed learning rate



4.2.4 Transfer Functions

As Figures 4.5 and 4.8 illustrate, with fixed learning rate the tanh family of functions clearly outperform MLPs using rectifier functions (with the exception of the combination of two hidden layers and complex-tanh discussed below). A possible cause for this behaviour could be an inadequate learning rate choice.

Networks using adaptive learning showed a similar learning progression regardless of transfer function used, with ReLUs leading to a faster convergence and clearly better

Figure 4.6: Two hidden layers with 10 and 10 neurons. Adagrad

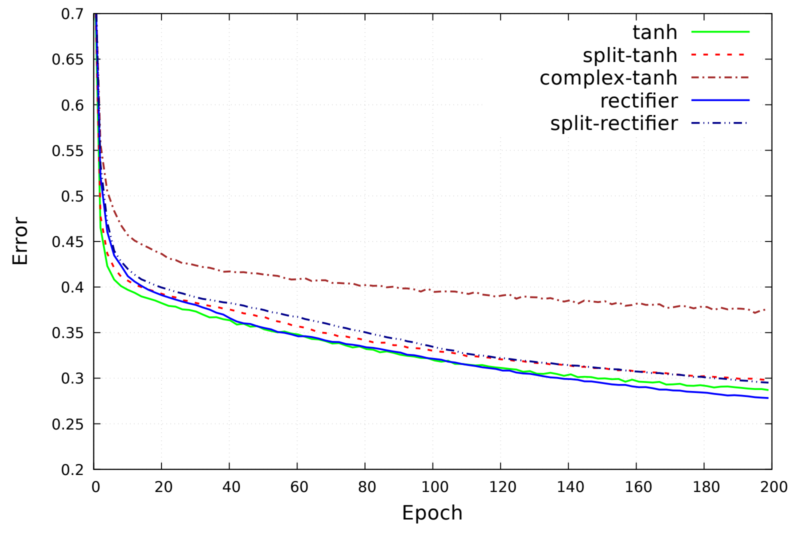


Figure 4.7: Two hidden layers with 10 and 10 neurons. Adam

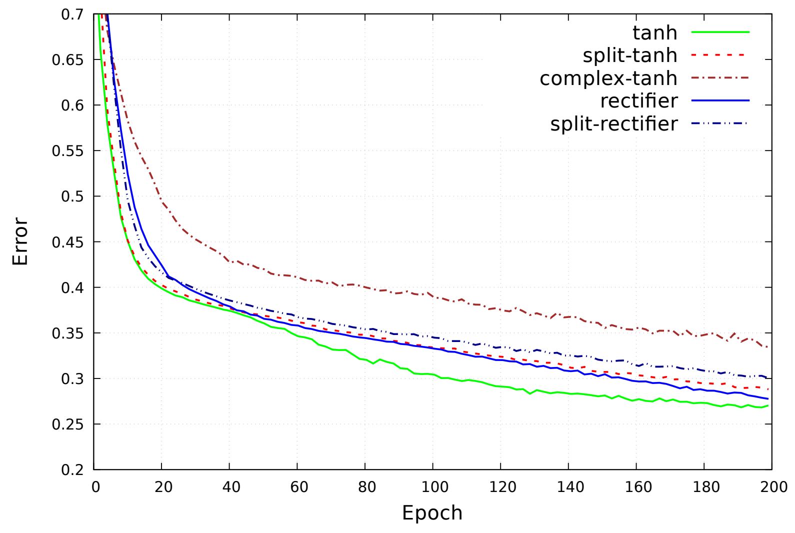


Figure 4.8: One hidden layer with 50 neurons. Fixed learning rate

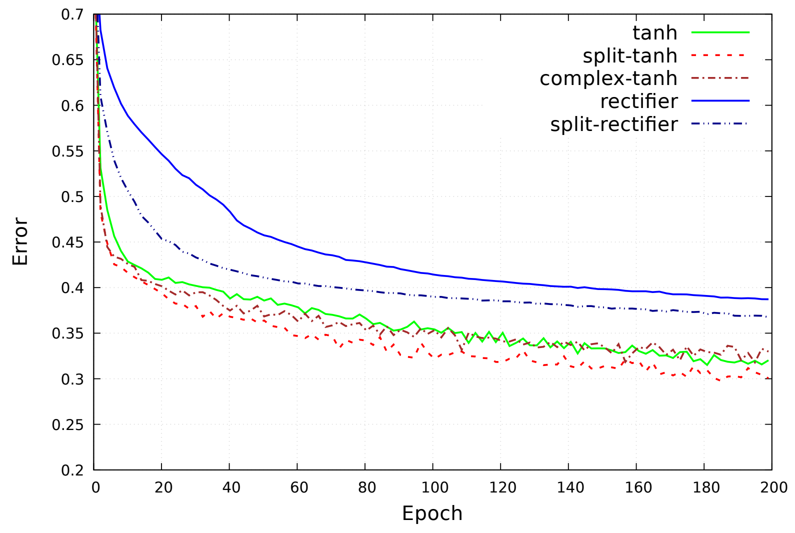


Figure 4.9: One hidden layer with 50 neurons. Adagrad

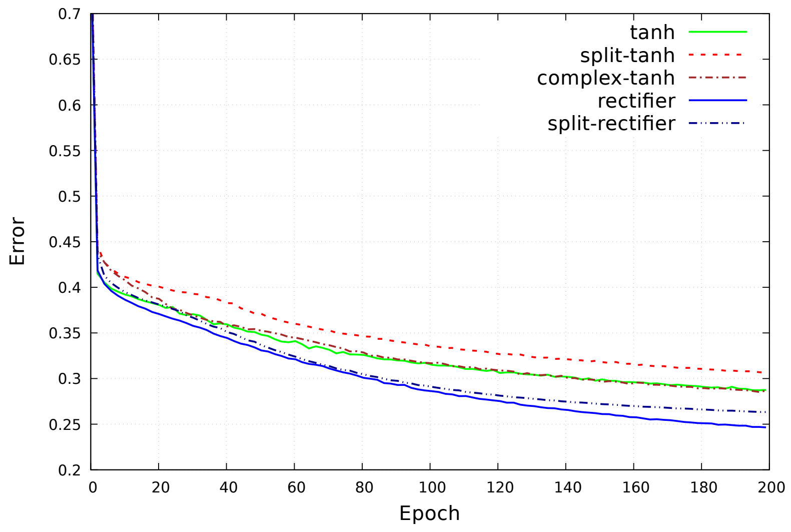
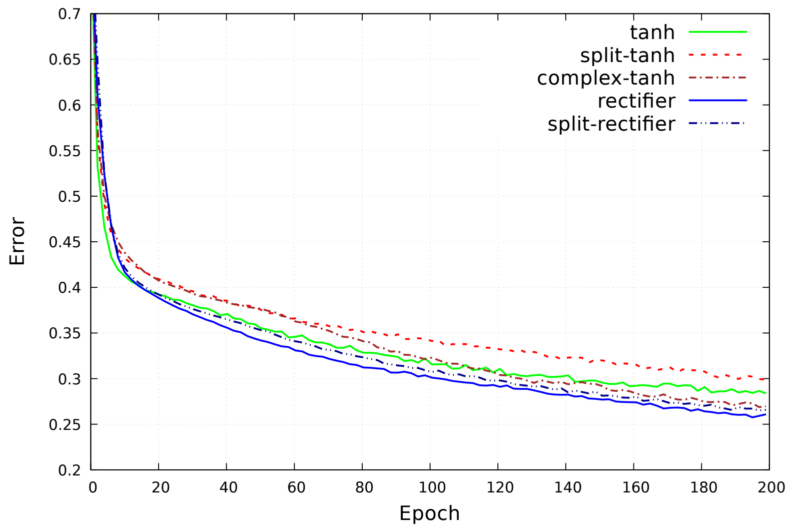


Figure 4.10: One hidden layer with 50 neurons. Adam



results when combined with Adagrad and a bigger hidden layer (see the corresponding test errors in Table 4.1).

The use of complex-tanh in hidden layers – combined with split-tanh for the output layer – lead to rather unstable results. When combined with adaptive learning and a single hidden layer, it could achieve similar results to those obtained by the split-tanh. But in the experiments with fixed learning rate or two hidden layers its performance was much worse. This behaviour is possibly explained by the singularities and overall behaviour of the complex-tanh function shown in Section 3.3, which could require extra measures in the hidden layers in order to properly use this function. Nevertheless the positive influence of adaptive learning suggests that further exploring of complex analytic functions should adopt such strategies.

Table 4.1: Balanced test error after 200 epochs obtained by CV-MLP. Best scores are underlined.

One hidden layer (10)

	split-rectifier	split-tanh	tanh (ℂ)
Fixed	41.5%±1.6%	37.8%±2.2%	41.2%±2.5%
Adagrad	33.2%±1.0%	33.1%±1.1%	36.1%±2.4%
Adam	$\underline{32.0\%\pm0.6\%}$	32.3%±1.6%	33.5%±1.3%

Two hidden layers (10-10)

	split-rectifier	split-tanh	tanh (ℂ)
Fixed	40.3%±0.9%	36.8%±2.3%	56.1%±2.3%
Adagrad	$\underline{30.7\%\pm1.5\%}$	31.1%±0.8%	40.7%±2.4%
Adam	34.8%±2.2%	32.2%±1.6%	38.7%±2.1%

One hidden layer (50)

	split-rectifier	split-tanh	tanh (ℂ)
Fixed	40.2%±1.0%	34.7%±2.8%	40.6%±2.1%
Adagrad	$\underline{27.5\%\pm0.5\%}$	31.1%±0.7%	31.8%±1.3%
Adam	30.1%±1.3%	32.7%±1.4%	31.6%±1.0%

Table 4.2: Balanced test error after 200 epochs obtained by RV-MLP. Best scores are underlined.

One hidden layer (10)

	rectifier ($\mathbf{c_{abs}}$)	tanh ($\mathbf{c_{abs}}$)	rectifier ($\mathbf{c_{split}}$)	tanh ($\mathbf{c_{split}}$)
Fixed	46.7%±3.6%	43.7%±1.5%	47.58%±2.1%	39.8%±1.4%
Adagrad	34.1%±2.2%	34.8%±1.8%	31.6%±1.7%	30.5%±0.7%
Adam	$\underline{34.2\%\pm2.6\%}$	36.4%±2.8%	$\underline{30.7\%\pm1.6\%}$	32.3%±1.5%

Two hidden layers (10-10)

	rectifier ($\mathbf{c_{abs}}$)	tanh ($\mathbf{c_{abs}}$)	rectifier ($\mathbf{c_{split}}$)	tanh ($\mathbf{c_{split}}$)
Fixed	45.8%±1.8%	40.3%±3.4%	47.6%±2.9%	35.8%±2.0%
Adagrad	33.6%±2.5%	33.1%±2.2%	29.6%±1.3%	30.6%±2.0%
Adam	35.2%±2.3%	$\underline{32.3}$%±1.3%	31.8%±1.8%	29.3%±0.8%

One hidden layer (50)

	rectifier ($\mathbf{c_{abs}}$)	tanh ($\mathbf{c_{abs}}$)	rectifier ($\mathbf{c_{split}}$)	tanh ($\mathbf{c_{split}}$)
Fixed	43.2%±2.6%	38.4%±2.4%	41.4%±1.9%	34.4%±1.6%
Adagrad	30.1%±1.5%	35.5%±1.8%	$\underline{26.5\%\pm0.6\%}$	30.3%±0.6%
Adam	35.2%±3.2%	38.4%±2.4%	30.3%±2.5%	31.0%±1.3%

4.2.5 Comparison between Complex-valued MLP and Real-valued MLP

Our implementation allows a direct comparison of a RV-MLP and a CV-MLP using the same dataset preparation and under similar conditions. The results vary depending on topology, learning rate approach, transfer function and feature vector used.

We notice that the CV-MLP performed overall better than a RV-MLP when fixed learning rate is used, at least when the topology contains rather smaller hidden layers (say 10 neurons). This happens independently of feature vector used, a result already shown by previous works (HÄNSCH, 2010).

As soon as adaptive learning is used though, we noticed that a RV-MLP using $\mathbf{c}_{split}$ could achieve similar or slightly better results than a CV-MLP. The same RV-MLP using $\mathbf{c}_{abs}$ still obtained worse results than a CV-MLP.

Table 4.3 contains confusion matrices of the CV-MLP and RV-MLP obtained from a single run with (split-)rectifier, Adagrad and a 50-sized hidden layer. As above, both networks were trained during 200 epochs over the Oberpfaffenhofen dataset using cross-validation. Each confusion matrix consists of a merge of confusion matrices regarding each image stripe used for testing. Each row i and column j represents the rate of samples from class i attributed to class j. The main diagonal contains thus the rate of correctly classified samples from each class. In this single run example, the CV-MLP achieved a BA of 72.8$\%$, while the RV-MLP 73.4$\%$.

Table 4.3: Confusion matrix regarding classification of the Oberpfaffenhofen data set. Results obtained after 200 epochs using (split)-rectifier and Adagrad. One hidden layers with 50 neurons were used.

CV-MLP

	City	Field	Forest	Shrubland	Road
City	**0.53**	0.06	0.17	0.19	0.05
Field	0.00	**0.68**	0.00	0.07	0.24
Forest	0.09	0.00	**0.83**	0.08	0.00
Shrubland	0.03	0.04	0.05	**0.87**	0.00
Road	0.04	0.17	0.00	0.05	**0.74**

RV-MLP

	City	Field	Forest	Shrubland	Road
City	**0.58**	0.06	0.16	0.17	0.03
Field	0.01	**0.78**	0.00	0.10	0.11
Forest	0.09	0.00	**0.84**	0.07	0.00
Shrubland	0.03	0.03	0.05	**0.88**	0.00
Road	0.05	0.30	0.00	0.05	**0.59**

The confusion matrix from Table 4.3 illustrates the difficulty of the network to learn some specific classes, for instance the samples containing the city. A possible cause could be a higher in-class variety regarding scattering patterns. This fact may be a consequence of the sample distribution in this concrete example. For instance, most samples of class *City* are concentrated on the predominantly residential area at the right-side of the image (which is almost entirely contained in a single test image) while samples from other areas are dominated by industrial buildings.

Figure 4.11 shows the classification results obtained by the CV-MLP, with the manually labeled image on the left side for visual comparison. The right-side image represents correctly classified pixels in green, and wrongly classified in red. Black pixels are non-classified pixels.

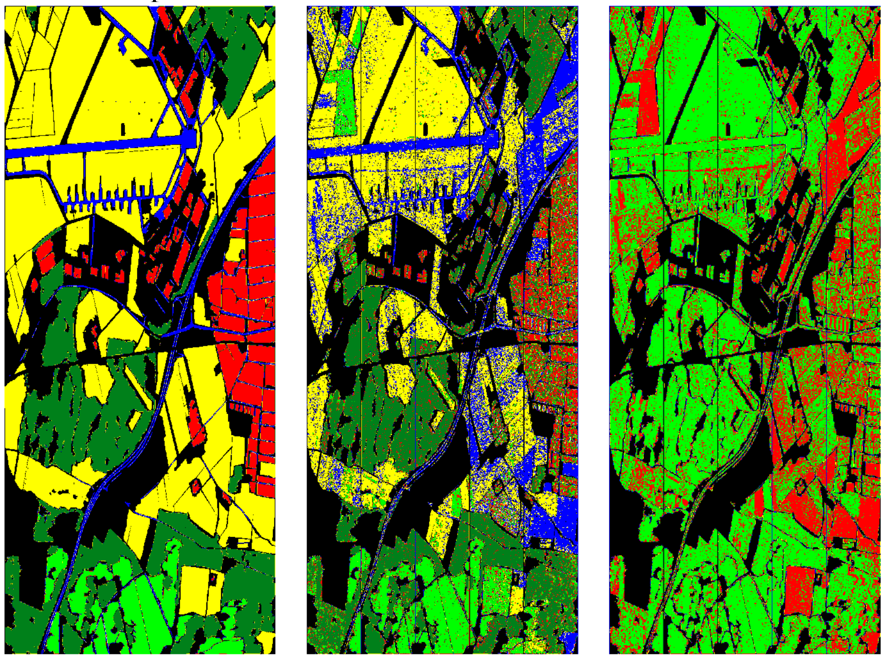
For further references, Table 4.4 illustrates the performance of both MLP's for categorization of the dataset of the city Allign, also provided by the DLR. The CV-MLP achieved a BA of 61.3%, while the RV-MLP 63.4%.

Table 4.4: Confusion matrix regarding classification of the Allign data set. Results obtained under same conditions as with the Oberpfaffenhofen dataset.

CV-MLP				
	City	Field	Forest	Road
City	**0.42**	0.02	0.39	0.18
Field	0.02	**0.85**	0.00	0.13
Forest	0.20	0.00	**0.74**	0.05
Road	0.13	0.36	0.06	**0.44**

RV-MLP				
	City	Field	Forest	Road
City	**0.40**	0.02	0.41	0.17
Field	0.01	**0.86**	0.01	0.11
Forest	0.16	0.00	**0.80**	0.04
Shrubland	0.16	0.33	0.04	**0.47**

Figure 4.11: Classification result from Oberpfaffenhofen dataset (DLR) using a CV-MLP with a single 50-sized hidden layer, Adagrad and split-rectifier after 200 epochs training. Left: manually labeled image; center: classification obtained by CV-MLP; right: classification comparison.
Color scheme for left and center images: road (blue), field (yellow), forest (dark green), shrubland (light green), city (red).
Color scheme for right image: correctly classified (green), misclassified (red).
In all schemes black pixels are non-classified.



5 CONCLUSION

This work investigated the merits of using a CV-MLP supporting different transfer functions, modern initialization strategies and adaptive learning rates for the task of object recognition in PolSAR images. It focused on adapting a selection of techniques usually applied to RV-MLPs to a CV-MLP and evaluating their influence on its performance for this task.

Early works (HÄNSCH, 2010) showed that a basic implementation of a CV-MLP could achieve similar results to those of a RV-MLP, what raised the question about how a more sophisticated CV-MLP would perform. Thus our first contribution was to select some well known techniques used in current RV-MLP implementations, implement them in a CV-MLP when needed and evaluate their benefits for general object recognition in PolSAR images. Most techniques could be directly applied to a CV-MLP, as the adaptive learning strategies. An exception is Xavier Initialization, that had to be slightly changed in order to achieve its goals.

This work also investigated the unbounded split-rectifier function seen in Popa (2017), which showed a faster convergence and better overall approximation capabilities when the topology had a bigger hidden layer. This result suggests that the split-rectifier could be a promising transfer function for bigger networks, where real-valued rectifier functions are commonly used.

We also have shown how adaptive learning rate algorithms as Adagrad and Adam allow a CV-MLP to much better categorize PolSAR images. These techniques also leveraged performance differences due to other factors considered, such as the transfer function choice or even whether a RV-MLP or a CV-MLP is used for this task.

Regarding the question of the relative performance of a CV-MLP and a RV-MLP, our results confirm that simpler implementations of both algorithms lead to a slightly advantage for a CV-MLP. The use of adaptive learning though makes them perform similarly, at least when using non-analytic functions that apply a real-valued function separately to the real and imaginary parts of its net input.

This work also investigated the complex-tanh function in combination with a split-tanh function in the output layer. Its results varied considerably, performing relatively well when using adaptive learning rate and a single hidden layer, and achieving much worse results otherwise. Future works could further investigate analytic, but unbounded complex functions, specially in combination with normalization techniques to avoid the pitfalls of these functions. Another path would involve exploring a polar representation of the complex data, as suggested in Hirose (2006). The use of Adagrad or Adam is advisable in these cases, since they avoid that fixed hyper-parameters of the model negatively impact the MLP's approximation abilities.

REFERENCES

AMIN, M. F. et al. **Complex-valued neural networks: learning algorithms and applications**. Thesis (PhD) — PhD. Thesis, University of Fukui, 2012.

ANTROPOV, O. et al. Land cover and soil type mapping from spaceborne PolSAR data at l-band with probabilistic neural network. **IEEE Transactions on Geoscience and Remote Sensing**, IEEE, v. 52, n. 9, p. 5256–5270, 2014.

BISHOP, C. M. **Neural networks for pattern recognition**. [S.l.]: Oxford university press, 1995.

BOVOLO, F.; BRUZZONE, L. A context-sensitive technique based on support vector machines for image classification. **Pattern Recognition and Machine Intelligence**, Springer, p. 260–265, 2005.

BRODERSEN, K. H. et al. The balanced accuracy and its posterior distribution. In: IEEE. **Pattern recognition (ICPR), 2010 20th international conference on**. [S.l.], 2010. p. 3121–3124.

DRUDE, L.; RAJ, B.; HAEB-UMBACH, R. On the appropriateness of complex-valued neural networks for speech enhancement. In: **INTERSPEECH**. [S.l.: s.n.], 2016. p. 1745–1749.

DUCHI, J.; HAZAN, E.; SINGER, Y. Adaptive subgradient methods for online learning and stochastic optimization. **Journal of Machine Learning Research**, v. 12, n. Jul, p. 2121–2159, 2011.

GEORGIOU, G. M.; KOUTSOUGERAS, C. Complex domain backpropagation. **IEEE transactions on Circuits and systems II: analog and digital signal processing**, IEEE, v. 39, n. 5, p. 330–334, 1992.

GLOROT, X.; BENGIO, Y. Understanding the difficulty of training deep feedforward neural networks. In: **Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics**. [S.l.: s.n.], 2010. p. 249–256.

GLOROT, X.; BORDES, A.; BENGIO, Y. Deep sparse rectifier neural networks. In: **Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics**. [S.l.: s.n.], 2011. p. 315–323.

HÄNSCH, R. Complex-valued multi-layer perceptrons–an application to polarimetric SAR data. **Photogrammetric Engineering & Remote Sensing**, American Society for Photogrammetry and Remote Sensing, v. 76, n. 9, p. 1081–1088, 2010.

HÄNSCH, R. **Generic object categorization in PolSAR images-and beyond**. [S.l.: s.n.], 2014.

HÄNSCH, R.; HELLWICH, O. Classification of polarimetric SAR data by complex valued neural networks. In: **Proceedings of ISPRS workshop**. [S.l.: s.n.], 2009.

HÄNSCH, R.; HELLWICH, O. Object recognition from polarimetric SAR images. In: **Radar Remote Sensing of Urban Areas**. [S.l.]: Springer, 2010. p. 109–131.

HIROSE, A. **Complex-valued neural networks**. [S.l.]: Springer Science & Business Media, 2006.

HUNGER, R. **An introduction to complex differentials and complex differentiability**. [S.l.]: Munich University of Technology, Inst. for Circuit Theory and Signal Processing, 2007.

JAIN, R.; KASTURI, R.; SCHUNCK, B. G. **Machine vision**. [S.l.]: McGraw-Hill New York, 1995.

KIM, M. S.; GUEST, C. C. Modification of backpropagation networks for complex-valued signal processing in frequency domain. In: IEEE. **Neural Networks, 1990., 1990 IJCNN International Joint Conference on**. [S.l.], 1990. p. 27–31.

KIM, T.; ADALI, T. Fully complex multi-layer perceptron network for nonlinear signal processing. **Journal of VLSI signal processing systems for signal, image and video technology**, Springer, v. 32, n. 1-2, p. 29–43, 2002.

KINGMA, D.; BA, J. Adam: A method for stochastic optimization. **arXiv preprint arXiv:1412.6980**, 2014.

KROGH, A.; HERTZ, J. A. A simple weight decay can improve generalization. In: **Advances in neural information processing systems**. [S.l.: s.n.], 1992. p. 950–957.

LECUN, Y. A. et al. Efficient backprop. In: **Neural networks: Tricks of the trade**. [S.l.]: Springer, 2012. p. 9–48.

LEE, J.-S.; GRUNES, M. R.; GRANDI, G. D. Polarimetric SAR speckle filtering and its implication for classification. **IEEE Transactions on Geoscience and remote sensing**, IEEE, v. 37, n. 5, p. 2363–2373, 1999.

MAAS, A. L.; HANNUN, A. Y.; NG, A. Y. Rectifier nonlinearities improve neural network acoustic models. In: **Proc. ICML**. [S.l.: s.n.], 2013. v. 30, n. 1.

MAGHSOUDI, Y. Analysis of radarsat-2 full polarimetric data for forest mapping. **Degree of PhD, Department of Geomatics Engineering, University of Calgary**, 2011.

MARSLAND, S. **Machine learning: an algorithmic perspective**. [S.l.]: CRC press, 2015.

MASJEDI, A.; ZOEJ, M. J. V.; MAGHSOUDI, Y. Classification of polarimetric SAR images based on modeling contextual information and using texture features. **IEEE Transactions on Geoscience and Remote Sensing**, IEEE, v. 54, n. 2, p. 932–943, 2016.

MINSKY, M.; PAPERT, S. Perceptrons: An introduction to computational geometry. **MA: MIT Press, Cambridge**, 1969.

PETERSEN, K. B.; PEDERSEN, M. S. et al. The matrix cookbook. **Technical University of Denmark**, v. 7, p. 15, 2008.

PINZ, A. Object categorization. **Foundations and Trends® in Computer Graphics and Vision**, Now Publishers Inc., v. 1, n. 4, p. 255–353, 2005.

POPA, C.-A. Complex-valued convolutional neural networks for real-valued image classification. In: IEEE. **Neural Networks (IJCNN), 2017 International Joint Conference on**. [S.l.], 2017. p. 816–822.

REIGBER, A.; HELLWICH, O. Rat (radar tools): A free SAR image analysis software package. In: **Proceedings of EUSAR**. [S.l.: s.n.], 2004. v. 4, p. 997–1000.

RUMELHART, D. E. et al. Learning representations by back-propagating errors. **Cognitive modeling**, v. 5, n. 3, p. 1, 1988.

SERPICO, S. B.; BRUZZONE, L.; ROLI, F. An experimental comparison of neural and statistical non-parametric algorithms for supervised classification of remote-sensing images. **Pattern recognition letters**, Elsevier, v. 17, n. 13, p. 1331–1341, 1996.

VIRTUE, P.; YU, S. X.; LUSTIG, M. Better than real: Complex-valued neural nets for mri fingerprinting. **arXiv preprint arXiv:1707.00070**, 2017.

ZEBKER, H. A.; ZYL, J. J. V. Imaging radar polarimetry: A review. **Proceedings of the IEEE**, IEEE, v. 79, n. 11, p. 1583–1606, 1991.

ZYL, J. J. van. **Synthetic aperture radar polarimetry**. [S.l.]: John Wiley & Sons, 2011.

APPENDIX A — COMPLEX PROPERTIES

A.1 Wirtinger Derivatives

As known from complex analysis, a complex function $f(z)$ is differentiable in an open domain region if it satisfies the *Cauchy-Riemann* equation (AMIN et al., 2012):

$$\frac{\partial f(z)}{\partial \Im z} = i\frac{\partial f(z)}{\partial \Re z} \tag{A.1}$$

Non-analytic complex functions can't be derived with the usual rules of analysis. One can though compute their partial derivatives according to the rules of Wirtinger Calculus:

$$\frac{df(z)}{dz} = \frac{1}{2}\left(\frac{\partial f(z)}{\partial \Re z} - i\frac{\partial f(z)}{\partial \Im z}\right) \tag{A.2}$$

$$\frac{df(z)}{dz^*} = \frac{1}{2}\left(\frac{\partial f(z)}{\partial \Re z} + i\frac{\partial f(z)}{\partial \Im z}\right) \tag{A.3}$$

For holomorphic functions, the Wirtinger derivatives have the nice property that they satisfy the Cauchy-Riemann equations, since $\frac{\partial f(z)}{\partial z^*} = 0$.

Another important result it that "z^* can be regarded as a constant when differentiating with respect to z, as well as z can be regarded constant when differentiating with respect to z^* (HUNGER, 2007, p. 10):

$$\frac{\partial z^*}{\partial z} = \frac{\partial z}{\partial z^*} = 0 \tag{A.4}$$

Also useful for deriving complex functions are following relations (HUNGER, 2007, p. 10):

$$\frac{\partial f^*(z)}{\partial z} = \left(\frac{\partial f(z)}{\partial z^*}\right)^* \text{ and } \frac{\partial f^*(z)}{\partial z^*} = \left(\frac{\partial f(z)}{\partial z}\right)^* \tag{A.5}$$

A.2 Complex Random Variable

$$\begin{aligned}
Var[Z] &= \mathbb{E}(|(Z-\mathbb{E}(Z))^2|) \\
&= \mathbb{E}[(Z-\mathbb{E}(Z))(Z-\mathbb{E}(Z))^*] \\
&= \mathbb{E}[ZZ^* - Z\mathbb{E}(Z)^* - \mathbb{E}(Z)Z^* + \mathbb{E}(Z)\mathbb{E}(Z)^*] \\
&\overset{(1)}{=} \mathbb{E}(ZZ^*) - \mathbb{E}(Z)\mathbb{E}(Z)^* - \mathbb{E}(Z)\mathbb{E}(Z)^* + \mathbb{E}(Z)\mathbb{E}(Z)^* \\
&= \mathbb{E}(ZZ^*) - \mathbb{E}(Z)\mathbb{E}(Z)^* \\
&= \mathbb{E}[(\Re Z)^2 + (\Im Z)^2] - [\Re\mathbb{E}(Z)]^2 - [\Im\mathbb{E}(Z)]^2 \\
&\overset{(2)}{=} \mathbb{E}[(\Re Z)^2] + \mathbb{E}[(\Im Z)^2] - \mathbb{E}[(\Re Z)]^2 - \mathbb{E}[(\Im Z)]^2 \\
&\overset{(1)}{=} \mathbb{E}[(\Re Z)^2] - \mathbb{E}[(\Re Z)]^2 + \mathbb{E}[(\Im Z)^2] - \mathbb{E}[(\Im Z)]^2 \\
&= Var[\Re Z] + Var[\Im Z] \qquad \text{(A.6)}
\end{aligned}$$

where (1) uses the linearity of the expectation and (2) the additivity of the real and imaginary operators.

A.3 Mathematical notation

z^*	complex conjugate
$\Re z$	real part of complex number z
$\Im z$	imaginary part of complex number z
$\mathbb{R}$	Real domain
$\mathbb{C}$	Complex domain
$\mathbf{k}$	simplified scattering vector
$\mathbf{S}$	Scattering matrix
$\mathbf{C}$	Covariance matrix
$\mathbf{c}$	vector containing the unique (modulo complex conjugate) elements of $\mathbf{C}$
$\mathbf{c}_{abs}$	Real-valued feature vector containing the magnitudes of the vector $\mathbf{c}$
$\mathbf{c}_{split}$	Real-valued feature vector containing the real and imaginary parts of the vector $\mathbf{c}$
$\mathbf{D}$	Training set
$\frac{\partial f(h)}{\partial h}$	partial derivative of f with respect to h
$\mathbb{E}(Z)$	expected value of random variable Z
$Var(Z)$	variance of random variable Z