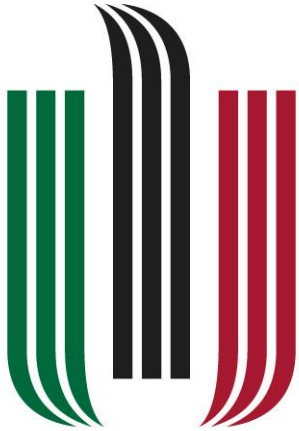

AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA
W KRAKOWIE

WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI, INFORMATYKI I INŻYNIERII BIOMEDYCZNEJ

KATEDRA AUTOMATYKI I ROBOTYKI

Praca dyplomowa inżynierska

Dobór parametrów dla uczenia głębokiego w zadaniach klasyfikacji
Tuning deep learning parameters for classification problems

Autor:	*Maciej Błaszczyk*
Kierunek studiów:	Automatyka i Robotyka
Opiekun pracy:	*dr inż. Patryk Orzechowski*

Kraków, 2019

Uprzedzony o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.): „ Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystyczne wykonanie albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, wideogram lub nadanie.", a także uprzedzony o odpowiedzialności dyscyplinarnej na podstawie art. 211 ust. 1 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym (t.j. Dz. U. z 2012 r. poz. 572, z późn. zm.) „Za naruszenie przepisów obowiązujących w uczelni oraz za czyny uchybiające godności studenta student ponosi odpowiedzialność dyscyplinarną przed komisją dyscyplinarną albo przed sądem koleżeńskim samorządu studenckiego, zwanym dalej „sądem koleżeńskim", oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i że nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

Składam serdeczne podziękowania dla dr Patryka Orzechowskiego za bezgraniczną cierpliwość i pomoc podczas przygotowywania tej pracy dyplomowej. Dziękuję również swoim rodzicom za wsparcie duchowe i motywacyjne.

Contents

1.Introduction

The concept of the neuron and neural network has been known for many years. However, only in the last 10-20 years deep learning began to develop very dynamically. Before it was not possible due to the following factors:

- the lack of adequate computing power
- the lack of sufficiently effective learning algorithms
- the unavailability of large datasets that are compulsory for deep networks

The first problem started to resolve in the early 20th century thanks to the development of the GPU. At that time, the main tool used for calculations were processors, but their architecture was not suitable for performing very many simple calculations at the same time (standard processors usually have several cores, so it was impossible to perform many tasks at the same time, which is crucial for effective learning in deep learning). In the years 2004-2005, NVIDIA carried out the first tests of using GPUs for calculations. They ended successfully, resulting in the development of CUDA architecture. It was a breakthrough achievement which raised popularity for GPUs and allowed them to become universal tools for quick computations.

The second problem was related to the lack of effective learning algorithm for deep networks, which had more than one hidden layer. It was clear that based on the difference in expected and calculated output, you can modify the weight of neuron that calculated this output. But taking into account that there are more layers, it was problematic what to do with those neurons in the earlier layers that were only indirectly involved in the calculation of the result. The back propagation algorithm turned out to be a solution to this problem. It was already developed in 1976, but because of the prevailing pessimistic moods regarding the future of deep learning ("AI winter"), nobody believed that it could work. Therefore, it was not until 1986 that it was disseminated and believed that multilayer networks can be finally taught effectively.

The third problem was the availability of data. Deep network, depending on its size, needed thousands, tens of thousands or even millions of different examples to generalize well and successfully learn. Currently on the Internet a lot of different portals could be found with properly structured datasets. However, 10-20 years ago those resources weren't easily available, as creating, describing and formatting a dataset of good quality required an effort.

In the last dozen years, a lot of different types of neural networks have been created, each of which is usually designed to solve a specific problem, which does not mean that it will not work in the solution of another one. Deep learning was applied to many image classifications or speech recognitions but they are not the only problems that could be solved. In this work, the basic task of classification has been addressed. For the tests, very popular and widely used architectures were chosen which are known for years (such as multi layer perceptron or

convolutional neural network) as well as those that have recently appeared in the world (capsule network, long-short term memory network) and are not yet well-tested.

Generally, every neural network consists of many layers, and each of them from many different computing units. Very often it is a neuron that performs simple calculations such as multiplication and addition. In other cases, it is a convolution filter that moves along the data and also based on multiplication and addition operations calculates specific values. Another form of a computing unit may be something more complicated, such as memory cells in which several small neural networks are contained, so that they are capable of remembering the data.

Deep learning models are usually configurable. That means they accept different hyperparameters which affect later received results. The problem of hyperparameters tuning exists since the first machine learning models have been created. It is about evaluating the algorithm and changing the hyperparameters basing on the obtained result. Every scientist is struggling with this task, therefore so many tuning techniques were created, such as grid search, random search or bayesian optimization. Nevertheless, still a lot of work is necessary to find the optimal hyperparameters.

2. Aims and scope

The aim of this work is to test various types of deep neural networks with different hyperparameters in the task of classification. Each type of neural network has its own set of hyperparameters that can be modified to change the course of learning and, consequently, get other results. There are no general rules that would say how to configure the model for specific data, so that a satisfying result could be accomplished. The most common method is the empirical search for the best settings of hyperparameters using a specific optimization technique and one of the cross validation variants to decrease the influence of luck.

Datasets on which experiments are carried out come from the PMLB benchmark. It is a collection which contains a wide range of different datasets, starting from small ones (50 - 1000 samples) to medium-sized (1000 - 10000 samples) and also to large ones (10000 samples +). On each of them a lot of classical machine learning algorithms have already been tested, which will be a reference point for the results obtained in this work.

The main research problem is whether for each of the selected models of deep neural networks are there hyperparameters that allow for achieving much better and more stable accuracy score?

Additional research problems:

2. Are deep neural networks competitive and sometimes able to outperform with classic machine-learning algorithms by a few per cent?
3. Are a lot of neurons needed in single hidden layer multi-layer perceptron to generalize well?
4. Are more than one hidden layer needed in multi-layer perceptron to generalize well?

The following hypotheses were formulated corresponding to the above questions:

1. For each of the selected models of deep neural networks there are hyperparameters that allow for achieving much better and more stable accuracy score.
2. Deep neural networks are competitive and sometimes able to outperform classic machine learning algorithms by a few per cent.
3. A lot of neurons is needed in single hidden layer multi-layer perceptron to generalize well.
4. More than one layer is needed in multi-layer perceptron to generalize well.

This dissertation is organized as follows:

Chapter 1 – Introduction – is a brief introduction about history of deep learning, problems that prevented it from developing earlier, types of neural networks chosen for this work, their most important element, most common applications and about hyperparameters tuning.

Chapter 2 – Aims and scope – covers objectives and scope of this dissertation. There is a superficial description of hyperparameters tuning and techniques used for it as well as quick presentation of collection of datasets used in this work. Also the research problems and hypotheses are stated here.

Chapter 3 – Background – contains more elaborations about deep learning, deep neural networks, their hyperparameters. There are two subsections – first called deep learning models where all four selected algorithms(multi-layer perceptron, convolutional neural network, capsule network, long-short term memory network) are written in detail and second named tools and frameworks where the main libraries(scikit-learn, tensorflow, keras) are described.

Chapter 4 – Datasets – there is presented a structure of PMLB – collection of datasets utilized in this work. It also contains a table in which datasets chosen for this work are described.

Chapter 5 – Methodology – covers more information on the techniques used for hyperparameter tuning along with a detailed description of those used in this work (grid search, k-fold cross validation, and balanced accuracy). What is more, there is a step-by-step description of each experiment, i.e. which models were used, how they were configured, how the results were achieved and how many resources and time these experiments took.

Chapter 6 – Results – is divided into 4 subsections - separately for every hypothesis. Each contains information about stated problem, description how the results were presented, received graphs and their discussion.

Chapter 7 – Conclusions and future work – contains recall of all four hypotheses and conclusions from the completed work for each of them. There are also listed possibilities of research development over the issue.

3. Background

Deep learning is a subclass of machine learning field which is defined as following [1]. It uses a model which consists of a few layers of some processing units that perform nonlinear calculations for feature extraction and transformation. Learning is supervised or unsupervised. It learns different representation for each level of abstraction which form a hierarchy of concepts.

Deep neural networks have a lot of hyperparameters – they are like devices that require configuration [27]. Each model has a loss function which can be optimized by tuning those hyperparameters. Some of them are selected immediately, because they do not directly affect the result or it is generally known that for a particular task there is one best specific parameter. The rest must be chosen in a proper way so as to obtain a satisfactory result. It is called hyperparameters optimization or tuning and is regarded as a different field of science.

3.1 Deep learning models

There are a lot of different deep learning networks. In this section the four chosen to be used in this work are described. For each one of them the detailed architecture, applications, principle of operation and auxiliary figure are presented.

3.1.1 Multilayer perceptron

Multilayer perceptron (MLP) is the least complicated type of neural network. It consists of perceptrons, which are the most basic and the oldest type of linear classifiers without any significant applications nowadays. They become more powerful device after stacking in layers and adding non-linear functions.

As it is show in the Figure 1, MLP consists of the input layer, hidden layers and the output layer [7]. Each layer contains a defined number of neurons and has a specific task. The input layer has a dimension corresponding to the input data and enters them into the network. The output layer has as many neurons as there are classes in the dataset and is responsible for calculating the final prediction of the network. The middle layers can be any number and can have any number of neurons.

Neuron is something like a computing unit. On his input comes data from all neurons from the previous layer, and he multiplies them by appropriate weights and fire the result. To increase neuron's power to separate hard groups of data here comes the nonlinear part. Calculated output is multiplied by activation function. There are a lot of different activation functions used for different types of neural networks, for example: Softmax, Sigmoid, TanH or Rectified Linear Unit.

In addition, usually bias unit in each layer is included. Standard units are capable of scaling activation function, but are not able to translate it to the right or left. Here comes the bias, e.g. neuron with constant value present in each layer which allows to shift the output of whole layer.

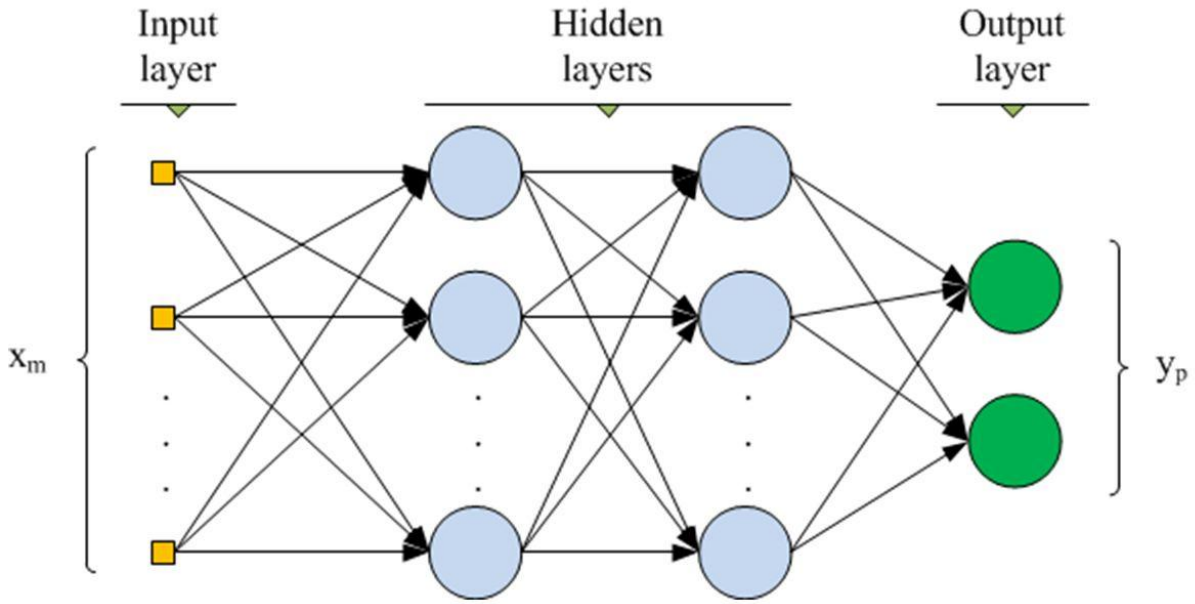

Figure 1: Architecture of multilayer perceptron. Source: [8]

After all the final output calculated from the neurons with activation functions and biases is passed to the next layer. This is called feed forward propagation, because we transfer information from the input to the output, where the final results are calculated. Then these results are compared with labels and the classification error is calculated based on the cost function. Common cost functions are for example mean squared error, mean absolute error, cross entropy loss.

Having a metric thanks to which the classifier's quality is rated, we face the usual task of function minimization. There are many algorithms designed for this purpose, starting with the simplest Stochastic Gradient Descent and ending with its different variations such as Adagrad, Adam or RMSProp. Cost function is minimized by adjusting neuron weights. The back propagation algorithm is used for this. For each neuron, the derivative of the function according to which it converted the received input is calculated. Then the weights are updated using following formula:

$$new\_weight = old\_weight - derivative\_rate * learning\_rate$$

All the above-mentioned activities are one learning iteration. To achieve satisfying result a lot of iterations is needed until algorithm convergence.

3.1.2 Convolutional neural network

Convolutional neural networks (CNN) gained a lot of popularity in 2012 when the ImageNet competition was won for the first time with their help. Their creation was inspired by research on the perception of the image by the human eye. It turned out that the seen picture is divided into fragments that are recorded by individual cortical neurons in visual cortex. Then they are combined into a whole so that it creates an entire image that our brain analyzes. Based on this knowledge, convolutional networks have been created. As it is shown in the Figure 2, they consists of a few layers [9] [24]:

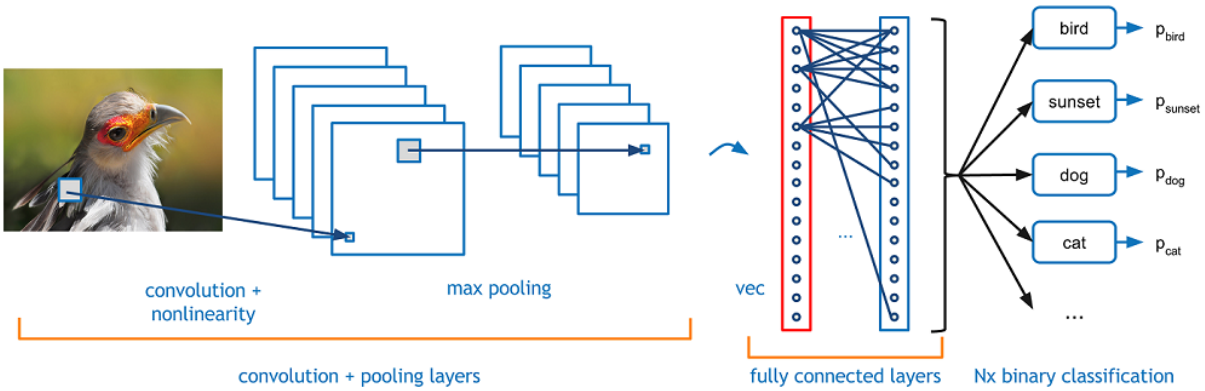

Figure 2: Architecture of convolutional neural network. Source: [10]

- **convolution layers**
  The most important layer in which convolution filters are applied. This is an operation that involves sliding the N x N filter all over the image and converting their dot product. As a result, we get the very basic extracted feature of the picture, for example some lines, or curves. Each filter is responsible for extracting different image's feature. Filter values are calculated automatically during network learning. Usually the network hyperparameters that we are tuning are the number of filters in each layer and their size.
- **pooling layer**
  The most popular examples of those are max and average pooling layers. Average pooling is about taking mean and max pooling is about taking max value from the selected area. The main purpose is to reduce spatial dimensions in the input and introduce some basic invariance in the image. On the other hand, pooling is just a disposal of information.
- **non-linearity layer**
  Almost always after convolution and pooling the Rectifier Linear Units(ReLU) function is applied. Generally it is just $f(x) = max(0, x)$, so it is very cheap in computing because of no exponentials, multiplication or division operations.

- **regularization layer**
  There are 2 mainly used regularization techniques called dropout and batch normalization. The first is simply about ignoring individual neurons during each of learning cycle. Usually the number of neurons in the entire layer to be ignored is determined in percents. The second method - batch normalization [19], is about providing any layer in the network with inputs that are zero mean / unit variance. Mainly it is an optimization method, however, it also provides quite good regularization.
- **fully connected layer**
  Here the output is flattened. These are the layers containing normal artificial neurons combined with each other. The last layer gives the decisions of the network. Usually the softmax nonlinearity function is used here, to approximate a probability distribution and avoid hesitation.

3.1.3 Capsule network

Capsule networks were invented by Geoffrey Hinton et al. a few years ago and they are believed to be a successor to the traditional convolutional neural networks which suffer from some fundamental problems.

First and foremost CNNs are not able to see a spatial hierarchy between objects. They focus on simple features and easily recognize them, but don't take into account their location in space. For example there is no difference to CNN if an eye and a mouth are swapped, only important thing is that they are present in the picture. [11]

Secondly CNNs are very susceptible to any rotation of an object. Kernels in convolution layers detect edges and shapes at a specific angle, which is very important, because when it is changed there is no mechanism to remember it so that the rotated feature is recognized as a completely new feature. Hence CNNs don't perform well on datasets with many rotated objects. [11]

To solve those problems, the capsule networks were designed. They are "*an attempt to formalize and abstract the creation of subnetworks inside a neural network"* [12] which give them greater abilities to learn more complicated patterns.

Thus the most important features which are responsible for overcoming mentioned above issues are as following:

- **routing by agreement instead of max pooling**
  First of all there is no loss of information. Therefore lower level capsules pass the found patterns only to those higher level capsules (parent capsules) which agree with them i.e. they recognize lower level features as similar to these represented by themselves. In that way, we avoid distributing useless noise.

- **vector output representation (a capsule) instead of single scalar output**
  By using a vector there is an opportunity to store more data about found pattern. Falling into details, a capsule is able to not only save the information whether object exists or not but also about thickness, width, rotation and others. Vector's length contains information about probability that the feature exists whereas the angle describes the rest of object's features.
- **reconstruction regularization**
  it's about adding a small number, which is based on difference between original and reconstructed image, to the loss function. As it is shown in [6] and [13] it does reduce overfitting and help to generalize better.

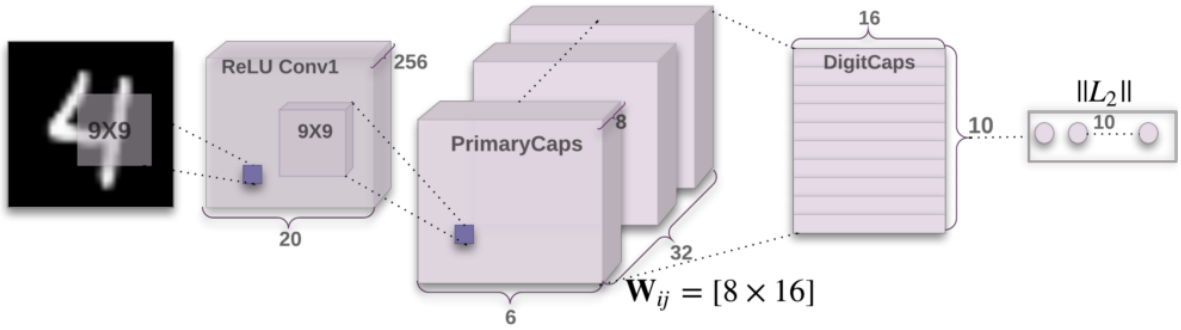

Figure 3: Architecture of capsule neural network. Source: [6]

As it is shown on the Figure 3, capsule network consists of:

- **standard convolution layer** that takes [$n\_features,\ n\_filters$] tensor and outputs the same volume.
- **primary capsule layer** created from convolution layer reshaped and squashed. The output tensor is of shape [$n\_channels\ *\ n\_features\ x\ capsule\_dim$].
- **capsule layer** that takes previous shape and outputs [$n\_classes\ x\ capsule\_dim$]

If it comes to application of capsule networks, so far the majority of internet sources describe them as the state-of-the-art technology to recognize images. Up to now they have been tested mainly on datasets containing pictures to prove that they are more efficient than traditional CNNs. However there was an attempt to try applying capsule networks on text classification. [14] Achieved performance looks very promising and ensure that capsule's idea being very effective is regarded deservedly as paradigm shift.

3.1.4 Long short-term memory network

Long-short term memory network (LSTM) is a type of recursive network that was invented in 1997 by Hochreiter and Schmidhuber and revolutionized text research, speech recognition and many other areas in which the context is the most important thing taken into account. Recursive networks are constructed similarly as MLP - they are composed of many layers, many neurons in each one. However, as it is shown in Figure 4, despite the inputs from the previous layer, each neuron has also an input from the previous time step, which gives the network so-called memory. [15][16]

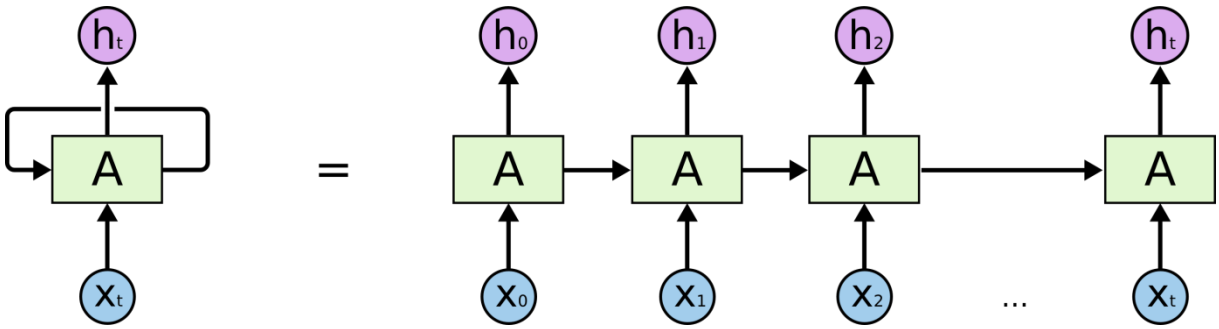

Figure 4: Architecture of recurrent neural network. Source: [17]

Thanks to this, in every learning step the network uses not only current data, but also data from the past, which allows for very effective learning in the fields that use the data context:

- speech recognition, where words are important, not individual letters
- predicting next word, where the meaning of the whole sentence is crucial, not single words
- time series anomaly detection where each value in time is strictly dependent on the previous value.

However, classic RNNs suffer from significant problem, especially vanishing gradient. Usually activation function outputs values in the range $(0, 1)$ and error backpropagated to the previous layers is computed by chain rule, so that there are a lot of small numbers multiplications which lead to effectively preventing weights from changing. Switching activation function for firing higher values is risky because on the other hand there is an exploding gradient problem.

The most popular and effective solution to this problem are the long-short term memory cells (LSTM), which were specially designed to overcome it [25]. They are naturally capable of learning long-term dependencies thanks to their structure, showed on the picture below. [18]

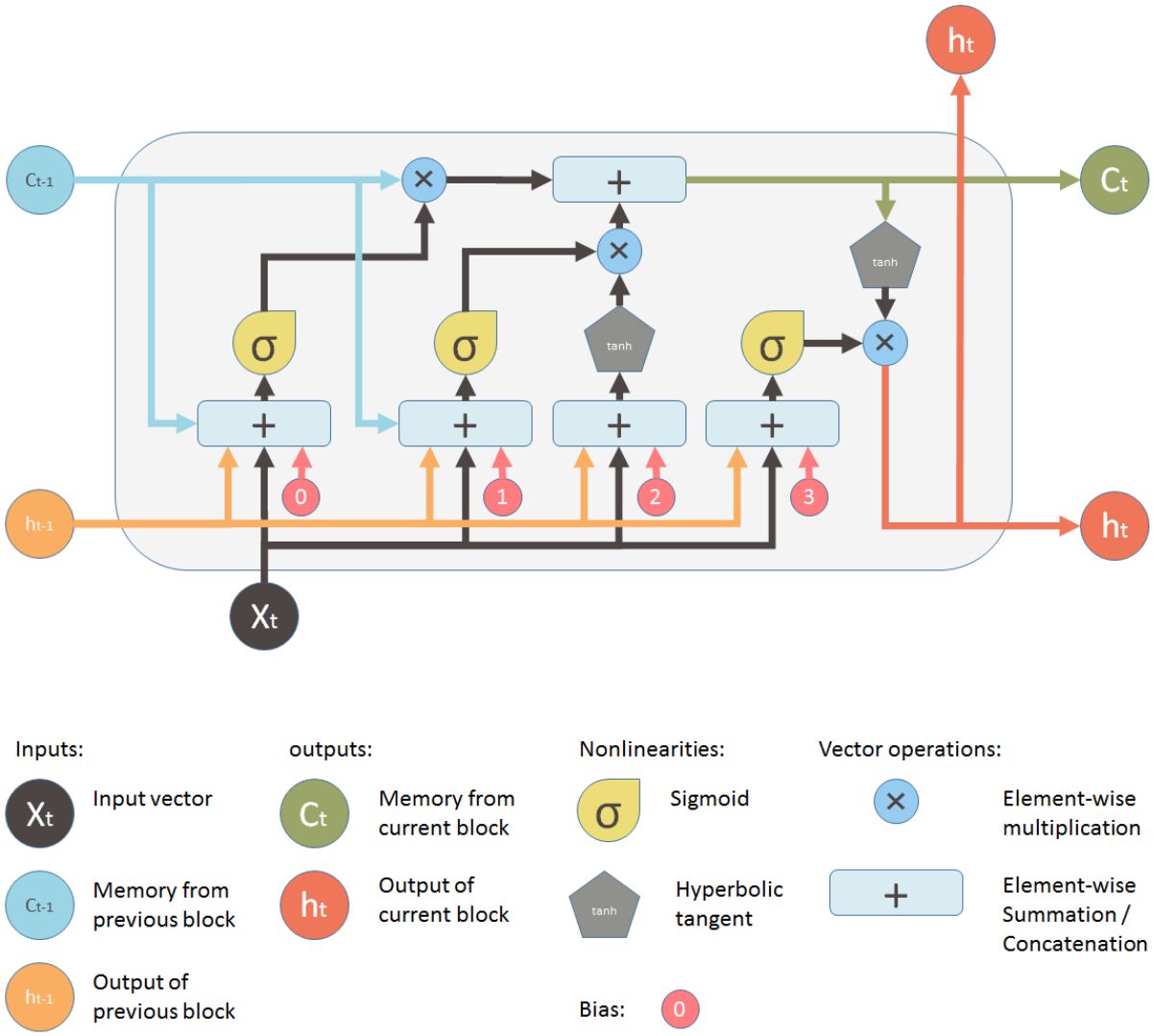

Figure 5: Architecture of LSTM neural network. Source: [18]

Generally LSTM block is divided into a few parts which are presented in Figure 5. Each of those separated small one layer neural networks do their job to remember long-term dependencies.

The left part of the LSTM block contains a network responsible for the calculation (through element-wise multiplication) of number of old memories that will be used. For values close to 0 old memories are forget, for values close to 1 are fully utilized. The value range 0-1 is provided by the sigmoid activation function. As input we have input vector, previous block's output, bias and old memories.

The central part of the LSTM block is responsible for the number of new memories that will be applied. The inner-left network with the sigmoid activation function, the same as mentioned previously, through element-wise multiplication is responsible for how many new memories will be admitted. The inner-right part, however, calculate these new memories using the current input vector, previous output vector and the hyperbolic tangent activation function. Then new memories are concatenated with the old ones.

The last, right part of the block is responsible for calculating the final output. The new memory is multiplied with summed input vector, previous output vector and bias which gives the final output of the block.

3.2 Tools and frameworks

There are many frameworks and libraries in which you can do machine learning and deep learning. This work uses the most popular and leading ones with active developers community and proven capabilities. In this chapter, they are briefly described, as well as their applications are shown.

3.2.1 Scikit-learn

Scikit-learn is a free software machine learning library for the Python programming language. It was created in 2007 as a Google summer of code project by David Cournapeau.

Scikit-learn is a python library that provides a lot of unsupervised and supervised learning algorithms. It is built using well-known and frequently used technologies such as NumPy, Pandas, Matplotlib. Functionality that provides scikit-learn [21]:

- Regression, for example linear regression, logistic regression
- Classification, for example k-nearest neighbors
- Clustering, for example k-means, k-means++
- Dimensionality reduction, for example PCA
- Model selection, for example cross validation
- Preprocessing, for example standard scaling, labels binarizer

Scikit-learn is very simple and intuitive API. It is well maintained and very popular. The vision of the library mostly focus on concerns such as ease of use, code quality, collaboration, documentation and performance.

3.2.2 Tensorflow

Tensorflow is an open-source library for numerical computing using data-flow graphs. In the beginning it was developed by the Google brain team for internal machine learning and deep neural networks research, but in 2015 it was released under the open-source license.

Tensorflow is cross platform [22]. It runs on almost every processing unit, i.e. GPUs, CPUs (including embedded and mobile platforms), and even on TPUs (Tensor Processing Units – hardware to do tensor math).

Main use cases of tensorflow:

- voice/sound recognition
- text based applications
- image recognition
- time series
- video detection

But it is general enough to be applicated for wide variety of other domains as well.

It uses Python as a convenient frontend high-level API for building scripts with the framework, but on the other hand those scripts are executed underneath using high-performance C++. On top of it often sit some even higher level API like Keras which makes building models much simpler and faster.

3.2.3 Keras

Keras is free open source high level API for fast prototyping neural network models [23]. It was developed by Francois Chollet, a Google engineer, as a part of a research project ONEIROS(Open-ended Neuro-Electronic Intelligent Robot Operating System). In 2017 Google's tensorflow team decided to support keras in tensorflow's library. Keras is able to run on top of the tensorflow, cntk or theano depending on the preferences. The main advantages are:

- user-friendliness (very smooth learning curve),
- modularity (intuitive division into modules),
- extensibility(compatible with scikit-learn, a few backends available)

Thanks to very high level of abstraction provided by Keras it is really easy to learn and create complicated models in short amount of time.

4. Datasets

PMLB stands for Penn Machine Learning Benchmark and it is a large benchmark suite for machine learning evaluation and comparison [20]. It includes 165 datasets without any missing values not to cause any problems during preprocessing. The meta features of these datasets are defined as follows:

- Instances – the number of observations in each dataset
- Features – the number of features in each dataset
- Binary Features – the number of binary features
- Categorical and Ordinal Features – the number of discrete features with more than 2 levels (integers)
- Continuous Features – the number of continuous features (floats)
- Endpoint Type – whether it is a binary or multi classification problem
- Classes – the number of classes to separate in each dataset
- Class Imbalance – metric of data balance in range (0,1). When the value is close to 0, data is perfectly balanced, eg. all classes have the same number of instances. When the value is close to 1, almost all instances belong to one class. It is calculated based on following formula:

$$I = K \sum_{i=1}^{K} (\frac{n_i}{N} - \frac{1}{K})^2$$

Not all datasets from this collection are used in this work due to lack of appropriate computing resources. Some of them are also too small for training deep neural networks. Chosen datasets are presented in Table 1.

Table 1: Description of the chosen datasets

Name	Instances	Features	Binary Features	Integer Features	Float Features	Endpoint	Classes	Imbalance Metric
Allhyper	**3771**	**29**	**21**	**8**	**0**	**Integer**	**4**	**0.93**
Allhypo	**3770**	**29**	**21**	**8**	**0**	**Integer**	**3**	**0.78**
Allrep	**3772**	**29**	**21**	**8**	**0**	**Integer**	**4**	**0.91**
Confidence	**72**	**3**	**0**	**0**	**3**	**Integer**	**6**	**0.0**
Wine Quality White	**4893**	**11**	**0**	**0**	**11**	**Integer**	**6**	**0.19**
Page Blocks	**5473**	**10**	**0**	**0**	**10**	**Integer**	**5**	**0.76**
Car	**1728**	**6**	**0**	**6**	**0**	**Integer**	**4**	**0.39**
Car Evaluation	**1728**	**21**	**21**	**0**	**0**	**Integer**	**4**	**0.39**

5. Methodology

In this chapter, the purpose and reason of the investigation is recalled. Next, the technique of hyperparameters tuning and metrics used for measurements are presented. What is more, each experiment is accurately described.

5.1 Subject and purpose of research

The subjects of the research are deep neural networks of different types, and more specifically, how the changes in their hyperparameters affect classification score.

The purpose of the research is to determine whether modifying the values of neural network hyperparameters make sense or the achieved difference in efficiency is so small that it is not worth that effort.

Hyperparameters tuning is a very demanding task when implementing a deep learning model. Most commonly they are considered as a bunch of settings that control the behavior of the model. Tuning means slightly changing the model to minimize the testing error. The model usually has a lot of hyperparameters. Each of them can accept different types of values, depending on its type - whether it is an integer, boolean or float. Checking all combinations is impossible and would be very time-consuming because with each subsequent hyperparameter the number of possible combinations (cartesian product) increases exponentially, according to the following formula:

$$Number\ of\ different\ sets\ of\ parameters\ = p_0 \times p_1 \times \ldots \times p_n$$

$$p_i = number\ of\ i-th\ parameter\ values, \qquad n = number\ of\ parameters$$

5.2 Terminology

Research tools:

- **grid search –** exhaustive search through selected subset of hyperparameter space. Collection is chosen manually in a most reasonable way based on theoretical knowledge of the model and the dataset. Metric chosen to rely on was balanced accuracy (due to high imbalance of datasets).
- **k-fold cross validation –** original dataset is randomly divided into k folds [29]. One of k subsamples is retained as a validation data and used for testing the model. The rest is used for training. For every next iteration another fold is taken as a test data and the rest as a training data. Afterall mean from scores from all iterations is computed as a final score. Mainly applied in order to allow model to better generalize and decrease the possibility to achieve outstanding result by “lucky shot”.

Research method:

- **experiment –** for each dataset and each neural network a lot of experiments with different hyperparameters were conducted using grid search and k-fold cross validation.

Research metric:

- **balanced accuracy –** applied to highly unbalanced datasets. Accuracy is calculated for each class and then everything is summed up and divided by number of classes.

5.3 Research procedure

Experiments were conducted based on balanced accuracy metric and 5-fold cross validation to exclude randomness influence. Before learning, each feature in data was standarized (removed mean and scaling to unit variance). Labels were transformed from integer to one-hot encoded to maximize performance of the networks. Every dataset had class weight calculated which were used during learning to overcome imbalance of data. For each type of neural network there were different hyperparameters which were not tuned:

- Multilayer perceptron:
    - softmax activation function for each layer (mainly used activation function for multiclass classification) [2]
    - Nadam optimizer as it is the best overall optimizer [3]
    - batch size equal to 128, it has been proved that large batches can decrease effect of learning [4]
    - categorical cross-entropy loss function as it is the most recommended function for multiclass classification [5]
- Convolutional neural network:
    - relu activation function for each convolutional layer (mainly used activation function for convolutional networks) [2]
    - softmax activation function for fully connected layer (mainly used activation function for fully connected layer because probability sum up to 1) [2]
    - Nadam optimizer as it is the best overall optimizer [3]
    - batch size equal to 128, it has been proved that large batches can decrease effect of learning [4]
    - categorical crossentropy loss function as it is the most recommended function for multiclass classification [5]
- Capsule network:
    - relu activation function for each convolutional layer (mainly used activation function for convolutional networks) [2]

    - softmax activation function for fully connected layer (mainly used activation function for fully connected layer because probability sum up to 1) [2]
    - Nadam optimizer as it is the best overall optimizer [3]
    - batch size equal to 128, it has been proved that large batches can decrease effect of learning [4]
    - margin loss function as it is loss function recommended by the creator of the the network [6]
    - 3 routings with agreement as it is the most recommended value by creators of the network [6]
- Long-short term memory network:
    - Nadam optimizer as it is the best overall optimizer [3]
    - batch size equal to 128, it has been proved that large batches can decrease effect of learning [4]
    - categorical crossentropy loss function as it is the most recommended function for multiclass classification [5]

There were general assumptions which apply for all experiments. Further each experiment will be presented in details.

In the first experiment each neural network was trained on every dataset using grid search with following pre-established hyperparameters:

- Multi layer perceptron:
    - number of hidden layers: $[1, 2, 3, 4]$
    - number of neurons in each hidden layer: $[5, 10, 20, 50, 100]$
- Convolutional neural network:
    - number of hidden layers: $[1, 2, 3]$
    - number of convolutional filters: $[5, 10, 20, 50]$
    - convolutional filter size: $[2, 4, 6]$
- Capsule neural network:
    - number of channels: $[4, 10, 15]$
    - capsules dimension: $[4, 10, 15]$
    - number of convolutional filters: $[5, 10, 20, 50]$
    - convolutional filter size: $[3, 6]$
- Long-short term memory network:
    - number of hidden layers: $[1, 2, 3]$
    - number of cells in each layer: $[5, 10, 20, 50]$
    - embedded vector length: $[4, 8, 12, 16]$

In total, taking into account 5-fold cross validation and 8 datasets $8 * 5[(4 * 5) + (3 * 4 * 3) + (3 * 3 * 4 * 2) + (3 * 4 * 4)] = 7040$ models were trained and the accuracy scores obtained for them are considered as a ‘grid search results’.

Then they were sorted and for each neural network and each dataset model with highest accuracy was chosen to be trained again for 10 times using 5-fold cross validation($5 * 10 * 8 * 4 = 1600$ more models). Therefore those results were considered as a 'results for best set of hyperparameters'.

For the second experiment results for best set of hyperparameters from previous experiment were used. In addition XGBOOST models were trained using default hyperparameters and k fold cross validation. Results obtained for them were compared against results for best set of hyperparameters.

In the third experiment number of neurons in MLP was tested single hidden layer and standard configuration which was mentioned earlier. The number of neurons ranged from 1 to 50 so there were $8 * 5 * 50 = 2000$ models trained.

In the fourth experiment number of layers in MLP was tested with pre-established overall number of neurons in network and standard configuration mentioned in the beginning of this chapter. It was decided to test 60 neurons divided into 1 to 10 layers. The number 60 doesn't have to be optimal, it was chosen because 1 hidden layer model with 60 seemed not to be too big and also when model had 10 hidden layers, 6 neurons in each seemed to be enough. General rule was to avoid having like hundreds of neurons in 1 layer or layers with less neurons than classes(6 is the largest number of classes in chosen datasets). So there were $8 * 5 * 10 = 400$ additional models trained.

6. Results

In this chapter, charts summarizing the research and allowing the verification of the hypotheses made are shown. It was decided that the charts will be of the box plot type, because the results shown in this way are the most statistically reliable. We see the median of the results as well as the distribution and the outliers that are the least desirable.

Each of the datasets is different and has its own meta features, so as it was difficult to create a model that would efficiently learn on each of them. Typically, each element of the model is adapted to the specific features of the dataset, in this case, however, because it is a comparison of many collections, it was not possible. Though, generally efficient model was created with some hyperparameters just widely used and considered as sufficient.

6.1 Tuning parameters – looking for best settings

In this paragraph, for each neural network architecture there is one figure on which there is a comparison of the results achieved for each dataset individually. Results from grid search using 5-fold CV are plotted in blue and the results for best set of hyperparameters using 5-fold CV are plotted in orange. Outliers are marked as red dots. Taking into account that model with best hyperparameters comes from the collection of models from grid search – it is assumed that when median for results for best set of hyperparameters is in upper part of the results from grid search – it means that tuning has been effective. Another important factor is box plot shrinking – it proves that classifier is more stable.

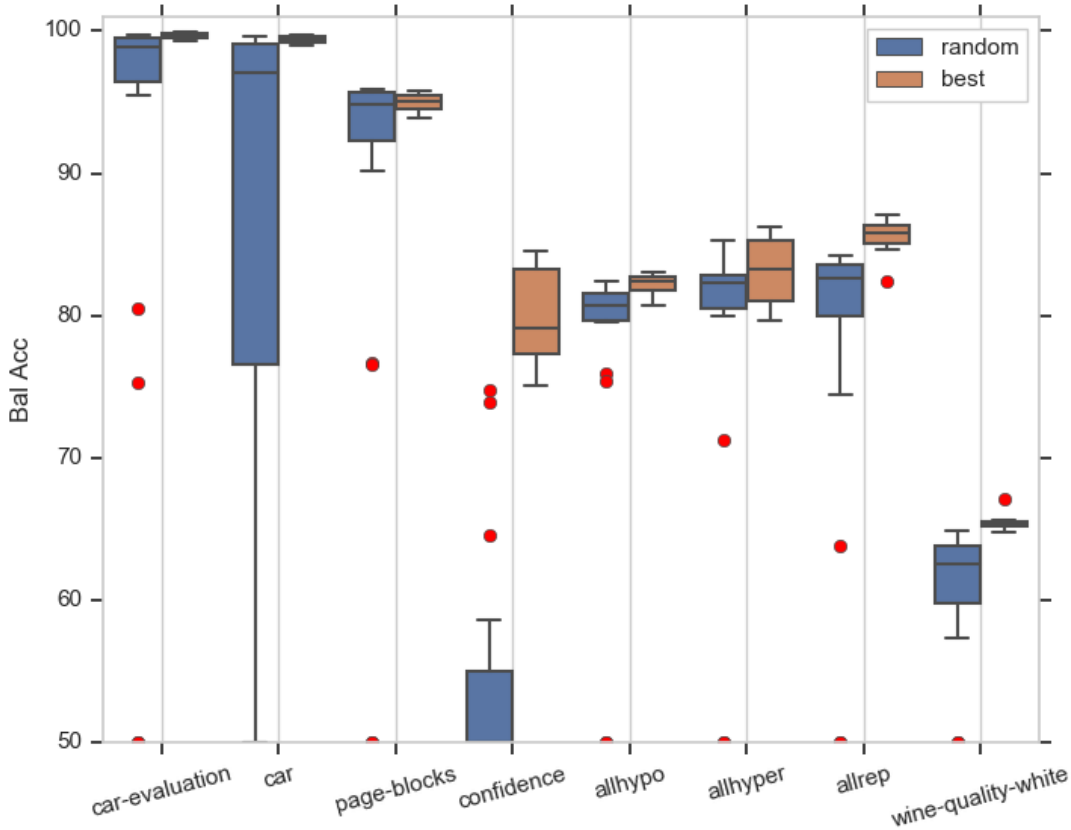

Figure 6: Comparison of the accuracies on each dataset for multi linear perceptron

First of all a lot of outliers for each dataset in Figure 6 are surprising. However, most of them belong to results from grid search, there is almost no significant outlier for results for best set of hyperparameters. This means that the model created with best settings learns much better and more stable - we have received more focused results around a specific value. This indicates a high sensitivity to changing the hyperparameters.

It can be concluded that car-evaluations, car and page-blocks datasets are relatively easy (accuracy almost 100%). Their box plots for results for best set of hyperparameters extend to a maximum of 1-3%, which is a satisfactory result and confirms the stability of the classifier. Charts for results from grid search are more stretched and achieve lower accuracy – tuning has been effective.

Grid search results for the confidence set are very low - this is due to the very small number of samples in this set (about 50). This is not enough for the classifier to be able to effectively learn knowledge, hence high instability and randomness. However results for best set of hyperparameters are much better. Regardless of amount of samples in dataset, tuning has been effective.

The allhypo, allrep collections can be assessed as relatively moderate (results at 80-90 % level). Box plots for results for best set of hyperparameters are located a bit higher and are

more squeezed than for results from grid search – the effect of tuning can be seen. This is not a very big improvement - at the level of a few per cents.

Allhyper is a very similar dataset to allrep, so the results should be also similar. However box plot for results for best set of hyperparameters is just a bit higher and is more stretched. Probably the effect of different shuffling or bumping into local maxima from which the model was not able to escape in the time left to learn.

Wine-quality-white dataset turned out to be a bit more difficult than the rest (60-70 % accuracy), although there is a clear increase in accuracy for results for best set of hyperparameters. In addition to this, the box plot also shrunk very much. In this example the effect of tuning can be seen clearly.

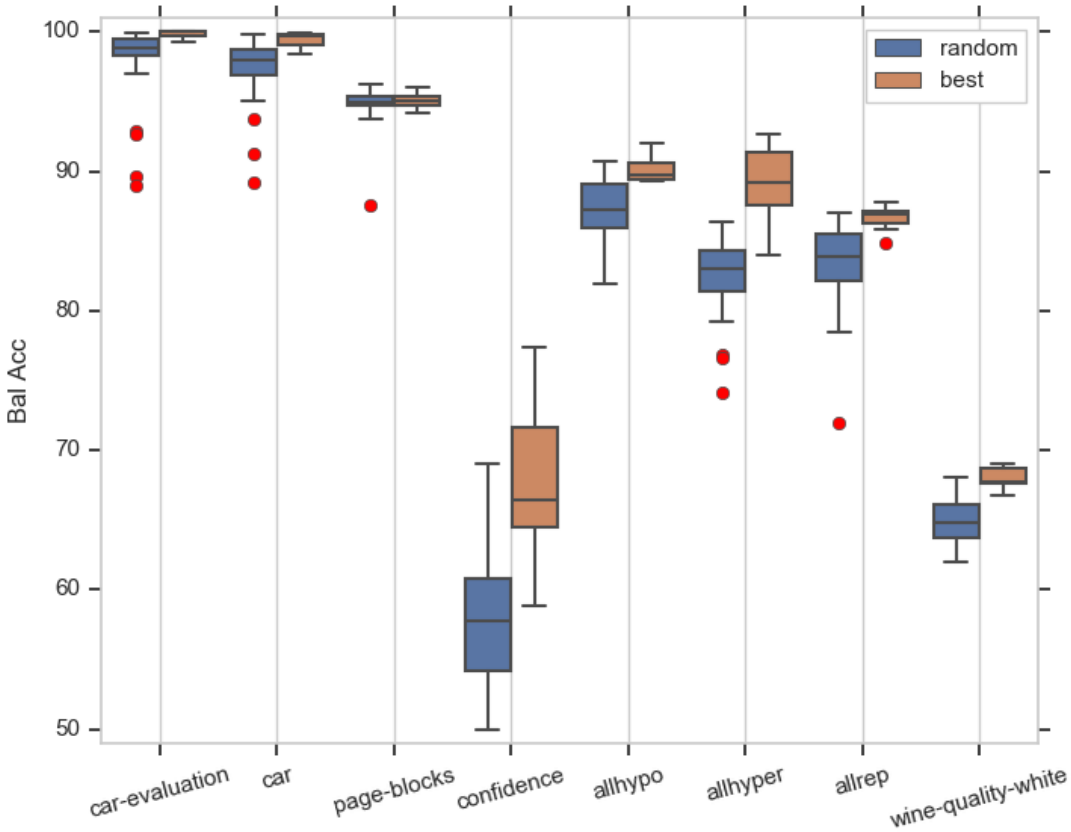

Figure 7: Comparison of the accuracies on each dataset convolutional neural network

In the case of convolutional network, in Figure 7 outliers are much less and they do not present much lower values comparing to the rest. It means that the model is less susceptible to tuning hyperparameters and has more advanced structure.

For the car-evaluation and car sets, the effects of tuning can be seen - tighter box plots, higher accuracy, reduced outliers.

The distribution of results for the page-blocks dataset looks very similar for best settings and from grid search settings - only one outlier has been reduced - but this is not meaningful improvement.

The confidence dataset gives very sparse results in this case - it is simply too small.

Results for allhyper, allrep and allhypo datasets show the effects of tuning - accuracy values are clearly higher, no outliers for results for best set of hyperparameters. However, the results for allhyper are again a bit more stretched - probably the effect of bumping on the local maximum.

The wine-quality-white collection, once again, presents the most clear effect of tuning - the whole box plot for results for best set of hyperparameters is above the one for results from grid search and it is not very stretched.

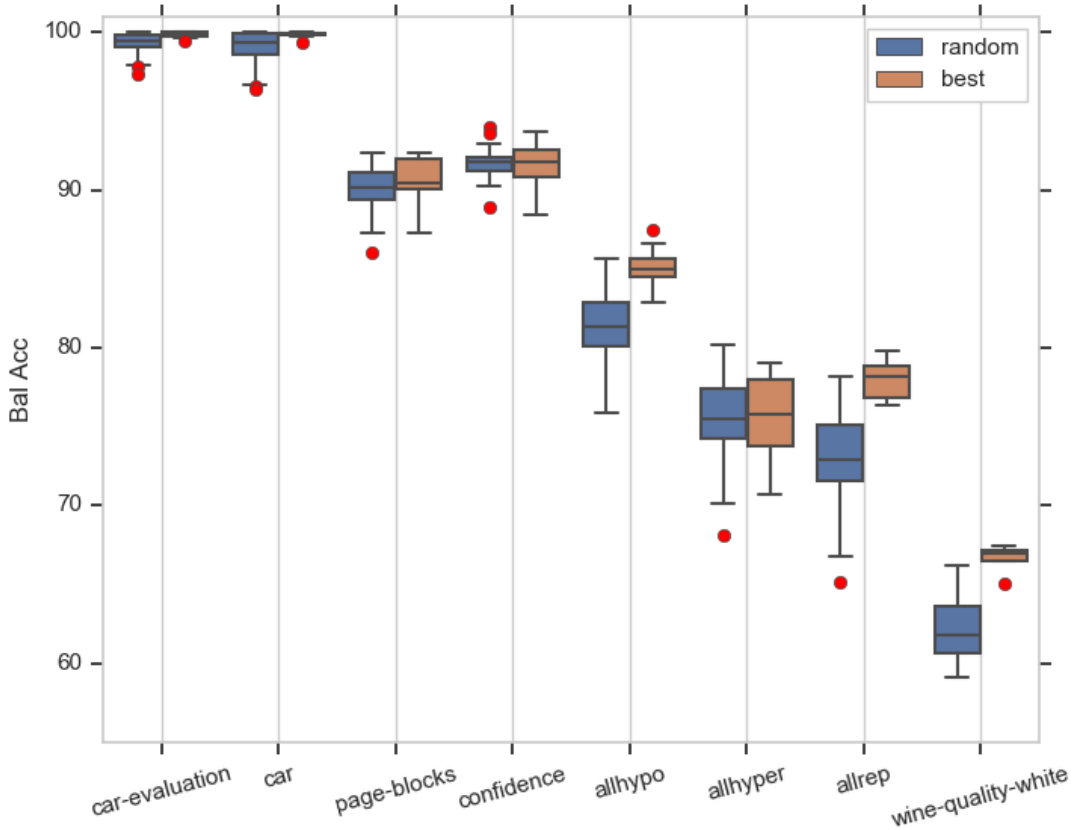

Figure 8: Comparison of the accuracies on each dataset for capsule neural network

There are almost no outliers in Figure 8 – probably because capsule network is quite advanced structure and even with bad hyperparameters is capable of learning well enough.

As always car and car-evaluation datasets show small effect of tuning – thinner boxes and higher accuracy score. Most results from grid search as well as results for best set of hyperparameters are about 99-100%, so each result like 97-98% is marked as negligible outlier.

Box plots for page-block, confidence and allhyper datasets are similar, there is no significant differences. Tuning did not improve anything.

Results for allrep, and allhypo datasets show quite good effect of tuning. Medians for results for best set of hyperparameters are higher than equivalents for results from grid search. Boxes are not very sparse.

For wine-quality-white dataset box plot connected with results for best set of hyperparameters is much higher than box plot for results from grid search and also it is much tighter – very good effect of tuning.

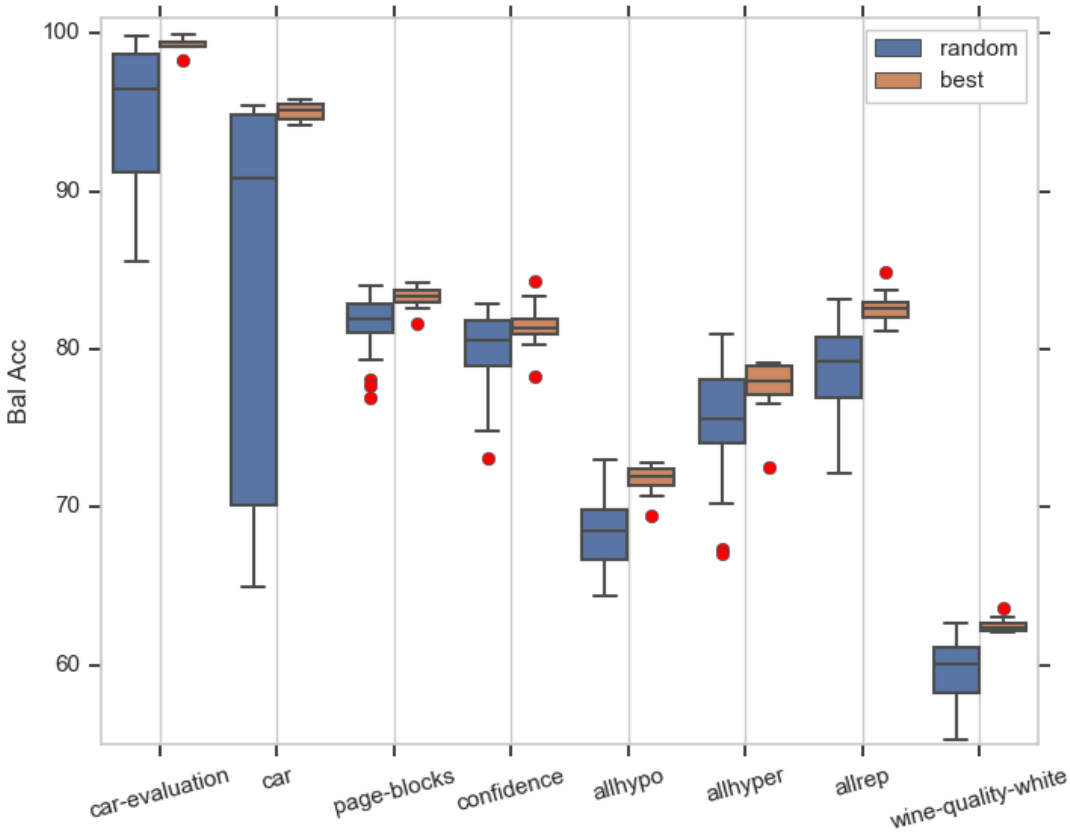

Figure 9: Comparison of the accuracies on each dataset for long-short memory network

Similarly to the capsule network – in Figure 9 there are barely any outliers with much lower values. LSTM is quite complicated architecture inside so it generally learns well.

For car and car-evaluation datasets huge improvement can be seen thanks to hyperparameters tuning. Boxes shrank noticeably and the accuracy improved by a few per cent.

Results for confidence does not look as bad as in the previous examples. Box plot for best settings results is a bit higher as well as it is not so wide. Small improvement thanks to tuning can be seen.

For the first time there is a difference comparing results for best set of hyperparameters and from grid search. Box plot is a bit higher and is more shrunk.

Allhypo and allrep datasets show clearly the effect of tuning – results for best set of hyperparameters are significantly higher and more shrunk than the results from grid search.

allhyper dataset show some improvement but difference is not so big. However, the box plot shrunk so there is some effect.

For the fourth time results for the hardest dataset – wine-quality-white - show significant improve in accuracy score and are much more consistent after tuning

6.2 Comparing results with XGBoost

Deep learning is developing very dynamically nowadays and finds more and more applications, but it does not mean that it is the best solution for every encountered problem. For many years, a lot of issues were resolved with high efficiency by machine learning algorithms. Arrival of a new concept does not mean that the old one stops working. There are very many machine learning algorithms with multiple applications. Recently one of the most famous ones are decision trees. It is simple weak classifier which splits the source dataset based on attribute value. It is repeated recursively n times, until in one final group there are only samples with the same labels or splitting does not bring any more value to the predictions. Generally decision trees are ensembled into a groups to make a strong classifier. One of the most popular strong classifiers compound from decision trees now is Gradient Boosted Machine. It build trees one by one and every next tree tries to correct errors made by the previous one.

In this work XGBoost framework is used [28]. It is a library that provides scalable and well optimized gradient boosting methods. As a comparison, XGBoost models with default hyperparameters are used against the deep neural networks.

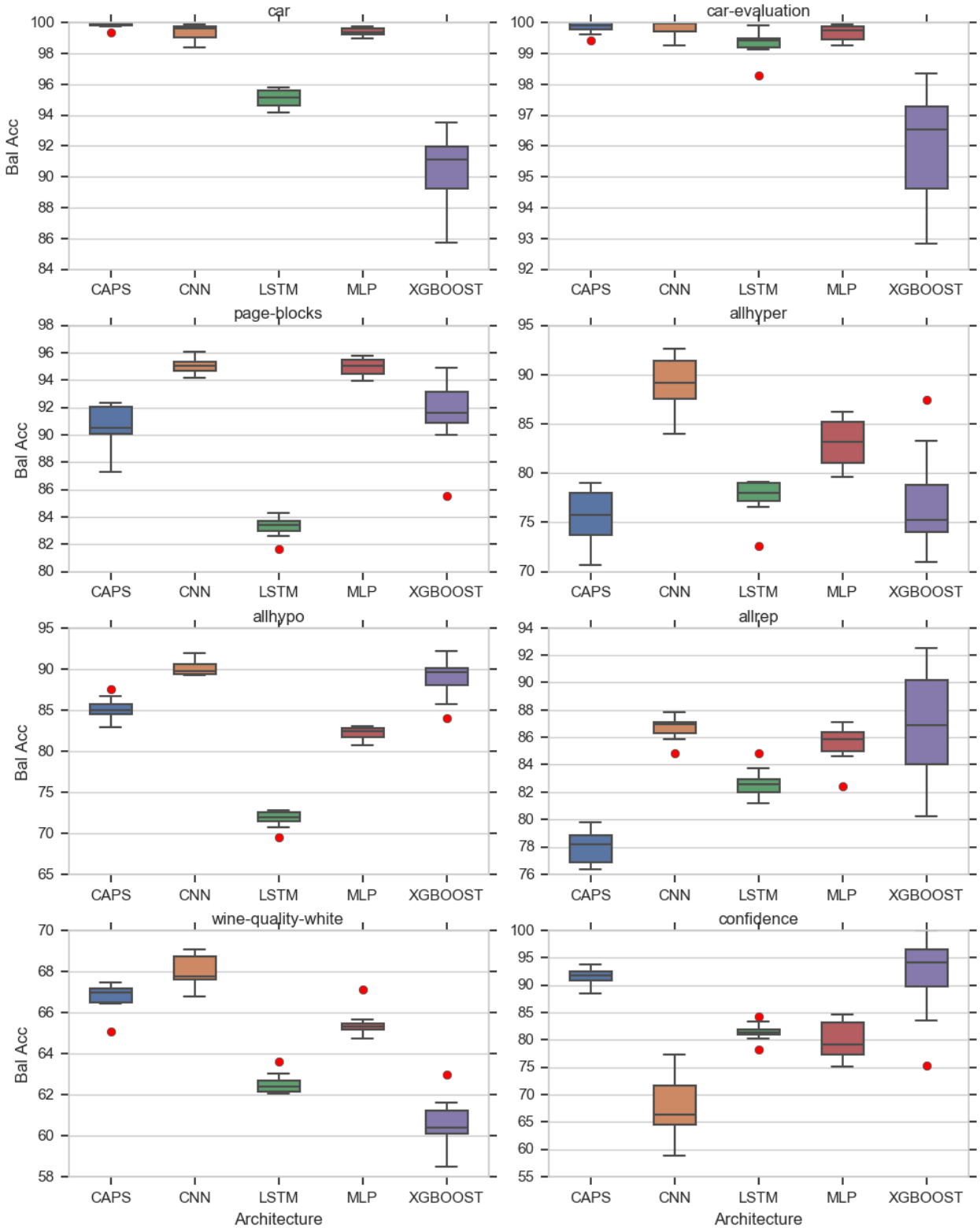

Figure 10: Comparison of the accuracies on each dataset for CAPS, MLP, CNN, LSTM and XGBOOST

Explanation for each dataset shown in Figure 10:

- **car** - the best networks – MLP, CNN and CAPS - achieved about 99-100% accuracy. LSTM proved to be a few per cents worse, but for all of them the box plots are very narrow. XGBOOST results range from 86 to 94% - they are much lower and more sparse. Generally neural networks performed better and in this case seemed to be more stable classifiers.
- **car evaluation** – All neural networks scored similar accuracy – about 99-100%. Plots are narrow. XGBOOST did more sparse and lower results – box plot range from 93 to 98%. Again the networks made it better.
- **page-blocks –** CNN and MLP are ahead in this case. They did about 95% and are little stretched. Second place for CAPS and XGBOOST classifiers – achieved about 91%, but they have a bit more sparse box plots. In the end LSTM with only 83% accuracy.
- **allhyper –** Single leader – CNN made about 90%. Then the MLP with average of 83%. CAPS, LSTM and XGBOOST results oscillate around 76%. Generally despite of LSTM, which is quite narrow, for all classifiers box plots are about 10 % stretched. It looks like it is hard to achieve stable results on allhyper dataset.
- **allhypo –** For the first time XGBOOST equally with CNN scored the best - about 90%. CAPS and MLP proved to be a few per cents worse. LSTM did around 73%. All box plots for neural networks are rather narrow, just the XGBOOST is a bit more sparse.
- **allrep –** In this case CNN, MLP and XGBOOST achieved similar results – about 86%, but box plot for XGBOOST is widely stretched for around 13%. The rest of classifiers made generally concentrated results. LSTM is one before last with 82% score and the CAPS is in the end with 78%.
- **wine-quality-white –** CAPS and CNN are ahead, they did accuracy around 67-69%. Second place for MLP with 65% score. LSTM achieved about 62%. Last but not least is XGBOOST which made 60%. All box plots are rather narrow, it looks like all classifiers lerarn very stable on this dataset.
- **confidence –** XGBOOST is leading here despite of quite sparse results – range from 85 to 100% with median around 95%. Then there is CAPS with score about 92%. Then the LSTM equally with MLP which achieved 80%. Surprisingly in the end is CNN classifier with a bit stretched box plot and relatively low accuracy of 65%. It looks like deep networks are not performing well on small datasets. XGBOOST even with default hyperparameters outclassed the other classifiers.

Generally, CNN performs best for most datasets. MLP was usually in the second place. Caps sometimes gave very good results and sometimes very weak. LSTM fell out the worst. Achieved results are also largely dependent on the data set, not just the classifier

6.3 How many neurons in MLP?

Building your own neural network architecture is not an easy task. When the model is too small, it will underfit, on the other hand when it is too complicated, it is highly likely that it will overfit. The aim of this paragraph is to check how fast MLP with 1 hidden layer and number of neurons increasing from 1 to 50 will stop improving the accuracy. This means that more neurons are not needed. Experiments were carried out using 5 fold CV.

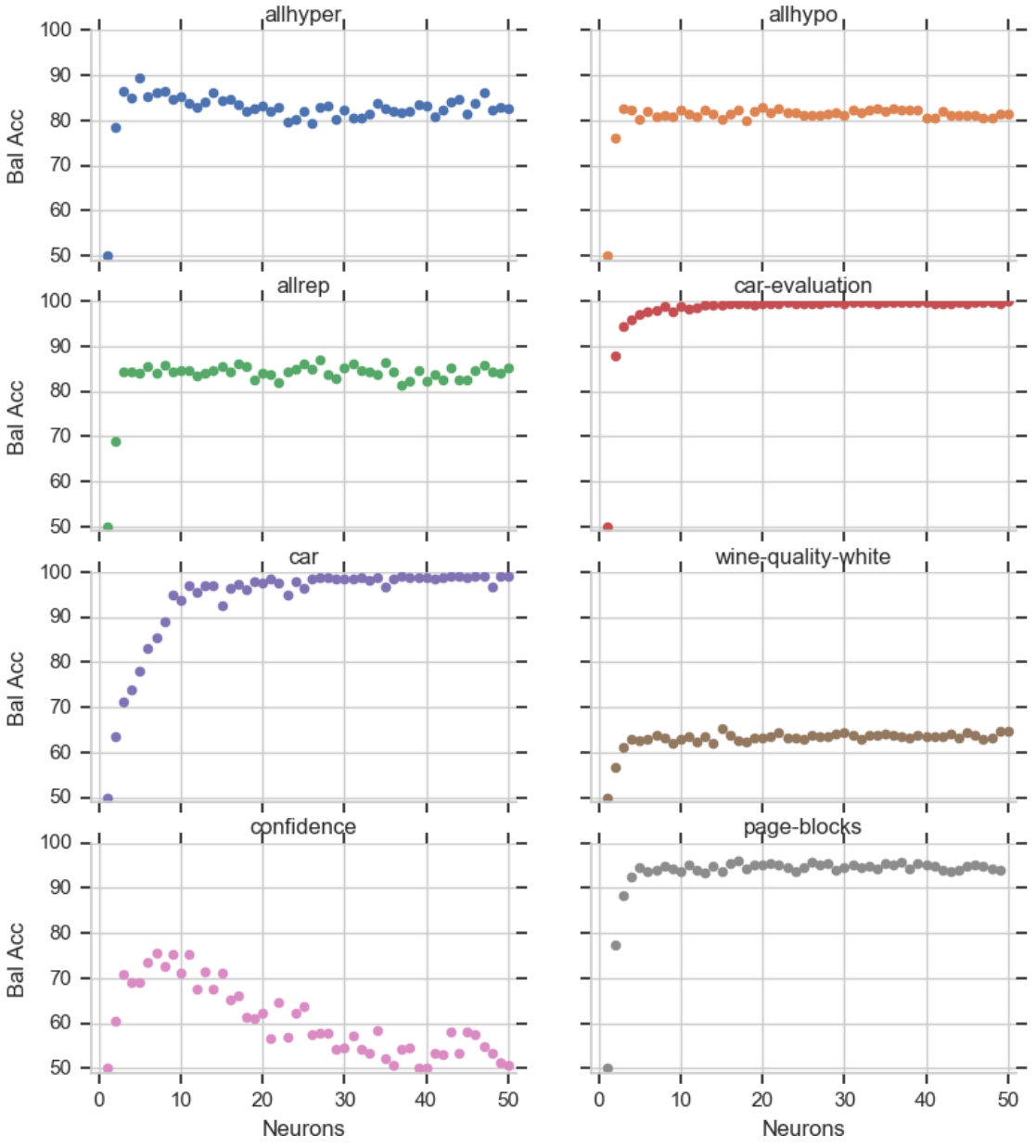

Figure 11: Comparison of the accuracies on each dataset for MLP for 1 hidden layer and 1-50 neurons

Figure 11 shows that for most datasets just 3-4 neurons in hidden layer are capable of scoring relatively well for this classifier. In two cases, for car and car-evaluation about 10 neurons were necessary. Confidence is a really small dataset – here the overfitting can be seen. Using more than 10 neurons leads to accuracy score decreasing. Rest of the datasets are big enough not to be susceptible to overfitting.

6.4 How many layers in MLP?

Despite of the number of neurons in each layer, the number of layers itself is also important. Similarly, if too many layers are used, model can be too complex and tends to overfit. Objective of this paragraph is to check whether adding more layers to MLP while keeping the same overall number of neurons in network can bring any improvements in performance.

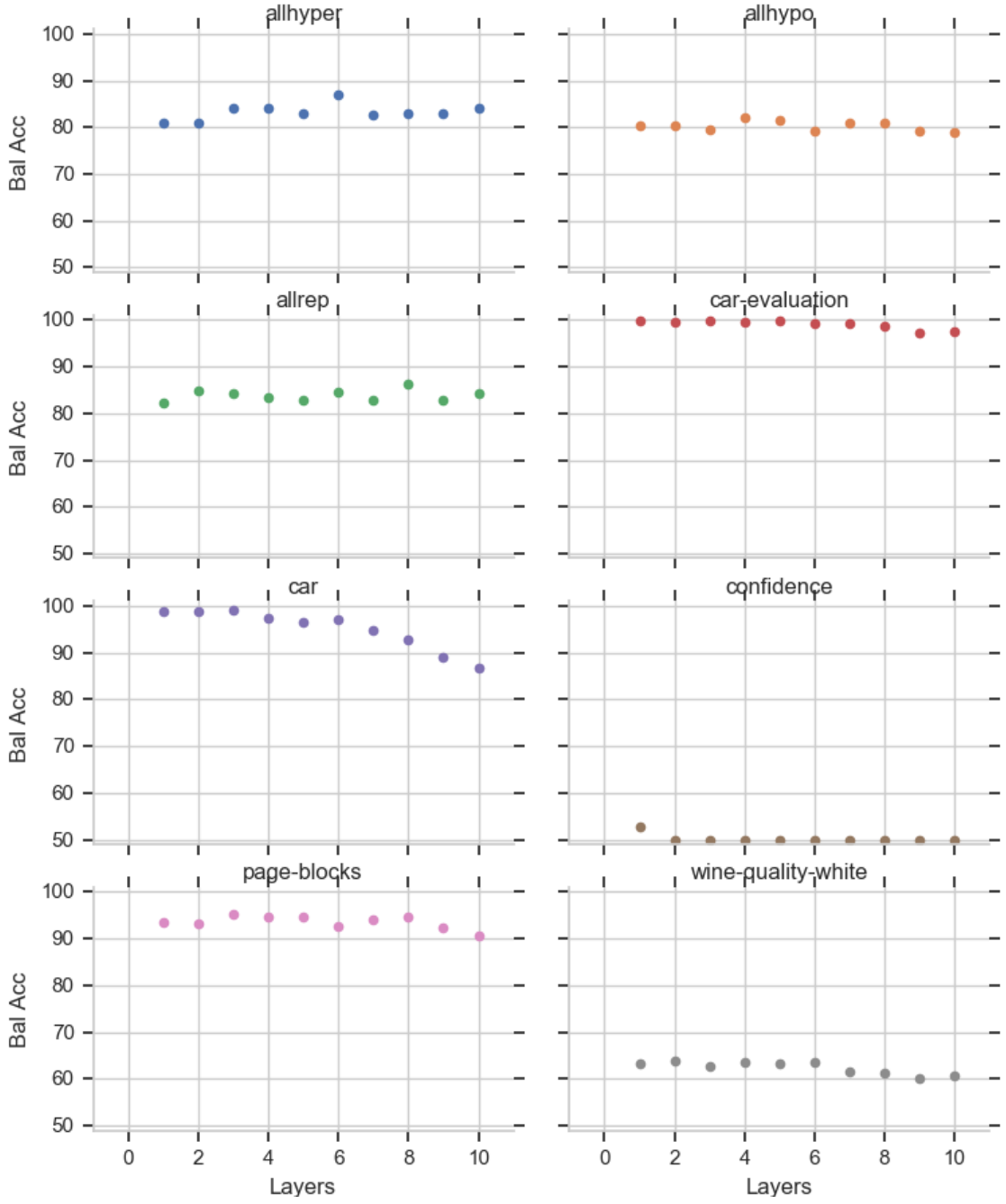

Figure 12: Comparison of the accuracies on each dataset for MLP for 60 neurons and 1-10 hidden layers

As it is shown in Figure 12, mostly results achieved by network with more layers is similar to those with 1 layer with little tendency to decrease. Generally 1 layer with appropriate number of neurons looks enough for datasets tested in this work.

7.Conclusions and future work

This chapter will present the lessons learned from the analysis of the results from the previous chapter. In total, about 11,000 models were taught for testing, which for an average learning length of 3 minutes gives about 550 hours using a normal computer.

7.1 Tuning parameters – looking for best settings

The first stated hypothesis was as follows:

For each of the selected models of deep neural networks there are hyperparameters that allow for achieving much better accuracy score and more stable results.

First and foremost MLP and LSTM network had much more stretched results from grid search than CNN and Capsule network. From this it follows that in task of classification models based on convolutional filters are less sensitive to hyperparameters tuning compared to those based on neurons.

Secondly, for five of eight datasets (car, car-evaluation, allhypo, allrep, wine-quality-white) medians for best sets of hyperparameters are a few per cent higher than their equivalents from grid search results. Also box plots are more shrunk. It looks good.

Thirdly there are two datasets which show moderate improvements. For allhyper there were improvements in accuracy and stability for CNN and LSTM network, but not for Capsule network and MLP. On the other hand for page-blocks there were improvements in accuracy for MLP and LSTM network but not for Capsule network and CNN.

Lastly the smallest dataset – confidence. Surprisingly results are not totally random. 2 of 4 times box plot shrunk and 3 of 4 times median for best set of hyperparameters is higher than median from grid search. From tested hyperparameters it was possible to build small models and avoid overfitting. Probably that’s why received results are positive.

Summing up, in most cases the tuning has succeeded. There are some exceptions that can be caused by many distortions, such as randomness, stuck in a local minimum, or a small amount of data.

The hypothesis is partially confirmed.

7.2 Comparing results with XGBoost

The second stated hypothesis:

Deep neural networks are competitive and sometimes able to outperform classic machine learning algorithms by a few per cent.

First of all box plots for results obtained with XGBoost were more sparse than box plots for tested neural networks. From this it follows that this machine learning classifier with default hyperparameters is less stable.

Omitting the small confidence dataset, for all others CNN achieved the best accuracy scores. It can be concluded that convolutional filters gather knowledge efficiently not only from images, but generally from n-dimensional vectors. MLP also proved to be not much worse than the CNN. For 6 out of 7 datasets MLP took the 2nd place, so it is still working quite good. LSTM was usually the worst. This network is well-suited to data based on time-series. It can be an explanation why it performs so poorly when applied to data without any internal links.

For the confidence dataset XGBoost occurred to be the best. Small number of samples did not influence the performance because machine learning algorithms don't need as much data as deep neural networks to learn efficiently.

Failure of capsule or LSTM network may be explained that those networks were not designed to work on this type of data, so they just perform average.

The hypothesis is confirmed.

7.3 How many neurons in MLP?

The third hypothesis was as follows:

A lot of neurons is needed in single hidden layer multi-layer perceptron to generalize well.

For five of eight datasets, the accuracy score achieved for just 3-4 neurons did not differ from that achieved for several dozen neurons. For the next 2 datasets, this threshold was at the level of 10-11 neurons. The last smallest one began to achieve good results for 3-4 neurons, and for a dozen or so they began to deteriorate - the effect of overfitting.

In summary, when looking for the optimal number of neurons, it is better to start with small values and increase it gradually until the improvement of results is no longer visible. We then maintain a balance between the speed of learning and the quality of the results achieved.

The hypothesis is rejected.

7.4 How many layers in MLP?

The fourth hypothesis was as follows:

More than one layer is needed in multi-layer perceptron to generalize well.

The results are clear - for all analyzed datasets, adding additional layers does not bring a positive effect. In some cases, the results achieved even fall. From this it follows that it is not worth using more than one hidden layer. With more layers the model is more complicated and learning becomes more difficult, which results in reduced effectiveness.

The hypothesis is rejected.

7.5 Future work

The issues presented in this work do not fully cover the subject regarding hyperparameters tuning. There are many ways that this work can be developed and some of them will be mentioned there:

First of all, only 8 data sets were used in this work. This was caused by limited computing resources. PMLB offers many more datasets that could be utilized. In addition, there is another collection of datasets - OpenML100, which also offers a lot of well-prepared data. However, to perform tests with more data, more computing power is needed. Therefore, in this case, it would be necessary to use a computing cluster to speed up the research.

In this work, the results obtained for 4 types of neural networks were compared with only one machine learning classifier - XGBoost with default hyperparameters. The work could be extended to other types of neural networks or other types of machine learning algorithms for comparison. XGBoost is a tool with a lot of potential, so after hyperparameters tuning is very likely that it would be much more competitive for neural networks.

Also due to the lack of computing power, calculations were performed only for a small amount of hyperparameters. With more capabilities, more hyperparameters could be tested. and also plotted on charts to examine the change in which hyperparameter affects a given model of the neural network.

What's more, a deeper analysis of the datasets could be conducted so as to find connections between datasets' metafeatures and the results obtained.

Bibliography

[1] Jurgen Schmidhuber. “Deep Learning in Neural Networks: An Overview”, 8 Oct. 2014, arXiv:1404.7828v4

[2] Sagar Sharma. “Activation Functions: Neural Networks”, 6 Sep. 2017, https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

[3] Sebastian Ruder. “An overview of gradient descent optimization algorithms”, 19 Jan. 2016, http://ruder.io/optimizing-gradient-descent/

[4] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, Ping Tak Peter Tang. “On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima”, 9 Feb. 2017, arXiv:1609.04836v2

[5] Ravindra Parmar. “Common Loss functions in machine learning”, 2 Sep. 2018, https://towardsdatascience.com/common-loss-functions-in-machine-learning-46af0ffc4d23

[6] Sabour Sara, Nicholas Frosst and Geoffrey E. Hinton. "Dynamic Routing Between Capsules", 7 Nov. 2017, arXiv:1710.09829.

[7] Assaad Moawad. “Neural networks and back-propagation explained in a simple way”, 1 Feb. 2018, https://medium.com/datathings/neural-networks-and-backpropagation-explained-in-a-simple-way-f540a3611f5e

[8] Manish Saraswat. “Understanding Deep Learning & Parameter Tuning in R”, 30 Jan. 2017, https://www.hackerearth.com/blog/machine-learning/understanding-deep-learning-parameter-tuning-with-mxnet-h2o-package-in-r/

[9] Lightning Blade. “Demystifying Convolutional Neural Networks”, 02 Sep. 2018, https://medium.com/@eternalzer0dayx/demystifying-convolutional-neural-networks-ca17bdc75559

[10] Adit Deshpande. “A Beginner’s Guide To Understanding Convolutional Neural Networks”, 20 Jul. 2016, https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/

[11] Bourdakos Nick. “Understanding Capsule Networks - AI's Alluring New Architecture”, 12 Feb. 2018, medium.freecodecamp.org/understanding-capsule-networks-ais-alluring-new-architecture-bdb228173ddc

[12] Matthew Graves. “How does Hinton’s capsules theory work?”, 4 Aug. 2016, ai.stackexchange.com/a/1313

[13] Edgar Xi, Selina Bing, and Yang Jin. “Capsule network performance on complex data”, 10 Dec. 2017, arXiv:1712.03480.

[14] Wei Zhao, Jianbo Ye, Min Yang, Zeyang Lei, Suofei Zhang, Zhou Zhao. “Investigating Capsule Networks with Dynamic Routing for Text Classification.” 29 Mar. 2018 arXiv preprint arXiv:1804.00538.

[15] Denny Britz. “Recurrent Neural Networks Tutorial, Part 1 – Introduction to RNNs”, 17 Sep. 2015, http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/

[16] Andrej Karpathy. “The unreasonable Effectiveness of Recurrent Neural Networks”, 21 May 2015, http://karpathy.github.io/2015/05/21/rnn-effectiveness/

[17] Christopher Olah. “Understanding LSTM Networks”, 27 Aug. 2015, http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[18] Shi Yan. “Understanding LSTM and its diagrams”, 13 Mar. 2016, https://medium.com/mlreview/understanding-lstm-and-its-diagrams-37e2f46f1714

[19] Sergey Ioffe, Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”, 2 Mar 2015, arXiv:1502.03167v3

[20] Randal S. Olson, William La Cava, Patryk Orzechowski, Ryan J. Urbanowicz, Jason H. Moore. “PMLB: A Large Benchmark Suite for Machine Learning Evaluation and Comparison”, 1 Mar 2017, arXiv:1703.00512v1

[21] Scikit-learn documentation. https://scikit-learn.org/stable/

[22] Tensorflow documentation. https://www.tensorflow.org/

[23] Keras documentation. https://keras.io/

[24] C.-C. Jay Kuo, Min Zhang, Siyang Li, Jiali Duan, Yueru Chen. “Interpretable Convolutional Neural Networks via Feedforward Design”, 21 Oct 2018, arXiv:1810.02786v2

[25] Alex Sherstinsky. “Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network”, 4 Nov 2018, arXiv:1808.03314v4

[26] Dawei Dai, Weimin Tan, Hong Zhan. “Understanding the Feedforward Artificial Neural Network Model From the Perspective of Network Flow”, 26 Apr 2017, arXiv:1704.08068v1

[27] Liam Li, Kevin Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Moritz Hardt, Benjamin Recht, Ameet Talwalkar. “Massively Parallel Hyperparameter Tuning”, 29 Nov 2018, arXiv:1810.05934v3

[28] Tianqi Chen, Carlos Guestrin. “XGBoost: A Scalable Tree Boosting System”, 10 Jun 2016, arXiv:1603.02754v3

[29] Henry B.Moss, David S.Leslie, Paul Rayson. “Using J-K fold Cross Validation to Reduce Variance When Tuning NLP Models”, 19 Jun 2018, arXiv:1806.07139v1