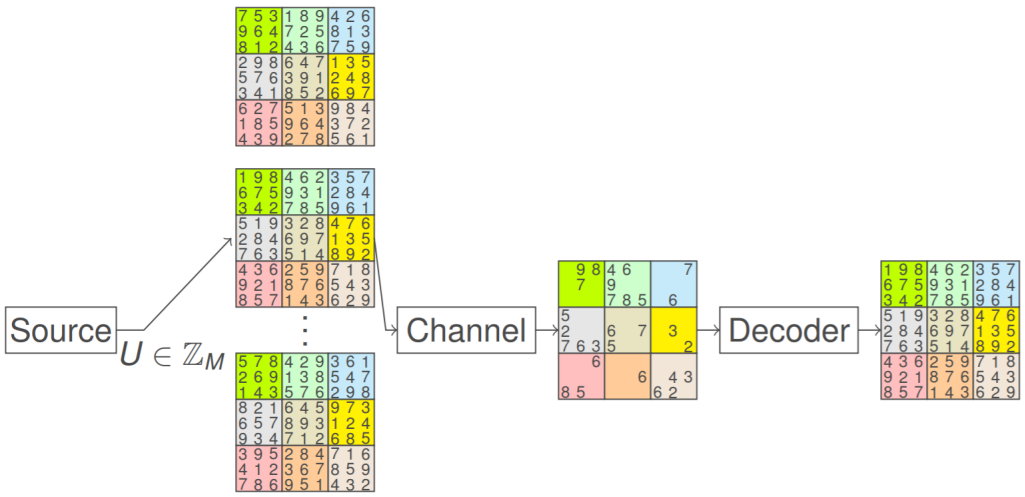
University of Stuttgart
Germany

Institute of Telecommunications
Professor Dr.-Ing. Stephan ten Brink

Study Thesis

Neural Network-based decoding of Non-linear Codes

Deepthi Sreenivasaiah



Date of hand out: May 15, 2018
Date of hand in: July 19, 2018
Supervisor: Sebastian Cammerer
Dr.-Ing. Jakob Hoydis
Prof. Stephan ten Brink

Abstract

Sudoku codes are non-linear codes with some constraints. This report discusses the idea of using neural networks for decoding the sudoku codes. Two approaches along with their results are presented. Although a simple convolution neural network was found to be not very effective in spite of the high computational cost, a deeper residual inception network which combines the benefits of inception module and the shortcut connection was observed to improve the performance to a larger extent. This residual connection improves the training speed greatly and also overcomes the problem of vanishing gradient. The inception module further boosts the accuracy without adding more parameters into the model.

Title page image: Transmission and Reception of Suduko codes

Contents

Acronyms

LDPC	Low-Density Parity Check
BP	Belief Propagation
NN	Neural Networks
ReLU	Rectifier Linear Unit
CNN	Convolution Neural Networks
GPU	Graphics Processing Unit

Notations

x	input to the first layer
$F(x)$	Residual function
$H(x)$	Original function

1 Introduction

Sudoku puzzle is a grid of 9x9 squares to be filled in with numbers from 1 to 9. It is required to fill each empty box in such a way that each row, each column, each 3x3 square gets a digit from 1 to 9 exactly once. Sudoku codes can be viewed as non-linear, forward error correcting codes that are based on rules of sudoku puzzles. It can be thought of as a model wherein the transmitter sends the sequence of codes from a solved sudoku puzzle, to an erasure channel as shown in Figure1.1. Due to the noise present in the channel, some of the symbols may be erased or corrupted and the receiver might receive the code with partially filled symbols. The task of the decoder present at the receiver is to fill the missing entries using the constraints of a sudoku puzzle.

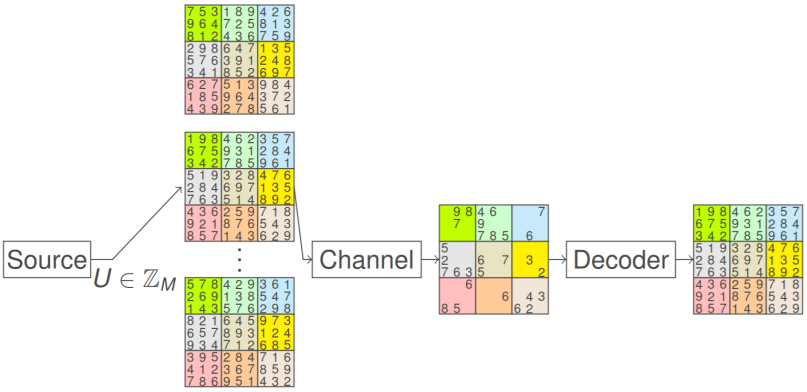


Figure 1.1: Transmission and Reception of Suduko codes [1]

One of the methods to decode sudoku codes is by using belief propagation in which sudoku is modeled as a probabilistic graphical model [1]. Structurally, solving a Low-Density Parity Check (LDPC) codes is similar to solving a Sudoku puzzle. In belief propagation, sudoku is represented using Tanner graph. Nodes in the Tanner graph send information regarding probabilities about the nodes to each other. There are three types of constraints: Each number should appear once in each row, once in each column and once in every 3x3 squares. These constraints are mapped to constraint nodes in the graph.

Considering 4x4 sudoku for a better understanding of Belief Propagation (BP) algorithm, the row constraint node (in green) connects to a number in every row of the sudoku as shown in Figure1.2. The column constraint node (in blue) connects to a number in each column of the sudoku and similarly, the 2x2 square constraint (in yellow) connects to every number in a 2x2 square. A partially complete sudoku puzzle is transformed into cell nodes in tanner graph.

The cell node sends a message about the probabilities of the various cell contents, given information about the constraints associated with that cell and a constraint node sends a message about the probability that the constraint is satisfied, which it computes using information from the cell nodes about the probabilities of the cell contents [2]. This process is repeated until all the cell nodes are filled with the correct number or until some maximum number of iterations. After every iteration, the graph is unrolled by providing the output of constraint nodes as an input to cell nodes in the next stage.

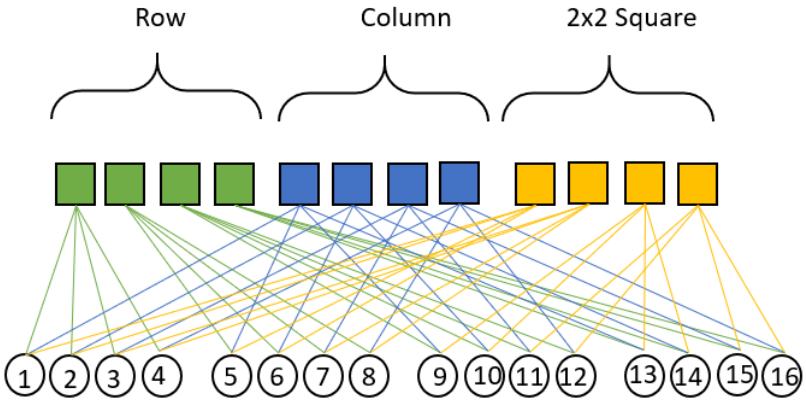


Figure 1.2: Belief propagation for 4X4 Sudoku

1.1 Decoding using Neural networks

The idea of using Neural Networks (NN) for decoding is inspired by the belief propagation algorithm for LDPC codes. A NN consists of many neurons which are connected to a set of neurons in the next layer and weights are assigned to them. In each layer, these weights are added up along with an optional bias, and the result is propagated through a non-linear activation function like sigmoid or Rectifier Linear Unit (ReLU). To find the optimal weights, a set of known input-output mapping is required and a loss function must be defined. By using gradient descent optimization techniques and backpropagation algorithm, optimal weights of each neuron can be computed. In order to generalize the NN, another dataset called validation set must be considered on which the loss is computed to further optimize the weights. NN are able to learn the mapping between noisy input patterns and codewords during the learning process. The output nodes in the final layer represent the bits of the codeword or, one-hot coding can be used where one output node corresponds to one unique codeword. At the transmitter, k message bits are encoded into N bit length codewords. They are modulated and transmitted over a noisy channel. At the receiver, the NN-based decoder should be able to recover the message bits from the noisy codewords.

Neural networks can be used for only shortcodes since as the length of the code increases, the number of possible codewords also increase and it is not possible to fully train a neural

network using all the codewords. For example, for length N=100 and rate r=0.5, 2^{50} codewords exist and it is difficult to use all the codewords to train a NN [3]. In this report, solving sudoku puzzle which is similar to decoding non-linear codes using different neural network architectures will be discussed and the results will be compared. One million dataset of sudoku puzzles with solutions are considered. Keras, a neural network library as a high-level front-end for TensorFlow is used to implement the NN model.

2 Decoding sudoku codes using neural network

In this section, two different approaches used to decode sudoku codes are discussed in detail. The first approach uses a simple Convolution Neural Networks (CNN) and the second solution uses a residual network with inception architecture. One million Sudoku puzzles with corresponding solutions are imported as NumPy arrays. Empty locations in the solution are filled with zeros. Solutions are represented as one-hot codes where given each number is converted into 10-bit wide one-hot code representation. In both the approaches, the output layer has 10 neurons with Softmax as the activation function which is used to represent a probability distribution over 9 different outcomes (1 to 9). Adam optimizer algorithm is used to update the weights during backpropagation which calculates an exponential moving average of the gradients. The loss function used is categorical cross entropy since this is a multiclass classification problem.

2.1 Convolution neural network

Convolution neural networks are inspired by the brain. Convolution layer is one of the main building blocks of CNN. Convolution is a mathematical operation performed on two functions to produce a third function [4]. It combines two sets of information. Each neuron in the Conv layer receives only some inputs and performs a dot product called convolution, followed by a non-linear function. In contrast to dense NN, the neurons in each layer of CNN will only be connected to a small region of the layer before it. CNN are usually used in image classification where the input is a 2D image in case of grey-scale images and 3D if it is an RGB image.

Conv layer is the core layer in CNN since it performs most of the heavy computations. The Conv layer's parameters consist of a set of learnable filters. Every filter connects to a small spatial area of the input which is called filter dimension. During the forward propagation, each filter is moved across the width and height of the input volume and dot product is computed between the entries of the filter and the input at any position producing a 2-dimensional activation map that gives the responses of that filter at every spatial position. The output dimensions of each Conv layer are decided by three hyperparameters namely depth, stride and zero-padding. The depth of the output corresponds to the number of filters used in Conv layer. Each filter looks for something different in the input. Stride gives the amount by which a filter is shifted. A stride of 1 moves the filter one position at a time. Zero-padding is used to maintain the same spatial dimensions of the output as that of the input.

```
#################################################################################
#Parameters
optimizer=optimizers.Adam(lr=0.0001)
#################################################################################
input_shape = (9,9,1)
blocks=3
#define conv net
model = Sequential()
model.add(Conv2D(512, kernel_size=(3, 3), strides=(1, 1),padding='same',activation='relu',
                 kernel_initializer='glorot_uniform',input_shape=input_shape))
for j in range(blocks):
    model.add(Conv2D(512, kernel_size=(1,9), padding='same', activation='relu'))
for j in range(blocks):
    model.add(Conv2D(512, kernel_size=(9,1), padding='same', activation='relu'))


model.add(Conv2D(10, kernel_size=1,activation='softmax'))

#compile model
model.compile(optimizer=optimizer, loss=losses.categorical_crossentropy)
model.summary()
```

Figure 2.1: Convolution neural network model

A NN model with convolution layers consists of a 2D convolution layer since the sudoku codes are arranged as a 9x9 array. Conv2D layer in Keras is used to create a sequential model as shown in Figure2.1. The input Conv layer has 512 filters with a dimension of 3x3 which corresponds to the 3x3 square constraint in the sudoku code. Stride of 1 is used as the filter should convolve with each and every number. Since output dimensions should also be 9x9, the padding value of 'same' is added to preserve the output volume. CNN is initialized by random weights drawn from the Gaussian distribution. Hence Glorot initializer in Keras is used in the input layer. Following this are the two blocks of Conv layers stacked one after the other. Both the blocks have the same hyperparameters as the input layer except for the filter dimensions. The first block has a kernel size of 1x9 corresponding to the row constraints while the second block contains kernel size of 9x1 depicting the column constraints. The idea of adding three different filter dimensions is to simply imitate the three constraint nodes of the BP algorithm. The final output layer has 10 neurons corresponding to 10-bit wide one hot coded solutions with Softmax activation function which gives the probability scores for each number. To compile the model, Adam optimizer with a learning rate of 0.0001 is used. During training, the hyperparameters are tuned to various values to find the optimal solution which is able to predict more number of symbols in a given puzzle. The results are discussed in Chapter 3.

2.2 Residual network with Inception architecture

The problems of a deep network are solved using Resnets. The performance can be further increased using inception module with less computational cost.

2.2.1 Resnets

Deep NN extract different low and high-level features and, the level of features increases as the number of stacked layers increase. These additional layers better approximate the mapping

than its shallower counter-part and reduce the error significantly. But with deep NN, the problem of vanishing gradient is seen which affects convergence of loss function. When the NN is able to converge, the accuracy of the network becomes saturated and it decreases rapidly which results in higher training error. This problem can be addressed by using deep residual learning framework [5].

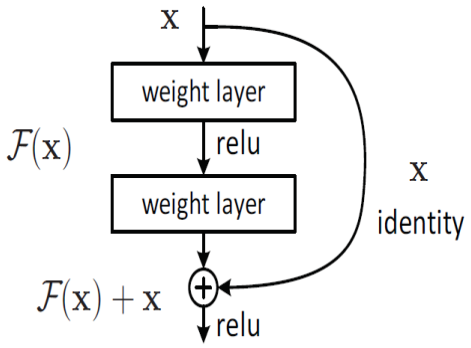


Figure 2.2: ResNet [5]

Instead of directly mapping the underlying layers using a function $H(x)$, a new mapping called residual mapping can be used to let the non-linear layers fit resulting in $F(x) = H(x) - x$ where $F(x)$ is the residual function and x denotes the input to the first layer as shown in Figure2.2. Hence, instead of approximating $H(x)$, the stacked layers now approximate $F(x)$ and the original function becomes $H(x) = F(x) + x$. It has been proved that optimizing the residual mapping is better than optimizing the original function. The formula $H(x) = F(x) + x$ can be realized using forward pass NN with a shortcut connection. The shortcut connections or skip connections simply skip one or more layers. These connections perform identity mapping. They neither add extra parameters nor computational complexity. The operation $F + x$ is performed by shortcut connection and element-wise addition. Non-linearity operation like ReLU is applied to the second layer after the addition. The residual networks can be adopted along with convolution layers to achieve better performance in terms of training speed.

2.2.2 Inception Module

The basic idea of inception module is to use many mini models inside a bigger module. Often there will be a confusion to decide on the type of convolution to use in each layer, whether 3x3, 5x5 or 1x1 convolution filter or max-pooling layers. Including all these layers will be beneficial to the modeling power of the network. The inception module combines all the different filters to be used allowing the model to pick the best. Hence the different convolutions are performed in parallel and the resulting feature map is concatenated before giving it as an input to the next layer, as seen in Figure2.3. In this way, the model will be able to extract both local features using smaller filters and high abstracted features using larger filters.

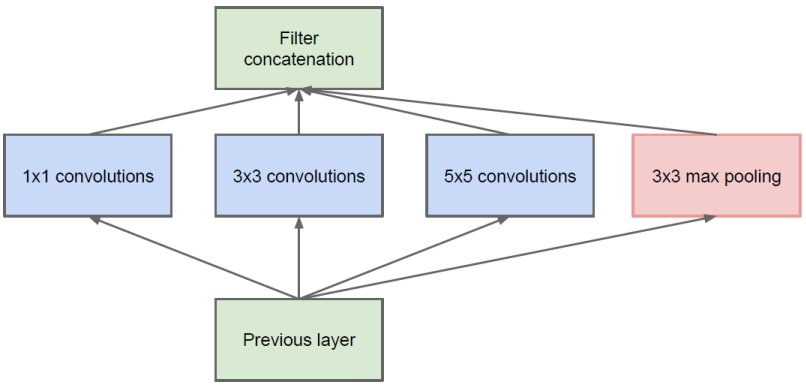


Figure 2.3: Inception module [6]

2.2.3 Residual Inception Network

The second approach uses Resnets to get a deeper network since they have proven to give better results when the depth of the model is increased. The inception module discussed in the previous section has been proven to improve the accuracy with less computational cost. Hence the second approach is inspired by Resnets and inception architecture and, combines them to give increased performance. The two variants of Residual inception networks along with the training methodology are as follows:

In the first variant, there are 8 blocks of inception modules. The input layer consists of a 3x3 convolution filter to depict the 3x3 square constraint of a sudoku puzzle. The input layer has 256 neurons which are divided among the two branches of the inception module into 128 neurons each. Each block in the network consists of 2 convolution branches with 1x9 and 9x1 filter, corresponding to the row and column constraints of a sudoku puzzle. The activation function used in these branches is linear. The resulting feature map is then concatenated which resembles the inception module. To this, a shortcut connection from the input layer is added. The non-linear activation ReLU is then applied after the addition. This forms one mini block of the network (Figure2.4). The output of each block is fed as an input to the branches of the next block.

Another variant of this network consists of an extra branch with 3x3 convolution (Figure2.5) and a total of 384 neurons in the input layer. Difference between the first variant is that the non-linear activation ReLU is applied in the three branches before concatenation and addition.

2.2.4 Training Methodology

Incremental training procedure is adopted to train the whole network having several blocks. In the first variant, two intermediate outputs are stemmed out. The output with 5 blocks is termed as the short model, while the output with 7 blocks is termed as the medium model and the final output is termed as long model. The total parameters with all three outputs sum up to 4.7M.

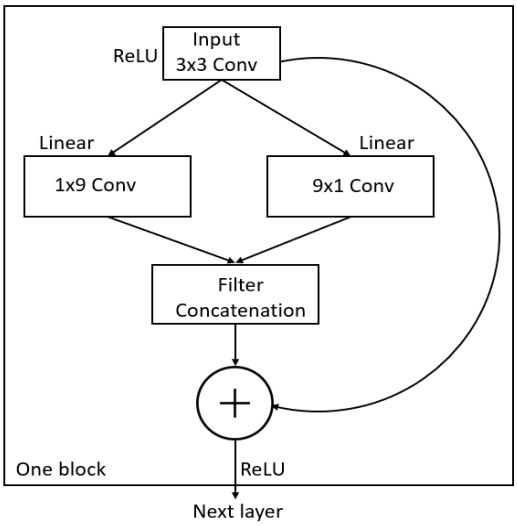


Figure 2.4: Representation of one block in Variant 1

In the second variant, the short and medium models are with 5 and 8 blocks respectively. The total parameters in the long model are 10.7M which is almost twice as that of the first variant.

In both the variants, the entire network is trained incrementally starting with short model followed by the medium one and finally the long model by selecting different values for hyperparameters like batch size, numbers of epochs. This kind of training methodology helps in training the deeper networks.

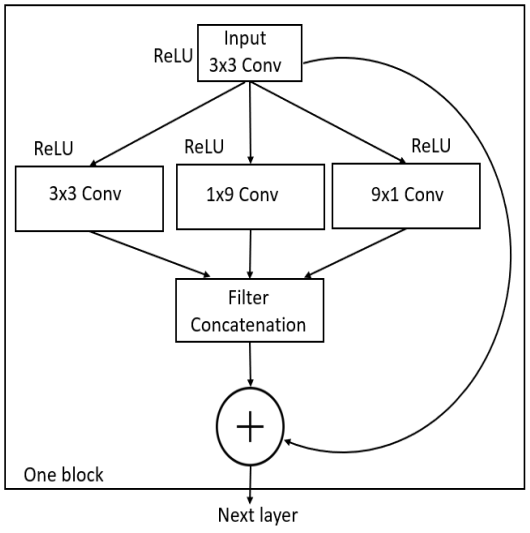


Figure 2.5: Representation of one block in Variant 2

3 Training outcome

Training of all the models was done on Nvidia GeForce Graphics Processing Unit (GPU). They were trained using random weights initialized by the Glorot initializer. Various hyper-parameters were tuned to obtain an optimal solution. The results of few CNN models are as follows:

Hyperparameters	Model 1	Model 2	Model 3	Model 4
No. of Conv layers	8	8	8	8
No. of filters	256	384	384	512
Kernel size	(3,3), (9,1), (1,9)	(3,3), (9,1), (1,9)	(3,3), (9,1), (1,9)	(3,3), (9,1), (1,9)
Learning rate	0.0001	0.0001	0.0001	0.0001
Batch size	256	256	256	256
No. of Epochs	20	30	30	30
Validation error	0.33	0.27	0.23	0.14
Average correct symbols (out of 81)	71	73	74.3	75.8
No. of completely solved Sudoku (out of 1000)	0	0	0	18

Table 3.1: Hyperparameters of CNN model

The observations are summarized as follows:

- Initially, the model was trained using the learning rate =0.001. It was observed after 15 epochs, the model stopped converging. Hence for all other models, the learning rate was set to 0.0001.
- The number of filters in each layer was incremented from 256 to 512. Model 1 with 256 filters was trained several times using different values of batch size and for various epochs. The minimum validation error achieved was 0.33 with 20 epochs. But with this, only 71 symbols out of 81 were corrected. Hence the number of filters was increased to 384 in each layer.
- With 384 filters in each Conv layer, the performance was slightly improved up to 76 symbols and a validation error of 0.23. But this model could not predict even a single Sudoku completely.

- However, when the number of filters was increased to 512 in each Conv layer, Model 4 was able to solve 18 Sudokus completely out of 1000. But this is very less in comparison to the number of parameters and it is computationally very expensive adding no improvement to the validation error.
- This shows the limitation of simple CNN in decoding the Sudoku codes.

The results of residual inception network are as follows:

Hyperparameters		Variant 1	Variant 2
No. of blocks		8	8
No. of branches		2	3
Kernel size		(3,3), (9,1), (1,9)	(3,3), (9,1), (1,9)
Intermediate outputs at		4, 6	4, 7
Total trainable parameters		4,730,910	10,692,884
No. of Epochs	Short model	10	10
	Medium model	20	1000
	Long model	40	100
Batch size		100 & 500	100 & 500
Validation error		0.025	0.035
Average correct symbols (out of 81)		80.27	80.02
No. of completely solved Sudoku (out of 1000)		620	512

Table 3.2: Hyperparameters of Residual Inception Network

The observations are summarized as follows:

- The second variant was trained using hyperparameters as shown in the table 3.2. After the complete training of the model for 12 hours, the model solved 512 Sudoku out of 1000. Hence the accuracy of the second variant is 51.2%.
- The first variant was trained starting with the short model for 5 epochs. This reduced the validation error to 0.3. After this, the medium model was trained for 10 epochs with a batch size of 100 and then for 10 epochs in batches of 500. The validation error drastically reduced to 0.025 after the training of the long model for 20 epochs.
- After the entire training of the network, it predicted 345 Sudoku completely increasing the accuracy to 34.5%.
- The network was again trained extensively using several epochs, which improved the accuracy to 51.4%.
- With a total of 10 epochs on the short model, 20 epochs on the medium and 40 epochs on the long model, the residual inception network solved 620 Sudokus out of 1000 (Figure3.1) .

- The 10% increase in accuracy in the first variant could be likely because of the addition of activation function ReLU after the shortcut connection.
- This kind of three-phase training methodology with intermediate models is beneficial to train a deeper network in a better way.
- A deeper network with the residual connection has proven to predict more symbols than a simple CNN model.
- However, if the number of blocks in the intermediate outputs are not properly selected, the model fails to converge. It was observed that when the short model has 2 blocks and the medium model has 5 blocks, due to the huge gap between the two outputs, the validation error in the medium model was always struck at 14.32.
- Similar behaviour was observed where the medium model failed to converge when the short model was trained using 10 epochs all at once, instead of incremental training.

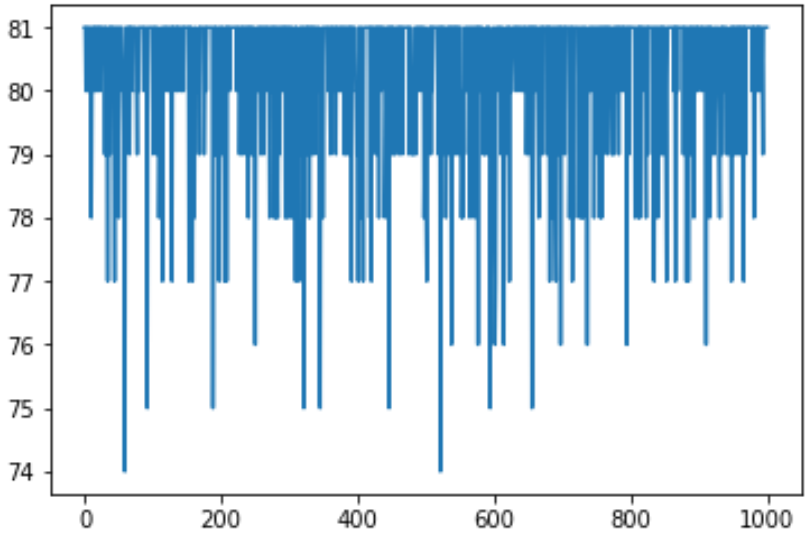


Figure 3.1: Distribution of correct symbols predicted by the model (Variant 1)

4 Conclusion

Neural networks are the powerful models that can be used in many applications. In communications, they can be used for channel decoding of polar codes. Sudoku codes, being non-linear codes can be decoded efficiently using neural networks. In this report, two different approaches to decode Sudoku codes with neural networks are discussed. A simple CNN model is able to recover 76 symbols out of 81. However, the accuracy is 1.8% which is very less compared to the total computational cost in terms of parameter count. In the second approach, the emphasis was on how to combine the residual networks and inception module to get satisfying results. With the residual inception network, the model was able to predict 620 Sudoku completely out of 1000 improving the accuracy to 62%. Hence this approach has proven to be beneficial despite having fewer parameters and computationally less expensive in comparison to a simple CNN model. Therefore, it can be inferred that neural networks have the potential to decode erroneous non-linear code words at the receiver.

Bibliography

[1] J. Sayir, "Sudoku codes, a class of non-linear iteratively decodable codes," Oct. 2014, http://www.inc.cuhk.edu.hk/sites/default/files/seminars/slides/talk (p. 1)

[2] T. K. Moon and J. H. Gunther, "Multiple constraint satisfaction by belief propagation: An example using sudoku," in *2006 IEEE Mountain Workshop on Adaptive and Learning Systems*, July 2006, pp. 122–126. DOI: 10.1109/SMCALS.2006.250702 (p. 2)

[3] T. Gruber, S. Cammerer, J. Hoydis, and S. ten Brink, "On deep learning-based channel decoding," *CoRR*, vol. abs/1701.07738, 2017. [Online]. Available: http://arxiv.org/abs/1701.07738 (p. 3)

[4] "https://cs231n.github.io/convolutional-networks/." (p. 4)

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385 (p. 6)

[6] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *CoRR*, vol. abs/1409.4842, 2014. [Online]. Available: http://arxiv.org/abs/1409.4842 (p. 7)

Erklärung

Hiermit erkläre ich, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie alle wörtlich oder sinngemäß aus anderen Werken übernommenen Aussagen als solche gekennzeichnet habe.

Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht. Das elektronische Exemplar stimmt mit den gedruckten Exemplaren überein.

Stuttgart, July 19, 2018

Deepthi Sreenivasaiah