Praca Magisterska

Agnieszka Pocha

September 23, 2015

Abstract

The goal of this work is to build a model that would classify ligands as active or inactive based on their fingerprints. Convolutional neural network are used. Two approaches are described. Two new (?) methods to include unlabeled samples in the learning procedure are presented.

Contents

Chapter 1

Introduction

1.1 The goal of this paper

Virtual screening is a part of drug design. It involves screening great amounts of molecules in order to find those which have interesting features. In this paper we describe how convolutional neural networks can be used to approach this problem. Our goal was to build a model that would classify ligands as active or inactive. Two approaches have been proposed. Moreover, we show new approach to make use of unlabeled examples during the learning process of convolutional neural networks.

1.2 related work, że to ma oparcie w czymś i o czym jest praca

sth sth

1.3 Problem

One of the problems in drug design is telling whether *protein* and *compound* will react and produce a new complex, called *ligand*. If the ligand indeed will be created, it is important to know whether this new complex will be *active* or *inactive*.

Proteins are molecules built from *amino acid residues* forming a single *chain*. Some proteins have less than 100 residues while others might have even few thousands of them. The order of amino acids in the chain and their spatial arrangement carry a lot of information.

Our goal was to build a model that would classify the ligand as *active* or *inactive* based on its *fingerprint*. Fingerprints are a widely used (?) representation of molecules. There are many types of fingerprint designed. They vary in length and the type of information they carry. Some of them are designed for specific tasks. (citation needed). Conventionally, fingerprints represent a molecule as a vector. Each element of this vector describes whether a specific

structure is present in the molecule or not, e.g. whether the molecule has a hydrophobic group or not.

Vectors are commonly used in machine learning as data representation because they can be stored in matrices which allows easy calculations. Even though representing a protein as a vector means loosing a lot of information about its topology, it enables effective computation and still provides us with reasonable results.

As already said, there are many different fingerprints designed. They vary in length and the features included. In this paper we used 2D SIFt [2] fingerprints. We used convolutional neural network because of they are eligible to work with data which topological carries information.

1.4 How this paper is organised

At first, the convolutional neural networks are described. We present such concepts as convolution or pooling. The properties of convolutional neural networks are given.

Chapter 3 is the most important part of the paper. In this chapter we present the two approaches that we used to address the problem. We describe in detail the data on which we performed experiments and the preprocessing that we used to extract most interesting features from the data. A detailed explanation of the architecture that we used is given. We describe in detail both training methods that we created to tackle the problem of using unlabeled samples in the learning algorithm - the naive approach and the fancy approach.

Our results are described in the next chapter. Finally, in the last chapter we discuss possible ways to further develop our approach. At the end of this paper there is a short appendix which includes some less important concepts used in this work.

Chapter 2

Convolutional Neural Networks

The simplest definition of convolutional neural networks is probably: neural networks that take advantage of using the convolution operation. CNN might be seen as a network consisting of two parts - first part is the convolutional part and it is responsible for extracting the features from the data. The second part might be a softmax layer responsible for classifying samples based on features provided by the convolutional part. This schema is shown on figure 2.1.

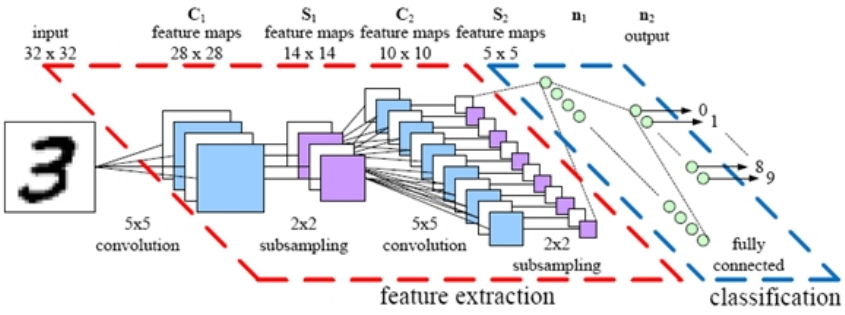

Figure 2.1: An example of convolutional neural network. The figure was downloaded from http://parse.ele.tue.nl/cluster/2/CNNArchitecture.jpg

In this section we will show what motivation stands behind using convolutional neural networks, explain what is the convolution operation, and what is pooling. Types of activation functions that might be used in convolutional neural networks will be presented as well as the learning algorithm.

2.1 Motivation

Convolutional neural networks take advantage of data which topological order carries some information. Therefore, CNNs are often applied to image recognition, video analysis and natural language processing problems.
These attempts *are often succesful/often give better results than (any other) models*.
Since the the topological order of the data we are working on carries information (due to how the proteins are built), we expect that using convolutional neural networks might bring good results.

2.2 Computation Flow

As stated above, CNNs can be conceptually divided into two subnetworks. In this subsection it will be described how the data is processed within the convolutional subnetwork. Not much attention will be put to classifying subnetwork as there is a wide variety of possible approaches.

There are three elemental operations that are performed in each layer of the convolutional subnetwork. At first, the input is convoluted with a convolution matrix. The result of this operation becomes an input to the activation function, which output becomes the input to the pooling function. The output of the pooling function becomes the input to the next layer which might be another convolutional layer and the whole process will begin again. This process is schematically depicted on figure 2.2 and it can be also described by the following formula:

$$output = p(\sigma(c(input))),$$

where c is the convolution function, σ is the activation function, and p is the pooling function.

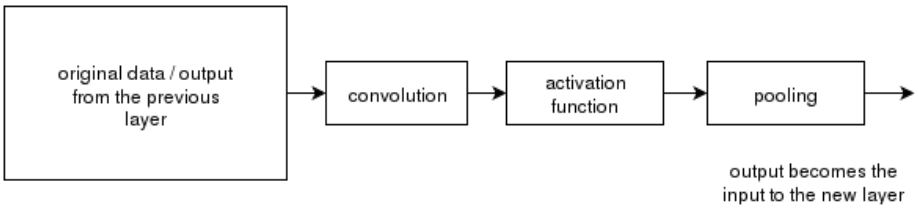

Figure 2.2: The three elemental operations performed in each convolutional layer.

One might also imagine three consecutive seperate layers: a convolutional layer, a classical layer that applies activation function and finally a pooling layer.

Each of these operations will be described in details in the following subsections.

2.2.1 Convolution

Convolution operation takes as operands two functions and returns a new function as a result. Mathematically, convolution is defined as:

$$c(t) = \int_{-\infty}^{\infty} f(x)g(t-x)dx,$$

where c is a function returned by convolution operation and t is a point in which function c is evaluated. c is defined as an integral over two other functions: f is often called the input, while g is often reffered to as a weighting function.

Convolution might be seen as a measure how well two functions fit. Function g is a pattern that moves along function f and it each place it measures how well the two functions fit. If the similarity is high, the returned value is high as well.

Convolution for neural networks

It is worth considering how this equation can be applied to neural networks. Since representing real values in the computers is rather troublesome, it is desirable to express this equation in a discrete form, which is shown below:

$$\overline{c}(t) = \sum_{x=-\infty}^{\infty} \overline{f}(x)\overline{g}(t-x)$$

If we want to apply this equation to neural networks we might want to think of $\overline{f}$ as a data sample. In such approach $\overline{g}$ would become a matrix that is moving around the data sample producing a single value in each place. From now on, we will call f an input, g a filter or a convolution matrix or a convolution window and c an output.

Usually, the convolution window is much smaller then the input and convolution is appplied multiple times. Each time a submatrix of input is multiplied by the convolution window. The result of this operation is a scalar, and a result of a whole process is a matrix. Each layer of neural network might include multiple filters and thus each layer might produce an output of higher dimensionality than the provided input - the additional dimension is produced due to using multiple filters.

It is worth noting, that there are different filters in each layer. Filters in the next layer work on the features extracted from the previous layers. Therefore, the later the layer is in the neural network, the more complicated patterns it may discover.

Size and stride, spatial invariance

It is worth taking a closer glance at how convolution works in details. First, we have to define two important parameters of the convolution windows - their size and stride. The size is simply the size of the convolution matrix. The stride defines which part of the input will be convoluted next with respect to the part of the input that is currently being convoluted. This concept is shown in figure 2.3.

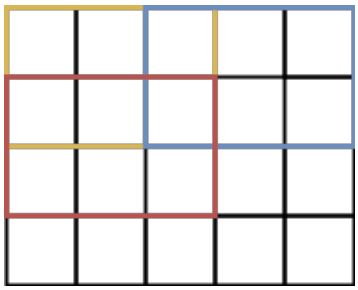

Figure 2.3: The figure shows three consecutive positions of a convolution window - yellow, blue and red. The input is of size 4×5, the convolution window has size 2×3 and stride 1×2, which means the window can move two elements to the right or one to the bottom.

Three consecutive positions of a convolution window are depicted. Each position defines which submatrix of the input will be multiplied by the convolution matrix. Each multiplication will produce a single value - these values when combined will produce an output matrix.

It should be noted that each time the convolution matrix is the same, only the part of the input that is being convoluted changes. Each filter is specialised in finding a specific pattern - because it is convoluted with many submatrices of the input it will find that pattern regardless of where the patter is on the input matrix. This property is called spatial invariance. Figure 2.4 shows this concept more clearly.

As stated above, in each layer usually there are plenty of convolution matrices - each is specialised in finding a specific pattern. Each pattern might be seen as a useful feature of the data and convolution provides us with some sort of map that shows where in the input this feature exists and where it does not. Therefore, it is often useful to see convolution windows as filters or feature extractors.

Data size reduction due to convolution operation

Let the input be an $I \times J$ matrix and the convolution filter a $K \times L$ matrix with a 1×1 stride. Therefore, the first submatrix to multiply by the convolution filter will be $[1 : K] \times [1 : L]$ and it will return a single value. The last submatrix will be $[I - K + 1 : I] \times [J - L + 1 : J]$ and it will also produce a single value. As a result the output will be a $[I - K + 1] \times [J - L + 1]$ matrix. This process

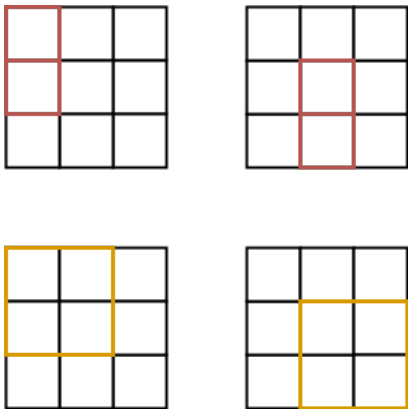

Figure 2.4: In the upper row two locations of a certain feature are shown in red. In the bottom row the location of a convolution window that will discover this feature is shown in yellow. As can be observed, by moving the convolution window it is possible to extract a certain feature regardless of its location.

is shown on picture 2.5.

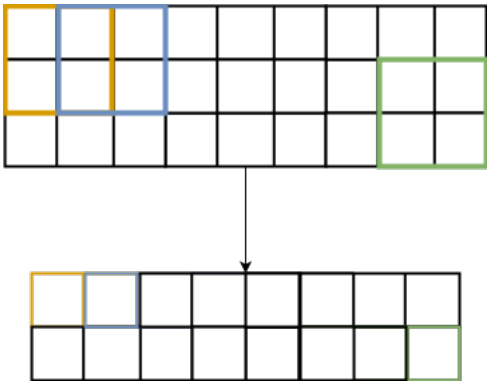

Figure 2.5: In this picture $I = 3, J = 9, K$ and $L = 2$. First submatrix and the output it produces are shown in yellow, second are shown in blue and the last are shown in green. The dimensionality reduction might be observed.

It can be observed that the values laying close to the edge of the input matrix will be underrepresented. The upper left corner of the input matrix will have influence on only one element of the output (the yellow one) while its right neighbour will already have influence on two elements of the output (the yellow and the blue one). The elements in the middle of the input matrix will influence four elements of the output. Such unequalities of the influence might be undesirable. There is a variety of ways to address this problem, e.g. zero-padding, but we will not cover them. More information about such methods can be found in [1].

2.2.2 Activation function

There are few possible activation functions that might be used in convolution neural networks [3].

- the sigmoid activation function $\sigma(x) = \frac{1}{1+e^{-x}}$
- the hyperbolic tangent activation function $tanh(x) = \frac{e^{2x}-1}{e^{2x}+1}$
- the rectifier linear function [4] $rect(x) = max(0, x)$
- the maxout activation function

$$maxout_i(x) = \max_{j\in[1,k]} z_{ij}$$
$$z_{ij} = x^T W_{...ij} + b_{ij}$$

2.2.3 Pooling

Pooling is an operation, usually applied to a matrix, that takes as an input multiple values and returns a single value describing the input. Typical pooling functions are [3]:

- max pooling - the max value of input is returned
- average pooling - the average value of input is returned
- stochastic max pooling - one element is chosen from the input to become the result. Probability of choosing an element is proportional to its value. [7]

Pooling is defined not only by its type but also by its size and stride. The size of pooling defines how many values will be taken as an input - the bigger the size of pooling, the more information is accumulated in a single value. The pooling stride defines where will be the next submatrix with respect to its present location. Fig 2.6 illustrates this concept.

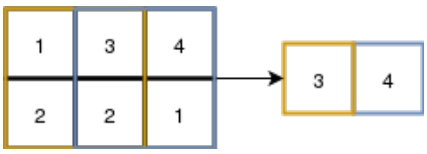

Figure 2.6: The result of applying the max pooling of size 2×2 with stride 1×1 to the input. Yellow square is the first pooling window, blue square is the second one. The values of the output are coloured accordingly.

When pooling is applied on the edges of the input same problems might be encountered as with the convolutions, namely some values will be underrepresented. It is worth noting that the output of the pooling is off smaller size than the input. To address these problems same techniques might be applied.

2.2.4 Summary

In the convolutional subnetwork each layer applies three operations to the input, namely: convolution, which provides us with spatial invariance, activation function and pooling. As a result of this processing, the features are extracted from the data. These features are then used by the classifying subnetwork.

Convolution provides us with spatial invariance and is responsible for extracting features. Each layer of the convolutional part of the network has its own filters. These filters take advantage of the features that were already extracted in the previous layers. Therefore, the later in the network, the more complicated pattern the filter may discover. It is worth noting that the patterns discovered by the filter are local - the filters are smaller than the input and find patterns that are also smaller than the input, which may appear few times in each sample.

Pooling ***

2.3 Learning Algorithm

Even though convolution operation and pooling might seem to introduce a lot of changes, the whole procedure remains a supervised learning problem and therefore, the classical algorithms for learning neural networks, such as stochastic gradient decent, might be used.

2.4 Other properties of convolutional neural networks

2.4.1 Fast computation

Convolutional neural networks have properties that make the computation fast, namely *parameter sharing* and *sparse interactions* [1]. Both these properties are connected to using convolution.

Parameter sharing means that the same weight is used multiple times during calculating the activation for a sample. Convolution matrices are much smaller than the input and are moved around it, so each time the same set of parameters is used to compute the output. Thanks to that, there is no need to have multiple sets of parameters that would discover the same feature in multiple places of the input - it is enough to have one filter that moves around that input. Because of that, the number of parameters is greatly reduced, and therefore the learning process speeds up.

In a typical neural network each element of input has influence on each element of the output which means that the interactions are dense. In convolutional neural networks the situation is different - the filters are much smaller than the input and discover local patterns, what leads to sparse interactions. This property again causes a reduction in the number of parameters in the

model and makes the learning process much faster.

2.4.2 equivariant representations

Equivariance is another interesting property caused by convolution. Equivariance means that if the input changes than output changes the same way. Mathematically it can be written as: $f(g(x)) = g(f(x))$. The intuition that stands behind it is: detecting feature in a particuler place - feature elsewhere - we find it elsewhere. To what types of transformation is convolution equivariant and to which transformations it isn‘t?

Chapter 3

The Model

3.1 Goal

The goal of this work was to build a model that will well perform the task of classification of the provided data. To complete this task multiple obstacles had to be overcomed, i.e. small data size, missing labels, a big number of hyperparameters that had to be adjusted.

3.2 Data

The dataset describes interactions between a single protein and multiple ligands. One might choose to see the dataset as a four dimensional matrix with axes dimensions described by: the number of ligands, length of the protein, 6 standard pharmacophore features of ligand and 9 types of interactions with amino acid[2]. One data sample can be seen as a 3-dimensional matrix that describes how a protein bounds with a specific ligand. The 3 dimensions are: the length of the protein (number of its residues), 6 standard pharmacophore features and 9 types of interactions with amino acid. A single data sample is presented on figure 3.1.

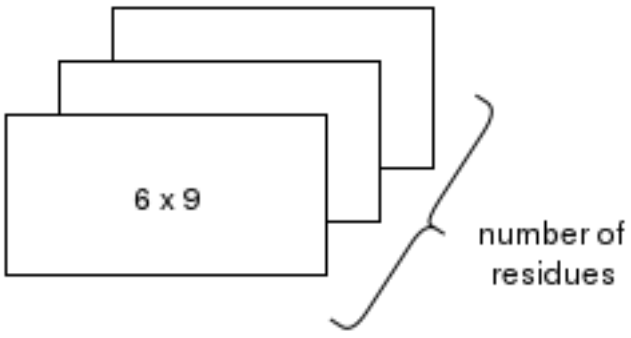

Figure 3.1: A single data sample.

The 6 pharmacophore features are: hydrogen bond acceptor, hydrogen bond donor, hydrophobic, negatively charged group, positively charged group, aromatic. The 9 types of interactions with amino acid are: any, with a backbone, interaction with sidechain, polar, hydrophobic, hydrogen bond acceptor, hydrogen bond donor, charged interaction, aromatic.

Even though it might be intuitive to look at this data as if it were 4-dimensional, it was stored in the memory in a 3-dimensional form by placing the 6 x 9 matrices adjacent to each other. If we had a protein with only three residues, name them A, B and C, a single sample would look like on figure 3.2.

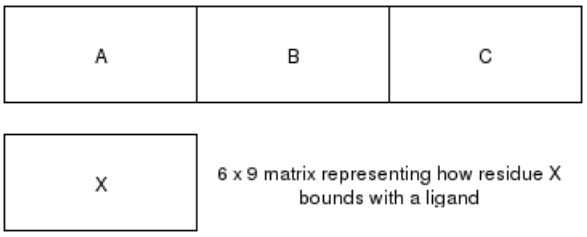

Figure 3.2: Data stored in the memory

The values constituting the dataset are discrete numbers in range 0 to 9. 0 means there is no interaction of specific kind. 1 and 2? The data was very sparse - more than 99% of all values were zeros.

The dataset was stored in 3 files - each file contained samples of only one type: active, inactive, middle (not labled). Out of 5844 samples 2655 were labeled as active, 1945 was labeled as inactive and there were also 1244 unlabeled examples.

3.2.1 Data preprocessing

In order to extract most promising features, the data has been preprocessed. Our goal was to enable the model building such patterns that could detect whether a bound of a specific kind was present in both adjacent residues. We expect that such approach might lead to discovering of interesting correlations.

To achieve such a form of the dataset that would enable this approach, the dataset was extended in the following way:

1. three copies of the dataset have been created.
2. Each copy was put just below the previous one.
3. Each copy was shifted in such a way that going from top to bottom and from left to right would preserve the order of the residues. The shift forced us to either complement each row with zeros or to cut off the residues that would stick out.

4. We decided not to cut off any residues in order to have each of them the same number of times in the dataset - this way no residue will be underrepresented.

As a result each sample had 18 instead of 6 rows and 18 columns more.

The schema of this approach is shown on figure 3.3 which depicts the simple example of a protein with only three residues. It can be observed that a convolution window broader than 9 would be able to detect whether an interaction of some type is present in both adjacent residues while convolution window higher than 6 would be able to detect if two adjacent residues have same pharmacophore features.

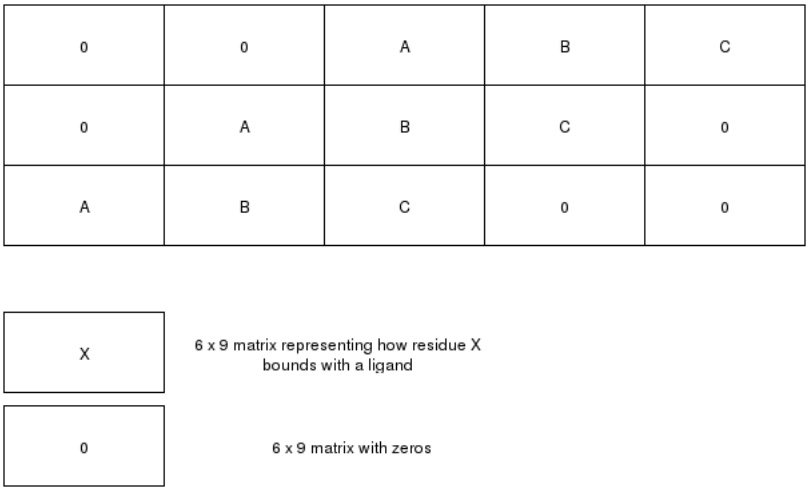

Figure 3.3: Data after preprocessing.

3.3 The Architecture

In this section we will describe the type of architecture which we used for experiments. All the models we have trained were convolutional neural networks with one or two convolutional rectified linear layers (those are layers that use as activation function the rectifier linear function, see 2.2.2). Each layer had 16 or 32 output channels that correspond to number of filters created in each layer. The number of layers have been chosen in such a way that the learning process will not take too much time and the data size will not be reduced too much. Rectifier activation function was used because of its eligible properties [3].

The convolution window's values were chosen in a way that would enable the model to find filters catching correlation between same types of interaction in two adjacent residues, what was described in section 3.2.1.

The convolution windows were of size $(width, height) \in \{6, 8, 10, 12\} \times \{4, 5, 6, 7, 8\}$.The convolution window's strides were of size $(width, height) \in$

$\{2, 4, 6\} \times \{2, 3\}$.

If both convolution window size and stride had big values in the first layer, it could happen that the data size in the second layer would be too small to satisfy the conditions above. In such cases the convolution window's size and stride in the second layer were reduced in such a way that the convolution window's size would always be smaller than the data size. Moreover, the convolution window's stride would always be smaller than the convolution window and, if only it was possible (i.e. all dimensions of the convolution window were smaller then the corresponding dimensions of the data), small enough to enable existence of at least two "windows" in each dimension.

The shape of pooling windows was (1, 1), (2, 1) or (2, 2) and smaller by at least one than the data size in each dimension so moving the pooling window was always possible. Pooling stride was always equal to or smaller by half than the pooling window in each dimension. Max pooling was used.

The last layer of the network was a softmax layer with two neurons. It was used to classify the sample based on features extracted by the convolutional part of the network.

3.3.1 Finding best architecture

Due to many hyperparameters, there exist many models that fulfill our architecture restrictions and therefore it is not possible to train and measure the performance of all possible architectures. To find the best one we used the tree of Parzen estimators algorithm (see A) provided by a Python library - Hyperopt [6] and let it sample 20 models.

Each architecture that was tested was chosen by hyperopt module. Based on the performance of the already tested architectures hyperopt was choosing another one. Each architecture was passed from hyperopt to the function responsible for measuring the performance of the model. We will call this function an objective function.

Objective function for hyperopt

Each time the objective function created five models of a given architecture and then trained and measured the performance of each. Cross validation procedure was used to obtain different training data for each model. Validation and test set included only labeled examples because classifying unlabeled examples would not be possible. All the unlabeled samples were added only to the training set. The cross validation proceeded as follows: the active and inactive samples were split into five parts of even size. One part became the validation set, one part the test set and the other three parts along with all the unlabeled examples became the training set. The whole procedure was repeated five times.

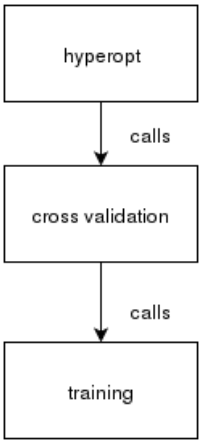

Figure 3.4: The control flow of the experiments.

Each time after training the model its performance on testing data was measured and stored. At the end the mean value of scores of all five models was returned to the hyperopt module. The score used for measuring the performance of the model was receiver operating characteristic (ROC) with Youden's J statistic. We will cover this topic in details later in this section. Algorithm 1 shows pseudo code for the cross validation procedure.

Algorithm 1 Cross validation

```
1:  procedure OBJECTIVE_FUNC(sample, data_labeled, data_unlabeled)
2:
3:      scores ← empty list
4:
5:      for k ∈ [0...4] do
6:          train_set, validation_set, test_set ← split(data_labeled, k)
7:          train_set ← train_set + data_unlabeled
8:          model ← build_model(sample)
9:          model ← train(model, train_set, validation_set)
10:         score ← measure_performance(model, test_set)
11:         scores.append(score)
12:
13:     return mean(scores)
```

Training the model

The model provided by the hyperopt module was built five times and every time it was trained on another part of the data which was obtained using the cross validation procedure. After each epoch of training the optimal threshold was calculated on the validation set. All samples, for which activation value was above the threshold, were then classified as active samples and those, for which activation value was below the threshold, were classified as inactive. Keep in

mind, that there were no unlabeled examples in the validation set.

In order to compute the optimal threshold, the receiver operating characteristic (ROC) procedure was used (see A for more information). To measure the quality of the threshold, the Youden's J statistic [5] (see A for more information) was used. Afterwards, the model's performance on validation data was measured. The score was the Youden's score. If the performance for this model was best until this point of time, the whole model (i.e. all its parameters along with the computed threshold) was dumped to hard drive for later reference. As a result, at the end of the learning process the model's best version was remembered and could be read in order to measure its performance on the testing set and append its score to the list in the cross validation procedure. The threshold calculated during the learning phase was used. The score to measure the performance of the model was the Youden's score.

The details of the learning algorithm are covered in section 3.4.

3.4 The Learning Algorithm

The provided data included unlabeled examples. Two approaches that would enable using these examples to training the model were tested.

3.4.1 Naïve Approach

Training set was constructed in the following way: all examples were included in the training set two times - the labeled samples were included with their label and the unlabeled examples were included once with positive label, and once with negative label. This way the impact on classification of unlabeled data was minimised while the unalabeled data could have been used to improve parameters in the convolutional part of the model.

Example:

If there were the following samples: [A, B, C] along with the following labels: [act, inact, unlabeled] then the training set would look like this: [A, A, B, B, C, C] and the labels would be [act, act, inact, inact, act, inact].

For this approach the stochastic gradient descent algorithm included in pylearn2 package (include version) was used.

3.4.2 Fancy Approach

In this approach each sample was included in the dataset only once. In order to train the model, a variation of stochastic gradient descent algorithm was written. This enabled using unlabeled examples during the learning process. The SGD implementation provided in pylearn2 (version here) was used as a base. Major changes were introduced in the training function in such a way that the

unlabeled examples were used to adjust the parameters of the convolutional part of the model only and had no impact on the classification part. The pseudo-code of this algorithm can be found below as Algorithm 2.

Algorithm 2 Learning

```
1:  procedure TRAIN(sample, label)
2:      if sample is unclassified then
3:          parameters_on_enter ← current_parameters
4:
5:          SGD(sample, inactive)
6:          diff_vec_1 ← current_parameters − parameters_on_enter
7:          current_parameters ← parameters_on_enter
8:
9:          SGD(sample, active)
10:         diff_vec_2 ← current_parameters − parameters_on_enter
11:         current_parameters ← parameters_on_enter
12:
13:         update_vector = new vector of length same to difference vectors
14:         for el1, el2, up_el ∈ zip(diff_vec_1, diff_vec_2, update_vec) do
15:             if sign(el1) == sign(el2) then
16:                 up_el ← combination_function(el1, el2)
17:             else
18:                 up_el ← 0
19:
20:         for up_el ∈ update_vec do
21:             if up_el is responsible for updating the classification part then
22:                 up_el ← 0
23:
24:         current_parameters ← parameters_on_enter + update_vector
25:     else
26:         SGD(sample, label)
27:
```

For labeled samples the learning process was performed with no changes. When the sample was unlabeled the network parameters were stored and then the sample was presented to the network as if it was labeled as inactive. During this process the network parameters were updated. The difference in the network parameters was stored and old parameters were restored. Afterwards, the sample was presented to the network again - this time as an active sample. The procedure was the same as before. After calculating the difference and restoring the old parameters the two vectors of differences were compared to produce the final vector of updates.

The final vector had the following properties:

1. the elements responsible for updating the classification part of the network were all zeros, therefore the unlabeled examples had only impact on learning parameter of the convolutional subnetwork and did not influence

the classification part of the network.

2. the elements responsible for updating the convolutional part of the model were calculated in the following way:

   - if the corresponding elements of the two vectors had the opposite sign, then the corresponding element in the final vector was zero. As a result, the unlabeled samples were used by the network to learn only these filters that were useful for classifying samples of both classes
   - if the corresponding elements in both vectors had the same sign, then the corresponding element in the final vector was calculated using the values of the two elements. The final value could be:
     - minimum by absolute value of the two elements
     - maximum by absolute value of the two elements
     - mean of the two elements
     - softmax mean of the two elements, i.e. having $x, y \in \mathbb{R}$ the softmax mean σ is equal to $x \cdot \frac{e^x}{e^x+e^y} + y \cdot \frac{e^y}{e^x+e^y}$.

       Remark

       It can be observed that $\frac{e^x}{e^x+e^y} \in [0, 1]$ for any x, $y \in \mathbb{R}$ and that $\frac{e^x}{e^x+e^y} + \frac{e^y}{e^x+e^y} = 1$, therefore $\sigma = x \cdot \frac{e^x}{e^x+e^y} + y \cdot \frac{e^y}{e^x+e^y}$ is a convex combination of x and y, so σ will be between x and y.

Concluding, the update vector had zeros in part responsible for classification. If two corresponding values in the vectors of differences had opposite sign, then the corresponding value of the update vector was zero. All other elements were calculated using one of the combination functions.

Example

Let $[+2, +5, +1, -3, +5, +7]$ and $[-2, +3, -1, -7, -7, +7]$ be the vectors of differences, elements 1 to 4 were responsible for updating the convolutional part, elements 5 and 6 were responsible for updating the classifying part of the network and the combination function used was minimum, then the final vector would be $[0, 3, 0, -3, 0, 0]$. Elements 5 and 6 are zeros because they are responsible for updating the classification part of the network. Elements 1 and 3 are zeros because the corresponding values in two vectors have opposite signs. Elements 2 and 4 are minimums by absolute value of the two corresponding values. This example is ilustrated in figure 3.5.

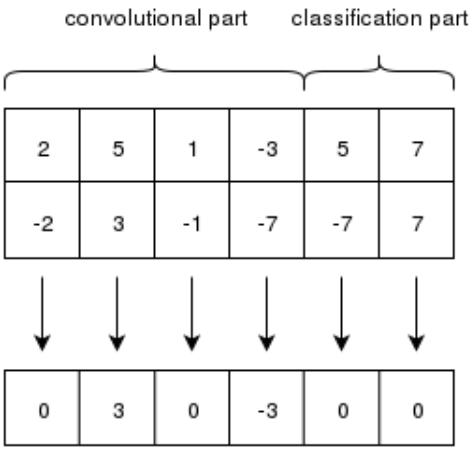

Figure 3.5: Example of the min combining function

Chapter 4

Results

4.1 not yet

Chapter 5

Discussion

5.1 napisać co się udało

5.2 Future work

There are still many ways to improve the proposed method.

It is important to investigate if representing the data in the memory in a 4-dimensional instead of 3-dimensional form can improve the performance of the model (see 3.2 for details). Using another form of representing the data in the memory might lead to discovering other patterns which can be more relevant for this data.

Moreover, new combination functions might be tested, e. g. ...

Further, it is interesting to explore new methods of finding the optimal threshold for classification. In particular, we would like to investigate the approach which would enable finding two thresholds. Note that the model that created the dataset also used two thresholds so the new approach will lead to a greater resemblance (see the paragraph that doesn't yet exist for more information).

The two-threshold model will leave some samples unlabeled. It is important to explore what error functions can be used in such approach.

During the experiments we realised that creating three copies of the data during preprocessing (see 3.2 for details) might be redundant. We want to inspect if having only two copies would cause a big difference in performance. Reducing the size of data will also lead to shorter time of learning pahse.

Finally, there exist some techniques for deep and convolutional neural networks that might further improve the performance of our model, i.e dropout and dropconnect methods [3].

Appendices

Appendix A

Dictionary

from: http://optunity.readthedocs.org/en/latest/user/solvers/TPE.html

The Tree-structured Parzen Estimator (TPE) is a sequential model-based optimization (SMBO) approach. SMBO methods sequentially construct models to approximate the performance of hyperparameters based on historical measurements, and then subsequently choose new hyperparameters to test based on this model.

Receiver Operating Characteristic (ROC) measures ratio of false positive rate (FPR) to true positive rate (TPR). $FPR = \frac{FP}{FP+TN}$, where FP is the number of false positive examples and TN is the number of true negative examples. $TPR = \frac{TP}{TP+FN}$, where TP is the number of true positive examples and FN is the number of false negative examples. When the threshold defining the classification moves the way the samples are classified changes. The goal is to find such a threshold that the score will be most optimal. There are plenty of ways for measuring the score. *Youden J Statistic* is one of such methods. It measures the difference between the point on ROC curve and corresponding point on ROC curve for random classifier. The picture **??** shows this concept.

Bibliography

[1] Yoshua Bengio, Ian J. Goodfellow, Aaron Courville - *Deep Learning*

[2] Stefan Mordalski, Igor Podolak, Andrzej J. Bojarski - *2D SIFt - a matrix of ligand-receptor interactions*

[3] Joost van Doorn - *Analysis of Deep Convolutional Neural Network Architectures*

[4] Yoshua Bengio, Antoine Bordes, Xavier Glorot - *Deep Sparse Rectifier Neural Networks*

[5] William John Youden - *Index for rating diagnostic tests*

[6] James Bergstra, Dan Yamins, David D. Cox - *Hyperopt: A Python Library for Optimizing the Hyperparameters of Machine Learning Algorithms*

[7] M. D. Zeiler, R. Fergus - *Stochastic pooling for regularization of deep convolutional neural networks*

Chapter 6

Irrelevant

6.1 zero-pad methods in detail

The easiest way is to let these values stay underrepresented (in MATLAB *citation?* this methodology is called valid), another one is to enlarge the input matrix by adding zeros *at the edges* - this is called zero-pad. One can either add enough zeros for each element of the original matrix to be convoluted exactly the same number of times (in MATLAB *citation?* this methodology is called full) or take only enough zeros for the output matrix to have the same size as the input matrix (in MATLAB *citation?* this methodology is called same).

{obrazek} ilustrujący te przykłady

One can question *legitimacy* of such approach. Adding zeros invites new information into the matrix and might cause additional noise. Instead of adding zeros one might try to change a matrix into torus or instead of zeros use the values that already are present in the original matrix. The added values might be symmetrical *lustrzane odbicie.*

{obrazek} ilustrujący te przykłady

The problems with a classical backpropagation

diminishing gradient flow, niedouczanie się, przeuczanie się, obczaić co o tym mówił Larochelle, on to chyba jednak mówił o głębokich. Wtedy to i tak napisać i przerzucić do głębokich.

Diminishing gradient flow

co to jest, skąd się bierze, można się wesprzeć wykładami Larochelle, on poleca dużo paperów zawsze.

Backpropagation

See (Goodfellow, 2010) from Bengio