Neural Processing Letters
https://doi.org/10.1007/s11063-019-10140-7

N2SkyC: User Friendly and Efficient Neural Network Simulation Fostering Cloud Containers

Aliaksandr Adamenko[1] · Andrii Fedorenko[1] · Benjamin Nussbaum[1] · Erich Schikuta[1]

Abstract
Sky computing is a new computing paradigm leveraging resources of multiple Cloud providers to create a large scale distributed infrastructure. N2Sky is a research initiative promising a framework for the utilization of Neural Networks as services across many Clouds. This involves a number of challenges ranging from the provision, discovery and utilization of services to the management, monitoring, metering and accounting of the infrastructure. Cloud Container technology offers fast deployment, good portability, and high resource efficiency to run large-scale and distributed systems. In recent years, container-based virtualization for applications has gained immense popularity. This paper presents the new N2SkyC system, a framework for the utilization of Neural Networks as services, aiming for higher flexibility, portability, dynamic orchestration, and performance by fostering microservices and Cloud container technology.

Keywords Problem solving environment · Neural networks · Cloud computing · Containers · Microservices

1 Introduction

The Cloud computing paradigm provides access to a set of computational power by aggregating heterogeneous resources and software and offering them as a single system composition. It hides the details of implementation and management of software and hardware from the

✉ Erich Schikuta
erich.schikuta@univie.ac.at
http://www.cs.univie.ac.at

Aliaksandr Adamenko
alexadamenko@gmail.com

Andrii Fedorenko
andriifedorenko@gmail.com

Benjamin Nussbaum
benjamin.nussbaum@gmail.com

[1] University of Vienna, Faculty of Computer Science, Währingerstr. 29, 1090 Vienna, Austria

Published online: 29 October 2019

end user. Cloud computing has been evolved from technologies like Cluster computing and Grid computing and has given rise to Sky computing [11].

Sky computing combines multiple Cloud-based infrastructures in such a way that the Sky providers aggregate the services scattered across several Clouds thus becoming the consumers of the Cloud providers. Sky computing in this way copes with the problem of vendor lock-in and extends the flexibility, transparency, and elasticity of the integrated infrastructure as compared to that of a single Cloud. Sky computing has taken another step forward towards the realization of virtual collaborations, where resources are logical, and solutions are virtual. The exchange of information and resources among researchers is one driving stimulus for development. That is just as valid for the neural information processing community as for any other research community. As described by the UK e-Science initiative [6] several goals can be reached by the usage of new stimulating techniques, such as enabling more efficient and seamless collaboration of dispersed communities, both scientific and commercial.

In the course of our research we designed and developed N2SkyC, a micro-service oriented Cloud container enabled neural network simulation environment, which is based on N2Sky [21], a virtual organization for the computational intelligence community. It provides access to neural network resources and enables infrastructures fostering multi Cloud resources. On the one hand, neural network resources can be generic neural network objects trained by a specific learning paradigm and training data for given problems whereas on the other hand these objects can also represent already trained neural networks, which can be used for given application problems. The vision of N2SkyC is the provisioning of a neural network problem solving virtual organization where any member of the community can access or contribute neural network objects all over the Internet.

In this paper, we present the new N2SkyC. The original system was developed as a monolithic application. That proved feasible for image-based virtualization mechanisms as in classical Cloud infrastructures. However, by the actual advent of containers and microservice architecture aiming for higher flexibility, portability, dynamic orchestration, and performance there was a need for a redesign of N2SKy to foster this new technology. By interpreting N2SkyC as a highly-distributed, service-based organizational body, a break-up of the monolithic system into various, self-contained N2SkyC microservices was done.

The layout of this paper is as follows: The state of the art of neural network simulators, the baseline research and a comparison of neural network simulation systems is given in Sect. 2. The system architecture of N2SkyC is presented in Sect. 3 followed by a description of the responsive user interface of the new N2SkyC platform in Sect. 4. The neural network execution stack for deployment in a container environment is presented in Sect. 5. In the following Sect. 6 we show the user experience by a sample workflow of a neural network training. The paper is closed by a summary of the state and availability of the N2SkyC project (Sect. 7) and a short lookout for current and future research issues.

2 Related Work and Baseline Research

The driving stimulus for development in the computational science domain is the exchange of knowledge and resources between researchers. This principle is just as valid for any other research community too.

The UK e-Science initiative [6] describes several goals to be reached by fostering new stimulating techniques:

– Enabling more effective and seamless collaboration of dispersed communities, both scientific and commercial.

– enable large-scale applications comprising of thousands of computers, large-scale pipelines etc.
– Transparent access to "high-end" resources from the desktop.
– Provide a uniform "look & feel" to a wide range of resources
– Location independence of computational resources as well as data.

However, these targets are not reached in the computational intelligence community until now. As an example, we examine the situation of neural network simulation. Over the last decades, a very large number of artificial neural network simulation environments has been developed which aim to mimic the behaviour of biological neural networks [17]. It started with systems which were developed for specific network families, as Aspirin/MIGRAINES [14], SOM-PAK [13]. Some systems aimed for a more comprehensive environment, as SNNS [24]. New technology shifts enabled new concepts, as distributed cooperative environments over the Internet, as NeuroWeb [18]. With the advance of virtual resources by Grid and Cloud computing new collaborative environments motivated the authors of this paper to aim for an "everything about sharing" approach leading the way towards virtual collaborative organisations, as N2Grid [22] and N2Cloud [9].

However, all these systems, reaching from programming language extensions over proprietary stand alone systems to distributed platforms, share the same common problems:

– *Complex tool*, which mostly do not present an intuitive interface to the user.
– *Proprietary system* with missing interconnection and data exchange to other software systems.
– *Lacking* provisioning of arbitrary *computing resources*, as CPUs, disks, network, on demand.

These problems lead to the situation that quite a number of simulation systems exist, but which are rarely used. Hence, scientists invent the wheel over and over again and develop their own neural network systems for their specific needs. We believe that this situation is one of the reasons for an obstructed open information and data exchange within the scientific community.

A promising project, totally in line with our motivation, was the CIML (Computational Intelligence and Machine Learning) community [25]. The goal of CIML was to create an online virtual scientific community wherein anyone interested in computational intelligence and machine learning can share research, obtain resources, or simply learn more. Sorry to say, but CIML failed. One reason was that CIML targeted a too huge and dispersed community and offered too many and different resources. Due to lack of automated guidance of the system it was difficult for the user "to find his specific needle in the haystack".

Having this situation in mind we realized N2Sky [21], with a clear focus on neural networks aiming for intuitive user guidance and transparent resource access. N2Sky was designed as an artificial neural network provisioning environment facilitating the users to create, train, evaluate neural networks fostering different types of computing resources. The system is cloud based in order to allow for a growing virtual user community. N2Sky supports experienced users to easily run their simulations by accessing data related neural network objects. Moreover, N2Sky provides a facility to end users to solve their problems by using predefined objects and paradigms. For the purpose of thin clients a simple Web browser, which can execute on a PC or a smart phone, can be used to access the front-end, the N2Sky (Mobile) Web Portal. N2Sky aroused strong interest even beyond the computational intelligence community.[1]

[1] http://cacm.acm.org/news/171642-neural-nets-now-available-in-the-Cloud/.

Table 1 Feature comparison of artificial neural network simulation systems

Features versus systems	SNNS	DL4J	Neural designer	NeuroWeb	N2Grid	N2Cloud	N2Sky	N2SkyC
User interface type	App	Web	App	App	App/Web	Web	Web	Web
Responsive design	×	×	×	×	×	×	×	✓
User management	×	×	✓	×	×	×	✓	✓
Container support	×	×	×	×	×	×	×	✓
System architecture	Mono	Mono	Mono	Mono	Mono	Mono	Mono	MS
Licence	OS	Comm	OS	OS	OS	OS	OS	OS

A general survey of existing neural network simulation environments and comparison of features can be found in [17]. However, in the following we provide a specific comparison of well known systems with the focus on the design principles guiding the development of N2SkyC (see Table 1). We focus on the following systems, which are on the one hand renowned and actual neural network simulation environments, and on the other hand are influencing precursors to N2SkyC:

- *SNNS* [24] aka Stuttgart Neural Network Simulator is a renowned neural network simulation environment for Unix workstations and PCs and is free of charge. It was developed at the University of Stuttgart and is maintained by the University of Tübingen.
- *DL4J* [15] aka Deeplearning4j is an open-source, distributed deep-learning project and was developed by Skymind in San Francisco 2014. The distributed project is an open-source deep learning library for JVM and written in Java and Scala.
- *Neural Designer* [2] is a software tool for data analytic based on neural networks and contains a graphical user interface which simplifies data entry and interpretation of results.
- *NeuroWeb* [18] This project implements an Internet-based framework for the simulation of neural networks and provisioning of processing resources.
- *N2Grid* [22] employs the novel infrastructure of the grid as a transparent environment to allow users the exchange of information (neural network objects, neural network paradigms) and the exploitation of available computing resources
- *N2Cloud* [9] is Cloud-based neural network simulation system, which enables the exchange of knowledge, as neural network objects and paradigms, by a virtual organization environment and delivers ample resources by exploiting the Cloud computing principle.
- *N2Sky* [21] provides a framework for neural network simulation by a virtual organization environment. It follows the sky computing paradigm delivering ample resources by the usage of federated Clouds.
- *N2SkyC* represents our novel microservice oriented and cloud container enabled neural network simulation environment.

Specifically we evaluate the following properties, which defined the requirements for the N2SkyC design:

- *User Interface Type*: Application (App) versus Web Interface (Web)
- *Responsive Design*: support of flexible layout and images scaling to the output device
- *User Management*: support of user administration allowing for security and business models
- *Container Support*: embedded cloud container support within the system
- *System Architecture*: Monolithic (Mono) versus MicroServices (MS)

– *Licence*: Open Source (OS) versus Commercial (Comm)

The pillar of our envisioned system is ViNNSL, the Vienna Neural Network Specification Language [5]. It is key for easy sharing of resources between the paradigm provider and the customers. ViNNSL is an XML-based domain specific language providing mechanisms to specify neural network objects in a standardized way by attributing them with semantic information. Originally it was developed as communication framework to support service-oriented architecture based neural network environments. Even more, ViNNSL is capable of describing the static structure, the training and execution phase of neural network objects in a distributed infrastructure, as grids and clouds. Its last extension [19] supports semantic information too, describing the usage scenario of network objects for a given problem domain.

The following short example illustrates the use of ViNNSL. The listing 1.1 defines a 2-5-1 backpropagation network for the well-known XOR problem. Besides the description of the structure of the network, also the application domain for using this network is specified. This is the basis for smart searching of feasible neural networks for given problems.

```
<definition xmlns="http://www.pri.univie.ac.at/../vinnsl">
  <identifier>7</identifier>
  <problemdomain>
    <backpropagation/>
    <classifier/>
      <XOR-net/>
    </classifier>
  </problemdomain>
  <executionenvironment>
  ...
  </executionenvironment>
  <structure>
    <input>
      <id>Input1</id>
      <dimension>2</dimension>
      <size>1</size>
    </input>
    <hidden>
      <id>Hidden1</id>
      <dimension>5</dimension>
      <size>1</size>
    </hidden>
    <output>
      <id>Output1</id>
      <dimension>1</dimension>
      <size>1</size>
    </output>
    <connections>
      ...
    </connections>
  </structure>
  <parameters>
    <valueparameter>
      <name>lrate</name>
      <label>Learning Rate</label>
      <value>0.3</value>
    </valueparameter>
    ...
  </parameters>
</definition>
```

Listing 1.1. ViNNSL: XOR-Backpropagation Neural Network Definition

3 N2Sky Architecture

Service migration to the cloud in the recent years is a clear trend in the IT-industry, and it is not surprising that some of the companies are trying to leverage existing cloud infrastructure to bring deep and machine learning solutions to the market. IBM SPSS, Amazon AWS, Google Cloud AI services are meant to provide software tools to the different stakeholders, but they are primarily focusing on the business users, which primary goal is to use applied statistics techniques to fulfill existing business needs. Services are split into components, each of which hides internal implementation of the solution and provides an interface for a specific task: speech recognition, text analysis, etc. Variety of pricing models, a necessity to use other software products of the company, proprietary code and low level of scalability and extensibility are making these products less attractive for the scientific and academic community. Vendor lock-in and inability to control execution and development process are making such software systems and services non-flexible and nontransparent, leading to the scenario where each service provider tends to aggregate and try to sell as many proprietary services as possible.

We consider these aspects as limiting factors for the development of the framework. We believe that such properties as an ability to share knowledge and resources, re-usage of the existing solutions and collaborative work are key features which should be put as a foundation of distributed neural network execution platform.

Technically speaking, N2Sky is an artificial neural network simulator, which purpose is to provide different stakeholders with access to robust and efficient computing resource [21]. It was designed to provide natural support for Cloud deployment with distributed computational resources. However, the current N2Sky implementation is based on the Java programming language and deployed as a single monolithic application, which is not well aligned with the chosen paradigm. Main bottlenecks of such architecture are the inability to split the infrastructure workload according to the available computational resources within the "shared-nothing" infrastructure which leads to the inefficient usability of the system. That led us to a redesign of the N2Sky platform using microservices approach and the new technology stack for the Cloud infrastructure, which will allow us to utilize the benefits of Cloud computing to its full extent. This approach was not only used on the backend services but also on the frontend and its services. Microservices application architecture is a considerably new approach in software architecture, and like any new idea, it is evaluated in the variety of projects. The conclusions from practice are:

- It is simpler to develop and maintain small applications. However the complexity of such an approach does not disappear, it just transfers on the more abstract level: to the relationships between components.
- Microservices are easier to split between developers or other parties—as each part is not strongly bound to others, it is possible to work on the same project without a huge amount of collisions.
- Technological stack flexibility—each component can be implemented in a language that suits better the concrete task.
- Horizontal scalability—as services are small, it is cheaper and faster to deploy them horizontally. With a Cloud infrastructure, this feature becomes very important.

On the other hand, there are some drawbacks, which have to be considered:

- The complexity of the distributed system is higher than with a monolithic approach.
- Maintenance costs are also higher, due to the constant redeployment, monitoring, and logging.

There is no definitive answer about the feasibility of microservices architecture; it depends on the domain. But in the case of a Virtual Organization for computational science, where resources and scientists work together in parallel there is a necessity to have control over the executing environment. Thus, new components integration and deployment costs are lower, and researchers can allocate computational resources according to their demands without additional difficulties.

From the general point of view, the environment system components are divided into four main layers:

- Frontend UI: the graphical user interface supporting different user roles,
- Orchestration tool layer: the management layer coordinating communication and cooperation between microservices,
- Cloud platform layer: the resource provisioning infrastructure for service execution, and
- Neural Network Execution Stack: the neural network simulation layer implementing all neural network specific tasks, as creation, training, and evaluation.

In such architecture all communication with the Cloud infrastructure level is handled through the orchestration tool layer: spawning and stopping containers and virtual machines, uploading new containers, and updating them. That approach allows to focus on the deployment and maintenance process. In cases, when there is a need to edit and access internal components of the Cloud infrastructure directly, it can be done by the Cloud API. The Cloud platform provides infrastructure and environment for container deployment only, and it is not related to the current software application design. It provides great support for microservices paradigm implementation but implies no boundaries on the software developers.

The original N2Sky Java implementation is well designed: Java is a powerful programming language, which allows splitting the functionality between classes and maps them to the corresponding services of the platform. Source code and interaction between components are part of the same monolithic application. Even that the application is well designed it creates the additional level of bindings between components of the software.

To redesigning the original architecture according to the microservices paradigm, it was necessary to perform additional steps:

- Decomposition of the business logic into microservices,
- Design storage according to data, stored by services,
- Neural network data archive—database with data, which is used to train and test neural networks, and
- Define endpoints and API for each service.

As a result, apart from technological and infrastructure improvements, the following changes to the main components of the system were performed:

- Neural network training and Neural network evaluation components are now separate microservices, which are run on demand. Resource allocation in such case can be performed on the orchestration layer, according to the computational cost of the task,
- Neural network data archive is a separate service on top of the database. It is not associated with a single database, as was in the previous implementation. It provides a specific API to add any external data sources,
- N2Sky Database has its own database storage,
- Business model module is split into two different parts: one part is responsible for the internal business logic of the application, and the second part is responsible for infrastructure management and handled by the orchestration component. Internal business logic functionality is also encapsulated into microservices.

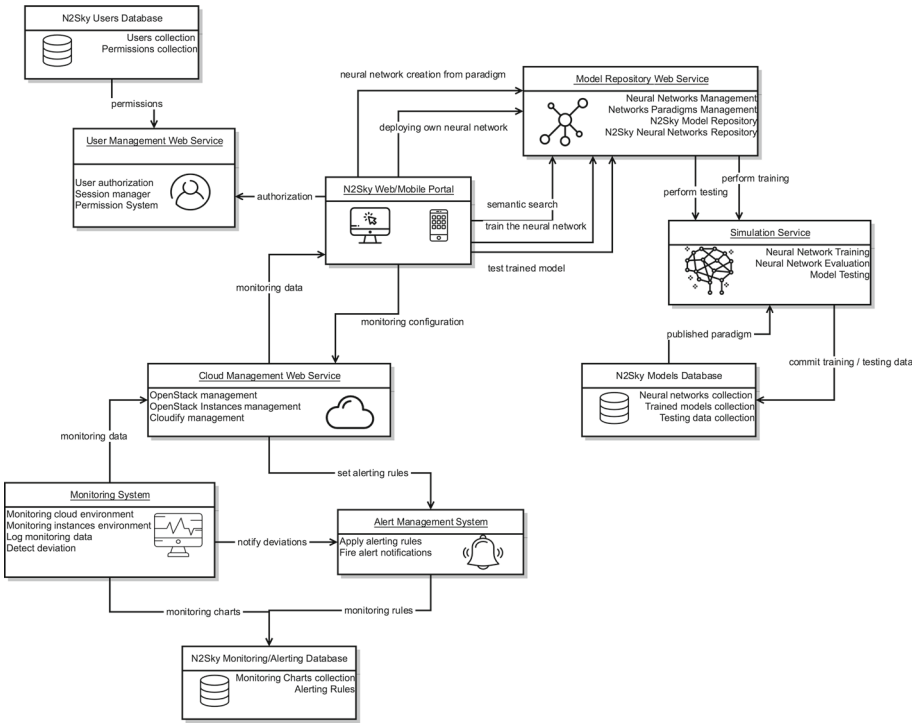

Fig. 1 New N2Sky architecture

The current working version of the new architecture with the described changes is presented in Fig. 1.

By implementing architectural changes to both infrastructure and software levels of the N2Sky platform, we achieve higher scalability and make the platform more suitable for usage in a Cloud environment. The shift to new technologies, as container base execution environment and orchestration tools for load and deployment management, increase efficiency and convenience of N2Sky neural network platform rigorously.

4 User-Centered Interface Based on Modular Frontend

The basis of N2Sky is the user-centered design. Moving from the complex monolithic design to a more comprehensible one we fostered past user experiences. Thus, a fundamental requirement was to gain the functionality and increase the performance of the application.

4.1 User Role as an Independent User Interface

The N2Sky web interface, as well as the mobile portal, provide an intuitive user interface. Since N2Sky supports different knowledge level users it was decided to differentiate between various user roles. So, every user role has its own user interface, which deeply reflects the specific user needs:

- *Arbitrary User.* The Arbitrary User must not have deep knowledge of the neural network field or programming language skills. Her main goal is to find a suitable neural network to a given problem. Thus she uses already existing, trained neural networks and evaluates them.
- *Neural Network Engineer.* The Neural Network Engineer is allowed to create new neural network resources based on existing paradigms. She has access to her own dashboard and publicly available resources on the main application module. She can perform semantic search for available neural network paradigms and use them. This user can create own neural network instances from existing neural network paradigms and can also train the running neural network instances and evaluate the trained models. She can share her trained neural networks by making them public.
- *Contributor.* The Contributor is an expert user, which has enough knowledge and experience to develop her own new neural network paradigm by using the ViNNSL template schema [5] and publish them on N2Sky. This user can deploy neural networks on the N2Sky environment as well as on her own environment by providing training and testing endpoints. The goal of the contributor is to study how networks will behave with different network structures, input parameters and training data that can also be provided by other users.
- *System Administrator.* The System Administrator is a user who has full access to the application including environment management, monitoring and alerting features. The administrator can manage OpenStack and Cloudify instances. She also can shadow any N2Sky user to observe the application from different perspectives. The administrator has access to all dashboards in every module.

4.2 Modular Frontend Application Design

Maintaining a large application like N2Sky with the monolithic approach is unwieldy. Since N2Sky supports microservices approach in the backend it was a design decision to apply the same approach on the frontend too.

Microservices in the frontend are small independent web applications, which are consolidated into one application. The main benefits of this approach are:

- *Maintainability.* It is possible to divide application between different teams. Developers do not even need to have some knowledge about other parts of the application.
- *Diversity of technologies.* Monolithic approach makes the whole application stick to one framework. Microservices allow to use any technology without the need to rewriting the application.
- *Independent deployment.* Every application has some releases periods, every release is accompanied by redeployment procedure. There is no need to redeploy the whole application, but just only the required components.

The microservices approach in frontend application is shown in "Fig. 2". It breaks the whole application into the following small micro applications [12]:

- *Shell.* Is a top-level component, which wraps Components picker and Container for the component. It contains application specific configuration.
- *Component picker.* Is a router, which manages the micro applications.
- *Container for Component.* Container, where the component will be injected.
- *Micro Application.* Independent application, which can be written in any programming language, but has to use one of the shards.

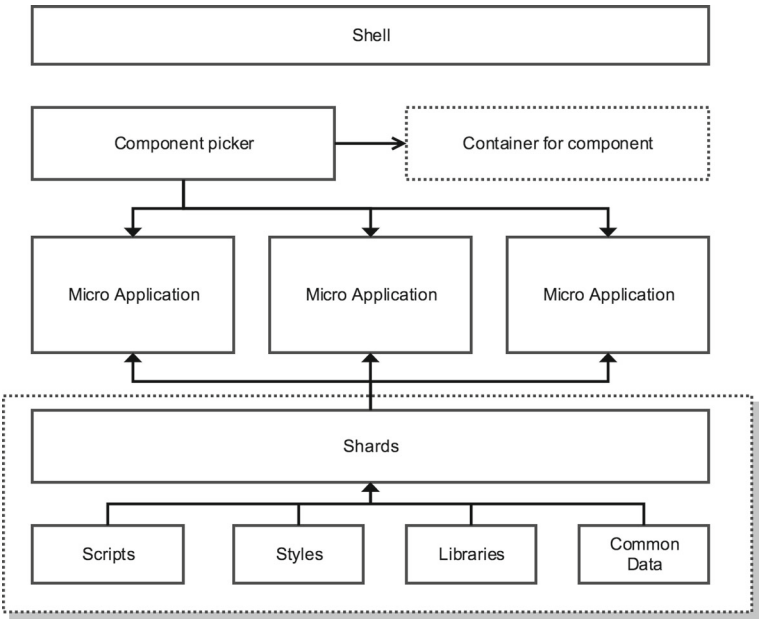

Fig. 2 Microservices approach in frontend application

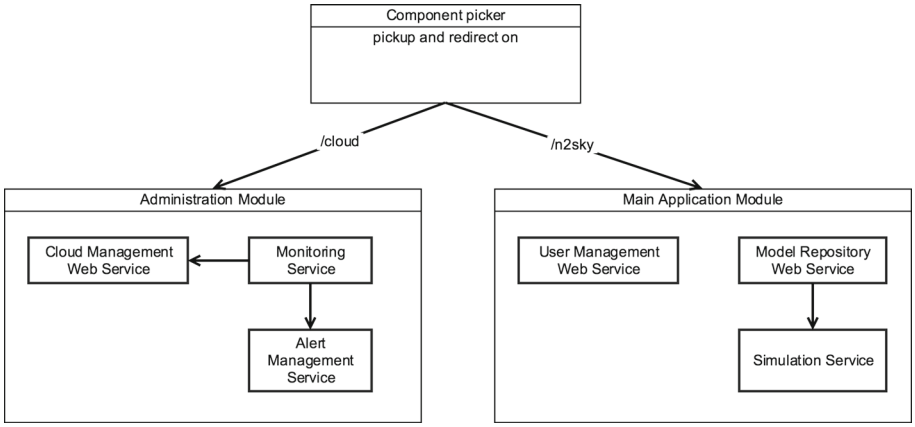

Fig. 3 N2Sky frontend application and its services

– *Shards.* Is a code base, which is shared between Micro Applications. Shards can have multiple levels.

4.3 N2Sky Modular Frontend

The central design concept of the application is to support the Software as a Service (SaaS) and Platform as a Service (PaaS) service model [23]. N2Sky consists of two modules: administration module and main application module as shown in Fig. 3.

– *N2Sky component picker.* When the user accesses the N2Sky web portal, first she will be dealing with a component picker. The component picker is a small service, which redirects the user depending on URL path.
  - */cloud* redirects to "Administration module"
  - */n2sky* redirects to "Main application module"
– *Administration module.* The administration module allows the system administrator to control the virtualized environment. The module supports OpenStack and Cloudify monitoring. Managing is possible by the application dashboard. It also contains custom monitoring and an alerting management system, which can be installed on any server within the N2Sky user interface. The administration module implements PaaS. It is fully configurable and wrapped into an open source project in order to make the module accessible to the third-party applications.
– *Main application module.* The main application module is the central module of N2Sky. Within this module, users can use, train and test existing neural networks. It is possible to reuse existing or create own neural network objects or paradigms. N2Sky allows to deploy own neural network objects or store data in the Cloud. Module services support the SaaS model. Experts can use an application directly through the N2Sky API or they can integrate N2Sky services into their own application.

5 Neural Network Execution Stack

For the execution of neural network objects basically two possibilities provided by the system:
– Handcrafting own neural network objects, and
– Ready-made neural network objects

5.1 Handcrafting Neural Network Objects

The user, specifically the neuralnetwork contributor, provides self-coded neural network components in an arbitrary programming language. To be useable in the N2SkyC environment, the following process has to be followed:
– Code your own neural network object in arbitrary programming language or use existing implementations (e.g. Tensorflow)
– Each neural network object is a webservice,
– Each object provides train and test endpoints,
– Each object is packed into a Docker container with all the libraries and dependencies,
– Provide data format specification—description on which data formats are accepted by their service
– Describe object by ViNNSL

Respective developed neural network objects can be integrated or imported into the N2Sky repository and are available for further usage.

5.2 Ready-Made Neural Network Objects: N2Container

Due to the situation that many N2SkyC user have no experience in programming of neural network objects, the system provides the possibility to use existing neural network frameworks, which provide ready-made components. For integration of such components, N2SkyC

implements an execution stack, called N2Container, which allows the easy embedment of neural network objects and management of executable containers.

N2Container fosters the *Kubernetes*[2] container orchestration and a Java based microservice architecture, which is exposed to users and other systems via RESTful web services and/or a web frontend. The whole workflow including creating, training, evaluating, and importing a neural network model is purely service oriented. The presented stack runs on popular Cloud platforms, like *Google Cloud Platform*,[3] *Amazon AWS*[4] and *Microsoft Azure*.[5] Furthermore it is scalable and each component is extensible and interchangeable.

For the design and development of N2Container we chose a number of proven and widely accepted paradigms and tools. Guiding principles was acceptance and support by the software developer community and open-source availability:

RESTful services Representational State Transfer (REST) is an architectural style for the development of webservices. A *REST client* is an application program interface that uses RESTful API HTTP requests to GET, PUT, POST and DELETE data.

Microservices The micoservice architecture pattern is a variant of a service-oriented architecture (SOA). Monolithic applications bundle user interface, data access layer and business logic together in a single unit. In the microservice architecture each task has its own service. The user interface puts information together from multiple services. This leads in our approach to separation of the specific services in own components, as *database, storage, worker, GUI services etc.*

Docker Containers Containers enable software developers to deploy applications that are portable and consistent across different environments and providers by running isolated on top of the operating system's kernel. As an organisation, Docker[6] has seen an increase of popularity very quickly, mainly because of its advantages, which are speed, portability, scalability, rapid delivery, and density [4] compared to other solutions. Building a Docker container is fast, because images do not include a guest operating system. The container format itself is standardized, which means that developers only have to ensure that their application runs inside the container, which is then bundled into a single unit. The unit can be deployed on any Linux system as well as on various Cloud environments and therefore easily be scaled. Not using a full operating system makes containers use less resources than virtual machines, which ensures higher workloads with greater density [10]. *Docker containers build the execution framework* for all our ViNNSL services.

Kubernetes Container Orchestration Technologies As every single microservice runs as a container, we need a tool to manage, organise and replace these containers. Services should also be able to speak to each other and restarted if they fail. Services under heavy load should be scaled for better performance. To deal with these challenges container orchestration technologies come into place. According to a study from 2017 published by Portworx, Kubernetes [3] is the most frequently used container orchestration tool in organizations. *Kubernetes* realizes the *orchestration of all Docker services*.

Neural Network Execution As starting point we chose Deeplearning4J,[7] which is a Neural Network Execution Frameworks providing a deep learning programming library written for Java and the Java virtual machine (JVM)and a computing framework with wide

[2] https://kubernetes.io.

[3] https://cloud.google.com/kubernetes-engine.

[4] https://aws.amazon.com/eks.

[5] https://azure.microsoft.com/services/container-service.

[6] https://docker.com.

[7] https://deeplearning4j.org/.

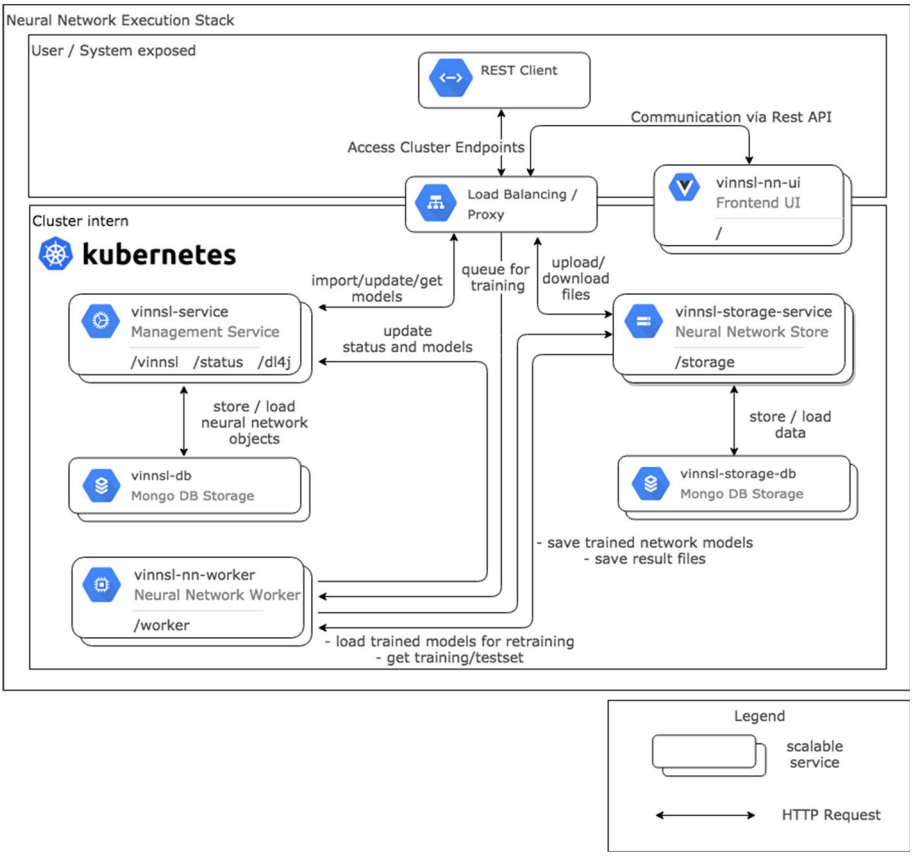

Fig. 4 Architecture of N2Container

support for deep learning algorithms. *Deeplearning4J* implements the *neural network worker service*. However, any other simulation framework can be easily integrated. For example, as prototype also a Tensorflow[8] binding is implemented.

Putting all these components together leads to the following architecture of our envisioned system (see Fig. 4), which is part of the overall N2SkyC design.

The functionality of the whole neural network simulation framework is defined by eight use cases realized by respective HTTP service calls. These service calls are typically executed by N2SkyC operations, but can also be used by N2SkyC users directly. To cope with this concept we refer to call executor as actor in the following.

1. **Import Neural Network:** An existing ViNNSL XML file with a neural network description is imported via the ViNNSL web service into the database.
   Call: The actor sends a *POST* request to the ViNNSL web service including a XML body.
2. **Train Neural Network:** An imported neural network is trained by passing the configuration over to the worker service.
   Call: The actor sends a *POST* request to the working service including the identifier of the neural network that should be trained.

[8] https://www.tensorflow.org/.

3. **Monitor Training Status:** The Data Scientist monitors the training status to evaluate the trained network afterwards.
   Call: The actor sends a *GET* request to the status endpoint of the ViNNSL service including the identifier of the neural network that is in progress.
4. **Evaluate Neural Network:** The Data Scientist evaluates the accuracy of the network after its training.
   Call: The actor sends a *GET* request to the status endpoint of the ViNNSL service including the identifier of the neural network that is finished.
5. **Upload Files:** The Data Scientist uploads files, that are usable as datasets (e.g. CSV files or pictures) to the storage service.
   Call: The actor sends a *POST* request to the storage service endpoint containing a multipart file.
6. **List Neural Networks:** Imported neural networks are listed.
   Call: The actor sends a *GET* request to the ViNNSL web service optionally including a neural network identifier.
7. **Extend Service:** An existing micro service can be extended by developers.
   Call: The developer downloads the source code and extends functionality of a micro service.
8. **Replace Service:** An existing micro service can be replaced by developers.
   Call: The developer writes a new implementation of an existing service respecting the API definition.

6 A Sample Workflow

Workflows of solving problems as contributor, engineer, arbitrary user, and administrator differ. For example, a user needs an easy step-by-step workflow, the engineer just wants to perform procedures quickly without any complex distractions. There are only a few prerequirements, which will remain the same: authentication within N2Sky and the creation of a first project from the N2Sky dashboard, which describes the problem field.

To support Software as a Service (SaaS) distribution every web service can work independently. The different user roles can use N2SkyC both via the web portal and the N2SkyC/N2Container API directly:

- Authorise users in the System.
- Create new neural network objects from existing paradigm.
- Deploy own neural network objects on N2SkyC environment.
- Perform training against own as well published neural network objects.
- Perform testing and evaluating trained models.

Sample workflow overviews for different user roles show the microservices architecture in action, see Fig. 1:

1. *Contributor*
   (a) The Contributor authorizes via the N2SkyC portal using a browser on desktop PC or mobile device. She is redirected to her own dashboard according to permissions, which will be received from User Management Web Service.
   (b) The user described her own neural network paradigm using the ViNNSL template and deploying it in the N2SkyC Cloud as it shown in Fig. 5.

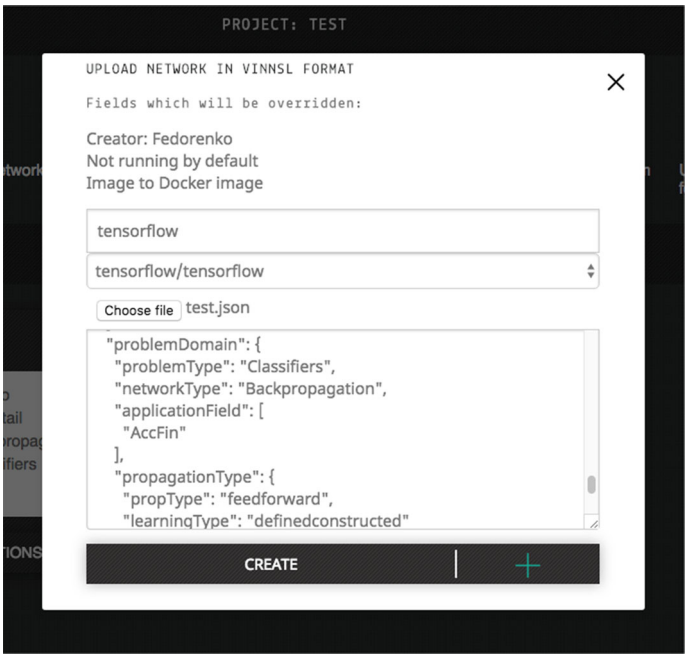

Fig. 5 The user uploads the neural network paradigm description in ViNNSL format

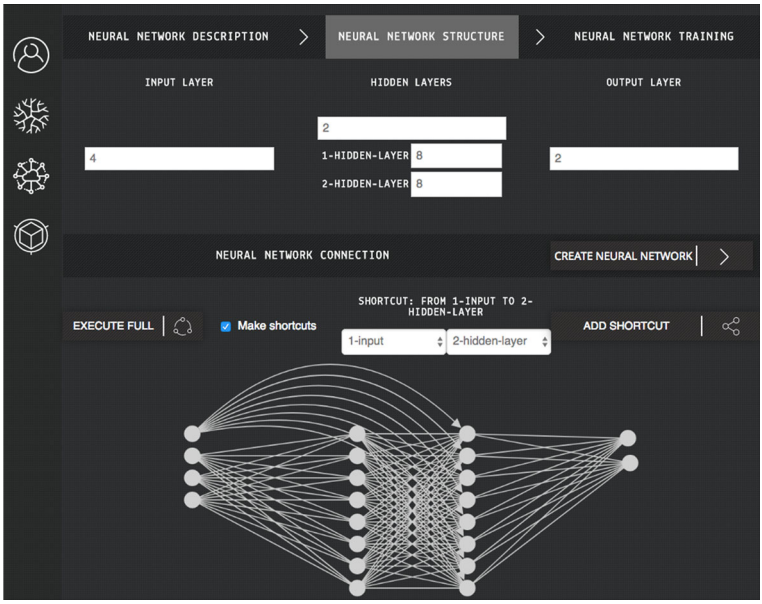

Fig. 6 The user specifies the neural network structure

(c) The Contributor performs training of her neural networks. Since she is an expert she can perform this operation using Simulation Service via Model Repository Web Service API.

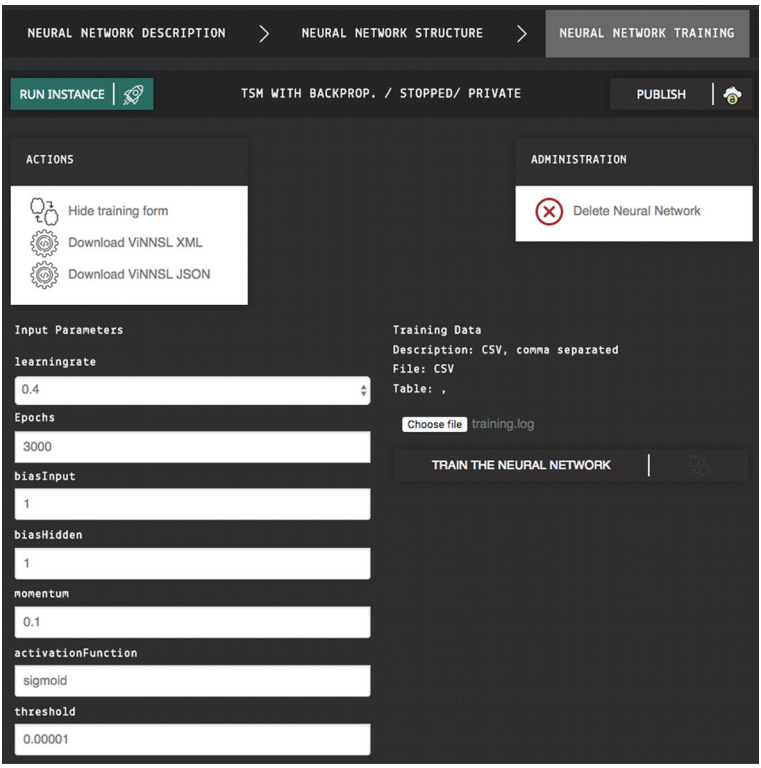

Fig. 7 Neural network training view

(d) The user publishes her paradigm via N2SkyC UI or available API.
(e) The Contributor awaits that other N2SkyC users will use her neural network paradigm in order to monitor the behavior of the neural network.
(f) The user modifies, redeploys and retrains her neural network after first results.

2. *Neural Network Engineer*

(a) The Neural Network Engineer authorizes via the N2SkyC portal using a browser on a desktop PC or mobile device. She is redirected to her own dashboard according to permissions, which will be received from User Management Web Service.
(b) From the dashboard, the user creates a neural network from existing paradigms using Model Repository Web Service. The user specifies the neural network structure as shown in Fig. 6.
(c) After creating a neural network, the user will be redirected to the training view which is shown in Fig. 7. From here, she can run the neural network instance on N2Sky and publish it to make this neural network available for the rest of the users.
(d) After training is completed, users can perform testing, see Fig. 8. The visual representation of evaluation will be shown. The learning curve is constantly updated during neural network training.
(e) If the user is satisfied with a trained model, she can perform data analysis using N2SkyC.

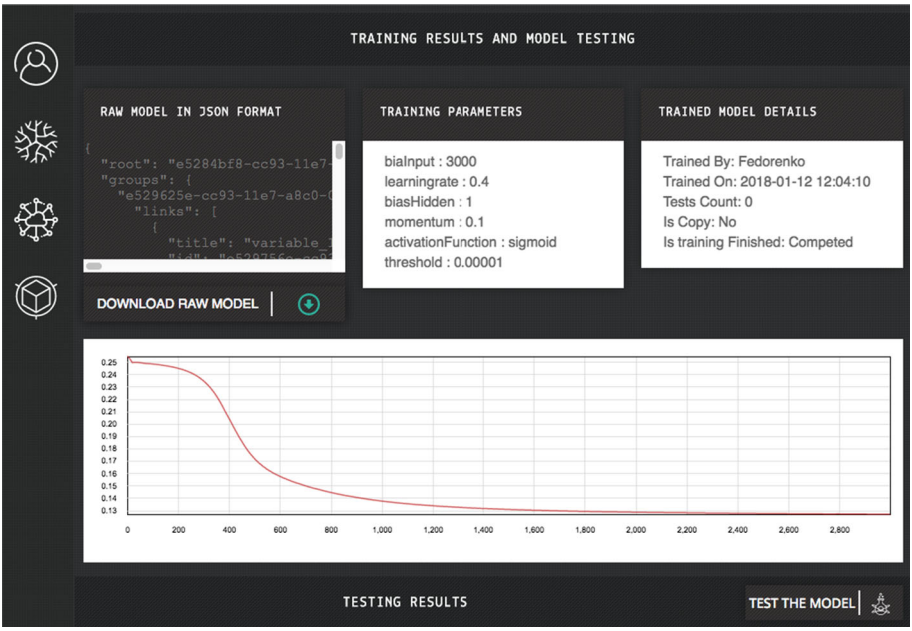

Fig. 8 The user performs neural network training

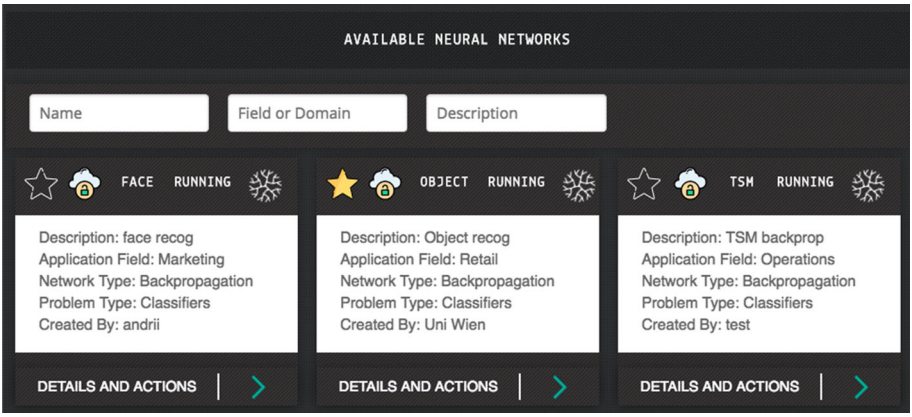

Fig. 9 The user performs semantic search in Neural Network Repository

(f) The neural network engineer user publish her neural network and trained model in order to make it available to other N2SkyC users.

3. *Arbitrary User*

(a) The Arbitrary User authorizes via the N2SkyC portal using a browser on desktop PC or mobile device. She will be redirected to her own dashboard according to permissions, which will be received from User Management Web Service.
(b) She performs a semantic search in order to find possible neural networks, see Fig. 9, as well as inspect available trained models if fulfilling her needs, see Fig. 10.

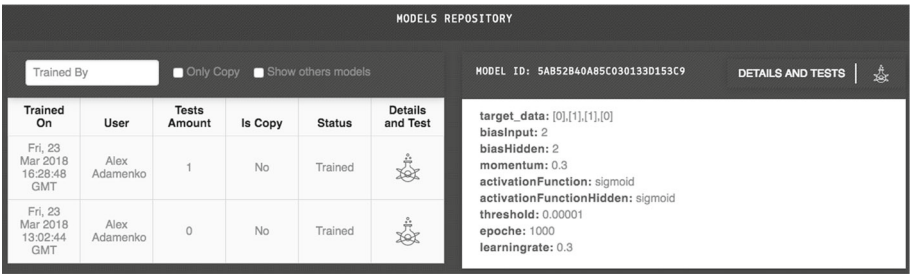

Fig. 10 Trained Neural Network models repository

(c) The user copies existing neural networks and trained models into her project.
(d) The user performs training of copied neural networks with default input parameters data.
(e) The user evaluates trained neural network models with default parameters.

4. *System Administrator*

(a) The System Administrator authorizes via the N2SkyC portal using her browser on a desktop PC or mobile device. She will be redirected to the administration dashboard.
(b) The user observes the Cloud environment.
(c) She creates new monitoring charts showing specific metrics and adds it to the administration dashboard.
(d) She creates alerts for newly created monitoring
(e) The user is notified by the Alert Management System, if some events occur.

7 State and Availability of N2SkyC Project

N2SkyC is an academic research project, which is used in University courses on neural networks and information systems. However, until now no commercialization is planned and the source code is available as open-source and open for the public community.

The development is ongoing. We are now focussing on the, for the user transparent, integration of N2Container as execution engine into the N2SkyC framework.

The novel N2SkyC system and the N2Container execution stack, specifically their documentation, demos, virtual images and code, can be accessed and downloaded from the respective sites.[9,10]

For update requests and/or error reports email links to the authors are provided via the N2SkyC web site.

Examples and use cases going beyond this paper showing the capabilities of the N2SkyC system can be found in several master theses created in the course of this project, as [1,7,16].

[9] http://www.wst.univie.ac.at/projects/n2skyc.
[10] http://www.wst.univie.ac.at/projects/n2container.

8 Conclusion and Future Work

We presented the novel microservice based N2SkyC architecture, which aims for increased extensibility, portability, dynamic orchestration and performance fostering Cloud container technology.

Tied to the presented new architectural design is also a shift of the application domain of N2SkyC from neural networks to arbitrary machine learning operators. Hereby a new specification of ViNNSL is under development, which aims for compatibility with PMML (Predictive Model Markup Language) [8]. A further new development of N2SkyC is N2Query [20], which allows the semantic discovery of N2SkyC services through a natural language querying mechanism using ontology alignment mechanisms.

Acknowledgements Open access funding provided by University of Vienna.

References

1. Adamenko A (2018) N2SkyC : cloud container-based problem solving environment. Master thesis, University of Vienna, Wien
2. Artificial Intelligence Techniques S (2019) Neural designer. https://www.neuraldesigner.com. Accessed Aug 2019 (**online**)
3. Baier J (2015) Getting started with kubernetes. Packt Publishing, Birmingham
4. Bashari Rad B, Bhatti H, Ahmadi M (2017) An introduction to docker and analysis of its performance. IJCSNS Int J Comput Sci Netw Secur 17(3):228–235
5. Beran PP, Vinek E, Schikuta E, Weishaupl T (2008) Vinnsl—the Vienna neural network specification language. In: IEEE international joint conference on neural networks, 2008. IJCNN 2008 (IEEE World Congress on Computational Intelligence), pp 1872–1879. IEEE
6. e-Science U (2016) UK e-science programme. http://www.escience-grid.org.uk. Accessed Aug 2019 (**online**)
7. Fedorenko A (2018) Functional user interfaces for controlling and monitoring the N2Sky and its services. Master thesis, University of Vienna, Wien
8. Guazzelli A, Zeller M, Lin WC, Williams G et al (2009) PMML: an open standard for sharing models. R J 1(1):60–65
9. Huqqani AA, Li X, Beran PP, Schikuta E (2010) N2Cloud: cloud based neural network simulation application. In: The 2010 international joint conference on neural networks (IJCNN), pp 1–5. IEEE
10. Joy AM (2015) Performance comparison between linux containers and virtual machines. In: 2015 International conference on advances in computer engineering and applications, pp 342–346 (2015)
11. Keahey K, Tsugawa M, Matsunaga A, Fortes J (2009) Sky computing. IEEE Internet Comput 13(5):43–51
12. Kobvel M (2017) Front-end microservices with web components. https://hackernoon.com/front-end-microservices-with-web-components-597759313393. Accessed Aug 2019 (**online**)
13. Kohonen T, Hynninen J, Kangas J, Laaksonen J (1996) Som pak: the self-organizing map program package. Report A31, Helsinki University of Technology, Laboratory of Computer and Information Science
14. Leighton RR, Wieland A (1991) The aspirin/migraines software tools, user's manual. Technical report MP-91W00050
15. Nicholson AC, Gibson A (2019) Deeplearning4j: Open-source distributed deep learning for the jvm. [online]. https://deeplearning4j.org/. Last visited Aug 2019
16. Nussbaum B (2018) Container based execution stack for neural networks. Master thesis, University of Vienna, Wien
17. Prieto A, Prieto B, Ortigosa EM, Ros E, Pelayo F, Ortega J, Rojas I (2016) Neural networks: an overview of early research, current frameworks and new challenges. Neurocomputing 214:242–268

18. Schikuta E (2002) NeuroWeb: an internet-based neural network simulator. In: Proceedings 14th IEEE international conference on tools with artificial intelligence, 2002 (ICTAI 2002), pp 407–412. IEEE
19. Schikuta E, Huqqani A, Kopica T (2015) Semantic extensions to the Vienna neural network specification language. In: 2015 international joint conference on neural networks (IJCNN), pp 1–8. IEEE
20. Schikuta E, Magdy A, Mohamed AB (2016) A framework for ontology based management of neural network as a service. In: International conference on neural information processing, pp 236–243. Springer, Berlin
21. Schikuta E, Mann E (2013) N2Sky—neural networks as services in the clouds. In: The 2013 international joint conference on neural networks (IJCNN), pp 1–8. IEEE
22. Schikuta E, Weishaupl T (2004) N2Grid: neural networks in the grid. In: Proceedings 2004 IEEE international joint conference on neural networks, vol 2, pp 1409–1414. IEEE
23. Walraven S, Truyen E, Joosen W (2014) Comparing paas offerings in light of saas development. Computing 96(8):669–724
24. Zell A, Mache N, Huebner R, Mamier G, Vogt M, Schmalzl M, Herrmann KU (1994) SNNS (stuttgart neural network simulator). In: Neural network simulation environments, pp 165–186. Springer, Berlin
25. Zurada JM, Mazurowski MA, Ragade R, Abdullin A, Wojtudiak J, Gentle J (2009) Building virtual community in computational intelligence and machine learning [research frontier]. IEEE Comput Intell Mag 4(1):43–54