Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Deep Learning in Neural Networks

dissertation proposal

2016 Mgr. Tomáš Kuzma

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Deep Learning in Neural Networks

dissertation proposal

Study Programme: Informatics

Field of Study: 2508 Informatics

Department: Department of Computer Science

Advisor: prof. Ing. Igor Farkaš, Dr.

2016

Mgr. Tomáš Kuzma

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname:	Mgr. Tomáš Kuzma
Study programme:	Computer Science (Single degree study, Ph.D. III. deg., full time form)
Field of Study:	9.2.1. Computer Science, Informatics
Type of Thesis:	Dissertation thesis
Language of Thesis:	English
Secondary language:	Slovak

Title:	Deep learning in neural networks
Literature:	http://deeplearning.net/reading-list/
Aim:	Implement, test and analyze, using computational simulations, a selected hierarchical model of a neural network, using a selected training data.
Annotation:	Deep learning is a set of algorithms in machine learning that attempt to model high-level abstractions in data by using architectures composed of multiple non-linear transformations. The most frequently used models are based on artificial neural networks. In the last decade, it has been an active research area.
Keywords:	deep learning, neural network, deep belief network, feature extraction, abstraction

Tutor:	prof. Ing. Igor Farkaš, Dr.
Department:	FMFI.KAI - Department of Applied Informatics
Head of department:	prof. Ing. Igor Farkaš, Dr.
Assigned:	21.02.2014

Approved: 24.04.2014 prof. RNDr. Branislav Rovan, PhD.
Guarantor of Study Programme

.....
Student

.....
Tutor

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Mgr. Tomáš Kuzma
Študijný program: informatika (Jednoodborové štúdium, doktorandské III. st., denná forma)
Študijný odbor: 9.2.1. informatika
Typ záverečnej práce: dizertačná
Jazyk záverečnej práce: anglický
Sekundárny jazyk: slovenský

Názov: Deep learning in neural networks
Hlboké učenie v neurónových sieťach

Literatúra: http://deeplearning.net/reading-list/

Cieľ: Implementovať, otestovať a analyzovať pomocou počítačových simulácií vybraný model hierarchickej neurónovej siete, na zvolených trénovacích dátach.

Anotácia: Hlboké učenie predstavuje množinu algoritmov v strojovom učení ktoré sa snažia modelovať vysokoúrovňové abstrakcie v dátach použitím architektúr, zložených z viacnásobných nelineárnych transformácií. Najčastejším prípadom modelov sú umelé neurónové siete. V ostatnej dekáde rokov bolo hlboké učenie aktívnou výskumnou oblasťou.

Kľúčové slová: hlboké učenie, neurónová sieť, model DBN, extrakcia príznakov, abstrakcia

Školiteľ: prof. Ing. Igor Farkaš, Dr.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: prof. Ing. Igor Farkaš, Dr.

Dátum zadania: 21.02.2014

Dátum schválenia: 24.04.2014 prof. RNDr. Branislav Rovan, PhD.
garant študijného programu

.....
študent

.....
školiteľ

I hereby declare that the following thesis was written solely by me.
All non-original content was correctly acknowledged.

In Bratislava, March 7th, 2016 .....

Acknowledgements

I would like to express my gratitude to my thesis supervisor, prof. Ing. Igor Farkaš, Dr., for his patience and guidance.

“I’ll be honest, we’re throwing science at the wall here to see what sticks. No idea what it’ll do.”

– Cave Johnson

Abstrakt

Umelé neurónové siete sú trieda modelov v strojovom učení, ktoré sú schopné úspešne riešiť komplexné problémy v rôznych oblastiach; občas svojou úspešnosťou dokonca aj prevyšujúc ľudí. Hoci univerzálnosť týchto metód prilákala veľký teoretický i praktický záujem, úspešnosť v niektorých úlohách ostáva nedostatočná. Častým problémom na týchto ťažkých úlohách je buď primárna neschopnosť naučiť sa úlohu úspešne riešiť alebo sekundárna neschopnosť správne zvládať nové, dosiaľ neznáme inštancie problému.

Oba tieto nedostatky je možné zmierniť použitím metód postupného učenia – triedy stratégií učenia inšpirovaných interakciami medzi fyzickým i psychickým vývojom dieťaťa a jeho vysporiadavaním sa s úlohami postupne rastúcej náročnosti. Cieľom tejto práce je navhrnúť, vyvinúť a skúmať ďalšie rozšírenia týchto úspešných techník v kontexte neurónových sietí.

Kľúčové slová:

strojové učenie, umelé neurónové siete, hlboké učenie, učenie s učiteľom, postupné učenie

Abstract

Artificial neural networks are a class of powerful machine learning models, capable of solving a wide variety of challenging tasks, occasionally even surpassing human performance. While their near-universality has attracted a lot of both theoretical and practical interest, performance in some tasks remains lacking. The common pitfalls for these hard tasks are either a primary inability to learn the task, or a secondary inability to correctly generalize to novel instances.

Both of these concerns may be addressed by the methods of curriculum learning – a class of teaching strategies inspired by the interaction of physical and psychological development of children and their experience with tasks of ever-increasing difficulty. The aim of this thesis is to propose, develop and examine extensions to these successful techniques in the context of neural networks.

Keywords:

machine learning, artificial neural networks, deep learning, supervised learning, curriculum learning

Contents

Introduction

Inspired by the connected discrete neurons found in animal brains, *artificial neural networks* are a class of models in machine learning[1] which manifest useful computation as an emergent behaviour of simple interacting units. While many specific architectures of neural networks exist, we will concentrate on the *feed-forward* variety, in which information is propagated from the inputs to the output neurons in a linear fashion. Conventionally, neurons in such models are arranged into layers, which encourages the formation of hierarchical features while at the same time facilitates effective parallelized simulation.

The brain structures involved in vision – mainly *simple cells* and *complex cells* found in the primary visual cortex of mammals – have inspired specialized models of neural networks, including the *Neocognitron* and its modern iteration, *convolutional neural network*, that excel in visual tasks. These models decidedly outperform general models, and have recently dominated the field of visual pattern recognition, surpassing human performance in some cases.

Following both recent theoretical breakthroughs and gradual increase in available computational capacity, the training of *deep* neural networks, possessing several hidden layers and thus capable of modelling complex, highly nonlinear transformations, became practically achievable. While it was a long-held belief that complex concepts can not be represented compactly by *shallow* architectures, only a slew of new state-of-the-art results has made the case conclusively. This series of evolutionary improvements came to be known as *deep learning*.

Observing that most dramatic learning in children occurs during the time that the child is rapidly developing, both physically and mentally, Elman [1993]

[1] Artificial neural networks can also be seen as models for investigating the inner workings of the brain in cognitive science; this viewpoint, however, is out of the scope of this thesis.

proposed a method of *curriculum learning* – a way of mimicking this learning phenomenon when training machine learning models. One approach adjusts the distribution of training inputs in time, such that a correct broad-strokes generalizations can be learnt very soon on easier, smaller or less noisy examples, and the peculiarities of harder instances are learnt during the later phases. Another approach puts loosening constraints on the representational capacity of the model, imitating child's maturation.

Chapter 1 provides an overview of the origins of artificial neural networks, from the first logical models to modern multi-layer networks trained by gradient methods. Convolutional neural networks, a subtype specialized for visual tasks, are introduced in chapter 2. The modern paradigm of deep learning, from earlier difficulties to current state-of-the-art methods, is detailed in chapter 3. Curriculum learning, a pair of general approaches facilitating fast and correct learning, is analyzed in chapter 4. This thesis proposal culminates in chapter 5, which describes ideas suitable for further research.

Chapter 1

Origins of Neural Networks

The first solid evidence of brain as a complicated system of interconnected individual neurons was collected by Santiago Ramón y Cajal, frequently designated as the "father of modern neuroscience". Using the so called Golgi's staining method, he was able to visualize individual neurons with enough detail to discern synapses, disproving the popular *reticular theory*, which postulated that the brain is a single continuous network.

The discovery of discrete independent but connected neurons gave rise to *connectionism*, the prevailing approach in cognitive science of modelling cognitive processes as emergent behaviour of simple interacting neurons, and in turn *artificial neural networks* and their later uses in machine learning.

1.1 The Logical Neuron

The first conceptualization of an *artificial neural network* has been proposed by McCulloch and Pitts [1943]. Their model consists of independent units – *logical neurons* – which realize a simple logic function. In accordance with biological neurons, these units have a single logic output (corresponding to the *axon*) and an arbitrary number of logic inputs, which can be either excitatory (corresponding to *dendritic synapses*) or inhibitory (corresponding to *somatic synapses*). The output of such a unit is activated (true), if and only if there is no activated inhibitory input, and if at least a fixed number of excitatory inputs

are activated. The basic structure of an artificial neuron is shown in fig. 1.1.

As the authors demonstrate, the logical neuron with suitable inhibitory and excitatory connections can represent the basic logical operations, such as AND, OR or NOT. In fact, any boolean function can be realized by a network composing several of such neurons; networks of logical neurons can thus be used as a universal model of logic computation.

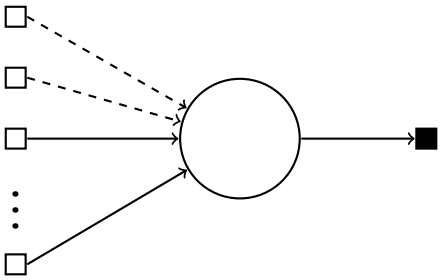


Figure 1.1: Logical Neuron. [*The black square is the output, white squares are inputs, circle is the cell body. Excitatory connections solid, inhibitory dashed.*]

1.2 The Perceptron

Based on the model of the logical neuron, the *perceptron* proposed by Rosenblatt [1958] generalizes the signals in the model from logical values to numeric quantities, with the output of each neuron still being all-or-nothing binary. The structure of such a perceptron, slightly generalized, is shown in fig. 1.2, with $w_1, w_2, \ldots, w_n$ and θ being the adaptable weights. The output activation of a perceptron on an input of $\mathbf{x}$ is given by:

$$y = f\left(\mathbf{w}^T\mathbf{x} - \theta\right)$$

with f being the threshold function: if the weighted input signals to the perceptron are larger than the bias, the neuron fires; otherwise is remains inactive. If firing (true) is interpreted as one and not firing (false) as zero, a network of logical neurons can be easily simulated by a network of perceptrons with correctly set weights. Therefore, like logical neurons, a sufficiently large network of perceptrons can represent an arbitrary boolean function.

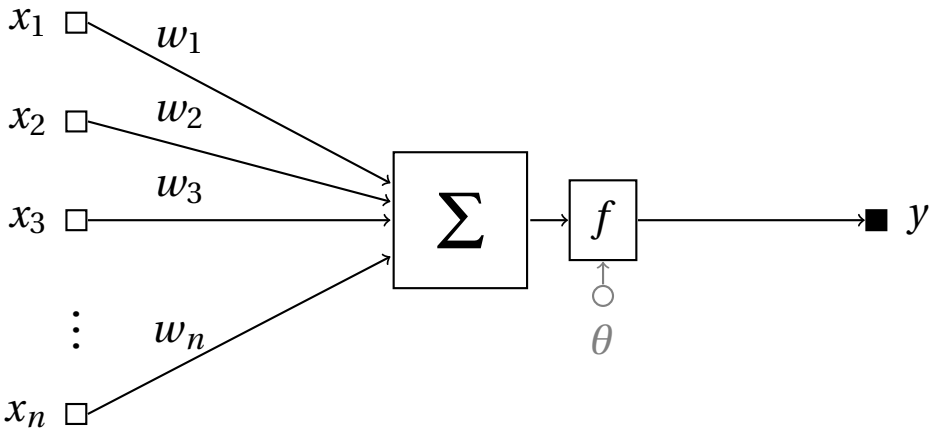


Figure 1.2: The Perceptron. [Simultaneously: the Adaptive Linear Combiner, the ADALINE or a modern artifical neuron, depending only on the choice of f.]

Real-valued signals aside, the most novel property of this model is that the weights are *adaptive* and the perceptron can therefore *learn*, if given the correct output, using a simple update rule:

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \alpha(d-y)\mathbf{x}$$

where $\mathbf{w}$ is the vector of weights, $\mathbf{x}$ is the vector of the current inputs, d and y are the desired and actual responses, respectively, and α is the learning rate.

A book dedicated to perceptrons was published by Minsky and Papert [1969], describing the model, its history and its properties in detail. Among a number of rigorous mathematical analyses of the model's behaviour in various circumstances, many were targeted at the model's representational and learning capabilities. In addition to positive results, such as that a single unit will successfully learn to discriminate two linearly separable classes, many results were negative: a single unit cannot represent an exclusive or (XOR) operation, or that a single unit cannot determine whether a presented shape is connected.

Of particular interest is a rather innocently-looking result, that for some tasks, neurons with connections to *all* the inputs are required. This observation is irreconcilable with the limited number of connections of the biological neurons, suggesting that they are in some way more capable. It also implies

that nearly all then-existing physical designs of learning machines based on the perceptron, which frequently did not feature dense connections, were non-universal.

In follow up appearances, the authors also stated their disbelief that adding layers could significantly improve perceptron's capabilities. The resulting negativity in the artificial intelligence community lead to the so-called *AI winter*, with many of the research groups abandoning their research projects in favour of *symbolic* learning systems.

1.3 Backpropagation

While neural models having multiple layers were proposed and experimented with before, the hidden layers were not trained in a conventional fashion, but were either fixed (and commonly constructed only of random connections), or used self-organization to learn their parameters. The first practical method for supervised training of models with hidden layers – *backpropagation* – was proposed by Werbos [1974] in his dissertation thesis.

Combining the method of gradient descent with the chain rule of derivatives, he arrived at an efficient layer-wise algorithm to compute the error gradients required for gradient descent. Using his technique, network parameters can be adjusted in two passes through the network: in the *forward* pass, the activations of the neurons are computed, and in the *backward* pass, the teaching error signal from the output layer is *propagated* in the reverse direction to calculate local gradients. The back-propagation involves computing the derivatives of the activation functions, which therefore must be continuous.[1] Unfortunately, his thesis failed to reach a wider audience and the method remained unknown.

The backpropagation algorithm in the context of neural networks was also independently rediscovered in 1985 by both Parker and LeCun, but the method became general knowledge only with publication of the book *Parallel Distributed*

[1] This means that a model possessing a neuron with a discontinuous activation, such as the threshold function of the perceptron, can not be trained by backpropagation.

Processing, where a chapter by Rumelhart et al. [1986] described and analyzed the rediscovered technique in detail, under the name of *generalized delta rule*.

Introducing a large number of artificial tasks, they have shown that multilayer networks can learn to represent a variety of transformations, including logical XOR, binary coding and decoding, binary addition, parity and recognized symmetric strings. In a model reminiscent of later convolutional neural networks, they managed to distinguish shifted and rotated "T" and "C" shapes. Some of the experiments managed to successfully train recurrent networks, with the technique now known as *backpropagation through time*.

Aside from demonstrating the success of the method and general usefulness of artificial neural networks, refuting Minsky and Papert [1969], their experiments introduced a number of techniques, that are still very relevant and used today, such as proper representations for inputs and outputs, initialization with small random values, sigmoid activations and training with momentum.

After the year 1990, despite the success of the backpropagation method, artificial neural network generally went out of fashion with the machine learning community, instead being replaced by simpler robust models, such as *support vector machines*, which were easier to apply, faster to train at the time and required almost no choices of architecture and less tuning of hyper-parameters.

Chapter 2

Convolutional Neural Networks

For tasks involving visual inputs, *convolutional neural networks* arose as a domain-specific variation of classical neural networks, incorporating the structure of biological neurons in visual areas of the brain into their artificial architectures (first in [Fukushima, 1980]). With neuron types modelled after the simple and complex cells of the primary visual cortex of mammalian brains (as described in [Hubel and Wiesel, 1959, 1962]), they quickly proved superior to other approaches, even surpassing human performance in many image classification tasks (e.g. flawed photos of traffic signs, see [Cireşan et al., 2012]).

Aside from the principal area of visual pattern recognition, variants of convolutional neural networks were also successfully employed to non-visual problems, such as speech recognition (starting with the Time-Delay Neural Networks of Waibel et al. [1989]), drug discovery (such as the AtomNet of Wallach et al. [2015]) and position evaluation in boardgames (historically beating a human Go expert, see [Silver et al., 2016]).

2.1 Biological Origins

The role of individual neurons in mammalian vision was first described by Hubel and Wiesel [1959] in a series of experiments on cats. Presenting various stationary or moving patterns of light to sedated animals, they measured the responses of individual neurons in the primary visual cortex using temporar-

ily implanted electrodes. While diffuse global illumination of the retina produced almost no response, small patterns of light were met with strong local responses, corroborating the then-new concept of *receptive fields*. For most of the studied neurons, a set of highly localized excitatory and inhibitory areas could be identified. Frequently, the receptive field for a single neuron is constructed in such a way that the cell presents a strong response only for a light boundary oriented in a specific way. These locally sensitive neurons, often identifiable as orientation-specific line or edge detectors, were later named *simple cells*.

In one of their subsequent papers, Hubel and Wiesel [1962] also described a more sophisticated type of cell, assigning it the name of *complex cell*. While the response of a simple cell is highly localized, the complex cells have receptive fields that tend to span a larger area.[1] They exhibit the same type of shape- and orientation- selectivity towards visual stimuli, but are sensitive regardless of the position of the stimulus within the receptive field – a form of *translation invariance* in a limited area. The authors also argue that the characteristic responses of complex cells arise as a fusion of responses from a number of simple cells.

2.2 Neocognitron

Inspired by these biological functional arrangements, Fukushima [1980] proposed a novel model of an artificial neural network specifically suitable for computer vision, which he titled *Neocognitron*, acknowledging the similarities to *Cognitron*, one of his earlier models (introduced in [Fukushima, 1975]). The model itself consists of alternating layers of *S-cells* and *C-cells*, containing cells organized into rectangular grids; the input, a virtual retina, being a raster image.

S-cells, modelled after the simple cells of the mammalian primary visual cortex, are the fundamental computational elements responsible for feature recognition. Opposed to standard perceptron-like artificial neurons, and in accor-

[1] usually in the order of single degrees of visual angle

dance with biological ones, the S-cells only process inputs in a localized area of the previous layer. These receptive fields are typically restricted to a certain radius from a central unit; the S-cells themselves are divided into *cell planes*, containing neurons selective for the same *feature*. The cells within a single plane are organized into a rectangular grid, possessing *shared weights* – recognizing the same feature, each at a slight different central position. Neural connections to the S-cells are plastic and provide the model with a capability to learn.

The recognized features from S-cells are then fed into a layer of *C-cells*, which mimic complex cells. While both biological complex cells and Neocognitron's C-cells possess elaborate response patterns, the C-cells only perform a simple consolidation of input signals – the cell fires when at least one of its afferent S-cells is firing. While these cells are organized in the same way as S-cells – preserving the cell planes representing distinct features – their simple operation needs no parameters and therefore they are unchanged by the learning process.

2.2.1 Training

In the original paper, Fukushima [1980] describes an *unsupervised learning* method for the model. The only cells with plastic connections – the S-cells – are presented with an input (either from the virtual retina or from another layer of C-cells) and, in a *winner-takes-all* fashion, the connections to the most strongly receptive S-cells are strengthened, while the connections to the rest of the cells are suppressed.[2] As the weights of the S-cells are shared across a cell plane, this variant of *competitive learning* leads to each plane recognizing a single feature at different positions. In a later paper, Fukushima [1988] showed that models consisting of several stacked layers of S-cells and C-cells can tolerate significant input deformations while retaining their detection capabilities.

[2]The original formulation involves fixed inhibitory *V-cells*, which provide a signal equal to the average activation of the preceding layer; incorporating the resulting dynamic directy into the update rule yields a clearer, but equivalent characterization.

2.3 Modern CNNs

From another perspective, researchers working on backpropagation driven supervised learning, struggling sometimes with slow convergence and insufficient generalization, have come to the conclusion that suitable application of *weight sharing* can frequently address both these concerns. In visual tasks, a natural arrangement of shared weights is the one employed in Neocognitron's S-cells: detection of a single optical feature with positional invariance, realized by a grid of neurons with shared parameters, but shifted receptive fields.

2.3.1 Weight Sharing

As a part of a suit of toy tasks, Rumelhart et al. [1986] used a single hidden layer of shared-weight neurons to distinguish between binary "T" and "C" shapes fixed in size, but variable in position and orientation. The hidden layer consisted of a two-dimensional grid of neurons, each having the same shared weights, feeding from a very limited 3×3 *receptive field* centered around the individual hidden neuron,[3] thus achieving rudimentary translational invariance.

2.3.2 Feature Maps

To tackle a much more complicated task of optical character recognition, LeCun [1989] examined this simple model along with several of its variations and extensions, many of them also paralleling the Neocognitron's structure. Experimenting on a medium-sized database of handwritten digits, he showed that the task can be easily solved by a simple multi-layer perceptron with a single hidden layer; while the training accuracy can reach 100%, the generalization capability suffers at 80%. Models that are more local and constrained lead to a better performance: receptive-fields alone lead to generalization accuracy of 88.5%, with simple weight sharing raises this to 94%. Further improvement to 98.4% can be achieved when two stacked grid-like layers extract hierarchical features, with

[3] The hidden layer sits on top of the rectangular input image in a conventional fashion.

each layer having several types of independent neurons – *feature maps*[4] – with their own shared weights; the resulting increase in the number of adjustable parameters is curbed by having fewer extracting neurons arranged with a stride over the inputs, thus virtually sub-sampling the inputs.

2.3.3 Pooling

This thinning method was further refined to an operation of *max-pooling* by Weng et al. [1993], where the layer of extracting neurons is kept dense, but their output activations are thinned by a pooling layer. The convolutional outputs are divided into small rectangular areas, commonly in non-overlapping 2×2 patches, with all the outputs from the area combined into a single activation, passing the largest value through. This in turn keeps the gradients sparse, as only one of the reduced activations actually contributes to the output value; this helps both the quality of generalization and the speed of learning. The max-pooling operation became widely used and displaced virtually all other reducing operations in practice.

[4] cf. the Neocognitron's cell planes

Chapter 3

Deep Learning

While there is no clear consensus of what constitutes a *deep neural network*, prior to the year 2006, neural network models possessing more than two hidden layers were very rarely involved in any practical results. While they have appeared in various unsupervised learning contexts, such as the Neocognitron of Fukushima [1980], attempts at training them in a supervised fashion were overwhelmingly unsuccessful.

The most significant problem in the training of deep models with gradient descent methods was identified by Hochreiter [1991]: as the error gradients are backpropagated from the output layer backwards through the model, their scale is affected on each layer. If this transformation is biased in a stable way, its effect is exponentially amplified with each layer. This in turn frequently causes the gradients in the layers distant from the output to be unsuitable large or small. He called this phenomenon the problem of *vanishing/exploding* gradients in his follow-up publications (his original thesis is in German).

3.1 The Pre-training Breakthrough

For the supervised training methods to work, the network has to be initialized cautiously to avoid these pathological scaling effects. One group of techniques that can mitigate the problem is *pre-training*, in which the supervised training phase is preceded by (usually) unsupervised training. During this stage,

the model learns meaningful internal representations for the inputs, which then serve as an adequate initialization of the next phase.

3.1.1 Restricted Boltzmann Machines

The breakthrough occurred with the work of Hinton et al. [2006], who used unsupervised pre-training of *stacked restricted Boltzmann machines*, matched to the desired architecture of the final network. While the Boltzmann machine was not a new model, and neither was its restricted variant – for which efficient training procedure of *contrastive divergence* was introduced also by Hinton [2002] – the core improvement of this approach is training the deep model greedily, one layer at a time. This efficient training method allowed the success of the model, now generally known as *deep belief networks*.

3.1.2 Stacked Auto-Encoders

Pre-training using deep belief networks was further investigated by Bengio et al. [2007], extending the method to continuous valued inputs and developing an analogous layer-wise greedy approach, replacing stacking relatively obscure restricted Boltzmann machines by *stacked auto-encoders*, a well-known artificial neural network model.

Using unsupervised pre-training, a similar conditioning effect can be obtained as with deep belief networks, albeit the resulting network appears to be less capable of correct generalization. *Supervised* pre-training also helps to train the final model, but the generalization potential is inferior compared the to unsupervised methods, both stacked auto-encoders and deep belief networks.

3.1.3 Denoising Auto-Encoders

The approach of stacking auto-encoders was further improved with a variant of the model, called *denoising autoencoder* [Vincent et al., 2008, 2010], reaching performance competitive with deep belief networks. While a standard auto-encoder is trained to reconstruct its input, the task of a denoising auto-encoder

is the reconstruction of a degraded input signal. Three types of corruption of the input $\mathbf{x} = \langle x_1, x_2, \ldots, x_k \rangle \in \mathbb{R}^k$ to a degraded input $\mathbf{x}' \in \mathbb{R}^k$ were considered:

- additive isotropic Gaussian noise:

$$(\forall i)\ x'_i := x_i + n;\quad n \sim \mathrm{N}\left(0, \sigma^2\right)$$

- masking noise:

$$(\forall i)\ x'_i := \begin{cases} 0, & c = 1;\quad c \sim \mathrm{Bern}\left(p\right) \\ x_i, & \text{otherwise} \end{cases}$$

- "salt and pepper" noise:

$$(\forall i)\ x'_i := \begin{cases} \pm 1, & c = 1;\quad c \sim \mathrm{Bern}\left(p\right) \\ x_i, & \text{otherwise} \end{cases}$$

The amount of applied noise is controlled by a single parameter, either standard deviation σ or p, representing a probability of a malfunctioning input unit. In the case of masking noise, the unit is supposed to be inactive, while the "salt and pepper" noise represents a unit stuck in a saturating state.[1]

Pre-training accomplished by this method is comparable in performance to stacking restricted Boltzmann machines, and superior to stacking classical auto-encoders; the choice of corruption noise and it's magnitude is not crucial. The authors also show that unlike standard auto-encoders, the denoising types learn internal representations for the training patterns that are crisp and salient.

3.2 The Evolution of Activation Functions

While the logical neuron of McCulloch and Pitts [1943] (see section 1.1) used only propositional logic to determine whether a neuron *fires* (is active) or not, the following models generally use a linear combination of inputs to determine

[1] The choice of ± 1 is therefore only valid for symmetric sigmoid activations with the given range; for the classical logistic sigmoid, 0 and 1 would be used.

the real-valued output activation. In this form, the output activation y of a single neuron is given by the vector of inputs $\mathbf{x}$, the weight vector $\mathbf{w}$ and the b bias[2] coefficient:

$$y = f\left(\mathbf{w}^T\mathbf{x} + b\right) = f\left(\mathbf{w}\cdot\mathbf{x} + b\right)$$

The product of the inputs and the weight vector, along with the bias coefficient, is commonly referred to as *net input* n (or in more biological contexts, *pre-synaptic activation*):

$$n = \mathbf{w}\cdot\mathbf{x} + b = \sum_i w_i x_i + b$$

And the output *activation* y (rarely, but more precisely, *post-synaptic activation*) is the result of applying an *activation function* (or, frequently in engineering contexts, a *non-linearity*) to n:

$$y = f\left(n\right)$$

The first model to use this real-valued computation for the purpose of classification was the perceptron of Rosenblatt [1958] (see section 1.2), utilizing a *threshold* activation function:

$$f(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

Later iterations of this model generally used *continuous* functions, that were *differentiable* at (nearly) every point. This in turn enables the use of *gradient descent* (see section 3.4.1) for tuning the parameters – instead of *ad hoc* update rules: the perceptron, for example, with its discontinuous threshold activation can be trained using a simple update rule:

$$\Delta\mathbf{w} := \alpha(d - y)\mathbf{x}$$

[2] There are also other equivalent ways of formalizing a bias, a common one is augmenting the input by a constant of ± 1 (correspondingly, the weight vector gains one additional coefficient).

where $\mathbf{x}$ is the input vector, d and y are the desired and actual output scalars respectively, α is the learning rate and $\Delta\mathbf{w}$ is the resulting update to the weight vector $\mathbf{w}$.

Linear Regression

While the perceptron model is designated for classification, it can be used for purposes of regression with a small modification – the output activation function will be purely linear,[3] that is:

$$f(x) = x$$

Using the same update rule as for classification with a threshold perceptron:

$$\Delta\mathbf{w} := \alpha(d - y)\mathbf{x}$$

The single linear neuron performs *linear regression* – it converges to a value of parameters that minimize the squared output error. Indeed, this update rule directly corresponds to a gradient descent step, when the cost function is quadratic.

Logistic Sigmoid

To perform classification on datasets that are not linearly separable, such as the XOR problem, a more complicated model is required. An easy biologically plausible way is to *stack* two (or more) perceptron-like layers, the outputs of the first layer serving as the inputs to the second layer:

$$\mathbf{h} = f\left(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}\right)$$
$$\mathbf{y} = g\left(\mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)}\right)$$

where the $\mathbf{h}$ is the activation of the hidden layer. It can easily be shown, that in order for the model to gain expressive power by this addition, the function f

[3] Other monotonous continuous functions can also be used; suitability varies with the desired distribution of modelled outputs. The update rule must also be modified accordingly.

must not be linear – if f were an identity[4] transformation:

$$\begin{aligned}
\mathbf{y} &= g\left(\mathbf{W}^{(2)} f\left(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}\right) + \mathbf{b}^{(2)}\right) \\
\mathbf{y} &= g\left(\mathbf{W}^{(2)} \quad \left(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}\right) + \mathbf{b}^{(2)}\right) \\
\mathbf{y} &= g\left(\underbrace{\mathbf{W}^{(2)}\mathbf{W}^{(1)}}_{=:\hat{\mathbf{W}}}\mathbf{x} + \underbrace{\mathbf{W}^{(2)}\mathbf{b}^{(1)} + \mathbf{b}^{(2)}}_{=:\hat{\mathbf{b}}}\right) \\
\mathbf{y} &= g\left(\hat{\mathbf{W}}\mathbf{x} + \hat{\mathbf{b}}\right)
\end{aligned}$$

As for the choice of *non-linear* activation function for the hidden layer, while the threshold function will lead to greater expressive power,[5] it is unclear how to find the parameters. Fortunately, gradient descent methods may be used when all the employed functions are continuous. The oldest such employed function is the *logistic sigmoid*:

$$f(x) = \frac{1}{1 + e^{-x}}$$

These oldest used activation functions are visualized in fig. 3.1.

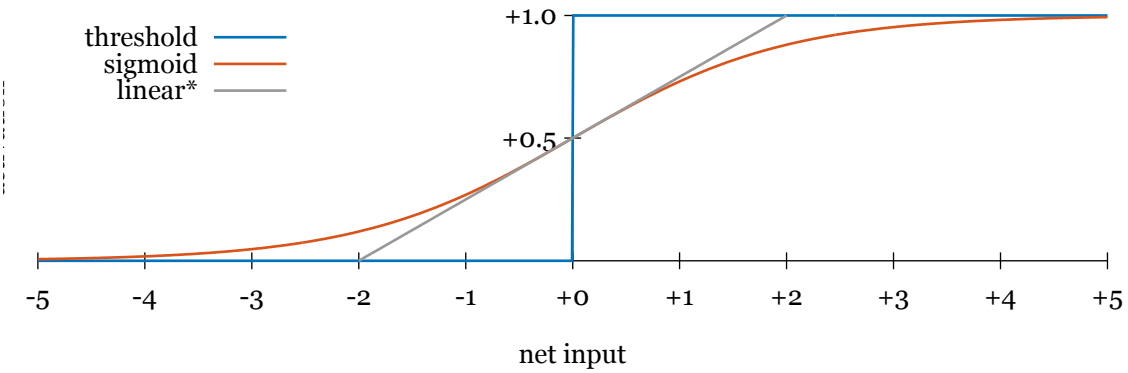


Figure 3.1: Unipolar activations. [*Linear is* $y = \frac{1}{4}x + \frac{1}{2}$*, to match the sigmoid.*]

[4]The same procedure would work for any linear function, but as inputs to f are already transformed linearly, it would only add to confusion, but not to generality.

[5]For example, any boolean function can be expressed with a sufficiently large network with one hidden layer and threshold activations all around.

3.2.1 Symmetric Activations

Both the threshold and logistic sigmoid activations have output values in the range of $[0; 1]$ – they can be therefore classified as *unipolar*. While this is biologically plausible, better results have been consistently achieved using *bipolar* activation functions, which attain both positive and negative values. These functions are frequently symmetric around zero, which in turn helps keep the mean activation values close to zero.

The first such widely used non-linear symmetric function is the *hyperbolic tangent* function, which is in fact a rescaled logistic sigmoid:

$$f(x) = \tanh(x)$$

When a sigmoid unit is pushed to saturation – its outputs close to extremal values – the gradient's magnitude drastically decreases and learning using gradient methods becomes very slow. To lessen this impeding effect, a new activation function was proposed by Glorot and Bengio [2010], which saturates more gradually than hyperbolic tangent – the *softsign* function:

$$f(x) = \frac{x}{1 + |x|}$$

These symmetric activations are compared in fig. 3.2.

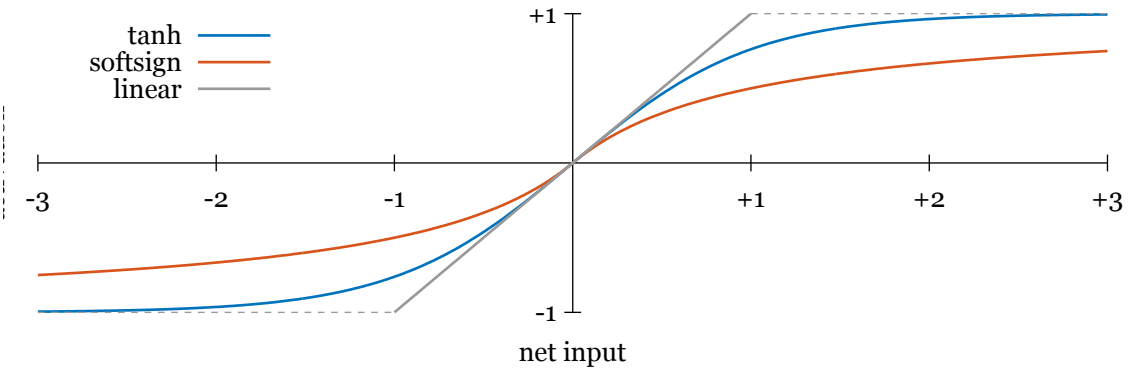


Figure 3.2: Bipolar activations. [*Linear is $y = x$, to match the central slope.*]

3.2.2 Rectifiers

Recently, the sigmoidal activation functions were overshadowed in the area of deep learning by the much simpler *rectifiers*, which are not symmetric. Their response for positive values is unbounded, which prevents saturation problems, while their response for negative values is either zero or very small, which provides sufficient non-linearity to model highly complex functions.

Rectified Linear Unit

The simplest kind of rectifier is the *rectified linar unit* (ReLU):

$$f(x) = \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

This very simple formulation has several advantages, the most important being the sidestepping of the problem of vanishing and exploding gradients, as the derivation for the positive values is a constant 1. The zero response for negative values forces *sparsity* of both activations and gradients, and can bolster generalisation as well as speed up training.

Leaky and Parametric Rectifiers

If a rectifier neuron is inactive, then the net activation is less than zero, and the local gradient is zero – an inactive rectifier thus cannot learn. To counter this zero gradient, several variants of rectifiers have been proposed. The simplest is the *leaky rectifier*, introduced by Maas et al. [2013], which has the slope of the negative field non-zero, but only small; the slope is controlled by a parameter α:

$$f(x) = \begin{cases} x, & x \geq 0 \\ \alpha x, & x < 0 \end{cases}$$

He et al. [2015] proposed also having the parameter α fully trainable, calling this modification *parametric rectified linear unit* (PReLU).

Exponential Linear Unit

Recently, Clevert et al. [2015] proposed a rectifier variant called *exponential linear unit* (ELU), in which while the positive field is linear, the negative field is unconventionally sigmoid, saturating at $-\alpha$:

$$f(x) = \begin{cases} x, & x \geq 0 \\ \alpha(e^x - 1), & x < 0 \end{cases}$$

The activation functions for the various rectifier types are show in fig. 3.3.

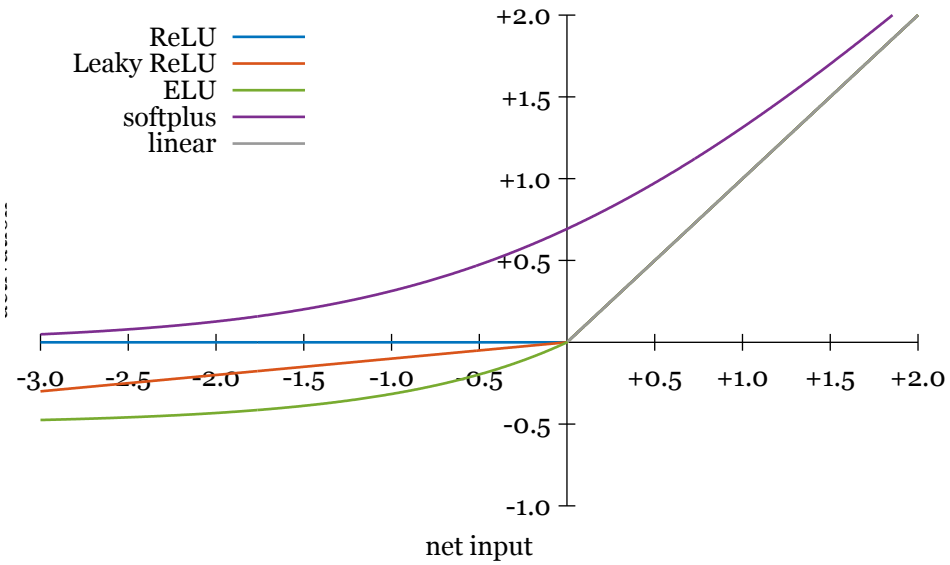


Figure 3.3: Common rectifier variants. [*Leaky ReLU:* $\alpha = 0.1$*; ELU:* $\alpha = 0.5$. *The grey line represents the shared positive slope of the ReLU and its variants.*]

3.3 Initialization Schemes

For a wide variety of iteratively-trained models in the area of machine learning, the initial values of parameters, if not zero, are frequently chosen to be "small random numbers". Random non-zero weights are an initial requirement for various gradient descent algorithms, providing an easy way of *breaking the*

symmetry – if multiple independent units of computation, such as artificial neurons or initial centers of clusters, were to be initialized with the same parameters, they would adapt in exactly the same way.[6] Artificial neural networks are no exception to this symmetry, but as it turns out, correctly chosen initialization is crucial for the learning capability of the network.

For most of the history of artificial neural networks, the initial weights were picked from a either a uniform or a normal zero-mean distribution, scaled[7] by a hand-picked coefficient s in the order of 10^{-3}:

$$\mathbf{W} \sim \mathrm{Uni}(-s, +s) \qquad \text{or} \qquad \mathbf{W} \sim \mathrm{N}(0, s')$$

The difference between the two distributions, with parameters chosen as to have the same mean and variance, is shown in fig. 3.4.

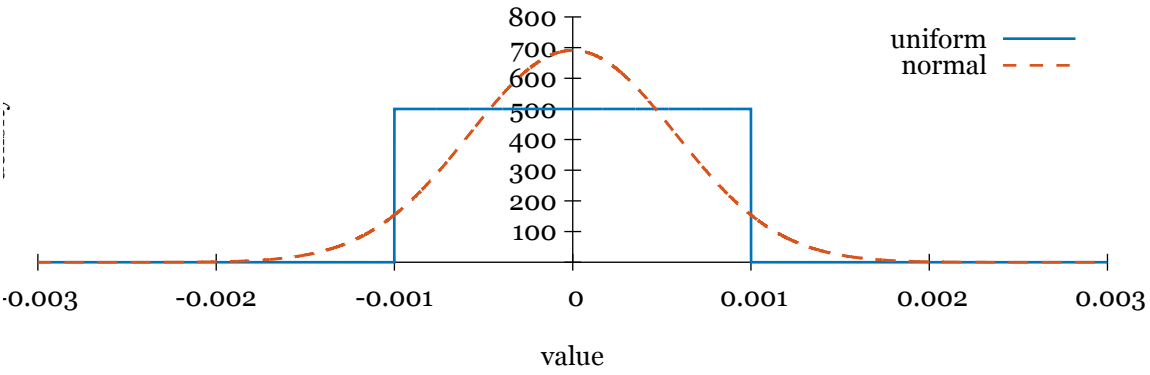


Figure 3.4: Probability density of uniform and normal initializations.
[*Blue:* $\mathrm{Uni}(\pm 10^{-3})$, *orange:* $\mathrm{N}(0, \sqrt{1/3} \cdot 10^{-3})$; *mean:* 0, *variance:* $1/3 \cdot 10^{-6}$.]

The initialization of *bias* weights is frequently done by sampling the same distribution, or, alternatively, a *zero* initialization is used; if the rest of the weights are random, this does not present a problem for symmetry breaking.

[6] Under the assumption that the training method is deterministic, which most of them are. Even *stochastic* gradient descent is deterministic, except for the order of input presentation; therefore, from the point of view of simultaneously adapting units, it's locally deterministic.

[7] For the distributions to have the same variance, $s' = \sqrt{1/3}s$ should hold.

3.3.1 Normalized Initialization

When advances in computational capabilities allowed experimenting on models with hundreds or thousands of hidden neurons, it became quickly obvious that the architecture of the network should play a role in the choice of the scale constant s. One common heuristic for this scale is:[8]

$$\mathbf{W} \sim \mathrm{Uni}\left(\pm\frac{1}{\sqrt{\deg_{in}}}\right)$$

While this scaling keeps the activations of the neurons mostly on the edge between saturation and the central linear regime, Bradley [2009] observed that no such useful property holds for the scale of backpropagated error gradients. The gradient of the output layer is the largest, as only the weights of the output layer are directly responsible[9] for the output error. As the gradient is backpropagated towards the inputs, the attribution of error becomes gradually less clear – in fact, the scale of the backpropagated gradients declines *exponentially* with the number of layers.

A thorough analysis of the statistical distributions of neuron activations and backpropagation gradients was performed by Glorot and Bengio [2010], both analytical and empirical. To maintain useful scales of both activations and gradients, a compromise is required – they proposed the so-called *normalized initialization*, where:[10]

$$\mathbf{W} \sim \mathrm{Uni}\left(\pm\sqrt{\frac{6}{\deg_{in}+\deg_{on}}}\right)$$

Using this carefully scaled initializations, they managed to successfully train deep networks, with the speed and quality of convergence either comparable, or only slightly inferior to the much more complex and computationally expensive pre-training methods. As a result, this scaling scheme is now commonly employed as a default choice of initialization in many neural network libraries.

[8] where deg_{in} is the size of the previous layer

[9] the resulting update scheme for the output layer very much resembles the Perceptron update rule, which just a special case for linear output layers

[10] where deg_{on} is the size of the *current* layer

Some of the assumptions of this derivation only hold for linear and sigmoidal activations, but not for rectifiers, as they are asymmetric around zero. To address the invalid assumptions, complementary analysis for rectifiers was performed by He et al. [2015], yielding:

$$\mathbf{W} \sim \mathrm{N}\left(0, \sqrt{\frac{2}{\deg_{in}}}\right)$$

While they chose to use the normal distribution in their paper, the analysis works regardless of the distribution: for uniform distribution, the scale would be $\sqrt{6/\deg_{in}}$. Conversely, the normalized initialization of Glorot and Bengio [2010] for normal distribution would be $\sqrt{2/\deg_{in} + \deg_{out}}$.

3.4 Variants of Stochastic Gradient Descent

For most but not all of supervised, unsupervised or reinforcement learning tasks involving neural methods, model- and configuration-specific update rules have given way to general error/energy minimization algorithms, the bulk of which is based around *gradient descent* techniques, and infrequently also *second-order methods*. While these methods are not new themselves, their spread to training neural network was hindered by ineffective formulations and insufficient computational power.

3.4.1 Stochastic Gradient Descent

In the simplest formulation for minimizing some error measure (loss function) E by changing the set of all trainable parameters $\mathbf{W}$ of the model, the general gradient descent algorithm update is as follows:

$$\begin{aligned} \mathbf{W}(t+1) &= \mathbf{W}(t) + \Delta\mathbf{W}(t) \\ \Delta\mathbf{W}(t) &= -\alpha \nabla E\big(\mathbf{W}(t)\big) \end{aligned}$$

where $\Delta\mathbf{W}$ is the computed weight update and α is the learning rate.

Layered neural network models, such as feed-forward neural networks, but also many types of recurrent neural networks, can be effectively trained by the *backpropagation algorithm*. While functionally equivalent to general gradient descent, this layer-wise formulation for artificial neural networks, arguably originally due to Werbos [1974] and popularized by Rumelhart et al. [1986], is particularly effective for computing gradients in layered models.

3.4.2 Loss Functions

As gradient descent methods minimize a selected measure of prediction error, the choice of such a loss function can be influential. When domain knowledge to accurately gauge the quality of a solution is unavailable, a standard choice of loss function in the fields of machine learning and statistics is the mean squared error (cf. various *least squares* methods):

$$E = \frac{1}{2}\|\mathbf{d} - \mathbf{y}\|^2 = \frac{1}{2}\sum_i \big(d_i - y_i\big)^2$$

where $\mathbf{d}$ is the desired output vector and $\mathbf{y}$ is the actual output vector; for classification tasks a *one-hot encoding* is used,[11] where an object belonging to a k-th of n classes is labeled by a vector $\mathbf{d} \in \mathbb{R}^n$, where $d_k = 1$ and $\big(\forall\, i \neq k\big)\; d_i = 0$.

In classification tasks, the mean squares error has been mostly supplanted with *cross-entropy loss* (sometimes also called *log-loss* or *negative log-likelihood* and closely related to entropy and the Kullback-Leibler divergence):

$$E = -\sum_i \big(y_i \log d_i\big)$$

The theoretical foundation for this loss function concerns the information loss between two probability distributions; therefore a correct representation of both the actual output and the target label must be chosen. For the labels, the one-hot encoding, optionally with ϵ, is sufficient; to be able to interpret the network output as a vector of probabilities, the output must be normalized as

[11] Sometimes, the extremal values are $0+\epsilon$ and $1-\epsilon$, for numerical stability, and in case of bipolar functions, -1 or $-1+\epsilon$ must be used.

to have only non-negative entries that add up to 1. The most common method is to have the output layer of neurons use the *softmax* activation function:

$$\hat{y}_i = \frac{e^{y_i}}{\sum_j e^{y_j}}$$

Alternative loss functions for classification do exist: one such has been proposed by Tang [2013], based on the *hinge loss* of support vector machines. While it has been found to improve classification performance slightly, it remain relatively unknown and unused.

3.4.3 The Importance of Momentum

The update step of gradient descent can be modified with the addition of a *momentum* term. Each update step changes the parameters in the linearly combined direction of two components: the gradient at the current position and the step of the previous update. Due to a number of variants of this technique, it is commonly referred to as *classic momentum*. The update step is as follows:

$$\Delta\mathbf{W}(t) = \underbrace{-\alpha\nabla E\big(\mathbf{W}(t)\big)}_{\text{gradient step}} + \underbrace{\gamma\Delta\mathbf{W}(t-1)}_{\text{momentum step}}$$

A significant effect of the momentum is that when the sequential update steps happen in the same general direction, the magnitude of the updates *increases*; when they differ significantly, the magnitude decreases. Gradient descent techniques enhanced by momentum therefore gain some of the benefits of second-order methods: large steps in the areas of low-curvature and small careful steps in the areas of high curvature. This can significantly speedup, and even enable convergence; strong momentum may also exert a regularizing effect on the parameters.

Nesterov Accelerated Gradient

The classic momentum technique can be modified slightly: the gradient for an update will not computed at the previous position, but at the previous po-

sition shifted by the imminent momentum step. This modified technique is known as the *Nesterov accelerated gradient*:

$$\Delta\mathbf{W}(t) = -\alpha\nabla E\big(\mathbf{W}(t) + \gamma\Delta\mathbf{W}(t-1)\big) + \gamma\Delta\mathbf{W}(t-1)$$

This method has been proven to converge quadratically faster than classic momentum for convex optimization problems and is generally believed to be more responsive to changing curvature during training, to a similar degree than proper second-order methods. While the optimization of artificial neural networks, is generally strongly non-convex, advantageous behaviour was still documented on various tasks by Sutskever et al. [2013], exposing the little-known technique to a wider audience of artificial neural researchers and practitioners.

3.4.4 Second-order Methods

The aforementioned gradient descent techniques utilise only the *first-order* information, i.e. the value of a function at a point and the first derivatives[12] of the function at a point. These approaches, as they work with only finitely small steps, often have problems overcoming areas of high curvature, where the optimal direction (which is the gradient) changes quickly.

Second-order methods also utilize second derivatives of the function, the *Hessian matrix*; in particular, the basic *Newton's method*[13] uses the inverse of the Hessian. However, for a function $f : \mathbb{R}^n \to \mathbb{R}$ the gradient – the vector of first derivatives – has n entries, while the Hessian – the matrix of second derivatives – has n^2 derivatives. For artificial neural networks in current practice, where the number of parameters can easily reach hundreds of millions, the storage itself (disregarding the costs of actual computation) of the Hessian matrix is infeasible, rendering direct methods futile. To overcome this limitation, various *Quasi-Newton* methods have been proposed that do not employ the Hessian matrix (or its inverse) in its full form. Popular methods such as *BFGS* and

[12] That is the *gradient*, as the typical function to be optimized in machine learning is a real-valued function from a large number of parameters to a scalar loss.

[13] for optimization, but closely related to the new Newton's method for finding roots

its limited-memory variant *L-BFGS*, that only compute an approximation of the Hessian, have been successfully used to train neural networks.

Of particular importance is the method of *Hessian-free optimizations* – as introduced by Martens [2010] specifically for the training of artificial neural networks – that only computes a product of the Hessian and a vector, avoiding the computation of the full Hessian. While the method is able to successfully train networks even without a pre-training requirement, recent advances in initialization, activations and batch normalization have enabled training by first-order methods without pre-training, and the Hessian-free optimization technique faded into background, being once again overshadowed by stochastic gradient descent and its variants.

3.4.5 Batch Normalization

A very recent improvement in training neural networks is the technique of *batch normalization* of Ioffe and Szegedy [2015], where linear rescaling layers are interspersed amidst the many-layered model. These non-parametric layers are not fixed, but change their transformation during the computation to curtail *internal covariate shift* – a phenomenon, where the distribution of activations, appropriate at the time of initialization, changes during the training as lower layers adapt. The transformation is recomputed for each training *mini-batch* and aims to keep the activations having zero mean and unit variance, that is normalized, hence the name. In practical experiments, this method greatly speeds up[14] the convergence, while making the training less fragile to correct initialization and the choice of hyper-parameters. In some cases, even positive effects on generalization capability have been reported.

3.5 Regularization

The progress in the area of training artificial neuron networks – sophisticated models, domain-specific layers and pre-processing, optimized initializa-

[14] As measured by the number of epochs. Time-wise, the added layers amount to a slowdown of about 30% on modern GPUs, which should still be a net improvement.

tions and effective variants of gradient descent – have led to a situation, where the training of the network is frequently not the limiting factor. Performance on both artificial and real-world tasks is now frequently limited by *overfitting*, where the model learns the peculiarities of the training data unrelated to a task, such as sampling irregularities of the dataset. When presented with the previously unseen testing data, the network fails to generalize properly, and the resulting testing error is significantly higher than the training one.

To overcome this difficulty, various methods have been proposed and used successfully. Explicit or implicit *regularization* techniques restrict or penalize unsuitable parameters, while *data augmentation* methods manufacture more training data[15] by various transformations.

3.5.1 Explicit Regularization

Explicit methods achieve regularization by adding a new term to the cost function, which in some ways penalizes unsuitable network parameters. This in turn steers the optimization procedure towards less penalized, and hopefully more suitable, parameters.

L_1-regularization

The simplest, both in its definitions and in its effects, is L_1-regularization, which penalizes the sum of the absolute values of the parameters:

$$L_1\left(\mathbf{W}^{(*)}, \mathbf{b}^{(*)}\right) = \sum_l \left(\|\mathbf{W}^{(l)}\|_1 + \|\mathbf{b}^{(l)}\|_1\right) = \sum_{l,i,j} \left|W_{i,j}^{(l)}\right| + \sum_{l,i} \left|b_i^{(l)}\right|$$

This penalizes all the non-zero weights equally, which quickly leads to irrelevant parameters approaching zero. This condition of *sparsity* leads to better generalization and the resulting more localist representations may even make the learnt parameters comprehensible to further analysis.

[15] Theoretically, with infinite training data, no generalization error will be present. While learning from infinite data would require infinite training time, manufacturing a finite amount of transformed training data can still be markedly beneficial.

L_2-regularization

The second, arguably more common[16] form of explicit regularization is L_2-regularization, which penalizes the sum of squared parameters:

$$L_2\left(\mathbf{W}^{(*)}, \mathbf{b}^{(*)}\right) = \sum_l \left(\|\mathbf{W}^{(l)}\|_2 + \|\mathbf{b}^{(l)}\|_2\right) = \sum_{l,i,j} \left(W_{i,j}^{(l)}\right)^2 + \sum_{l,i} \left(b_i^{(l)}\right)^2$$

The effect of L_2-regularization is more subtle: by penalizing large weights more, excessive weights will be discouraged. Of course, as it's the case with most regularization methods, weights that don't contribute to the models success tend to be pulled towards zero. (This may also sometimes speed up training, as excessive weights may easily push neurons to saturation.)

3.5.2 Implicit Regularization

Methods of *implicit* regularization achieve their ends indirectly, as a result of some modification of the architecture of the network or the training procedure. For example, the *weight decay* technique for gradient descent (see [Krogh and Hertz, 1991] for detailed analysis) adds a $-\lambda\mathbf{W}$ term to the weight update. This can be shown to be equivalent to multiplying the weights by $1-\lambda$, but more importantly, to replicate explicit L_2 regularization:

$$E'(\mathbf{W}^{(*)}) = E(\mathbf{W}^{(*)}) + \lambda L_2(\mathbf{W}^{(*)})$$

Dropout & DropConnect

Very successful in combating overfitting is the *dropout* method of Hinton et al. [2012]. At any given time during training, a fixed fraction p of neurons is masked: their output is set to zero. These switched-off neurons are not participating in the output and so are also not learning. The masks are generated randomly for the presentations of each input (or batch) during training; in the testing phase, no masking occurs. Instead, the input connections are scaled by $1/1-p$ to compensate for the now larger number of active inputs.

[16] It is also possible to use a (linear) combination of various explicit methods at the same time.

This technique aims to break co-adaptation between neighbouring neurons; the authors draw a parallel to the crossing-over process of sexual reproduction, which also discourages fragile interconnected constructions. Various experiments on different configurations of dropout on the input, hidden and output layers are detailed in [Srivastava et al., 2014].

As a refinement, Wan et al. [2013] proposed *DropConnect*, a technique that decreases the granularity of the binary dropout; instead, they perform the masking on the level of individual connections.[17]

3.5.3 Data Augmentation

To offset the limited amount of available training data, *data augmentation* techniques[18] may be utilized; constructing possibly unlimited amounts of new training samples from the data originally available. Such methods need to rely on problem- or domain-specific assumptions, as to which transformations leave the semantic content of the data intact.

For image data, frequent and relatively universal transformations are small shifts and limited resizing or cropping. Flipping or rotation of the input image can often be used in some contexts, but not in others (e.g. horizontal flipping of the letter “p” yields “q”, a rotated photo of a building is perplexing and so on). A particularly powerful and nearly universal method of *elastic distortions* was introduced by Simard et al. [2003], performing small and local but smooth relative displacements of the parts of images.

[17] Multiplicative correction is still done at the level of individual neurons.

[18] Data augmentation can also refer to modification or addition of new features; this is rare in artificial neural networks, but common in machine learning models not so prone to overfitting, such as support vector machines.

Chapter 4

Curriculum Learning

Curriculum learning is a class of teaching strategies in machine learning trying to force the training process to first focus on correctly grasping simple examples, and deferring the learning of complex examples to later stages of training. This can be done explicitly, by delaying the presentation of more difficult inputs, or implicitly, where a constrained model overlooks complex examples.

Though conceived by Elman [1993] as an analogy to natural language acquisition during childhood, curriculum learning closely parallels both *shaping* as applied to animals [Skinner, 1958] and teaching methods used in robotics. From an optimization or a machine learning perspective, curriculum learning can be viewed as a continuation method, is closely related to the technique of *boosting* and also exhibits a strong regularizing effect on the network parameters learnt.

4.1 Elman's Language Experiments

For some complex tasks, such as language acquisition, the greatest learning capacity is present at a very young age, when the brain is still developing, both physically and psychologically. Interested in this contra-intuitive observation, Elman [1993] proposed two approaches to model this phenomenon of benefiting from *"starting small"*. While the aim of his simulations was the connectionist modelling of these learning processes, the methods used and the results obtained are also appealing from a machine learning perspective – as a technique for speeding up the training process and for improving correct generalization.

For the modelling of this observation, he constructed an artificial language acquisition task. Sequences were constructed over a severely limited dictionary of twenty-six words, but with sophisticated grammar, with relative clauses that exhibited verb-object number agreement and included transitive, semi-transitive and intransitive verbs. This grammatical structure renders the task exceedingly difficult for statistical models and impossible to solve using finite state automata; however, the employed *recurrent neural networks* are suitably powerful. The network was presented with the sentence one word at a time and expected to output the most likely word completing the fragment.

In the first approach, the distribution of training examples presented to the model during the learning process is adjusted, with inputs deemed to be easy – represented in this task by short sentences – being more frequent in the early stages and complex samples – long sentences with subordinate clauses – introduced gradually during the later phases. Networks trained in this structured fashion performed significantly better than models trained uniformly.

In the second experiment, the memory of the recurrent network was being randomly erased every few words, simulating the limited working memory and attention span of children. This prevents long-term dependencies in the input sentences to be noticed by the network, focusing the learning on local structures. As the deletion effect is reduced, modelling longer dependencies becomes viable. As with the previous approach, this structured training drastically outperforms the standard method of using an unrestricted model.

Elman's conclusions, most importantly his assertion that limited cognitive capability may be advantageous or even *necessary* for some tasks, such as the examined language acquisition, have met a stern opposition from Rohde and Plaut [1997, 1999]. In a series of experiments with a more intricate network architecture and a more involved task, they have shown that starting small may possess no advantage, or even sabotage the learning process.

4.2 Bengio's Investigation

Disregarding the pessimism exhibited by Rohde and Plaut [1997], the first approach of starting small – varying the distribution of training samples in time – was studied in detail by Bengio et al. [2009], providing formalization, analysis and several illustrative experiments for the technique, assigning it the name of *curriculum learning*. They also discuss the similarities between the technique and the animal training method of *shaping* [Skinner, 1958], teaching methods used in robotics, and boosting and continuation methods of optimization.

Along with other simpler toy examples, a non-trivial visual task was introduced: within a small greyscale image distinguish whether the single geometric shape contained is an ellipse, a triangle or a rectangle. The model used was a standard multi-layer perceptron with three hidden layers, reaching a testing error of ~20% after 256 epochs. An easy version of the generated dataset was also produced, with no elongations[1] and smaller variations in position and size. A network trained for 128 epochs on the easy dataset and then fine-tuned for another 128 epochs on the full training set reached a testing error of ~15%.

The final experiment involves language modelling – circling back to the original area of Elman's paper – predicting the score of a word completing a given context (sequence fragment). The model opens with a word-embedding layer, where each word of the fragment is turned into a fixed-size vector of real values; in comparison, Elman's model represents a small vocabulary directly using a localist one-hot encoding. Using a large[2] vocabulary of twenty thousand words, they extract five-word sentence fragments from the complete text of the English-language Wikipedia, discarding fragments containing words not in the vocabulary. Curriculum learning is performed by gradually growing the size of the vocabulary in five thousand word increments, rejecting fragments in the same fashion. In the testing phase, the model trained in the standard way achieves a somewhat higher error than the model trained gradually.

[1] i.e. circles, equilateral triangles and squares

[2] in context of language modelling, as Elman used a vocabulary of twenty-six words

4.3 Computational Hardness Estimation

Employing the first approach of starting small in any form requires a way of assigning hardness scores to individual training instances. While this is not an obstacle in the artificial tasks used for demonstration by Elman and Bengio, it poses a problem for real-life tasks, for which no such information is commonly available. A partial solution is to invent a task-specific heuristic that would approximate this hardness, but that may not be easy (or possible), or the performance can be unsatisfactory. This requirement therefore hinders the application of the first approach to practical tasks.

To overcome this difficulty, we proposed a method that estimates the hardness in a task-oblivious, universal way for classification tasks [Kuzma and Farkaš, 2015]. The core of our method consists of training a simple model on the same task, and then using its performance on the training set to estimate the hardness of individual input instances. In our case, we trained a *soft-margin support vector machine* to discriminate pairs of input classes, and then computed the hardness scores using distances from the resulting decision plane. Using these hardness scores, an "easy" subset of the training data was selected, and training then proceeded in two phases, using only the easy subset in the first phase, and the complete set in the second.

In an experimental setup as described by Bengio et al. [2009], we tried this approach on his shape recognition demonstration task. Our method provided a modest gain compared to simple training on the full set, but was expectedly surpassed by the original curriculum. Another examination using a real-life handwritten digit recognition task failed to yield any improvements in performance. Full details are available in the original text of the article, see appendix A.

Chapter 5

Thesis Proposal

In parallel to a developing child acquiring complex skills, such as language proficiency, Elman [1993] proposed a pair of approaches capable of assisting computational models, such as artificial neural networks, in mastering these tasks quickly and completely. To demonstrate the merit of these approaches, he successfully applied them to a simple, artificial task of language acquisition. The principal aim of our research is to extend these approaches as to be useful when applying various neural network models to challenging real-life tasks.

Controlling the Presentation of Inputs

The first approach adjusts the distribution of training examples presented to the model during the learning process. At the start, correct broad-strokes generalizations can be learnt very soon on easier, smaller or less noisy examples, while the peculiarities of harder instances are only encountered later. This approach hinges on having a way of assessing the hardness of training instances, which is problematic for real-life problems. We have proposed and examined a computational way of doing such an assessment in a task-oblivious way using soft-margin support vector machines, but using the resulting hardness scores only leads to improvement on certain tasks. We would like to further extend the method and explore its limitations, particularly why this approach appears to be ineffective when using convolutional neural networks.

Limiting Model Complexity

The second approach proposed by Elman imposes constraints on the networks architecture, reducing the model's representation capacity. As the training progresses, these constraints are gradually lifted or lessened, expanding the learning capacity of the network. The assumed effect is focusing the early training phases to learning the broad strokes of the task, while the greater model complexity in the latter phases can subsequently acquire the finer details.

While Elman's original method limits the available memory capacity of recurrent neural networks, we propose to broaden to scope to other successful models of artificial neural networks, such as pure feed-forward networks and convolutional neural networks, with an emphasis on deep models. We would like to propose following general techniques:

- **regularization pressure** – Start the training under a high degree of regularization, promoting basic sparse solutions. When no further progress can be made, lessen the strength of regularization; fine details should be incorporated into the solution. Alternatively, regularization could be replaced with added noise, encouraging resilient internal representations.
- **bottleneck layers** – Represent some of the dense weight matrices as a product of two lower-rank matrices, creating a *bottleneck layer*. After a period of training, expand this product; while the transformation remains unchanged, the number of adaptable parameters increases.

For convolutional neural networks, prominent in visual tasks, we propose these model-specific constraints that warrant further study:

- **filter upscaling** – Build the initial model with small filters and train on downscaled inputs; on higher layers, downsampling of internal representations is also suitable. After the opening phase, relax these restrictions by upscaling the filters to a final size and training on full-sized inputs.
- **pooling constraints** – Start the training with an architecture, in which large areas of detected convolutional features are downsampled into a single dominant cue. High-level features learned in such a state will depend only on coarse positioning of low-level patterns; gradually shrinking the pooling areas should result in well-structured hierarchical features.

Bibliography

Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In *Advances in Neural Information Processing Systems 19 (NIPS'06)*, pages 153–160, 2007.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th International Conference on Machine Learning (ICML-09)*, pages 41–48, 2009.

David M. Bradley. *Learning in Modular Systems*. PhD thesis, The Robotics Institute, Carnegie Mellon University, 2009.

Dan Claudiu Cireşan, Ueli Meier, Jonathan Masci, and Jürgen Schmidhuber. Multi-column deep neural network for traffic sign classification. *Neural Networks*, 32:333–338, 2012.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (ELUs). *CoRR*, abs/1511.07289, 2015. URL `http://arxiv.org/abs/1511.07289`.

Jeffrey L. Elman. Learning and development in neural networks: the importance of starting small. *Cognition*, 48:71–99, 1993.

Kunihiko Fukushima. Cognitron: A self-organizing multilayered neural network. *Biological Cybernetics*, 20(3):121–136, 1975.

Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36:193–202, 1980.

Kunihiko Fukushima. A neural network for visual pattern recognition. *IEEE Computer*, 21(3):65–75, 1988.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the International conference on artificial intelligence and statistics*, pages 249–256, 2010.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015. URL `http://arxiv.org/abs/1502.01852`.

Geoffrey Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14:1771–1800, 2002.

Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18:1527–1554, 2006.

Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.

Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen Netzen. Master's thesis, Institut fur Informatik, Technische Universitat, Munchen, 1991.

David H. Hubel and Torsten N. Wiesel. Receptive fields of single neurons in the cat's striate cortex. *The Journal of Physiology*, 148:574–591, 1959.

David H. Hubel and Torsten N. Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of Physiology*, 60:106–154, 1962.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 448–456, 2015.

Anders Krogh and John A. Hertz. A simple weight decay can improve generalization. In *Advances in Neural Information Processing Systems 4*, pages 950–957, 1991.

Tomáš Kuzma and Igor Farkaš. Automatická tvorba učebných plánov pre strojové učenie. In *Zborník 15. konferencie Kognícia a umelý život (KUŽ 2015)*, pages 99–102, 2015.

Yann LeCun. Generalization and network design strategies. In R. Pfeifer, Z. Schreter, F. Fogelman, and L. Steels, editors, *Connectionism in Perspective*, Zurich, Switzerland, 1989. Elsevier.

Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, 2013.

James Martens. Deep learning via hessian-free optimization. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 735–742, 2010.

Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

Marvin Minsky and Seymour Papert. *Perceptrons – an introduction to computational geometry*. MIT Press, 1969.

Douglas L. T. Rohde and David C. Plaut. Simple recurrent networks and natural language: How important is starting small? In *Proceedings of the 19th Annual Conference of the Cognitive Science Society*, pages 656–661. Erlbaum, 1997.

Douglas L. T. Rohde and David C. Plaut. Language acquisition in the absence of explicit negative evidence: How important is starting small? *Cognition*, 72: 67–109, 1999.

Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, November 1958.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning internal representations by error propagation. In David E. Rumelhart, James L. McClelland, and CORPORATE PDP Research Group, editors, *Parallel Distributed*

Processing: *Explorations in the Microstructure of Cognition, Volume 1: Foundations*, chapter Learning Internal Representations by Error Propagation, pages 318–362. MIT Press, Cambridge, Massachusetts, 1986.

David Silver, Aja Huang, Christopher J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–503, 2016.

Patrice Y. Simard, Dave Steinkraus, and John C. Platt. Best practices for convolutional neural networks applied to visual document analysis. In *Proceedings of the 7th international conference on Document Analysis and Recognition*, pages 958–963, Aug 2003.

Burrhus Frederic Skinner. Reinforcement today. *American Psychologist*, 13:94–99, 1958.

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

Ilya Sutskever, James Martens, George E. Dahl, and Geoffrey E. Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 1139–1147, 2013.

Yichuan Tang. Deep learning using support vector machines. *CoRR*, abs/1306.0239, 2013.

Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine Learning*, pages 1096–1103, 2008.

Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11:3371–3408, December 2010.

Alexander H. Waibel, Toshiyuki Hanazawa, Geoffrey E. Hinton, Kiyohiro Shikano, and Kevin J. Lang. Phoneme recognition using time-delay neural networks. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37 (3):328–339, 1989.

Izhar Wallach, Michael Dzamba, and Abraham Heifets. Atomnet: A deep convolutional neural network for bioactivity prediction in structure-based drug discovery. *CoRR*, abs/1510.02855, 2015.

Li Wan, Matthew D. Zeiler, Sixin Zhang, Yann LeCun, and Rob Fergus. Regularization of neural networks using DropConnect. In *Proceedings of the 30th International Conference on Machine Learning*, pages 1058–1066, 2013.

John (Juyang) Weng, Narendra Ahuja, and Thomas S. Huang. Learning recognition and segmentation of 3-D objects from 2-D images. In *Proceedings of the 4th International Conference on Computer Vision*, pages 121–128, 1993.

Paul J. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, Cambridge, Massachusetts, 1974.

List of Figures

Appendix A

Published Materials

In 2015, we presented the preliminary results of applying curriculum learning to several tasks, contrasting our method for task-agnostic estimation of instance hardness (see section 4.3) to results obtained with hand-crafted curricula, at the *Kognícia a umelý život 2015*, a jointly organized Czech-Slovak cognitive science conference. We include a self-contained unmodified copy of the original article – [Kuzma and Farkaš, 2015] (in Slovak) – in the form published in the proceedings of the conference.

Automatická tvorba učebných plánov pre strojové učenie

Tomáš Kuzma, Igor Farkaš

Katedra aplikovanej informatiky, FMFI, Univerzita Komenského v Bratislave
Mlynská dolina, 842 48 Bratislava
Email: {tomas.kuzma,igor.farkas}@fmph.uniba.sk

Abstrakt

Pri tvorbe učebných plánov sa často používa jednoduchá zásada – na začiatku učenia sa študentovi predkladajú jednoduché príklady a s postupom času sa zaraďujú aj príklady ťažšie. Cieľom tohoto postupu je pomôcť správnemu zovšeobecňovaniu v danej úlohe a urýchliť a zjednodušiť proces učenia. S myšlienkou použiť túto zásadu aj na trénovanie umelých neurónových sietí prišiel Elman (1993), pričom zvýšil úspešnosť učenia na syntetickej jazykovej úlohe pomocou ručne vytvoreného plánu. Na túto prácu nadväzujú aj Bengio a spol. (2009), demonštrujúc efektivitu tejto metódy (pod názvom *curriculum learning*) na ručne vytvorených plánoch pre viacero modelov a syntetických úloh. V tejto práci navrhneme niekoľko metód, ako takýto učebný plán (*curriculum*) tvoriť automaticky, a otestujeme ich efektívnosť na syntetických, ale aj praktických problémoch.

1 Úvod

V oblasti vzdelávania sa pri tvorbe učebných plánov často využíva nasledujúci princíp: ako prvé sa v preberanej látke objavujú príklady, ktoré sú jednoduchšie, menšie, či majú menej súvislostí, a v priebehu výučby sa táto náročnosť zväčšuje. Úlohou tohoto postupu je najprv naučiť študentov základy, a potom postupne pridávať rôzne rozšírenia a špeciálne prípady, ktoré by inak mohli byť na začiatku mätúce.

Hypotézu, že takéto postupné učenie (*curriculum learning*) by mohlo byť prospešné aj pri učení (trénovaní) modelov v strojom učení, predstavil Elman (1993). Vo svojom prvom experimente učil jednoduchú rekurentnú neurónovú sieť s kontextovou vrstvou predikovať nasledujúce slovo vo vete, pričom tieto vety boli generované jednoduchou umelou gramatikou. Sieť sa podarilo úspešne natrénovať len vtedy, keď sa najprv trénovala na kratších a jednoduchších vetách. (Vo svojom druhom experimente tiež dosiahol podobne pozitívne výsledky inou metódou, kde postupný nárast zložitosti učenia nebol dosiahnutý zmenami distribúcie vstupných dát, ale postupným nárastom reprezentačných možností trénovaného modelu. Sieť počas trénovania zabúdala, čo bolo implementované pomocou mazania skrytej kontextovej vrstvy po istom počte krokov.)

V nedávnej dobe sa k tejto myšlienke pod názvom *curriculum learning* (*postupné učenie*) vrátili v rovnomennom článku Bengio a spol. (2009), kde k nej tiež poskytli formálnu definíciu – postupnosť distribúcií nad trénovacími príkladmi s narastajúcou entropiou – a demonštrovali jej efektivitu na zmesi syntetických úloh: hľadanie Bayesovského klasifikátora, klasifikácia lineárne oddeliteľných dát a rozpoznávanie geometrických útvarov; a tiež na predikcii reálneho jazyka, pri ktorej sa hľadá skóre pre možné pokračovania začatej vety. Vo všetkých týchto scenároch dosiahli mierne, no signifikantné zlepšenie schopnosti generalizácie.

V tomto článku sa ďalej budeme venovať skúmaniu metód, ktoré budú takéto učebné plány vytvárať bez ďalších znalostí či už o úlohe, alebo hlbšej štruktúre jej vstupných dát.

2 Tvorba plánu

Základom nášho prístupu je myšlienka, že lineárne oddeliteľné rozhodovacie problémy sú exaktne reprezentovateľné a ľahko naučiteľné pre skoro všetky modely používané v strojovom učení. Pre problémy, ktoré ale lineárne oddeliteľné nie sú, by v duchu postupného učenia dobrým začiatočným bodom mohla byť čo najväčšia lineárne oddeliteľná podmnožina vstupov.

Naším počiatočným problémom je teda rozhodovacia úloha, ktorej dve triedy nie sú (v priestore vstupov) lineárne oddeliteľné, a chceme z nich vybrať podmnožinu, ktorá by už lineárne oddeliteľná bola. Je zjavné, že táto formulácia úlohy je veľmi nejednoznačná. Keby sme uvažovali podmienku, že vybraná množina má byť najväčšia spomedzi všetkých vyhovujúcich, narazíme na problém – úloha síce má jednoznačné riešenie, no už nie je efektívne riešiteľná.

Približná verzia nášho problému sa ale rieši implicitne pri trénovaní akéhokoľvek lineárneho klasifikátora – po natrénovaní klasifikátora sú totiž dáta, ktoré klasifikuje správne, lineárne oddeliteľné. Pre dobre natrénovaný klasifikátor bude navyše táto množina *približne* najväčšia možná (no nie vždy exaktne najväčšia). Z dôvodov efektivity sa teda budeme musieť uspokojiť s takýmto rýchlim približným riešením.

2.1 Stroje s podpornými vektormi

S veľmi efektívnym riešením problému (nielen) lineárnej klasifikácie prišli Cortes a Vapnik (1995) vo svojom modeli *stroj s podpornými vektormi* (*Support Vector Machine*), kde sa hľadá oddeľujúca nadrovina, od ktorej sú všetky trénovacie body čo najďalej, a to v správnom smere (taká môže zjavne existovať len pre lineárne oddeliteľné problémy); vo variante s *mäkkými okrajmi* (*soft margin*) sa navyše umožňuje túto vlastnosť porušovať, ale takéto porušenie je penalizované.

Ak si označíme počet vstupov n, dimenziu vstupov d, vstupy $\mathbf{x}_i \in \mathbb{R}^d$ a triedy $y_i = \pm 1$, našou úlohou je nájsť optimálne hodnoty pre koeficienty deliacej roviny ($\mathbf{w} \in \mathbb{R}^d,\ b \in \mathbb{R}$) a vo variante *s mäkkými okrajmi* navyše aj hodnoty pre vedľajšie[1] premenné ($\xi_i \in \mathbb{R}_0^+$). Za cieľ si volíme maximalizovať vzdialenosť medzi triedami a navyše vedľajšie premenné penalizujeme regularizačným parametrom $C \in \mathbb{R}^+$; dostávame teda tento kvadratický program:

$$\text{min:} \qquad \tfrac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i$$

$$\text{s podmienkou:} \qquad y_i(\mathbf{w}\cdot\mathbf{x}_i - b) \geq 1 - \xi_i \quad (\forall\, 1 \leq i \leq n)$$

Nakoľko takto sformulovaný kvadratický problém je pozitívne definitný, je možné ho efektívne riešiť (všeobecné kvadratické programovanie je NP-ťažké) niekoľkými spôsobmi, a to v primálnej i duálnej forme.

2.2 Rozhodovacie úlohy

Náš postup teda spočíva v natrénovaní stroja s podpornými vektormi na vstupných dátach a použitím tohoto modelu na vybratie *jednoduchších* trénovacích vzorov z celkovej množiny. Pre každý vstupný vzor si vypočítame jeho orientovanú vzdialenosť od rozhodovacej nadroviny (ďalej len *okraj*), pričom vzory ležiace ďalej od tejto roviny v správnom smere budeme považovať za jednoduchšie. Do sady jednoduchších dát pripustíme iba vzory, kde tento *okraj* je väčší ako nejaká hodnota – *prah*. Pre kladné hodnoty tohoto *prahu* dostaneme jednoduchšiu množinu, ktorá je už lineárne oddeliteľná; záporné hodnoty tohto parametra vyberajú aj už lineárne neoddeliteľné vstupy, no stále zahadzujú viac *vytŕčajúce dáta* (*outliers*).

2.3 Klasifikačné úlohy

Pre úlohy, ktoré majú $k > 2$ tried, už takýto jednoduchý postup nie je možný – viac tried sa nedá oddeliť použitím jednej roviny. Dá sa ale uvažovať nad použitím súboru viacerých lineárnych klasifikátorov, ktorého jednotlivé binárne rozhodnutia budú určovať výslednú klasifikáciu.

[1] Tieto nám umožňujú, aby niektoré body boli bližšie k rovine, ako majú byť (geometrická vzdialenosť $\geq \frac{1}{\|\mathbf{w}\|}$). Vo formulácii *SVM* bez mäkkých okrajov sa tieto premenné nenachádzajú (sú rovné nule).

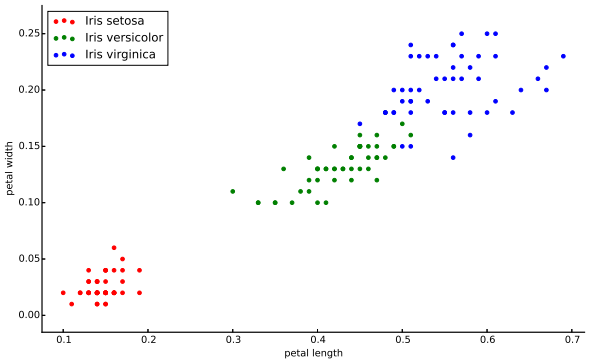
Ukázalo sa, že zmena vzájomnej početnosti jednotlivých tried má zásadný vplyv na úspešnosť trénovania; do našej sady jednoduchších dát preto budeme brať zástupcov všetkých tried rovnomerne.

Budeme skúmať dva takéto prístupy: *one–vs–rest*, ktorý používa k klasifikátorov, a *one–vs–one*, ktorý používa $\binom{k}{2}$ klasifikátorov.

2.3.1 One–vs–rest

V jednoduchšom variante sa pre k vstupných tried natrénuje k klasifikátorov, pričom i-ty oddeľuje triedu i od zvyšných. Výsledným *okrajom* pre vzor patriaci do i-tej triedy je v tomto prípade vzdialenosť daného bodu od i-tej rozhodovacej nadroviny. Výhodou tohoto postupu je jeho jednoduchosť a výpočtová nenáročnosť, no iba veľmi málo problémov sa dá úspešne riešiť takýmto súborom k klasifikátorov.

Obmedzenia tejto metódy si môžeme ilustrovať napríklad na známej dátovej sade *Iris flower data set* (Fisher, 1936; Anderson, 1935) (pozri obr. 1). Oddeliť červené body od zvyšných je možné exaktne, modré body od zvyšných s vysokou presnosťou. Zelené body sú ale obkolesené zvyšnými triedami z dvoch strán a ani približné oddelenie nie je možné. (Zobrazené sú iba posledné dva rozmery; oddelenie je ale podobne beznádejné aj s použitím všetkých štyroch rozmerov.)



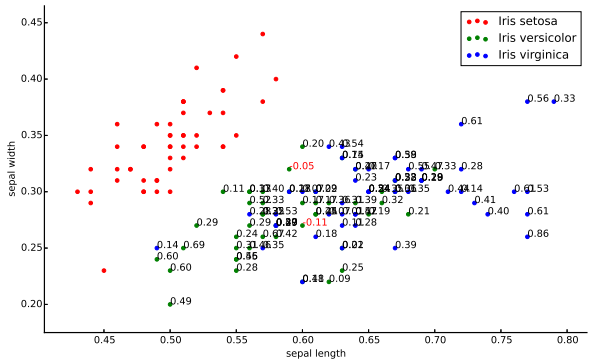
Obr. 1: Iris flower data set, zobrazené sú posledné dva rozmery: šírka a dĺžka okvetných lístkov.

2.3.2 One–vs–one

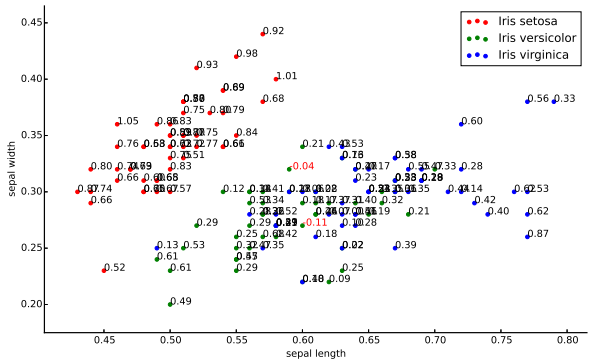
Zmysluplnejšie *okraje* sa dajú dosiahnuť trénovaním klasifikátorov, ktoré oddeľujú dvojicu tried i a j, takýchto klasifikátorov ale potrebujeme $\binom{k}{2} = O(k^2)$. Výhodou je, že každý klasifikátor počas svojho trénovania vidí len časť vstupných dát, a nie všetky, ako v prípade prístupu *one–vs–rest*, čo znižuje výpočtovú náročnosť trénovania veľkého množstva klasifikátorov.

Situáciu, kde mala metóda *one–vs–rest* ťažkosti, metóda *one–vs–one* rieši uspokojivo. Oddelenie dvojíc tried prebehne bez problémov, najťažšie je oddelenie *Iris versicolor* a *Iris virginica* (pozri obr. 2); výsledné skóre

ukazuje obr. 3. Je dobré si všimnúť, že konečné *okraje* sú celkom intuitívne: pre vzory na okrajoch tried sú vysoké, a na miestach, kde sa triedy prelínajú, sa pohybujú okolo nuly.



Obr. 2: Nedokonalé lineárne oddelenie kvetov *Iris versicolor* od kvetov *Iris virginica* – dva zo sto kvetov sa nachádzajú na nesprávnej strane oddeľovacej nadroviny. (Zobrazená je projekcia na prvé dva vstupné atribúty zo štyroch: šírka a dĺžka kališných listov; čísla nad bodmi znázorňujú vzdialenosť od oddeľujúcej nadroviny; záporné okraje sú kreslené červenou.)



Obr. 3: Konečné skóre pre všetky vstupné vzory. (Zobrazená je projekcia na prvé dva vstupné atribúty zo štyroch: šírka a dĺžka kališných listov; čísla nad bodmi znázorňujú vzdialenosť od oddeľujúcej nadroviny; záporné okraje sú kreslené červenou.)

3 Pokusy na syntetických úlohách

Úspešnosť nášho postupu sme skúmali na umelej úlohe klasifikácie geometrických tvarov, ktorú predstavili (Bengio a spol., 2009): cieľom je klasifikovať obrázky v odtieňoch šedej s veľkosťou 32×32 bodov do troch kategórií (elipsy, obdĺžniky a trojuholníky). Pre túto úlohu vytvorili aj učebný plán, čo nám umožňuje porovnať jeho efektivitu s naším učebným plánom.

V snahe priblížiť sa čo najviac pôvodným experimentom sme robili pokusy na umelej neurónovej sieti obsahujúcej tri skryté vrstvy (so *soft-sign* aktiváciou) po 400 neurónoch nasledované výstupnou *softmax* vrstvou (jednotlivé triedy boli reprezentované pomocou *one-hot* kódovania). Sieť sme trénovali *metódou poklesu gradientu* s využitím vysokého momenta po 256 epoch[2]. Výsledky pre 50 behov sú uvedené v tabuľke 1.

variant	estimácia	validácia	testovanie
bez plánu	13.43±0.98	29.52±0.94	29.76±1.00
one–vs–one	9.92±1.03	24.38±1.48	24.63±1.60
Bengiov plán	7.60±1.48	23.27±1.09	23.75±1.20

Tab. 1: Priemerné klasifikačné chyby (v %) dosiahnuté na pôvodnej verzii úlohy GeomShapes (odtiene šedej).

Ukázalo sa, že pre lineárny model je mimoriadne náročné mať v jednej triede aj svetlé útvary na tmavom pozadí, aj naopak. Vytvorili sme teda zjednodušenú monochromatickú verziu dát, kde je útvar vždy biely a pozadie vždy čierne. Táto úloha sa prekvapivo ukázala byť omnoho jednoduchšia aj pre nelineárnu neurónovú sieť; výsledky 50-tich behov sumarizuje tabuľka 2.

variant	estimácia	validácia	testovanie
bez plánu	0.03±0.06	8.84±0.41	9.57±0.48
one–vs–one	0.04±0.03	8.84±0.29	9.47±0.35
Bengiov plán	0.04±0.03	8.90±0.42	9.53±0.51

Tab. 2: Priemerné klasifikačné chyby (v %) dosiahnuté na monochromatickej verzii úlohy GeomShapes.

4 Pokusy na praktických úlohách

Ako praktickú úlohu sme si vybrali štandardnú sadu rukou písaných číslic MNIST (**?**). V tejto úlohe je potrebné klasifikovať číslice 0 až 9, ako vstup slúžia obrázky, ktoré majú majú 28×28 obrazových bodov v odtieňoch šedej. Na testovanie sme použili rovnaký postup a architektúru ako pri predchádzajúcej úlohe, nakoľko majú podobný charakter a typ vstupu. Jedinou zmenou je zmenšenie počtu vstupných neurónov z 32×32 na 28×28 a zníženie počtu epoch na 60. Výsledky pre 50 nezávislých behov uvádza tabuľka 3.

variant	estimácia	validácia	testovanie
bez plánu	0.03±0.03	1.63±0.10	1.66±0.08
one–vs–one	0.02±0.02	1.68±0.11	1.69±0.08

Tab. 3: Priemerné klasifikačné chyby (v %) dosiahnuté na úlohe MNIST; na validáciu sme vybrali náhodnú šestinu trénovacích dát.

[2] V prípade Bengiovho plánu sme trénovali sieť 128 epoch na ľahších a následne ďalších 128 epoch na ťažších dátach; pre náš plán, ktorý má v jednoduchej množine menej dát ako Bengio, sa ukázalo byť vhodné zvoliť dĺžku trénovania 16 + 240 epoch.

5 Implementačné detaily

Na efektívne trénovanie lineárnych SVM používame knižnicu `liblinear` (Fan a spol., 2008). Trénovanie na d-rozmernej dátovej množine obsahujúcej n vzorov má časovú zložitosť $O(d \cdot n)$, zatiaľ čo trénovanie všeobecnejších SVM má zložitosť $O(d \cdot n^2)$.

Pri použití metódy *one–vs–rest* sa trénuje k klasifikátorov – jeden pre každú triedu. Trénovanie jedného takéhoto klasifikátora má zložitosť $O(d \cdot n)$, pričom všetky trénovania sú nezávislé a dajú sa ľahko paralelizovať. Celková zložitosť je teda $O\left(k \cdot n \cdot d\right)$.

Pri použití metódy *one–vs–one* trénujeme dokonca $\binom{k}{2} = O(k^2)$ klasifikátorov, pričom každý oddeľuje nejakú triedu i od inej triedy j; tieto klasifikátory sa takisto dajú trénovať paralelne. Celková zložitosť je teda $O(k^2 \cdot d \cdot n)$, ale za predpokladu, že v každej triede je zhruba rovnaké množstvo dát, $O(n/k)$, čo platí napríklad pre všetky dátové množiny použité v tomto článku, je možné získať jemnejší odhad. Pre trénovanie klasifikátora rozhodujúceho sa medzi triedami i a j sú totiž potrebné len dáta patriace do týchto tried, ktorých nie je n, ale len $2 \cdot O(n/k)$ – získavame teda zložitosť:

$$O\left(\binom{k}{2} \cdot \tfrac{2}{k} n \cdot d\right) = O(k \cdot n \cdot d)$$

Na trénovanie dopredných neurónových sietí používame vlastnú implementáciu v jazyku Python, pričom na samotné výpočty používame knižnicu Theano (Bergstra a spol., 2010; Bastien a spol., 2012) – tá oproti naivnej implementácii prináša niekoľko výhod, vrátane optimalizácií na numerickú stabilitu a rýchlosť výpočtu a umožňuje *Just-in-Time* generovanie a kompiláciu zdrojového kódu pre výpočty. Výsledný strojový kód môže byť efektívne vykonávaný priamo na procesore (pričom využíva špecifickú inštrukčnú sadu konkrétneho procesora, dosahujúc zhruba 5-násobné zrýchlenie), alebo pomocou grafickej karty (ktorá dokáže mnohé operácie vykonávať paralelne na stovkách špecializovaných výpočtových jadier, s praktickým zrýchlením v ráde desiatok až stoviek).

6 Záver

Pokusy na syntetických dátach ukázali, že s použitím automaticky vytvoreného učebného plánu je možné dosiahnuť zlepšenie na skoro tak vysokej úrovni, ako s ručne vytvoreným plánom. Vytvorili sme tiež modifikovanú verziu tejto úlohy, ktorá je pre lineárny klasifikátor jednoduchšia. V tejto verzii získané okraje lepšie popisujú zložitosť jednotlivých vstupov a preto automaticky vytvorený plán dosahuje dokonca lepšie výsledky, ako plán vytvorený ručne.[3] Ďalšie skúmanie možno odhalí, prečo nám automaticky tvorený plán na praktických dátach nepriniesol žiadne zlepšenie.

Pôvodný článok (Elman, 1993) podrobili ostrej kritike Rohde a Plaut (1999), pričom ukázali, že na ich variantoch pôvodnej úlohy a pôvodného modelu postupné učenie nielen nepomáha, ale dokonca výsledky zhoršuje. Dá sa teda predpokladať, že techniky postupného učenia nie sú úplne univerzálne a nedajú sa úspešne použiť na ľubovoľný typ úloh.

[3] ktorý bol ale vytvorený pre nemodifikovanú úlohu

Poďakovanie

Tento príspevok vznikol za podpory grantovej agentúry VEGA v rámci grantovej úlohy 1/0898/14.

Literatúra

Anderson, E. (1935). The irises of the gaspe peninsula. *Bulletin of the American Iris Society*, 59:2–5.

Bastien, F., Lamblin, P., Pascanu, R., Bergstra, J., Goodfellow, I. J., Bergeron, A., Bouchard, N. a Bengio, Y. (2012). Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop.

Bengio, Y., Louradour, J., Collobert, R. a Weston, J. (2009). Curriculum learning. V *Proceedings of the 26th International Conference on Machine Learning (ICML-09)*, str. 41–48.

Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D. a Bengio, Y. (2010). Theano: a CPU and GPU math compiler in python. V *Proceedings of the 9th Python in Science Conference (SciPy 2010)*, str. 3–10.

Cortes, C. a Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20:273–297.

Elman, J. L. (1993). Learning and development in neural networks: the importance of starting small. *Cognition*, 48:71–99.

Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R. a Lin, C.-J. (2008). LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874.

Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7:179–188.

Rohde, D. L. T. a Plaut, D. C. (1999). Language acquisition in the absence of explicit negative evidence: how important is starting small? *Cognition*, 72:67–109.