Universiteit Leiden

Bayesian Optimization for Automatic Configuration of Convolutional Neural Networks for Text Classification

Author:
Mounir Hader

Supervisors:
Bas van Stein
Suzan Verberne

Master Thesis
MSc Computer Science and Advanced Data Analytics

Abstract

Convolutional neural networks (CNNs) can be a very effective and straightforward method for performing text classification tasks, mostly because feature engineering and feature extraction are completely taken care of by the network itself. However, global optimization of expensive black-box models, such as neural networks, is a very complex task that is highly unlikely to be solved by hand. The many different parameters that have to be configured lead to high dimensionality and heterogeneity of the search space and make the problem very difficult. The black-box nature of neural networks makes that the quality of a specific configuration can only be determined empirically (i.e., by training the network and evaluating the performance), which is a very time-consuming process. In this work, we adopt a framework for Bayesian Optimization (*MiP-EGO*) to automatically configure CNNs for binary text classification. To this end, we introduce a simple, configurable model (*TXT-CNN*) that achieves state-of-the-art performances after (automated) optimization with MiP-EGO. Furthermore, the original MiP-EGO algorithm is extended to allow asynchronous execution of parallel processes running on different GPUs to obtain maximal optimization efficiency.

Contents

1 Introduction

The availability of digital documents and other forms of (unstructured) textual data has been increasing rapidly with the growth of the World Wide Web over the last few decades. Much of this data is user-generated content that typically lacks a pre-defined data model, examples being web blogs, e-mails or customer reviews. One way of organizing this kind of data is to automatically assign them to one or more pre-defined classes (i.e. *text classification*). In this work we focus on sentiment analysis of movie reviews, an application of text classification where movie reviews are categorized as having either positive or negative sentiment [1, 2]. Text classification is an important and well-studied task on the border of information retrieval and machine learning, and many different approaches exist for solving this task [3]. Traditional techniques applied to this problem include Naive Bayes classification and Support Vector Machines [4, 5, 6]. In order to successfully apply such techniques to text, tedious feature engineering and selection are very important [7]. Recently, several papers have been published that involve convolutional neural networks (CNNs) for performing text classification without the need for manual feature engineering that achieved very promising results [8, 9].

Neural networks are widely used in (supervised) machine learning, for example for performing image classification tasks. With the rise of GPUs, deep learning software packages and well-established architectures like CNNs and RNNs, they are relatively easy to use and often provide more than satisfactory results. One of the most often praised properties of neural networks is their ability to automatically extract features from data, thus removing the need of manual feature engineering as in traditional techniques (e.g. Naive Bayes, Support Vector Machines). However, practitioners are still stuck with the task of designing the network architecture and selecting many specific parameters, which can be a very tedious job. The most straightforward approach is performing a grid-search: defining a grid of parameters with pre-defined value ranges, exhaustively changing one parameter at a time while keeping the others static in order to find the best performing model. The problem of such an approach is the high probability of getting stuck in a local optimum, and it has even been shown that a random search might perform better than a grid search for hyperparameter tuning of machine learning models [10]. More sophisticated methods are thus needed for optimizing neural network architecture and hyperparameter configuration.

In this work, we adopt a Bayesian Optimization approach for efficiently optimizing *CNN* architecture and hyperparameters for text classification. To that end, we use an adapted version of the popular *Efficient Global Optimization* (EGO) algorithm [11], which we will refer to as Mixed-integer Parallel *EGO* (MiP-EGO) [12]. The original *EGO*-algorithm has been developed for the optimization of expensive black-box functions. Examples of such functions are geographically determining location sites for oil drilling, measuring drug effectiveness in clinical trials and finding neural network performance where network training often takes long.

In this paper, we introduce a configurable model (TXT-CNN) and apply MiP-EGO to it for automatic configuration and optimization. Furthermore, the original MiP-EGO algorithm was extended to allow asynchronous execution of parallel processes running on different GPUs in order to obtain maximal efficiency during optimization (this is described in more detail in section 4). We test the algorithm under different setups on our dataset and compare the results of the found (sub)optimal solutions to networks that were tuned using grid-searches.

1.1 Problem statement

The problem of optimizing the architecture and hyperparameter configuration of a convolutional neural network can be stated concisely as

$$\max_{x \in S} f(x) \tag{1}$$

That is, we want to optimize the performance of our network (a non-linear function f that has as input a parameter configuration x) for all possible combinations of parameter configurations S. This is a complex task for several reasons. First, the black-box nature of neural networks in general makes it impossible to do a manual or intuition-based search: the performance of a specific configuration x can only be determined empirically. Second, training times for CNNs are relatively long even when using GPUs (especially when using much data), making (semi-)exhaustive approaches like grid search even more obsolete. Third and last, the search space for this task is very complex. The number of parameters to be set for configuring CNNs can be very big and heterogeneous, leading to high dimensionality and heterogeneity of the search space. Also, we often have to deal with conditional variables that only make sense in combination with other variables.

1.2 Report outline

The remainder of this report is organized as follows. First, we introduce neural networks in section 2.1. In particular, we discuss CNNs for text classification in subsection 2.1.2. Next, we cover the principles of Bayesian Optimization in section 2.2. We then look at related work in the field of automatic parameter configuration in section 2.3 and explain how MiP-EGO works in section 3. In section 4 we discuss how we extended the original MiP-EGO algorithm. Next, in section 5 we introduce our dataset, present our configurable CNN model for text classification (TXT-CNN) and discuss the experimental setup. We then present our results in section 6, followed by a discussion and conclusion in section 7.

2 Background and Related Research

2.1 Background: Neural Networks

Neural networks (NNs) are machine learning methods that are inspired by the functioning and architecture of biological nervous systems, such as the human brain [13]. They consist of many interconnected neurons, usually distributed over multiple layers, that work together to solve complex tasks [14]. The distributed arrangement of their neurons allows NNs to perform massively parallel computations when learning from the input data. Furthermore, by applying nonlinear functions to their neurons, NNs are able to learn and successfully classify non-linearly separable data without losing generalization ability of the model [15].

A brief history of NNs The birth of NNs dates back all the way to 1958, when Rosenblatt introduced the perceptron [16]. After the publication of the backpropagation algorithm for training multilayer perceptrons NNs gained much popularity [17], leading to the development of Convolutional and Recurrent Neural Networks (CNNs and RNNs). However, the AI community lost faith in NNs in the mid 90s and was mainly focused on the uprising Support Vector Machine (SVM). When Hinton *et al* showed that NNs could achieve better results than SVMs on the MNIST benchmark dataset in 2006 [18], excitement for NNs reemerged and ultimately led to the current machine learning revolution caused by deep learning [19]. The development of Graphics Processing Units (GPUs) for speeding up the training process of NNs has also been a very important factor for this reestablished interest.

Architecture of NNs Two examples of *feed-forward* NNs can be found in Figure 1: one network with 3 layers and another network consisting of 4 layers. This type of network is called feed-forward because the neurons in each layer pass on output signals to the neurons in the following layer.

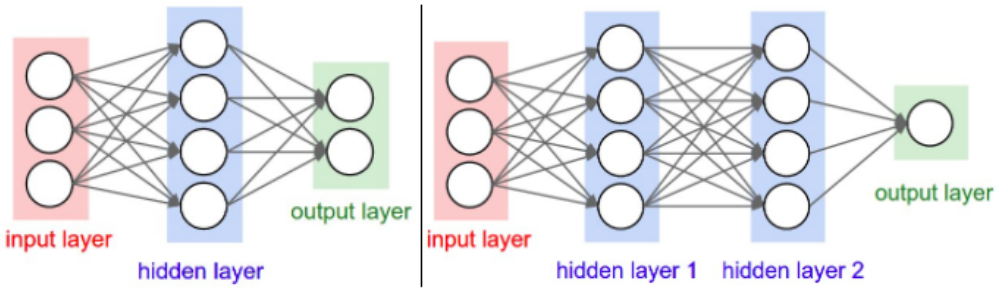

Figure 1: NNs with 3 (left) and 4 (right) layers (image retrieved from *http://cs231n.github.io/convolutional-networks/*).

We can see how the neurons of a neural network function in Figure 2. The left side of this figure shows a biological neural network, while the right side represents its artificial counterpart. An artificial neuron (or *node*) receives input signals $\mathbf{x}$ from neurons in the previous layer. In this example, the neuron receives three different inputs, so $\mathbf{x} = \{x_0, x_1, x_2\}$. Furthermore, the weights $\mathbf{w} = \{w_0, w_1, w_2\}$ and a bias b are associated with the neuron. Its output y is calculated by multiplying each input x_i with weight w_i and optionally adding a bias term b, before passing the result through an activation or transfer function f:

$$y = f\left(\sum_{i=0}^{2} w_i x_i + b\right) \tag{2}$$

Popular choices for f are e.g. the rectified linear unit ($f(x) = max(0, x)$) or the sigmoid function ($\sigma = \frac{1}{1+e^{-x}}$).

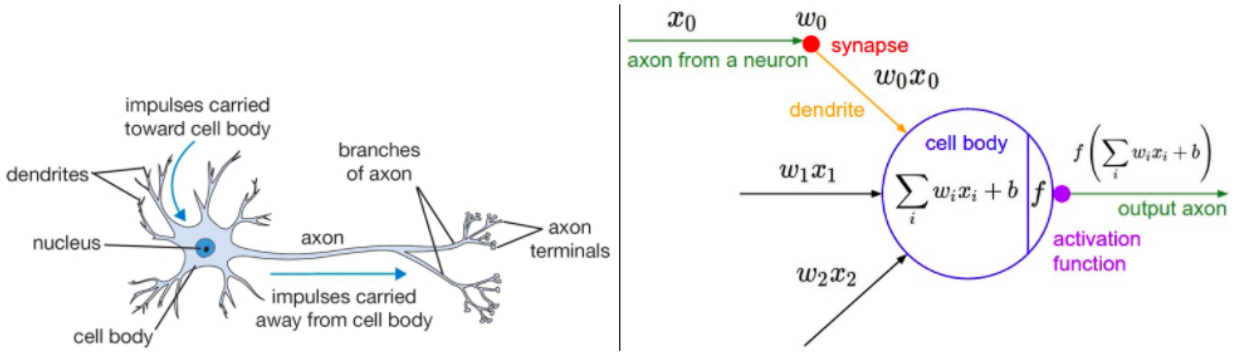

Figure 2: The output of an artificial neuron is computed by multiplying inputs **x** and weights **w** (and optionally the addition of a trainable bias term b) and passing the result to an activation function f (image retrieved from *http://cs231n.github.io/convolutional-networks/*).

2.1.1 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a type of feedforward networks that apply convolving filters to local features in the input data and are often used in the field of computer vision and object recognition [20]. As illustrated in Figure 3, the neurons in the convolutional layer of a CNN scan local regions of the input data and have shared weights across the full input. Furthermore, multiple filters of the same size can be used to scan the same input regions but look for complementary features.

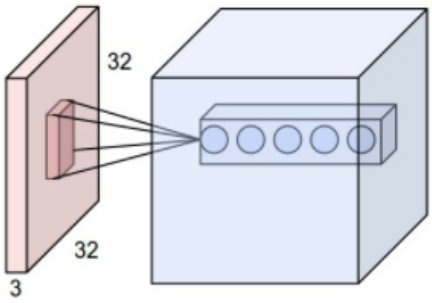

Figure 3: Schematic overview of how convolutional filters are applied to input data. The convolutions are projected onto the convolutional layer (blue volume): multiple neurons can have the same receptive field for learning complementary features (image retrieved from *http://cs231n.github.io/convolutional-networks/*).

2.1.2 CNNs for Text Classification

CNNs are a natural fit for image classification tasks because their aim is to learn features locally. As pixels in images have a higher probability to originate from the same object when they are located in the same local region, we can understand why CNNs are able to achieve excellent results in the field of computer vision and object recognition. It might be less straightforward to apply the same approach for text classification, but it has been shown that CNNs can achieve excellent results on tasks like sentiment analysis as well [8, 9]. We show which components are needed for applying CNNs to text and how they work in practice in the following paragraphs, followed by an introduction to word embeddings for representing words as vectors.

Sentence Matrix In order to apply convolutions to text in a similar way as they are applied to images, we need to represent the input similarly as well [21]. This is achieved by creating a sentence matrix: a matrix that represents a single sentence or document and where each row corresponds to a token or word. The row vectors have a fixed length, namely the length of the embedding vectors that are used to represent words (more on word vectors in Section 2.1.2).

Convolutions In images, convolutions are typically two-dimensional: the filters scan the input data in both the vertical and horizontal dimension, leading to a two-dimensional output. When we use a sentence matrix for representing text, however, it does not make sense to use a filter that scans the input horizontally: as one row in the matrix represents a single word, filters should be used that scan this word vector across its full dimensionality. The convolutional filters that are applied to the sentence matrix have different heights. As filters are of the same width as the input data, convolutions will lead to one-dimensional feature maps as output. It is common to use multiple filters of the same size for learning complementary features from the same local input.

Pooling As different convolutional filter sizes lead to feature maps of different lengths, pooling is applied to each feature map for extracting only the highest activation in that map. After concatenation of the resulting values the final set of high-level features is obtained.

Regularization In order for the network to generalize well on unseen data, regularization for preventing overfitting is important. One way for achieving this is by using dropout layers, where neurons are randomly set to zero ("dropped out") with dropout probability p. Another method for regularization is L2-norm constraint regularization: at each step in the training process, each node is re-scaled such that $||\mathbf{w}||_2 = L2max$ whenever $||\mathbf{w}||_2 > L2max$ [22].

Output The feature vector that was obtained after performing the convolutions is connected to a single output neuron via a hidden layer. The output is computed by applying a sigmoid function to the linear combination of the weights and incoming activations, such that the final output is in the range [0, 1]. The final binary classification is made by using a cut-off value at 0.5.

Word embeddings

In traditional document representation, the Bag-of-Words model is used for representing documents as vocabulary-sized feature vectors using term weights like *tf-idf*. Such huge and sparse vectors would obviously not be a good choice to use as input for a neural network, as we apply filters with the same dimensionality as the size of the word representation (leading to a huge number of weights). A better idea is to use word embeddings: these are vector representations of words with an arbitrary number of dimensions that can be learned in an unsupervised manner, for example using word2vec.

Word2vec Word2vec is a set of unsupervised machine learning techniques for creating word embeddings of high quality [23, 24]. This can be achieved by using either the Skip-gram model or the Continuous Bag-of-Words (CBOW) model. These models are neural networks with a single hidden layer that predict the probability of context words given a single input word, and the probability of a word given some input words, respectively. The context can take on an arbitrary size n. The true probabilities are easily computed by

simply counting the frequencies of words that are in the same context window of any other word among all documents in the training corpus. Figure 4 shows an example network of context size 1 (and can therefore be interpreted as both CBOW and skip-gram).

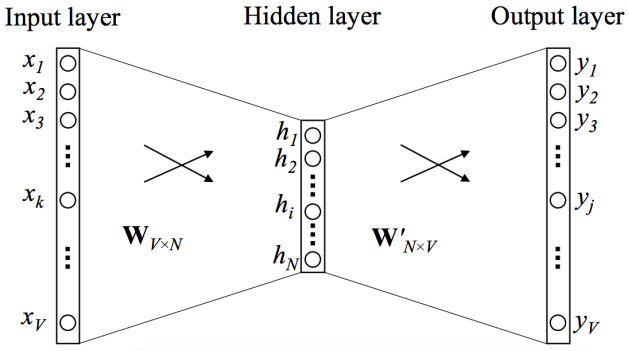

Figure 4: Network architecture in word2vec for a context of size one (figure from Rong [25]).

After training these networks on some text corpus, the matrices **W** and **W'** contain vectors (of arbitrary length N, i.e. the size of the hidden layer) for every word in vocabulary V: each word is thus represented twice (once as context word and once as middle word). Combining these vectors (e.g. averaging them), we end up with the final word embeddings.

The final word vectors generated with word2vec carry lots of semantic information and are therefor very suited for NLP tasks. Like any neural network, word2vec-networks are black boxes that don't explicitly clarify why they work well. One intuitive explanation is that in most texts, similar words have similar context. When looking, for example, at the sentence "*We pointed the door* ____", we expect the missing word to be {*red, green, blue,...*}. However, words like *yesterday* would obviously fit the sentence as well, without sharing the desired similarity of the other words.

The authors of the original papers additionally published a set of pre-trained word embeddings. These vectors were trained with word2vec on a Google News dataset of roughly 100 billion words, which resulted in very high quality word embeddings in terms of capturing semantic relationships between words, as shown in Figure 5.

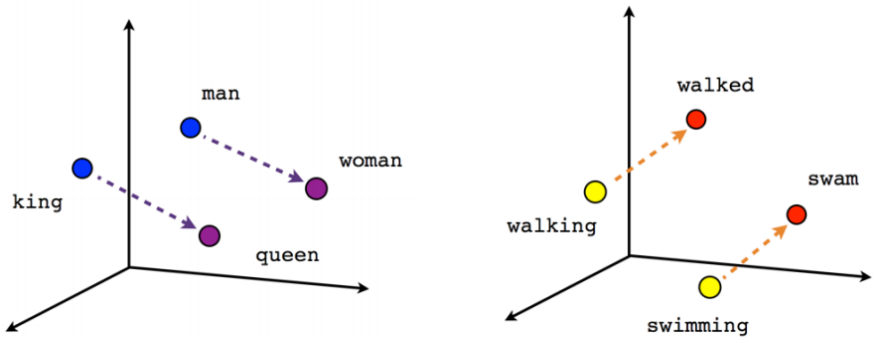

Figure 5: Because the word vectors for *man* and *woman* is are equally distant from resp. *king* and *queen*, very intuitive vector calculations can be made, e.g. **v**(king) - **v**(man) + **v**(woman) ≈ **v**(queen) (image retrieved from *https://www.tensorflow.org/images/linear-relationships.png*).

2.2 Background: Bayesian Optimization

Bayesian Optimization is a probabilistic method for global optimization of an objective function *f*. The key element of Bayesian Optimization is the assumption of a prior distribution of the objective function, from which a posterior distribution is obtained and iteratively updated using Bayes' Rule as new observations of the objective function are accumulated (i.e. *Bayesian inference*). As opposed to local search methods, e.g. gradient descent, Bayesian Optimization methods use all observations of f accumulated so far to determine the next evaluation point.

When considering machine learning models (neural networks in our work), the fitness function f is typically defined as the model performance in terms of e.g. accuracy or loss. Assuming a prior *P(f)* of this fitness function and letting $\mathbf{x}_i$ represent the i'th parameter configuration of the model and $f(\mathbf{x}_i)$ the true fitness function, we obtain the accumulated set of observations $\mathcal{D} = \{\mathbf{x}, f(\mathbf{x})\}$. The posterior distribution can then be found via Bayesian inference as in Eq. 3.

$$P(f|\mathcal{D}) \propto P(\mathcal{D})P(f) \tag{3}$$

The meta-model *P(f)* is known as a *surrogate model*, and for determining which point should be evaluated next (with the true fitness function), *acquisition functions* are used. These components are discussed in the next sections. An overview of a generic Bayesian Optimization algorithm can be found in Algorithm 1.

Algorithm 1 Bayesian Optimization

1: Initialize $\mathcal{D}_t = \{\mathbf{x}_{1:t}, f(\mathbf{x})_{1:t}\}$
2: Construct surrogate model with $\mathcal{D}_t$
3: **while** not stopping criterion **do**
4: Select $\mathbf{x}_{t+1}$ by maximizing an acquisition function $\mathcal{M}$:

$$\mathbf{x}_{t+1} = \arg\max_x \mathcal{M}(\mathbf{x}; \mathcal{D}_t)$$

5: Evaluate $\mathbf{x}_{t+1}$ on the true objective function to obtain $f(\mathbf{x})_{t+1}$
6: Augment data $\mathcal{D}_{t+1} = \{\mathcal{D}_t, (\mathbf{x}_{t+1}, f(\mathbf{x})_{t+1})\}$
7: Update surrogate model with $\mathcal{D}_{t+1}$
8: $t \leftarrow t + 1$
9: **end while**

2.2.1 Surrogate models

The estimate of f is thus constructed by using a finite number of samples of the true fitness function, and is known as a *surrogate* model. Intuitively, the surrogate model will have high uncertainty in regions that are not explored yet and vice versa.

Gaussian Processes (GPs) are a very common choice to use as surrogate models for Bayesian Optimization. A GP is a stochastic process that defines distributions over functions and is completely specified by a mean function $m(\mathbf{x})$ and a covariance function (or *kernel function*) $k(\mathbf{x}, \mathbf{x}')$ (see Figure 6).

$$f \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}'))) \tag{4}$$

Furthermore, the prior mean $m(\mathbf{x})$ is often assumed to be 0 for convenience. A popular choice for the covariance function $k(\mathbf{x}, \mathbf{x}')$ is

$$k(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{1}{2}||\mathbf{x} - \mathbf{x}'||^2\right) \tag{5}$$

This function gets closer to 1 as points lie closer together, and approaches 0 when points get further apart. Because all variables are weighted equally in Eq. (5), an additional parameter vector θ is commonly introduced, where each θ_i measures the importance and effectively regulates the activity of variable $\mathbf{x}_i$, as suggested by e.g. Jones[1] in the original EGO algorithm [11].

By the properties of a centered Gaussian Process, we know that any finite collection of its random variables are jointly Gaussian. Given a set of observations $\mathcal{D} = \{\mathbf{x}_{1:t}, f(\mathbf{x})_{1:t}\}$, we can thus model an arbitrary point $\mathbf{x}_{t+1}$ that we may want to evaluate next with the joint distribution

$$\begin{bmatrix} \mathbf{f}_{1:t} \\ f_{t+1} \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \mathbf{K} & \mathbf{k} \\ \mathbf{k}^T & k(\mathbf{x}_{t+1}, \mathbf{x}_{t+1}) \end{bmatrix}\right) \tag{6}$$

where $\mathbf{k}$ is a vector of covariance terms between $\mathbf{x}_{t+1}$ and $\mathbf{x}_{1:t}$. The posterior distribution can then be written as

$$P(f_{t+1}|\mathcal{D}_{1:t}, \mathbf{x}_{t+1}) = \mathcal{N}(\mu(\mathbf{x}_t + 1), \sigma^2(\mathbf{x}_{t+1})) \tag{7}$$

where

$$\mu(\mathbf{x}_t + 1) = k^T\mathbf{K}^{-1}\mathbf{f}_{1:t} \tag{8}$$

$$\sigma^2(\mathbf{x}_{t+1}) = k(\mathbf{x}_{t+1}, \mathbf{x}_{t+1}) - \mathbf{k}^T\mathbf{K}^{-1}\mathbf{k}. \tag{9}$$

2.2.2 Acquisition functions

Having defined a statistical model for representing the posterior distribution of the (unknown) function f as a combination of a mean function and an uncertainty term, we now focus on methods for selecting promising candidate points that we might want to evaluate next. In Bayesian Optimization, this is typically done by using an *acquisition function*, or *infill-criterion*. In order for the algorithm to converge to a global optimum, it is important that the acquisition function carefully balances exploration of new areas with high uncertainty and exploitation of regions with high estimated function values. Here we discuss *probability of improvement*, *expected improvement* and the *moment-generating function* for finding the global minimum of f.

Probability of Improvement The first acquisition function we discuss is the Probability of Improvement (PI). The function rewards solutions that may have an improved fitness value over the best observation found so far (i.e., the incumbent) $f(\mathbf{x}^+)$:

$$\text{PI}(x) = P(f(x) < f(\mathbf{x}^+)) \tag{10}$$

$$= \Phi\left(\frac{f(\mathbf{x}^+) - \mu(x)}{\sigma(x)}\right) \tag{11}$$

[1] Actually, Jones defines a *correlation* function for expressing variable distances, which is obviously very similar to using a covariance function

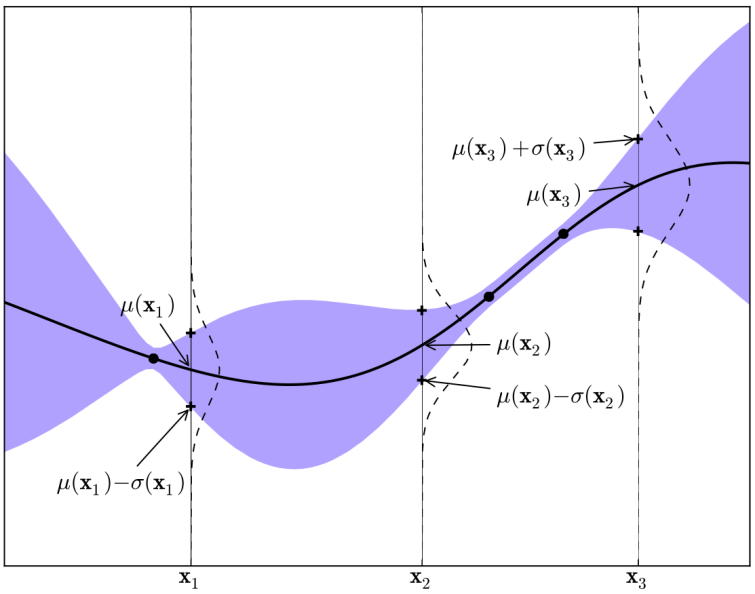

Figure 6: Toy example of a one-dimensional Gaussian Process prior on a fitness function f (solid black line). Four points have already been evaluated with the true fitness function and have zero variance (although not perfectly visible in the figure). Every other point $\mathbf{x}_i$ is modeled as a Gaussian distribution with mean $\mu(\mathbf{x}_i)$ and standard deviation $\sigma(\mathbf{x}_i)$ (figure from Brochu *et al* [26]).

Here, $\Phi(\cdot)$ is the normal cumulative distribution function. Because the PI-function chooses to draw points that are infinitesimally smaller than the incumbent over points that may offer greater gains but with high uncertainty, this acquisition function has a highly exploitative nature.

Expected Improvement The Expected Improvement (EI) extends PI by including the magnitude of the improvement a solution can potentially yield. The improvement is defined as

$$\mathrm{I}(x) = max\left\{0, f(\mathbf{x}^+) - f(x)\right\}. \tag{12}$$

The expected improvement is the expected value of (12) and can be expressed in closed form as

$$\mathrm{EI}(x) = (f(\mathbf{x}^+) - \mu(x))\Phi\left(\frac{f(\mathbf{x}^+) - \mu(x)}{\sigma(x)}\right) + \sigma(x)\phi\left(\frac{f(\mathbf{x}^+) - \mu(x)}{\sigma(x)}\right) \tag{13}$$

where ϕ is the normal probability density function and Φ the normal cumulative distribution function.

MGFI-acquisition The third and last acquisition function we discuss was proposed by Wang *et al* [27] and is based on the moment-generating function of the improvement (MGFI). The function takes all the higher moments into account and introduces a parameter (t) for explicitly balancing exploration and exploitation.

$$\mathrm{MGFI}(x;t) = \Phi\left(\frac{f(\mathbf{x}^-) - \mu'(x)}{\sigma(x)}\right)\exp\left((f(\mathbf{x}^-) - \mu(x) - 1)t + \frac{\sigma^2(x)}{2}t^2\right) \tag{14}$$

Lower values of t assign more weight to the lower moments, leading to a more exploitative search behaviour. Conversely, higher values of t lead to a more explorative search behaviour.

2.3 Related Research

The idea of automatic algorithm configuration is not new. As mentioned before, performing a grid search can be less efficient than a random search [10], underlining the need for more sophisticated methods for hyper-parameter optimization. Especially *model-free* algorithm configuration methods have been widely studied. Examples are evolutionary algorithms, *F-RACE* [28] (based on the *racing algorithm* [29]) and *ParamILS* [30] (based on iterated local search). A major disadvantage of these methods is that the search is guided locally, leading to a high probability of getting stuck in a local optimum.

Model-based algorithm configuration methods, on the other hand, use meta-models and iterate between fitting them and using them to select new points for evaluation (i.e. Bayesian Optimization). This approach is also referred to as Sequential Model-Based Optimization (SMBO) by Hutter *et al* [31]. The roots of these methods are all based on the work of Jones and the introduction of the *EGO*-algorithm [11]. Examples are ROAR (Random Online Adaptive Racing) and SMAC (Sequential Model-based Algorithm Configuration) by Hutter *et al* [31]. These methods have been successful in autoML tools like Auto-WEKA [32] and auto-sklearn [33].

More recently, an extension of *EGO* known as MiP-EGO (Mixed-integer Parallel-*EGO*) was introduced by Van Stein *et al* and applied to parameter optimization of CNNs for image classification tasks [12]. The method can handle mixed-integer data by using Random Forests as surrogate models, instead of Gaussian Processes as in traditional *EGO*. Furthermore, multiple candidate points can be selected and evaluated in parallel, which is extremely useful when optimizing CNNs and multiple GPUs are available for evaluating network configurations.

3 MiP-EGO

MiP-EGO follows the generic Baysian Optimization framework as described in Algorithm 1. The initial set of candidate points $\{\mathbf{x}_{1:t}\}$ is created using latin hypercube sampling (LHS). After evaluation of these points on the available GPUs and obtaining fitness values $\{f(\mathbf{x})_{1:t}\}$, the initial dataset is used to fit a Random Forest as a surrogate model for optimization (allowing mixed-integer data). An acquisition function is then maximized for selecting the next point for evaluation (section 3.1). Currently MiP-EGO has *PI*, *EI* and *MGFI* (section 2.2.2) implemented. After evaluation, the set of known configurations and corresponding fitness scores is augmented, the surrogate model is re-fitted and the process restarts.

MiP-EGO allows for multiple candidate points to be selected and evaluated in parallel, which is an extremely useful feature that was not supported in the traditional *EGO*-algorithm. Selecting multiple candidate points simultaneously can be done by constructing multiple instances of the *MGFI*-acquisition function using different values for t to ensure that the acquisition functions that are optimized are unique. This is implemented in MiP-EGO by using a log-normal distribution from which different values for t are sampled. The idea is that by using this long-tailed distribution, most values for t are small and thus well suited for exploitation, whereas few values will be large and guide the algorithm's explorative behaviour [12].

3.1 Optimization

Optimization of the acquisition function is done using evolutionary algorithms. In particular, *Mixed-Integer Evolution Strategies (MIES)* are applied to the problem [34]: a natural extension of CMA-ES [35] for mixed-integer optimization problems. An outline of this algorithm can be found in Algorithm 2. Since we use a (μ, λ)-selection scheme, the μ best individuals in each generational cycle are only selected from the offspring population λ, ignoring the parent population $\mathrm{P}(t)$.

Algorithm 2 (μ, λ)-Evolution strategy

$t \leftarrow 0$
initialize population $\mathrm{P}(t) \in \mathbb{I}^{\mu}$
evaluate the μ initial individuals with fitness function f
while not stopping criterion **do**
    for $i \in \{1, ..., \lambda\}$ **do**
        randomly select parents p_{i_1} and p_{i_2} from $\mathrm{P}(t)$
        $x_i \leftarrow mutate(recombine(p_{i_1}, p_{i_2}))$
        $Q(t) \leftarrow Q(t) \cup \{x_i\}$
    $P(t) \leftarrow \mu$ individuals with best fitness value from $Q(t)$
    $t \leftarrow t + 1$

4 Contribution

In this work, MiP-EGO is adopted and used for optimizing CNNs for text classification. Recall from section 2.1 that GPUs lead to a typical 100- to 1000-fold speedup of the training process and are therefore extremely useful for training and evaluating neural network configurations. In the original MiP-EGO software, multiple candidate points can be selected and evaluated in parallel if multiple GPUs are available (see section 3), which is an extremely useful extension to the original *EGO*-algorithm. The implementation of this feature in MiP-EGO was originally such that the evaluations of all candidate points in a single MiP-EGO iteration would run synchronously on the available GPUs. Thus, before the selection of a new set of candidate points for evaluation, all evaluations of the previous set would have to finish first (such that at the end of the optimization each GPU would have done the same number of function evaluations, see lower part of Figure 7).

Upon completion of an evaluation in the synchronous parallelization, a GPU thus has to wait on all other evaluations running in parallel to finish before getting assigned a new task. This is obviously very inefficient in terms of algorithm running time, even more so because training times of CNNs can vary substantially for different configurations (depending e.g. on the number and sizes of convolutional filters). One slow evaluation among all candidate points might therefore cause all other GPUs to be idle for hours, which is simply wasted time.

To overcome this problem, we extended the original software by implementing a more efficient parallelization of the GPUs: candidate points are evaluated asynchronously by using multithreading, allowing the GPU to evaluate a new candidate point as soon as the previous evaluation is finished. This way, all GPUs are constantly working instead of having to wait for all other GPUs to finish their task first (see upper part of Figure 7). To achieve this, we launch one thread per available GPU and use one *evaluation queue* that is shared across these threads. Now every time a GPU finishes an evaluation, its thread computes a new candidate point by re-fitting the surrogate model and maximizing the acquisition function on that model, adds this point to the evaluation queue and starts evaluating the first next point in the queue. Furthermore, the surrogate model is not shared across the different threads, as this may cause errors when different threads try to fit the surrogate model at the same time. We therefore make sure that each thread has its own unique copy of the surrogate model.

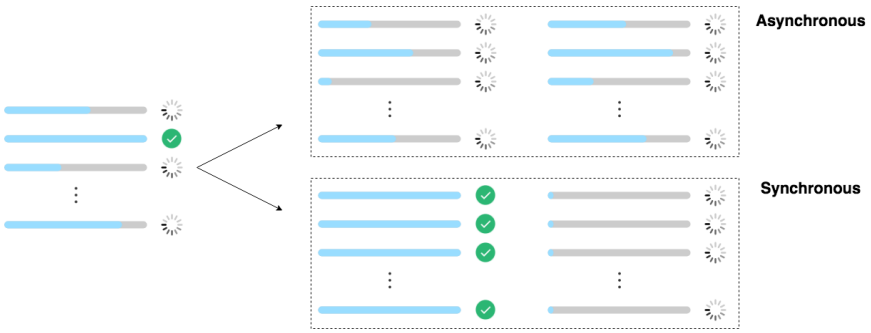

Figure 7: Synchronous (lower part) and asynchronous (upper part) execution of evaluations using multiple GPUs. In the asynchronous implementation, all GPUs are constantly working, while in the synchronous parallelization GPUs have to wait until all evaluations are finished before getting assigned a new job.

5 Dataset and experimental setup

5.1 Dataset

Experiments are conducted on the *MR* dataset: this dataset contains movie reviews from the *Rotten Tomatoes* website[2]. The reviews are approximately one sentence long and were automatically labeled positive or negative, based on the review being marked *fresh* or *rotten*, respectively.

The *MR* dataset consists of 10662 reviews, equally balanced among the positive and negative classes. The review lengths (after preprocessing) are in the range [1, 56], with an average review length of 20. The total vocabulary size is 18590, of which 16461 terms are available from the pre-trained word vectors (Google News data).

For the preprocessing steps, first all HTML tags like
were removed and all text was lowercased. Then the text was tokenized, treating commas, parentheses, exclamation marks and question marks as separate tokens and ignoring periods. Also, the expressions *'s*, *'ve*, *'re*, *n't*, *'d* and *'ll* are treated as separate tokens.

5.2 TXT-CNN model and configuration

Our main model for text classification is TXT-CNN, which is a configurable, shallow CNN with at most one hidden layer (Figure 8 represents the multichannel variant of this network, as described below). The network is based on the work of Kim [8] (the main difference being the possibility of an additional hidden layer in our model). Arguing that the vectors that are used for input representation (i.e. word2vec) already carry word information on multiple levels of abstraction (semantically and syntactically), we did not configure MiP-EGO to search for networks with multiple hidden layers. Furthermore, we adopt 3 variants for input representation:

- **Static**: input word vectors from word2vec's pre-trained Google News vectors that are not updated during training;
- **Non-static**: the input word vectors (same as above) are fine-tuned (updated) during training;
- **Multichannel**: a combination of static and non-static, where one set of inputs is kept static and the other is fine-tuned using SGD at training time.

The network receives as input a sentence matrix (two for the multichannel architecture), followed by a dropout layer d_0. Initially, the network was designed to apply convolutional filters of 3 sizes k_i to the input, with f_i different filters per filter size ($i \in \{1, 2, 3\}$). In later experiments, we treated the number of different filter sizes (s) as a tuneable hyperparameter of the model. The convolutions are calculated using an activation function a, and with or without an additional trainable bias term b. After *global-max* pooling of the resulting feature maps, the concatenated neurons are, via a dropout layer d_1, connected to a hidden layer of size h, using *relu* activation and a bias term. Finally, the output is calculated as a linear combination of the hidden neurons and a bias term using sigmoid activation, again via a dropout layer d_2. In the case that $h = 0$ (no hidden layer), the concatenated features are connected to the output via only a single dropout layer.

[2] https://www.cs.cornell.edu/people/pabo/movie-review-data/

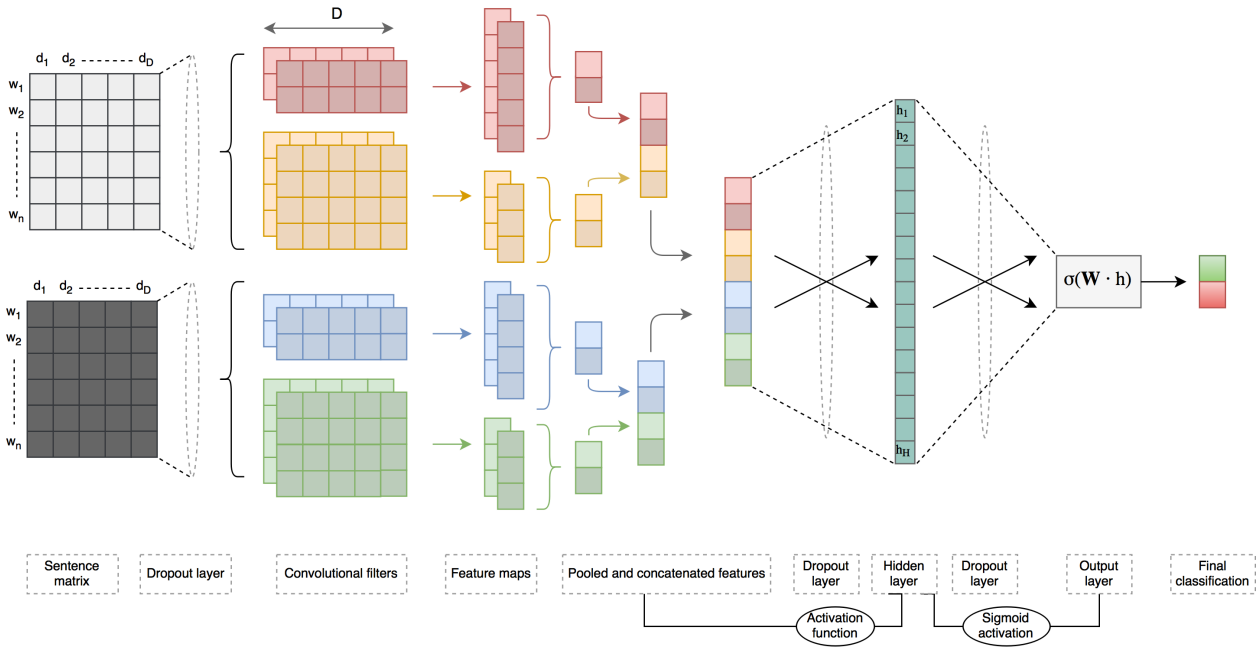

Figure 8: Multichannel architecture of our main model. The configurable hyperparameters we want MiP-EGO to learn are listed in Table 1, along with their corresponding ranges.

For all networks, the training objective is to minimize the binary cross-entropy loss function. Optimization is done using either *Adam* [36] or *Adadelta* [37], which both are widely used methods for SGD optimization. We perform early stopping whenever the loss on the test set does not improve in 15 consecutive epochs. Furthermore, we set the maximum number of epochs at 150 and we make sure to save the (up to that point) best performing model after every epoch, such that it can be loaded later for evaluation on the test set. We use 300 embedding dimensions for word2vec training of words that are not in the set of pre-trained word vectors (Google News dataset, section 2.1.2).

variable	range
a	[elu, relu, tanh, sigmoid, selu]
f	[1,100]
k	[1,15]
h	[0,100]
s	[1, 10]
d	[0,0.95]
b	[True, False]
balance	[True, False]
model	[static, nonstatic, multi]
optimizer	[Adam, Adadelta]

Table 1: Tuneable hyperparameters and ranges for MiP-EGO to optimize

Variable size inputs Because all input data must have the same dimensionality, all sentence matrices were padded with zeros up to the maximum review length in the dataset (56). Because convolutions on these padding vectors take dot products of zeros they are effectively ignored.

5.3 MiP-EGO configuration

Search space With the variables and corresponding ranges from Table 1 we can define the search space for MiP-EGO as the Cartesian product of the heterogeneous search spaces induced by the continuous (d), ordinal (f, k, h and a) and nominal (all remaining) variables.

Loss function Given a configuration θ, MIP-EGO calculates a loss value based on the classification accuracy of the network on the test set. The loss function is defined as

$$\text{loss}(\theta) = \log(1 - \text{test\_accuracy}) \tag{15}$$

Thus, the loss is at most 0 when the accuracy on the test set is 0.0 and decreases exponentially as the test accuracy increases.

Initialization Initialization of the algorithm is done by drawing 5 points using Latin Hypercube Sampling (LHS). We set the evaluation budget for evaluating parameter configurations at 1000.

Infill-criterion The infill-criterion (acquisition function) that we use in our experiments is either Expected Improvement (EI), Moment-Generation Function-based Infill (MGFI) or Probability of Improvement (PI).

MIES parameters Candidate points (configurations) are selected by maximizing the infill-criterion using MIES. We used $\mu = 4$ for the parent population and $\lambda = 10$ for the offspring population, and used an evaluation budget of 500 generational cycles.

Surrogate model We use Random Forest as our surrogate model for optimization. The number of trees is determined experimentally, as discussed in the next subsection.

5.4 Baseline Models and Experimental Setup

In order to compare the quality of MiP-EGO to alternative tuning methods (i.e. grid search), we first establish two baselines. The first is the network configuration that Kim found via grid search and tested on the same dataset [8]. We re-implemented this model and report the accuracy we found, i.e. 80.94 as opposed to the 81.5 as reported by Kim. This difference is most likely due to the fact that different folds were used for the cross-validation. The second baseline model is an extension of the first where a hidden layer was added and different regularization parameters were used that led to better results (this model was found after a new grid search that we performed on top of the first baseline). The performances of these networks were measured by 10-fold cross-validation of the test accuracy. That is, 10% of the data was randomly selected as test data and the remaining 90% as training data. The folds are chosen such that all data serve exactly once as test data and are used for training in all other folds. The exact baseline network configurations and their performance can be found in Table 2.

An important part of MiP-EGO is constructing the surrogate model (Random Forest) that is used for selecting candidate points to evaluate. We therefore first investigate the effect of the number of trees in our Random Forest on the search performance of our algorithm on a single fold of the data. Also, after search is completed, we use the complete dataset of sampled points for fitting the Random Forest and calculate the

Parameter	**Baseline 1**	**Baseline 2**
Number of hidden neurons (0 in absence of hidden layer)	0	10
Sizes of the conv. filters	3, 4, 5	3, 4, 5
Dropout probabilities for the dropout layers as shown in Figure 8 (0 when the dropout layer is absent)	0, 0.5, 0	0.5, 0.5, 0.8
The number of convolutional filters that are used per filter size	100	100
Batch size during SGD optimization	50	50
Boolean variable for adding an additional trainable bias term to the computed convolutions or not	*yes*	*yes*
L_2max constraint on weights when working with L2-norm constraint regularization	3	-
Boolean variable for weighting classes inversely proportional to their underlying class frequencies during training or not	*no*	*no*
Boolean variable for generating vectors that are not in the set of pre-trained w2v-vectors with word2vec on the training data (*no* means random vectors)	*no*	*yes*
Context window size for w2v training	-	10
Performance: test accuracy on 10-fold cross-validation	**80.94**	**81.52**

Table 2: Parameter configurations for our baseline models.

R^2-score for determining the quality of the fit.

In our experiments, when searching optimal network configurations with MiP-EGO, we only evaluate network performance on one fold of the data. After completion of an optimization run using a specific setup (number of trees in the Random Forest, acquisition function etc.), we use the 5 highest scoring configurations that were found along the full optimization run and perform the complete 10-fold cross-validation with these configurations (on the exact same train/test-splits as our baseline models to make comparison between different configurations on the same data possible).

6 Results

6.1 EI-based optimization

Figure 9 shows the test accuracy of the incumbent during optimization using Expected Improvement as acquisition function on a single fold of the data, for different sizes of the Random Forest. Additionally, in Table 3, the R^2-scores are given for Random Forests of different sizes after fitting them on their corresponding accumulated data set after optimization (i.e. 1000 samples), as well as the final incumbent accuracy.

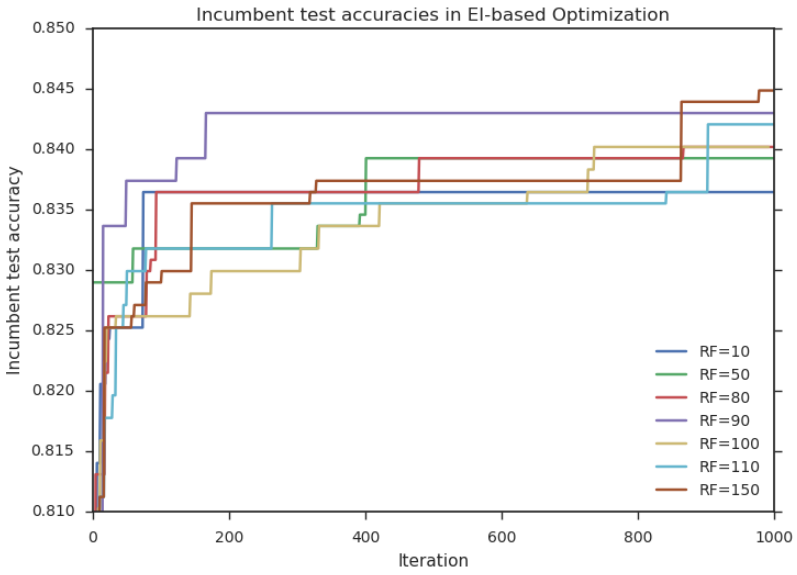

Figure 9: Test accuracies of the incumbent during EI-optimization of the surrogate model for different sizes of the *RF* (on a single data fold).

RF	Acc.(%)	R^2
10	83.64	0.862
50	84.02	0.879
80	84.02	0.879
90	84.30	0.874
100	84.02	0.882
110	84.21	**0.885**
150	**84.49**	0.879

Table 3: Test accuracies and R^2-scores of the *RF* for different sizes.

Highest test accuracy is achieved with 150 trees in the *RF*, while the best model in terms of R^2-score is the *RF* with 110 trees. Figure 10 shows the average, 10-fold cross-validated test accuracies after EI-optimization using different *RF* sizes. The figure shows the 5 highest scoring configurations that were found in a single optimization run. Table 4 summarizes the average 10-fold cross-validated accuracies. Highest cross-validation accuracies are achieved here by the surrogate model (*RF*) with 110 trees.

	Number of trees in the Random Forest						
	10	**50**	**80**	**90**	**100**	**110**	**150**
cv 1	75.09	81.36	81.45	63.73	**81.73**	**81.80**	75.10
cv 2	81.37	69.49	81.49	81.49	**81.58**	**81.74**	70.93
cv 3	81.42	81.46	75.10	75.64	73.75	**81.89**	72.15
cv 4	**81.67**	75.54	72.18	68.08	61.32	**81.66**	60.20
cv 5	81.11	69.34	75.92	78.37	72.24	81.43	77.86

Table 4: Average accuracies for 10-fold cross-validation of the 5 best configurations after EI-optimization for different sizes of the surrogate model (*RF* sizes), with accuracies that exceed our baseline in bold.

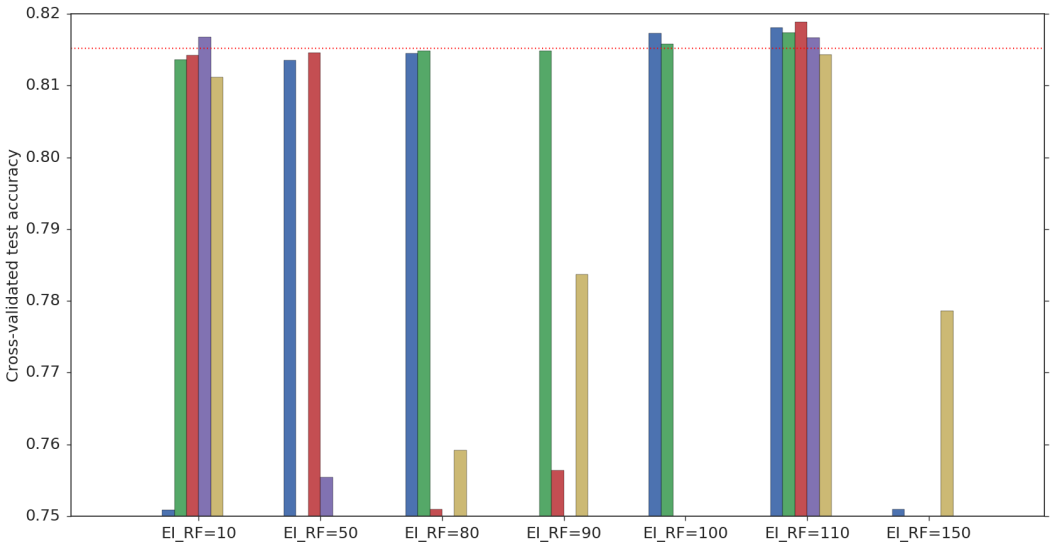

Figure 10: Cross-validated test accuracies of the 5 best configurations after EI-optimization of the surrogate model for different sizes (number of estimators/trees) of the *RF*. The red line corresponds to our main baseline (81.52).

6.2 MGFI-based optimization

Figure 11 summarizes the results for optimization using the *MGFI*-acquisition function on a single fold of the data. Figure 11a first shows how the test accuracy of the incumbent changes during optimization as different *RF* sizes are used, while keeping the t parameter at its default value (t=2). The corresponding R^2-scores are listed in Table 5. The results for different values of t (t0 in the figures) when using a forest with 110 trees are shown in Figure 11b. Finally, Figure 11c shows the effect of using a cooling scheme for the t parameter, cooling down from $t = 2$ to $t = 0.1$ and again using a forest of size 110. The final incumbent accuracies are listed under Table 6.

For the very small set of Random Forest sizes that were tested here, a forest with 110 trees achieves the highest R^2-score (Table 5). Furthermore, we see that $t = 2$ performs best here, and that the use of a cooling schedule does not lead to better results on this single data fold. However, the differences in performance of the optimal configurations are very small for the different setups that were tested.

From Figure 11b we can partly see the influence of the t-parameter on the convergence rate. With $t = 2$ the algorithm converges the fastest while $t = 5$ has the slowest convergence. The convergence rates for values of 3 and 4 for t are in between these two boundaries. Furthermore, from Figure 11c we can see that an exponential schedule leads to quicker convergence than a linear schedule.

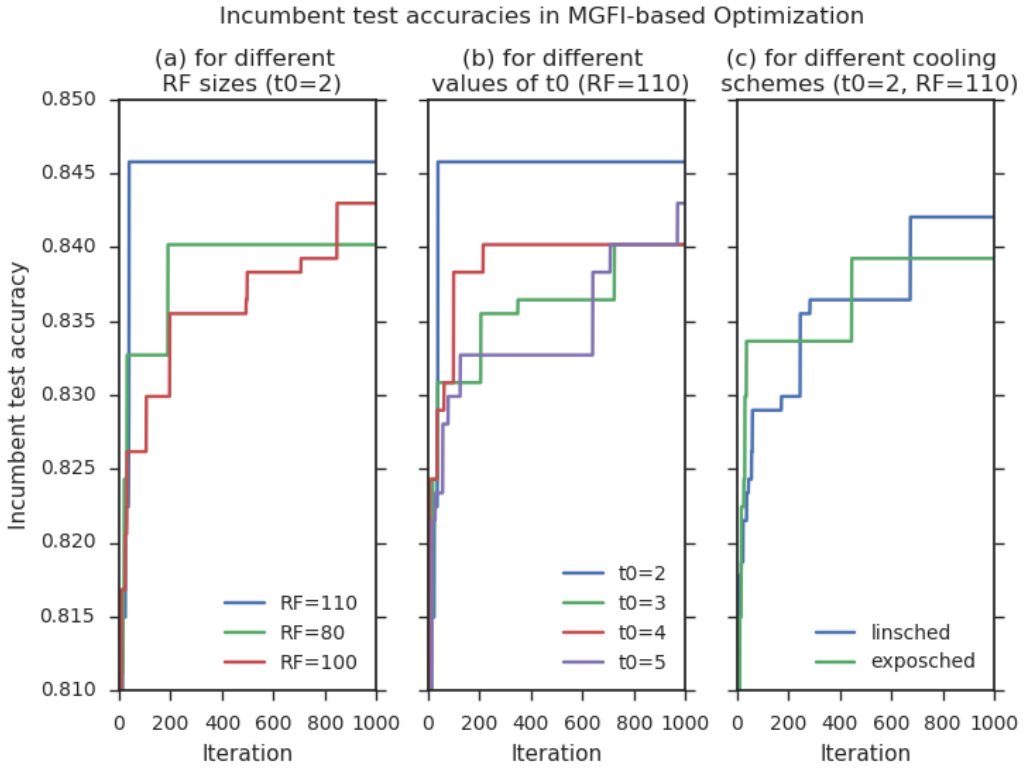

Figure 11: Test accuracies of the incumbent for MGFI-based optimization on one fold of the data.

RF	R^2
80	0.877
100	0.878
110	0.886

Table 5: R2 scores for surrogate models with different number of trees

RF	Acq.	t0	Schedule	Accuracy
80	MGFI	2	-	84.02
100	MGFI	2	-	84.30
110	MGFI	2	-	84.58
110	MGFI	3	-	84.02
110	MGFI	4	-	84.02
110	MGFI	5	-	84.30
110	MGFI	2	lin	84.21
110	MGFI	2	expo	83.93

Table 6: Final incumbent accuracies for one fold of the data with the optimization setups from Figure 11

The average 10-fold cross-validated results are also listed under Table 7, again for the 5 highest scoring configurations that were found in a single optimization run. For the cross-validation results of the incumbents from Table 6 (first line in Table 7), we see that the setup with an exponential cooling schedule (t0=2) performs best (81.83), closely followed by the setup with a fixed t of 2 (81.81).

	trees			t0			sched.	
	80	100	110	3	4	5	lin.	exp.
cv 1	78.64	78.51	**81.81**	**81.74**	78.52	81.42	81.36	**81.83**
cv 2	81.48	81.45	81.51	81.15	75.55	**81.54**	81.21	81.36
cv 3	75.24	**81.76**	**81.68**	78.29	81.43	**81.95**	81.11	71.97
cv 4	75.92	72.56	81.22	81.36	**81.87**	**81.58**	81.47	72.88
cv 5	81.51	81.41	**81.79**	**81.85**	**82.01**	81.41	**81.57**	81.35

Table 7: Average cross-validated accuracies (baseline-exceeding scores in bold)

6.3 PI and variable *s*

The remaining results are for PI-based optimization using a Random Forest with 110 trees, and optimization of a slightly modified version of the TXT-CNN model where the number of filter sizes s is a tuneable hyperparameter as well (as explained in section 5.2). The latter was optimized using the EI acquisition function and again using a Random Forest with 110 trees. Figure 12 shows the test accuracies of the incumbent during optimization of these two methods for a single fold of the data. The final incumbent accuracy is given in Table 8, as well as the average 10-fold cross-validated accuracies of the 5 highest scoring configurations in Figure 12.

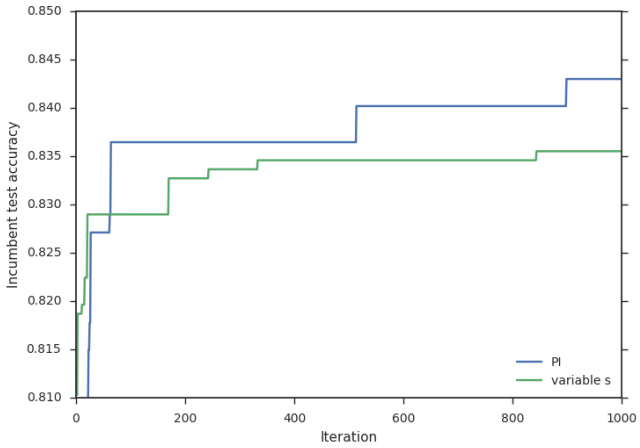

Figure 12: Test accuracies of the incumbent during PI-optimization (blue line) and EI-optimization with tuneable s on one fold (both with 110 trees)

	PI	var *s*
fold	84.30	83.55
cv 1	**81.66**	**81.54**
cv 2	**81.70**	78.34
cv 3	81.36	81.45
cv 4	78.35	81.50
cv 5	81.26	81.36

Table 8: Final incumbent test accuracies from Fig. 12 (first line) and cross-validated accuracies of the 5 best configurations

6.4 Best configurations

Table 9 summarizes which settings for MiP-EGO achieved the highest cross-validated test accuracies among all results that were previously discussed. The size of the Random Forest was 110 for all these setups. Furthermore, the outer right column specifies the rank of the configuration in the initial single-fold optimization (among the top 5 best scoring configurations) that led to this final cross-validated test accuracy. Table 10 shows the corresponding configurations that were found using the settings from Table 9.

#	**Acq.**	t_0	**sched.**	**Fold rank**
1	MGFI	5	-	3
2	EI	-	-	3
3	MGFI	4	-	4
4	MGFI	3	-	5
5	MGFI	2	exp	1

Table 9: MiP-EGO settings that led to the highest cross-validated test accuracies.

#	**model**	**balance**	**a**	**b**	$\mathbf{d}_0$	$\mathbf{d}_1$	$\mathbf{d}_2$	$\mathbf{k}_0$	$\mathbf{k}_1$	$\mathbf{k}_2$	**h**	$\mathbf{f}_0$	$\mathbf{f}_1$	$\mathbf{f}_2$	**opt.**	iter	cv
0	*multi*	*False*	*relu*	*True*	*0.500*	*0.500*	*0.800*	*3*	*4*	*5*	*10*	*100*	*100*	*100*	*Adadelta*	-	*81.52*
1	multi	True	relu	True	0.000	0.915	0.416	4	1	2	66	41	100	6	Adadelta	640	81.95
2	multi	True	elu	True	0.374	0.811	0.002	1	2	6	2	89	5	15	Adadelta	262	81.89
3	multi	True	relu	False	0.333	0.753	0.091	5	2	1	2	2	84	56	Adadelta	71	81.87
4	multi	False	relu	False	0.358	0.602	0.024	1	2	1	78	73	51	64	Adadelta	161	81.85
5	multi	False	relu	False	0.391	0.647	0.193	4	1	2	43	11	73	69	Adadelta	446	81.83

Table 10: The configurations that achieved the highest cross-validated test accuracies (outer right column), along with the number of iterations MiP-EGO needed to find them. Configuration corresponds to our baseline model.

7 Discussion and conclusions

Using CNNs for text classification is a very promising approach that is completely different from traditional techniques for performing this task. The combination of input representation using high quality word embeddings and a network that is able to automatically extract useful features from these is a fundamentally different approach than using e.g. SVMs for NLP tasks, which are typically based on the Bag-of-Words model. The major shortcoming of neural networks in general, however, is that their black-box nature makes it impossible to optimize the model by hand and without expert knowledge. Even for a relatively simple network like TXT-CNN, it is extremely difficult to determine good ranges for e.g. continuous variables (i.e. dropout layers), which increase the search space dramatically, as very small changes can already lead to very different performances.

We have seen that automatic configuration of our TXT-CNN model using MiP-EGO is able to achieve state-of-the-art performance on our movie review dataset without using much prior knowledge. The dropout probabilities, for example, were allowed to take on values ranging from 0 up to 0.95: a range that would be impractical for a grid search (in combination with all other variables). Not only is our algorithm able to find good solutions, it also takes away the tedious process of deciding which configurations to prioritize when performing a grid search: after initialization, no more user input or comparison is required and we can simply work with the highest scoring configurations that are found after optimization is completed.

Although we have only performed experiments on a single dataset where we applied a specific (configurable) model to a specific problem, MiP-EGO is also a very promising method for optimizing other problems like image classification [12]. This makes it a very useful technique for increasing autonomy of machine learning models in general and black-box models in particular.

Limitations and future work One limitation of our approach for optimizing CNNs is that we define a fixed search space before running the algorithm that is not able to grow dynamically. Because of this, interdependencies between conditional variables need to be taken care of manually. This is the case in our model, where we decided to use 3 different filter sizes, each with a different number of filters. We experimented with an additional masking variable s (ranging from 1 to 10) to control the number of active filter sizes in the network. In this approach, however, the initial search space is still fixed, but some dimensions are simply ignored by the network during optimization (and not by MiP-EGO). Furthermore, our approach for optimizing the model on a single fold of the data and looking at cross-validation results of the highest scoring configurations *afterwards* is very unlikely to find optimal configurations for the full cross-validation. The algorithm is prone to overfitting the network in this manner. This is clearly reflected by the fact that the best configurations that we found (Tables 9 and 10) were in 4 of the 5 cases *not* the highest scoring configuration in the corresponding single fold experiments. An experimental setup that performs a full cross-validation at each MiP-EGO iteration is thus a better approach for exceeding the baseline as much as possible. The downside of this is obviously that the evaluation time of a specific configuration will increase with a factor 10, but most likely fewer iterations will be needed for achieving similar (cross-validation) results.

All in all, we have seen that Bayesian Optimization (MiP-EGO) is a very effective way for optimizing CNN architecture for text classification and takes away much of the work and time that users would have to spend when designing these networks manually.

References

[1] Bing Liu and Lei Zhang. A survey of opinion mining and sentiment analysis. In *Mining text data*, pages 415–463. Springer, 2012.

[2] Vivek Kumar Singh, Rajesh Piryani, Ashraf Uddin, and Pranav Waila. Sentiment analysis of movie reviews: A new feature-based heuristic for aspect-level sentiment classification. In *Automation, computing, communication, control and compressed sensing (iMac4s), 2013 international multi-conference on*, pages 712–717. IEEE, 2013.

[3] Vandana Korde and C Namrata Mahender. Text classification and classifiers: A survey. *International Journal of Artificial Intelligence & Applications*, 3(2):85, 2012.

[4] Siwei Lai, Liheng Xu, Kang Liu, and Jun Zhao. Recurrent convolutional neural networks for text classification. In *AAAI*, volume 333, pages 2267–2273, 2015.

[5] Mehran Sahami, Susan Dumais, David Heckerman, and Eric Horvitz. A bayesian approach to filtering junk e-mail. In *Learning for Text Categorization: Papers from the 1998 workshop*, volume 62, pages 98–105, 1998.

[6] Thorsten Joachims. Transductive inference for text classification using support vector machines. In *ICML*, volume 99, pages 200–209, 1999.

[7] Sebastian Raschka. Naive bayes and text classification I - introduction and theory. *CoRR*, abs/1410.5329, 2014.

[8] Yoon Kim. Convolutional neural networks for sentence classification. *CoRR*, abs/1408.5882, 2014.

[9] Ye Zhang and Byron Wallace. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. *arXiv preprint arXiv:1510.03820*, 2015.

[10] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

[11] Donald R Jones, Matthias Schonlau, and William J Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998.

[12] B. van Stein, H. Wang, and T. Bäck. Automatic Configuration of Deep Neural Networks with EGO. *ArXiv e-prints*, October 2018.

[13] Jianmin Jiang, P Trundle, and Jinchang Ren. Medical image analysis with artificial neural networks. *Computerized Medical Imaging and Graphics*, 34(8):617–631, 2010.

[14] IA Basheer and M Hajmeer. Artificial neural networks: fundamentals, computing, design, and application. *Journal of microbiological methods*, 43(1):3–31, 2000.

[15] Anil K Jain, Jianchang Mao, and K Moidin Mohiuddin. Artificial neural networks: A tutorial. *Computer*, 29(3):31–44, 1996.

[16] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[17] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

[18] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[19] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[20] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[21] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM, 2008.

[22] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[23] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[24] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. *CoRR*, abs/1310.4546, 2013.

[25] Xin Rong. word2vec parameter learning explained. *arXiv preprint arXiv:1411.2738*, 2014.

[26] Eric Brochu, Vlad M Cora, and Nando De Freitas. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*, 2010.

[27] Hao Wang, Bas van Stein, Michael Emmerich, and Thomas Bäck. A new acquisition function for bayesian optimization based on the moment-generating function. In *SMC*, pages 507–512, 2017.

[28] Mauro Birattari, Zhi Yuan, Prasanna Balaprakash, and Thomas Stützle. F-race and iterated f-race: An overview. In *Experimental methods for the analysis of optimization algorithms*, pages 311–336. Springer, 2010.

[29] Oden Maron and Andrew W Moore. The racing algorithm: Model selection for lazy learners. *Artificial Intelligence Review*, 11(1-5):193–225, 1997.

[30] Frank Hutter, Holger H Hoos, Kevin Leyton-Brown, and Thomas Stützle. Paramils: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36:267–306, 2009.

[31] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International Conference on Learning and Intelligent Optimization*, pages 507–523. Springer, 2011.

[32] Lars Kotthoff, Chris Thornton, Holger H Hoos, Frank Hutter, and Kevin Leyton-Brown. Auto-weka 2.0: Automatic model selection and hyperparameter optimization in weka. *The Journal of Machine Learning Research*, 18(1):826–830, 2017.

[33] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In *Advances in Neural Information Processing Systems*, pages 2962–2970, 2015.

[34] Rui Li, Michael TM Emmerich, Jeroen Eggermont, Thomas Bäck, Martin Schütz, Jouke Dijkstra, and Johan HC Reiber. Mixed integer evolution strategies for parameter optimization. *Evolutionary computation*, 21(1):29–64, 2013.

[35] Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, 2001.

[36] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[37] Matthew D. Zeiler. ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701, 2012.