Sudoku Solver

Ingo Prötel, Christoph Hanke,
Benjamin Müller

Introduction Sudoku

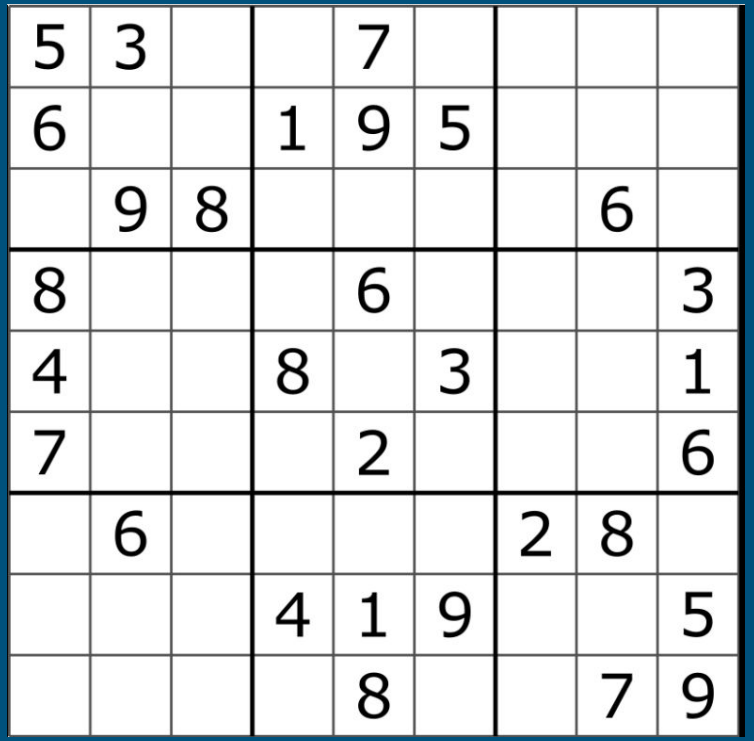
Fill in the digits 1 - 9 so that

each row and

each column and

each 3x3 sub-grid

contains each digit exactly once.



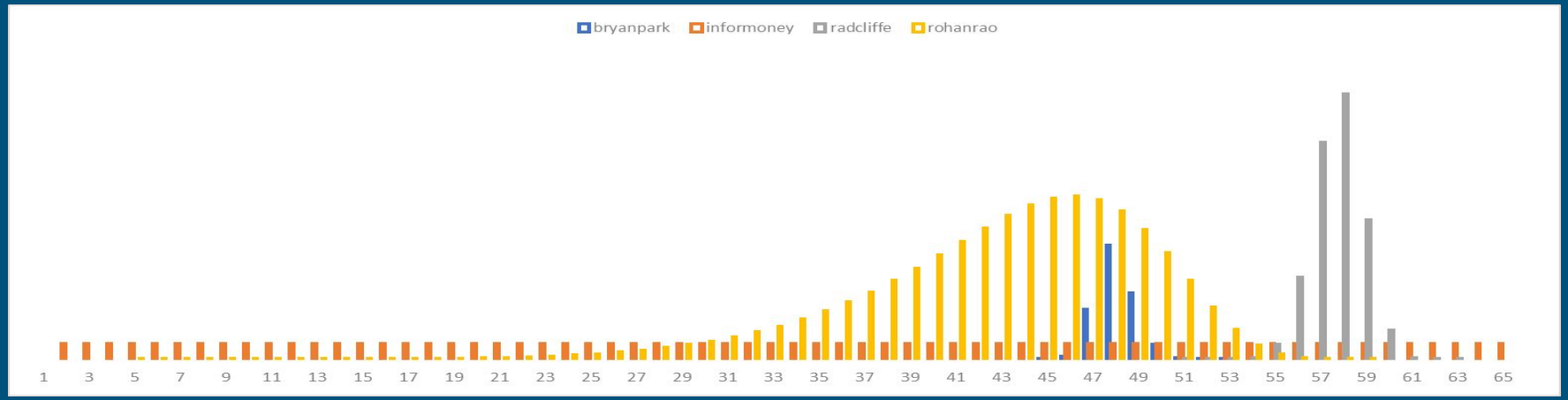
Data

Four different datasets from Kaggle:

- https://www.kaggle.com/datasets/bryanpark/sudoku
  1 million puzzles
- https://www.kaggle.com/datasets/rohanrao/sudoku
  9 million puzzles
- https://www.kaggle.com/datasets/radcliffe/3-million-sudoku-puzzles-with-ratings
  3 million puzzles
- https://www.kaggle.com/datasets/informoney/4-million-sudoku-puzzles-easytohard
  4 million puzzles

Data Content

- 17 000 000 solutions are 16 999 295 unique
- between 1 - 64 missing digits



Base Model

- Convolutional Neural Network-based Sudoku Solver
- Sudokus are zero mean-centred normalized ([-0.5, .05]) and represented by a `(9,9,1)`-Tensor
- the model transforms every sudoku field into a multi-class classification of size 9
- -> size of probability tensor is 81x9 which is converted to a 9x9 solution tensor using softmax
- Loss function is sparse_categorical_crossentropy

Base Model

```
keras.models.Sequential([
  Conv2D(filters=64, kernel_size=(3,3),  activation="relu",  padding='same', input_shape=(9,9,1)),
  BatchNormalization(),
  Conv2D(filters=64, kernel_size=(3,3),  activation="relu",  padding='same'),
  BatchNormalization(),
  Conv2D(filters=128, kernel_size=(1,1),   activation="relu",  padding='same'),

  Flatten(),
  Dense(81*9),
  Reshape((-1, 9)),
  Activation('softmax')
])
```

Base Model

- First approach: predict everything at once
  - -> bad performance
- Second approach: predict only one field per iteration
  - take the value with highest probability and start all over again with n+1 given values
  - human centered approach
  - takes very long (~ 20 seconds per sudoku)
  - higher reliability (99% on certain datasets)

Sums and Products

How to improve this already good base model?

- Looking for ways to make it faster and/or even more reliable
- Let’s support the model with additional information!
  - every column and row has a sum of 45 ($\sum i=[1;9]$)
  - every column and row has a product of 362.880 ($\prod i=[1;9]$)
  - expand the 9x9 tensor to a 10x10 resp. 11x11 tensor
  - Idea: the CNN uses some kind of linear regression which is interwoven to the inference process

Sums and Products

Unsolved Sudoku:

```
[ 0  0  4  3  0  0  2  0  9 18]
[ 0  0  5  0  0  9  0  0  1 15]
[ 0  7  0  0  6  0  0  4  3 20]
[ 0  0  6  0  0  2  0  8  7 23]
[ 1  9  0  0  0  7  4  0  0 21]
[ 0  5  0  0  8  3  0  0  0 16]
[ 6  0  0  0  0  0  1  0  5 12]
[ 0  0  3  5  0  8  6  9  0 31]
[ 0  4  2  9  1  0  3  0  0 19]
[ 7 25 20 17 15 29 16 21 25  0]
```

Correct Sudoku:

```
[ 8  6  4  3  7  1  2  5  9 45]
[ 3  2  5  8  4  9  7  6  1 45]
[ 9  7  1  2  6  5  8  4  3 45]
[ 4  3  6  1  9  2  5  8  7 45]
[ 1  9  8  6  5  7  4  3  2 45]
[ 2  5  7  4  8  3  9  1  6 45]
[ 6  8  9  7  3  4  1  2  5 45]
[ 7  1  3  5  2  8  6  9  4 45]
[ 5  4  2  9  1  6  3  7  8 45]
[45 45 45 45 45 45 45 45 45  0]
```

Sums and Products

Difficulties:

- Model becomes quite big:
  - `model.add(Dense(81*9))` `#(9x9x9)`
  - `-> model.add(Dense(100*45))` `#(10x10x45)`
  - `-> model.add(Dense(121*362.880))` `#(11x11x362.880)`

Sums and Products

Difficulties:

- Model becomes quite big:
  - `model.add(Dense(81*9))` `#(9x9x9)`
  - `-> model.add(Dense(100*45))` `#(10x10x45)`
  - `-> model.add(Dense(121*362.880))` `#(11x11x362.880)`
- “45” and “362.880” is always the correct answer

  -> Model doesn’t care at all about the additional information!

Custom loss function

- Make loss function aware of Sudoku specifics
- Check the difference in the count of each digit
- Use the difference to modify the given ‘regular’ loss function
- Process as tensors

Custom loss function

```
def custom_loss_function(y_true, y_pred):
    error = []
    y_pred1=tf.math.argmax(y_pred,axis=2)
    a, b = y_pred1.get_shape()
    for i in range(a):
        _, _, cnts = tf.unique_with_counts(y_pred1[i,:])
        cnts_diff = tf.abs(cnts-9)
        pred_error = tf.reduce_sum(cnts_diff) / 144
        error.append(pred_error)
    corrector = 1 + tf.square(tf.reduce_mean(error))
    corrector = tf.cast(corrector, tf.float32)
    scce = tf.keras.losses.SparseCategoricalCrossentropy()
    result = scce(y_true, y_pred)
    return result * corrector
```

Reduce runtime - reduce model complexity

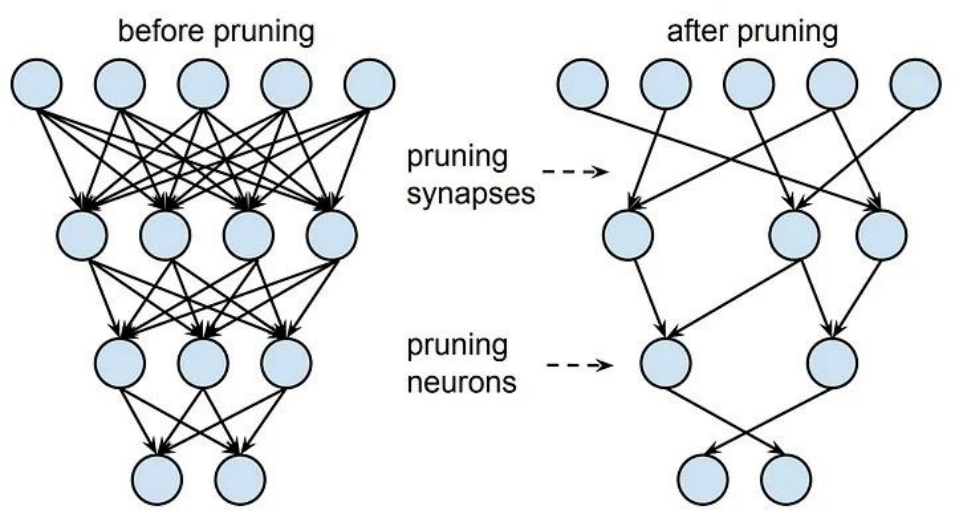
```
keras.models.Sequential([
  Conv2D(filters=64, kernel_size=(3,3),  activation="relu",  padding='same', input_shape=(9,9,1)),
  BatchNormalization(),
  Conv2D(filters=64, kernel_size=(3,3),  activation="relu",  padding='same'),
  BatchNormalization(),
  Conv2D(filters=128, kernel_size=(1,1),   activation="relu",  padding='same'),

  Flatten(),
  Dense(81*9),
  Reshape((-1, 9)),
  Activation('softmax')
])
```

1. Delete Layers
2. Reduce number of filters

Reduce runtime - pruning

- remove redundant neurons
- remove rarely activated neurons



Reduce runtime - Results

Model	Learntime	#Param (in Mio)		testtime	iterative complete	relative		testtime	oneshot complete	relative
64, 64, 64, 128	7214	7.6		257.72	1	1		6.32	0	0.70
64, 64, 128	1581	7.6		223.99	1	1		5.11	0	0.72
64, 64, 128 P	2529	7.6		252.42	1	1		6.08	0	0.72
64 ,64	1010	3.8		215.86	1	1		4.82	0	0.70
32, 32	575	1.9		211.36	0.96	0.64		5.10	0	0.67
16, 16	366	0.9		210.96	1	1		5.07	0	0.70
16, 16 P	467	7.6		220.55	1	1		5.35	0	0.70
8, 8	259	0.5		209.18	0	0.47		4.97	0	0.52
64	783	3.8		215.60	0	0.31		4.97	0	0.35

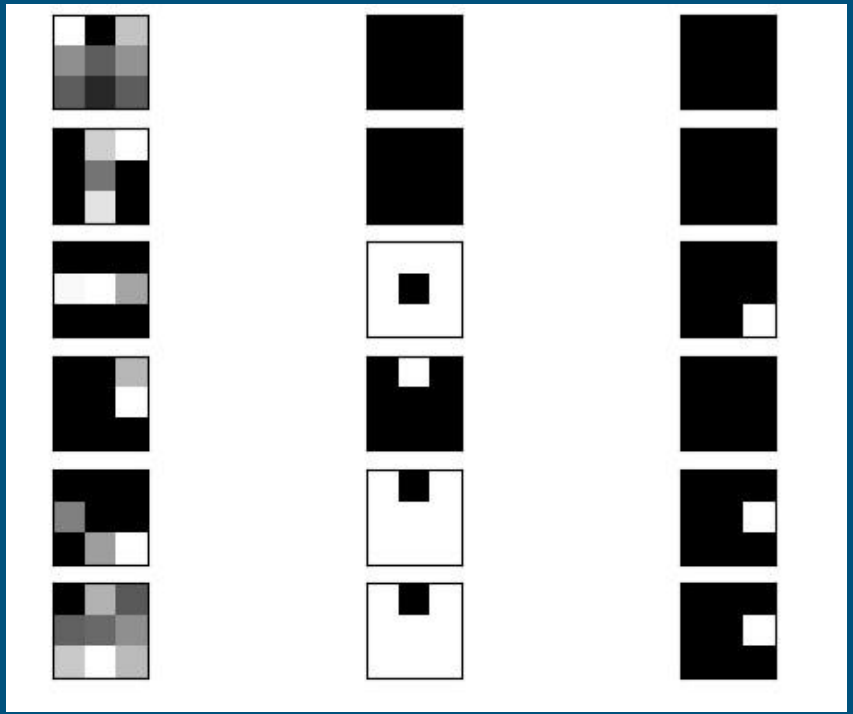
Reduce runtime - Result & followup measures

- model reduction without losing accuracy is possible
- BUT: reducing model complexity had a low effect on testtime

- use better Pruning methods
- time is lost in the interactive process due to array methods?
  - enhance by using faster array methods
- go for the oneshot approach
  - make larger models and prune them
  - Is it possible to reach 100% with this approach?

Picture of first 6 filters

- black means low activation
- white high activation



Lessons learned

- Get a better understanding at reading error messages
- Good looking learning curve doesn’t automatically mean good predictions
- Shaping is a pain!
- What is a tensor and what is a numpy array?
- Training takes time
- Batch size matters
- AI is not the solution to everything
- ChatGPT is better in generating a classical backtracking solution than solving sudokus itself

The End

Questions?