

Large Dataset Source Code Similarity Analysis

Checking capabilities of simple models

Vladimir Zolotov

January 13, 2021

Definition and types of similarity problem solved

- **Similarity of source code samples:**

- Solutions solving same problems are considered similar

1. Validating/Testing on code samples solving the problems seen at training

- Of course, all validation samples are different than training ones

	Prb 1	Prb 2	Prb 3	Prb 4	Prb 5
Train	Sol 1.1	Sol 2.1	Sol 3.1	Sol 4.1	Sol 5.1
	Sol 1.2	Sol 2.2	Sol 3.2	Sol 4.2	Sol 5.2
	Sol 1.3	Sol 2.3	Sol 3.3	Sol 4.3	Sol 5.3
Validate	Sol 1.4	Sol 2.4	Sol 3.4	Sol 4.4	Sol 5.3
	Sol 1.5	Sol 2.5	Sol 3.5	Sol 4.5	Sol 5.5

2. Validating/Testing on code samples solving problems not seen at training

- Not only samples but even problems solved with validation samples are different than training ones
- More difficult problem
 - Because similarity is checked for solutions of problems never seen with the analyzer

Prb 1	Prb 2	Prb 3	Prb 4	Prb 5
Sol 1.1	Sol 2.1	Sol 3.1	Sol 4.1	Sol 5.1
Sol 1.2	Sol 2.2	Sol 3.2	Sol 4.2	Sol 5.2
Sol 1.3	Sol 2.3	Sol 3.3	Sol 4.3	Sol 5.3
Sol 1.4	Sol 2.4	Sol 3.4	Sol 4.4	Sol 5.3
Sol 1.5	Sol 2.5	Sol 3.5	Sol 4.5	Sol 5.5
Train			Validate	

Similarity analysis of C++ source code

Old results

- **57 AIZU problems**
 - 90,029 solutions
 - Training 200,000 samples
 - Validation 50,000 samples
- Solutions of problems seen at training
 - **93.37% accuracy**
- Solutions of problems not seen at training
 - **77.06% accuracy**
- Details are in the attached old charts in the appendix.

New Results

- **536 AIZU problems**
 - 395,870 solutions
 - Training 2,000,000 samples
 - Validation 200,000 samples
- Solutions for problems seen at training
 - **96.33% accuracy**
- Solutions of problems not seen at training
 - **93.11% accuracy**
- **1163 AtCoder problems**
 - 3,733,548 solutions
 - Training 4,000,000 samples
 - Validation 400,000 samples
- Solutions for problems seen at training
 - **96.33% accuracy**
- Solutions of problems not seen at training
 - **93.28% accuracy**

Conclusions:

- Similarity analysis for our dataset can be done with high accuracy
 - Accuracy can be improved further by finer tuning DNNs and optimization

New vs Old classification techniques

Similarities

- Sequence of tokens model of source code (no comments, no identifiers)
 - Geert's tokenizer
- 2 input Siamese CNN with GlobalMax Pooling and Sigmoid classifier

Differences

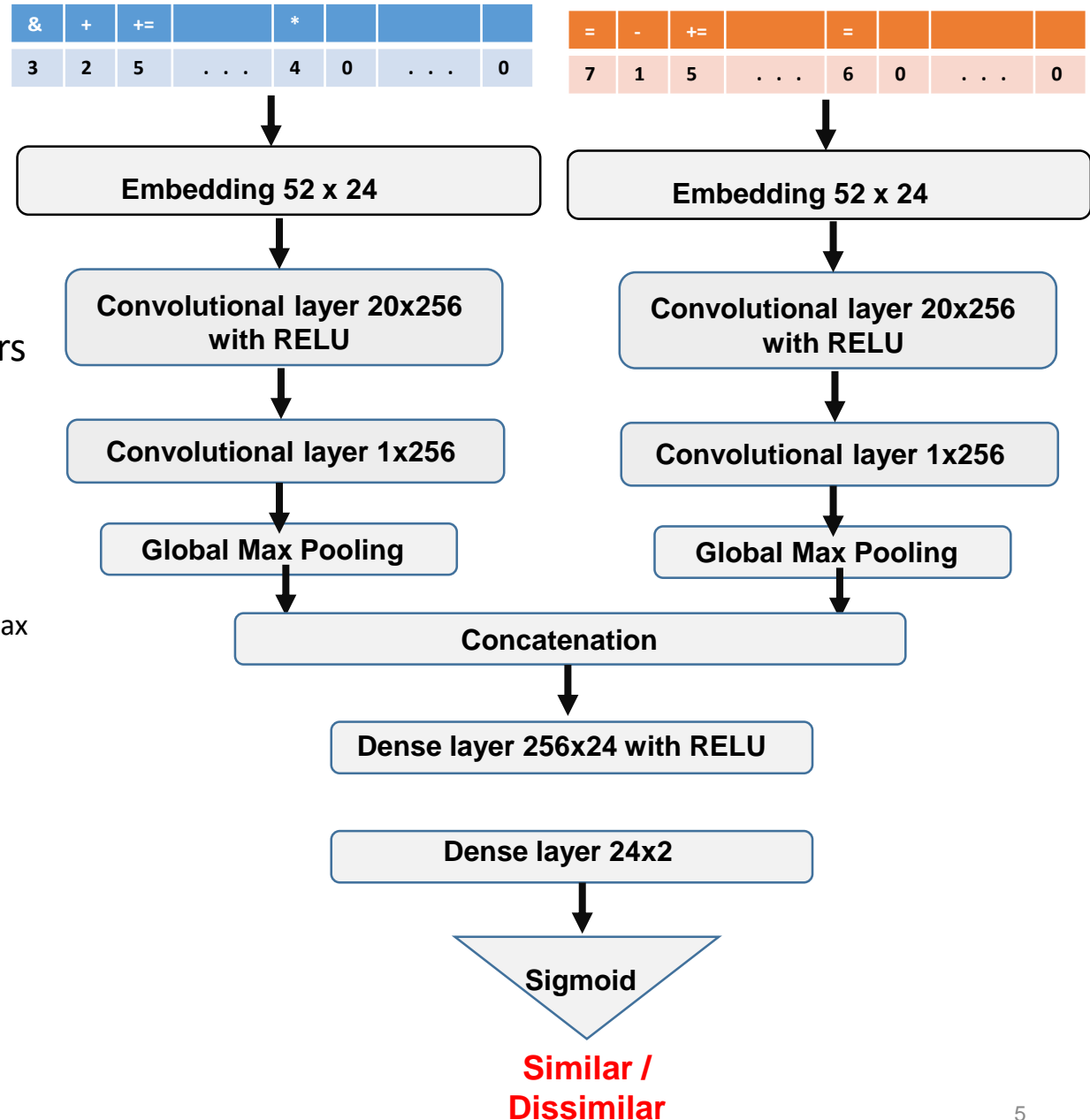
Old technique

- 17 groups of combined tokens
 - H. Ohashi and Y. Watanobe, Convolutional Neural Network for Classification of Source Codes, 2019
- One-hot coding
- No embedding layer
- 5 layers:
 - 1x 1D convolution 7x128
 - Global Max Pool,
 - 2 Dense: 64x16, 16x2
 - Sigmoid
- RMSPROP optimizer

New technique

- 52 tokens (no grouping)
 - Almost all operators and a few keywords
- Categorical coding
- Embedding layer of 24 width
 - Trainable embeddings
- 7 layers for Atcoder:
 - Trainable embedding layer 52x24
 - 2x 1D convolution: 20x256, 1x256
 - Global Max Pool,
 - 2x Dense: 256x24, 24x2
 - Sigmoid
- ADAM optimizer

Two input Siamese CNN for Atcoder-1163



- 2 inputs:
 - Each is supplied with a sequence of tokens
- 2 trained embedding layers
 - Sharing weight
- 2 towers of same convolutional layers:
 - Sharing weights
 - One with RELU the other without
 - No RELU before Global Max pool
- Global Max pooling after convolutions
- Concatenation layer
- 2 dense layers
 - One with RELU the other without
 - No RELU before Sigmoid
- Sigmoid classifier
- With regularization

Old and New sequence elements

Old groups of tokens

ALL KEYWORDS AND TOKEN NUMBERS

Assignment operator	Assigned number	=	4
=	0	^ =	4
Arithmetic operators	Assigned numbers	<<=	4
+	1	>>=	4
-	1	Comparison operators	Assigned numbers
*	1	==	5
/	1	!=	5
%	1	<	5
Bitwise Operators	Assigned numbers	<=	5
&	2	>	5
	2	>=	5
^	2	Logical operators	Assigned numbers
~	2	&&	6
^	2		6
<<	2	!	6
>>	2	Others	Assigned numbers
Compound arithmetic assignment operators	Assigned numbers	'if' control flow	7
+=	3	'else' control flow	8
-=	3	'for' control flow	9
*=	3	'while' control flow	10
/=	3	(11
%=	3)	12
++	3	{	13
--	3	}	14
Compound bitwise assignment operators	Assigned numbers	[15
&=	4]	16

New tokens

- =, +, -, *, /, %
 - Assignment and arithmetic
- &, |, ^, ~, <<, >>
 - Bitwise Operators
- +=, -=, *=, /=, %=, ++, --
 - Compound arithmetic assignment
- &=, |=, ^=, <<=, >>=
 - Compound bitwise assignment
- ==, !=, <, <=, >, >=,
 - Comparison operators
- &&, ||, !
 - Logical operators
- (,), {, }, [,], ->
- if, else, for, while, switch,
- int, char, short, long, float, double, bool

Current work and Future Plans

- **On-going work:**

- POJ-104 dataset
- mAP and other metrics from MISIM
- Comparison with published MISIM results

- **Plans:**

- Cross-language similarity analysis
- Confusion analysis:
 - Understand what source code are difficult for classification and similarity analysis
- Using sequences of detailed tokens constructed from parse trees:
 - Distinguish between using * for multiplication and dereferencing, etc.
 - Variable and operator types (int vs float)
- Using information from syntax trees to improve robustness and accuracy of similarity analysis
- Attention based technique
- Develop and try batch technique for efficient processing syntax trees by GPU
- Transforming CodeNet to dataset for code translation
 - Using clustering techniques

Appendix

Old charts on similarity analysis of AIZU 57 problems by sequence and bag of tokens techniques

Source Code Similarity Analysis by Sequence of Tokens Technique

Checking capabilities of simple models

Vladimir Zolotov

October 27, 2020

Outline

- Overview of background
 - Sequence of tokens technique
 - Formulation of similarity analysis for AIZU dataset
- CNN architecture for similarity analysis
 - Single tower and Siamese CNNs
- Results of code similarity analysis
 - Testing with source code solutions of problems seen and not seen at training
- Issues and difficulties
- Conclusions and directions of future work

Sequence of tokens model of source code

- Tokenization of source code
 - Geert Janssen tokenizer for C, C++
 - Very simple set of tokens:
 - Comments and macros are deleted
 - Groups of original language tokens are combined:
+, -, *, /, % -> token #1
 - Hiroki Ohashi and Yutaka Watanobe, "Convolutional Neural Network for Classification of Source Codes", 2019 Int. Symp. on Embedded Multicore/Many-core Systems-on-Chip (MCSoc)
- One hot coding of each token
- Zero padding at the end to the length of the longest sequence

```
#include <bits/stdc++.h>
using namespace std;
int dp[101][10001];
int main(){
    int N, W;
    cin >> N >> W;
    int v[100], w[100];
    for(int i = 0; i <= N; i++){
        cin >> v[i] >> w[i];
    }
    int ans = 0;
    for(int i = 0; i <= N; i++){
        for(int j = 0; j <= W; j++){
            dp[i+1][j] = max(dp[i+1][j], dp[i][j]);
            if(j + w[i] <= W){
                dp[i+1][j + w[i]] = max(dp[i+1][j + w[i]], dp[i][j] + v[i]);
                ans = max(ans, dp[i+1][j + w[i]]);
            }
        }
    }
    cout << ans << endl;
    return 0;
}
```

```
< / ++ > [ ] [ ]
( ) { >> >> [ ] [
] for ( = < ++ )
{ >> [ ] >> [ ] }
= for ( = < ++ )
{ for ( = <= ++ )
{ [ + ] [ ] = ( [
+ ] [ ] [ ] [ ] )
if ( + [ ] <= ) {
[ + ] [ + [ ] ] =
( [ + ] [ + [ ] ]
[ ] [ ] + [ ] ) =
( [ + ] [ + [ ] ]
) } } } } << << }
```

Assignment operator	Assigned number	=	4
=	0	^ =	4
Arithmetic operators	Assigned numbers	< <=	4
+	1	> >=	4
-	1	Comparison operators	Assigned numbers
*	1	==	5
/	1	!=	5
%	1	<	5
Bitwise Operators	Assigned numbers	<=	5
&	2	>	5
	2	>=	5
^	2	Logical operators	Assigned numbers
~	2	& &	6
~	2		6
<<	2	!	6
>>	2	Others	Assigned numbers
Compound arithmetic assignment operators	Assigned numbers	'if' control flow	7
+=	3	'else' control flow	8
--	3	'for' control flow	9
*=	3	'while' control flow	10
/=	3	(11
%=	3)	12
++	3	{	13
--	3	}	14
Compound bitwise assignment operators	Assigned numbers	[15
&=	4]	16

&	+	+=		=			
0	0	0	...	1	0	...	0
0	1	0	...	0	0	...	0
1	0	0	...	0	0	...	0
0	0	1	...	0	0	...	0
0	0	0	...	0	0	...	0

Code similarity problem for AIZU dataset

- Similarity samples are pairs of AIZU solutions
 - All solutions of same problem are similar
 - Not obvious assumption (possibly too broad)
 - For example, plagiarism is defined differently
 - It is possible that the same problem can be solved with very dissimilar code
 - Solutions of different problems are different
- Two types of similarity problem setups:
 1. Validating on code samples solving the same problems that used for training
 - All validation samples are different than training one
 - This similarity problem can be solved by a code classifier
 - Classify each code file : If they are of the same class, they are considered similar otherwise dissimilar.
 - I used a different approach
 2. Validating on code samples solving problems that were not used for training
 - More difficult problem
 - Because similarity is checked for solutions of problems never seen with the analyzer

	Prb 1	Prb 2	Prb 3	Prb 4	Prb 5
Train	Sol 1.1	Sol 2.1	Sol 3.1	Sol 4.1	Sol 5.1
	Sol 1.2	Sol 2.2	Sol 3.2	Sol 4.2	Sol 5.2
	Sol 1.3	Sol 2.3	Sol 3.3	Sol 4.3	Sol 5.3
Validate	Sol 1.4	Sol 2.4	Sol 3.4	Sol 4.4	Sol 5.3
	Sol 1.5	Sol 2.5	Sol 3.5	Sol 4.5	Sol 5.5

Prb 1	Prb 2	Prb 3	Prb 4	Prb 5
Sol 1.1	Sol 2.1	Sol 3.1	Sol 4.1	Sol 5.1
Sol 1.2	Sol 2.2	Sol 3.2	Sol 4.2	Sol 5.2
Sol 1.3	Sol 2.3	Sol 3.3	Sol 4.3	Sol 5.3
Sol 1.4	Sol 2.4	Sol 3.4	Sol 4.4	Sol 5.3
Sol 1.5	Sol 2.5	Sol 3.5	Sol 4.5	Sol 5.5
Train			Validate	

Similarity Dataset Construction

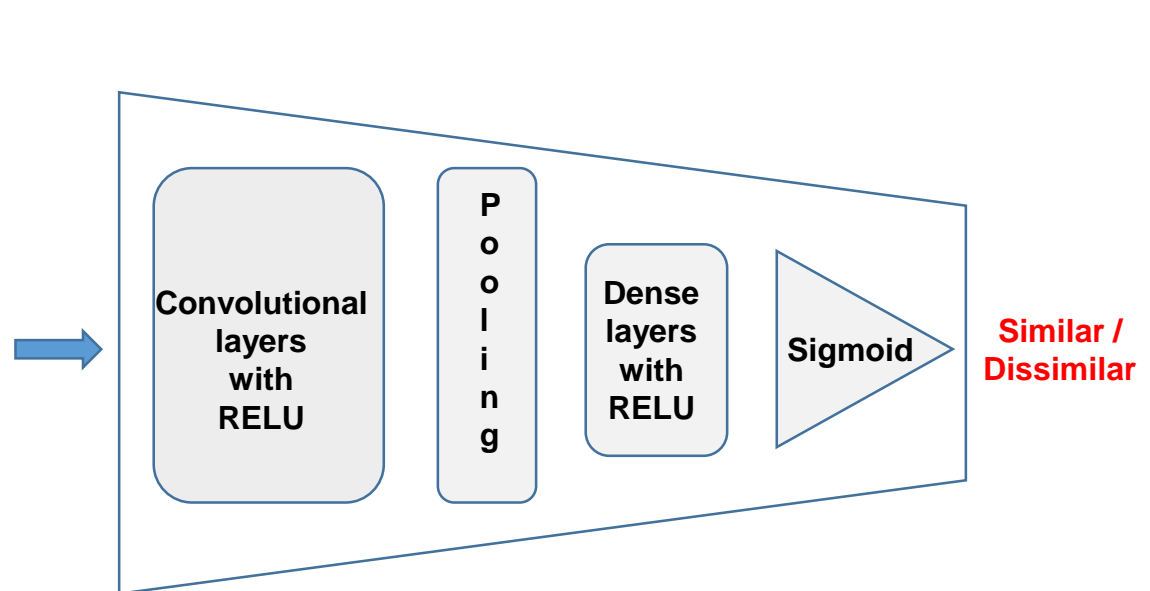
- Straightforward approach:
 - Randomly select a pair of code samples
 - If they solve same problem, it is a similar sample
 - If they solve different problem, it is different sample
- This method was not used:
 - Probability of dissimilar samples is much higher
 - Possible to construct a pair of same solutions of same problem
 - Resulted dataset is not clean and skewed
- Approach used:
 - Split original problem solutions into training and validation parts
 - Split either problems or solutions of each problem depending on the strategy used
 - Construct separately similar and dissimilar samples in required proportion
 - 50% of all samples are pairs of similar sample and 50% are dissimilar samples
 - For similar samples select different solutions of same problem
 - For dissimilar samples select solutions of different problems
 - Two step selection:
 - Random selection of problem with probability proportional its number of solutions
 - Random selection of samples from the selected problem
- Developed technique is intended for constructing homogeneous fair non-skewed dataset
 - Each problem is represented proportionally to its number of solution

Similarity Analysis by Sequence of Tokens

- Classifier input is a pair of token sequences
 - One hot coding of each sequence
 - Each sequence consists of vectors having 17 elements
- All sequences are padded to length of the longest sequence (3040)
- Sequences have large (>100x) length variation
 - Additional difficulties in classification
- Convolutional neural network
 - Convolutions, pooling, dense layers and sigmoid classifier
- Tried 2 different architectures, regularization and large training data sets
 - Because of having difficulties in generalization due to easy overfitting

&	+	+=		=			
0	0	0	...	1	0	...	0
0	1	0	...	0	0	...	0
1	0	0	...	0	0	...	0
0	0	1	...	0	0	...	0
0	0	0	...	0	0	...	0

=	-	+=		=			
1	0	0	...	1	0	...	0
0	1	0	...	0	0	...	0
0	0	0	...	0	0	...	0
0	0	1	...	0	0	...	0
0	0	0	...	0	0	...	0

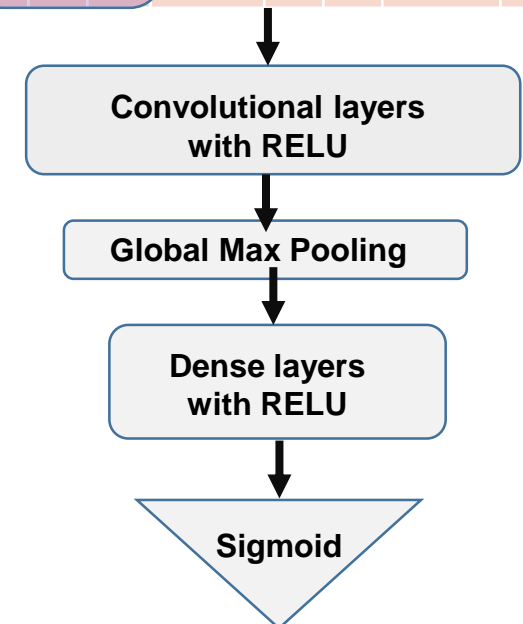


Single tower CNN

- Implemented with sequential model of Keras
 - Simple tower
- Two token sequences are combined into one sequence of vectors of $34=17*2$ elements
- 1 or more 1D convolutional layers
 - RELU at each layer except the last one
- Global Max pooling after last convolution
 - As it is nonlinear function, no RELU on last convolution
- 1 or more dense layers with RELU at each layer except the last one
- Sigmoid classifier
 - As it is nonlinear function, no RELU on last dense layer
- No regularization
- Average global pooling performs poorly
 - Because sequences of different length are averaged differently due to padding
 - Short sequences contribution is too small
 - They are averaged across to many padding zeros

Convolution Kernel

&	=	+	+=		*	=		=	
		-	+=		=				
0	0	0		...	0	1	...	1	
0	1	0		...	1	0	...	0	
1	0	0		...	0	0	...	0	
0	0	1		...	0	0	...	0	
0	0	0		...	0	0	...	0	
1	0	0		...	1	0	...	0	
0	1	0		...	0	0	...	0	
0	0	0		...	0	0	...	0	
0	0	1		...	0	0	...	0	
0	0	0		...	0	0	...	0	



Similar / Dissimilar

Two input Siamese CNN

Convolution Kernel

&	+	+=		*	=		=
0	0	0	...	0	1	...	1
0	1	0	...	1	0	...	0
1	0	0	...	0	0	...	0
0	0	1	...	0	0	...	0
0	0	0	...	0	0	...	0

=	-	+=		=			
1	0	0	...	1	0	...	0
0	1	0	...	0	0	...	0
0	0	0	...	0	0	...	0
0	0	1	...	0	0	...	0
0	0	0	...	0	0	...	0

Convolutional layers with RELU

Convolutional layers with RELU

Global Max Pooling

Global Max Pooling

Concatenation

Dense layers with RELU

Sigmoid

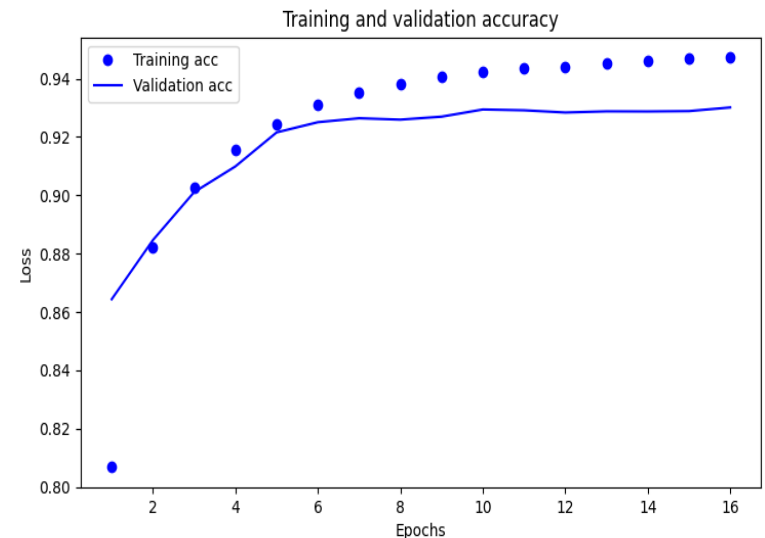
Similar / Dissimilar

- Two inputs:
 - Each is supplied with a sequence of tokens
- Two towers of 1 or more 1D same convolutional layers with RELu
 - Shared or non-shared weights
- Global Max pooling after convolutions
- Concatenation layer
- 1 or more dense layers with RELU
- Sigmoid classifier
- With and without regularization

Similarity of solutions for problems *seen* at training

- 57 problems with 90029 solutions
- Training: 200,000 samples
- Validation: 50,000 samples
- Single tower CNN
 - One convolutional layer:
 - 128 filters with width = 7
 - Two dense layers: 128x16 and 16x2
 - No regularization
 - **90.9%** accuracy at epoch 13
- Two inputs Siamese CNN with shared weights
 - One convolutional layer:
 - 32 filters with width = 7
 - Two dense layers: 64x16 and 16x2
 - No regularization
 - **93.37%** accuracy at epoch 13
- Significant overfitting
 - On Siamese network it is lower

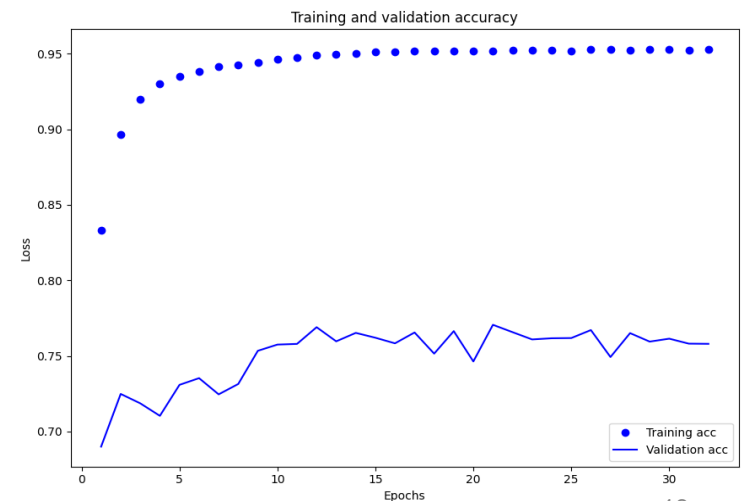
	Prb 1	Prb 2	Prb 3	Prb 4	Prb 5
Train 200000 samples	Sol 1.1	Sol 2.1	Sol 3.1	Sol 4.1	Sol 5.1
	Sol 1.2	Sol 2.2	Sol 3.2	Sol 4.2	Sol 5.2
	Sol 1.3	Sol 2.3	Sol 3.3	Sol 4.3	Sol 5.3
Validate 50000 samples	Sol 1.4	Sol 2.4	Sol 3.4	Sol 4.4	Sol 5.3
	Sol 1.5	Sol 2.5	Sol 3.5	Sol 4.5	Sol 5.5



Similarity of solutions for problems *not seen* at training

- 57 problems with 90029 solutions
- Training: 200,000 samples
- Validation: 50,000 samples
- Single tower CNN
 - One convolutional layer:
 - 256 filters with width = 7
 - Three dense layers: 256x6, 6x4, and 4x2
 - No regularization
 - **70.5%** accuracy at epoch 32
- Two inputs Siamese CNN with shared weights
 - Two convolutional layers:
 - 64 filters with width = 13
 - 32 filters with width = 7
 - Two dense layers: 128x16 and 16x2
 - Regularization: L1=0.00001, L2=0.001
 - **77.06%** accuracy at epoch 21
- Overfitting is even higher than for testing on solutions of problem seen at training
 - Poor generalization in spite of, many measures taken

Prb 1	Prb 2	Prb 3	Prb 4	Prb 5
Sol 1.1	Sol 2.1	Sol 3.1	Sol 4.1	Sol 5.1
Sol 1.2	Sol 2.2	Sol 3.2	Sol 4.2	Sol 5.2
Sol 1.3	Sol 2.3	Sol 3.3	Sol 4.3	Sol 5.3
Sol 1.4	Sol 2.4	Sol 3.4	Sol 4.4	Sol 5.3
Sol 1.5	Sol 2.5	Sol 3.5	Sol 4.5	Sol 5.5
Train 200000 samples			Validate 50000 samples	



- Much worse generalization and overfitting vs Bag of Tokens
 - Especially for testing on problems different than the ones used for training
- Several special measures to improve generalization:
 - Much large (10x) dataset had to be used: 200,000 vs 20,000 training samples
 - Siamese type of CNN with shared weights
 - Regularization: both L1 and L2
- Fortunately, large datasets for similarity is not a problem
 - There many possible combinations of source code problem solutions
- However, 1-hot coding and padding up to longest sequence length result in huge data:
 - $250,000 * 17 * 2 * 3000 * 4 = 102,000,000,000$ bytes = 102 Gbytes
 - Larger data will require smarter techniques:
 - Dynamic batch creation
 - Padding only up to longest sequence of the batch
- High variability of accuracy of code similarity analysis with testing on problems not seen at training

High variability of accuracy of code similarity analysis of problems not seen at training

- Previously bag of tokens showed accuracy 86.6%
 - Without any significant tuning
- Currently sequence of tokens technique showed accuracy 77.06%
 - Even after architecture selection, 10x larger dataset and regularization.
- Very strange unexpected result
- Rerunning bag of tokens showed best accuracy only ~68%
 - 18% difference between previous 86.6% and current results 68%
- DNN tuning, extensive search for bugs, checking suspicions on dyce1/dyce2, GPU/CPU, TF versions difference showed nothing relevant
- The cause was statistically small *inhomogeneous set of problems*:
 - Two runs generated validation dataset from different set of problems
 - The number of problem was not statistically large: only 57
 - Test set was generated from 11 problems, not seen at training 46 problems
 - Difficulty of detecting similarity of code solutions of different problems varies very high
 - Previous experiments with bag of tokens had lucky test set created from “easier” problems
- No same problem for classification and similarity analysis with testing on problems seen at training.
 - Number of solutions (90029) is much larger than the number of problems
 - Therefore the similarity datasets are more homogenous

Conclusions and future directions

- Neither classification nor similarity analysis of AIZU code with testing on problems seen on training is a difficult ML problem:
 - CNN gets 94.72% accuracy for classification among 57 classes
 - 2-input Siamese CNN gets 93.37% accuracy for similarity analysis
 - Better architectures, regularization and tuning may add a few %% to those results
 - But not large room for improvement
- Similarity analysis with testing on problems not seen at training is more difficult
 - Best achieved accuracy is only 77% : not very bad but not very good either
 - Poor generalization
 - Obvious issue as testing sees implementation of algorithms never seen at training
 - Therefore training need to find features indicating similarity of code with respect to algorithm it solves
 - However, training is inclined to select/construct features indicating similarity of code of specific algorithms
 - Aurora uses manually constructed features, MISIM reuses those features
- Facts to analyze and verify:
 - Increasing the number of problem/algorithms may improve generalization
 - Some problems are much more difficult to classify/analyze
 - Scaling of accuracy with increasing the number of problems
 - Different sets of tokens
 - ROC analysis and tuning based on it for similarity with testing on problems not seen at training
 - 3 set technique: training, validation testing

Source Code Classification and Similarity Analysis by Bag of Tokens Technique

Checking capabilities of simple models and techniques

Vladimir Zolotov

September 30, 2020

Outline

- Bag of tokens for classification with many classes
- Problems of similarity analysis
 - Dataset construction
- Bag of tokens for code similarity
- Experiments on similarity analysis by bag of tokens

Bag of tokens model

- Tokenization of source code

- Geert Janssen tokenizer for C, C++
- Very simple set of 17 tokens:
 - Comments and macros are deleted
 - Groups of original language tokens are combined:

+, -, *, /, % -> token #1

- “Convolutional Neural Network for Classification of Source Codes”,

Hiroki Ohashi and Yutaka Watanobe at 2019 Int. Symp. on Embedded Multicore/Many-core Systems-on-Chip (MCSoc)

- Bag of tokens:

- Make vector of number of tokens occurrences

=	+	-	*	. . .	{	}	[]
5	10	0	0	. . .	4	4	25	25

- Normalize it to get vector of token frequencies

$$V = W / \sqrt{W * W}$$

=	+	-	*	. . .	{	}	[]
0.13	0.21	0	0	. . .	0.12	0.12	0.47	0.47

- Advantages:

- Invariant to many types of code transformations:
 - Statement permutations, code factorization, etc.
- Many other models are not invariant to many code transformations preserving its algorithms

```
#include <bits/stdc++.h>
using namespace std;
int dp[101][10001];
int main()
{
    int N, W;
    cin >> N >> W;
    int v[1001], w[1001];
    for(int i = 0; i <= N; i++)
        cin >> v[i] >> w[i];
    int ans = 0;
    for(int i = 0; i <= N; i++){
        for(int j = 0; j <= W; j++){
            dp[i+1][j] = max(dp[i+1][j], dp[i][j]);
            if(j < w[i] <= W){
                dp[i+1][j] = max(dp[i+1][j], dp[i][j] + v[i]);
            }
            ans = max(ans, dp[i+1][j] + w[i]);
        }
    }
    cout << ans << endl;
    return 0;
}
```

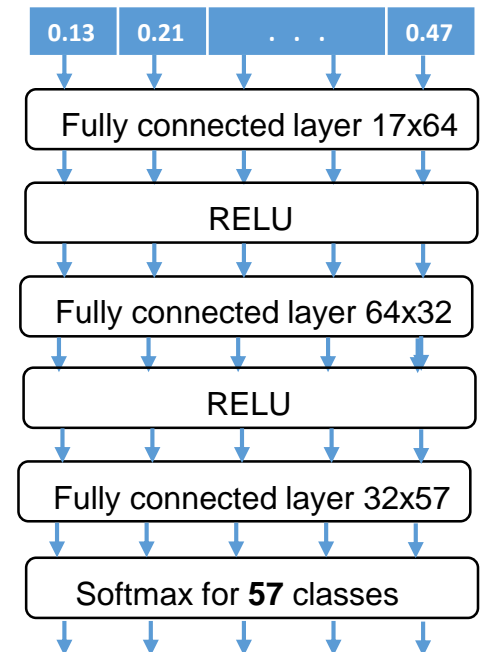
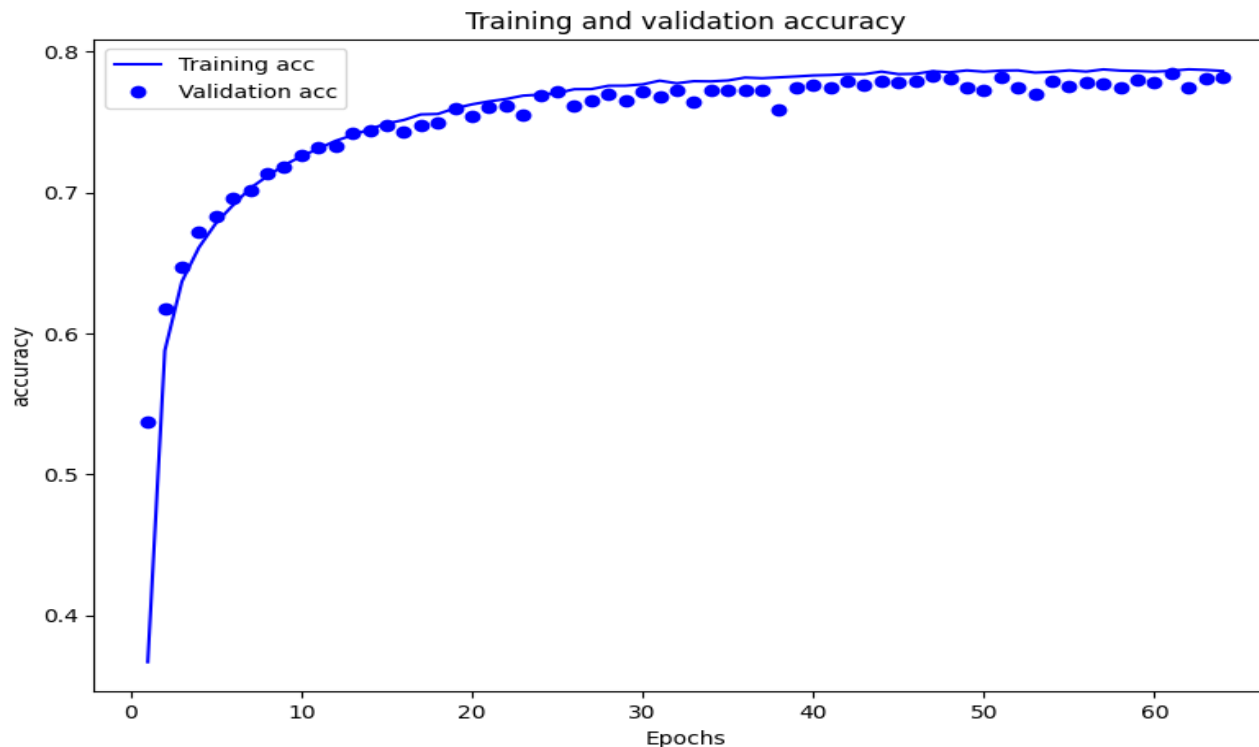
```
< / ++ > [ ] [ ]
( ) { >> >> [ ] [
] for ( = < ++ )
{ >> [ ] >> [ ] }
= for ( = < ++ )
{ for ( = <= ++ )
{ [ + ] [ ] = ( [
+ ] [ ] [ ] [ ] )
if ( + [ ] <= ) {
[ + ] [ + [ ] ] =
( [ + ] [ + [ ] ]
[ ] [ ] + [ ] ) =
( [ + ] [ + [ ] ]
) } } } << << }
```

ALL KEYWORDS AND TOKEN NUMBERS

Assignment operator	Assigned number	=	4
=	0	^=	4
Arithmetic operators	Assigned numbers	<<=	4
+	1	>>=	4
-	1	Comparison operators	Assigned numbers
*	1	==	5
/	1	!=	5
%	1	<	5
Bitwise Operators	Assigned numbers	<=	5
&	2	>	5
	2	>=	5
^	2	Logical operators	Assigned numbers
~	2	&&	6
^	2		6
<<	2	!	6
>>	2	Others	Assigned numbers
Compound arithmetic assignment operators	Assigned numbers	'if' control flow	7
+=	3	'else' control flow	8
-=	3	'for' control flow	9
*=	3	'while' control flow	10
/=	3	(11
%=	3)	12
++	3	{	13
--	3	}	14
Compound bitwise assignment operators	Assigned numbers	[15
&=	4]	16

Classification C++ source code files for many classes

- 57 classes are problems of AIZU dataset
 - Problems with 712 – 5099 *accepted* C++ solutions
 - Most problems have <1500 solutions
 - 80% - 20% training/validation split
 - Training on 72024 samples, Validating on 18005 samples
- 3 Layer Neural network
 - Fully connected layers: 17x64, 64x32, 32x57,
 - RELU and softmax activations
- 79.47% accuracy at epoch 61
 - Probability of random guess is only ~ 2%



Code similarity problem

- Similarity samples are pairs of AIZU solutions
 - All solutions of same problem are similar
 - Not obvious assumption (possibly too broad)
 - For example, plagiarism is defined differently
 - It is possible that the same problem can be solved with very dissimilar code
 - Solutions of different problems are different
- Two types of similarity problem setups:
 1. Validating on code samples solving the same problems that used for training
 - All validation samples are different than training one
 - This similarity problem can be solved by a code classifier
 - Classify each code file : If they are of the same class, they are considered similar otherwise dissimilar.
 - I used a different approach
 2. Validating on code samples solving problems that were not used for training
 - More difficult problem
 - Because similarity is checked for solutions of problems never seen with the analyzer

	Prb 1	Prb 2	Prb 3	Prb 4	Prb 5
Train	Sol 1.1	Sol 2.1	Sol 3.1	Sol 4.1	Sol 5.1
	Sol 1.2	Sol 2.2	Sol 3.2	Sol 4.2	Sol 5.2
	Sol 1.3	Sol 2.3	Sol 3.3	Sol 4.3	Sol 5.3
Validate	Sol 1.4	Sol 2.4	Sol 3.4	Sol 4.4	Sol 5.3
	Sol 1.5	Sol 2.5	Sol 3.5	Sol 4.5	Sol 5.5

Prb 1	Prb 2	Prb 3	Prb 4	Prb 5
Sol 1.1	Sol 2.1	Sol 3.1	Sol 4.1	Sol 5.1
Sol 1.2	Sol 2.2	Sol 3.2	Sol 4.2	Sol 5.2
Sol 1.3	Sol 2.3	Sol 3.3	Sol 4.3	Sol 5.3
Sol 1.4	Sol 2.4	Sol 3.4	Sol 4.4	Sol 5.3
Sol 1.5	Sol 2.5	Sol 3.5	Sol 4.5	Sol 5.5
Train			Validate	

Similarity Dataset Construction

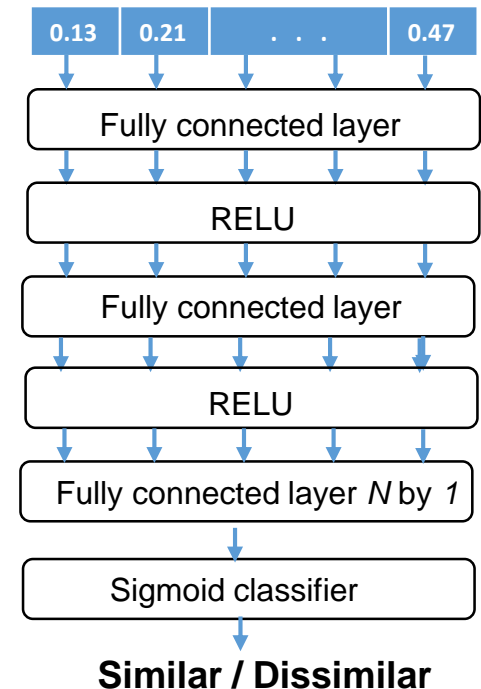
- Straightforward approach:
 - Randomly select a pair of code samples
 - If they solve same problem, it is a similar sample
 - If they solve different problem, it is different sample
- This method was not used:
 - Probability of dissimilar samples is much higher
 - Possible to construct a pair of same solutions of same problem
 - Resulted dataset is not clean and skewed
- Approach used:
 - Split original problem solutions into training and validation parts
 - Split either problems or solutions of each problem depending on the strategy used
 - Construct separately similar and dissimilar samples in required proportion
 - 50% of all samples are pairs of similar sample and 50% are dissimilar samples
 - For similar samples select different solutions of same problem
 - For dissimilar samples select solutions of different problems
 - Two step selection:
 - Random selection of problem with probability proportional its number of solutions
 - Random selection of samples from the selected problem
- Developed technique is intended for constructing homogeneous fair non-skewed dataset
 - Each problem is represented proportionally to its number of solution

Bag of Tokens Similarity Classification

- Same source code tokenization:
 - Geert Janssen tokenizer for C, C++
 - Very simple set of 17 tokens:
- Classifier input is concatenation of a pair of bag of tokens
 - $34=17*2$ long vector



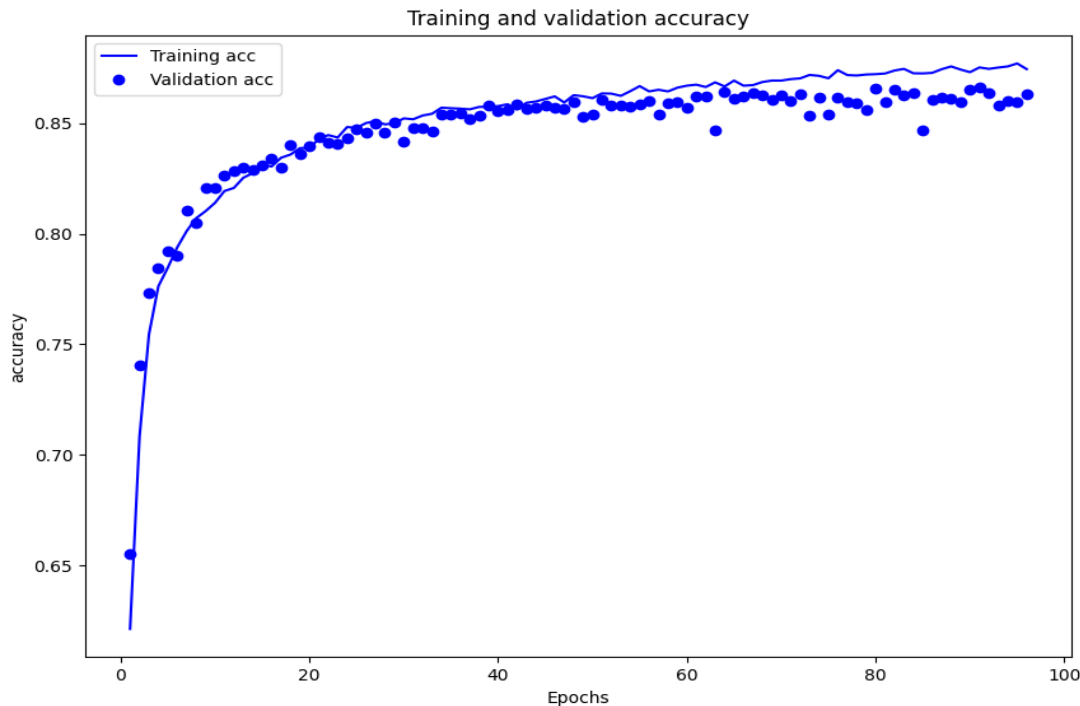
- Each subvector is the same bag of tokens that was used in classification experiments
- Similarity classification with 3-layer neural network
 - 3 fully connected layers:
 - 2 with RELU activation
 - Sigmoid classifier as output
- No regularization, no special tuning



Similarity of solutions for problems *seen* at training

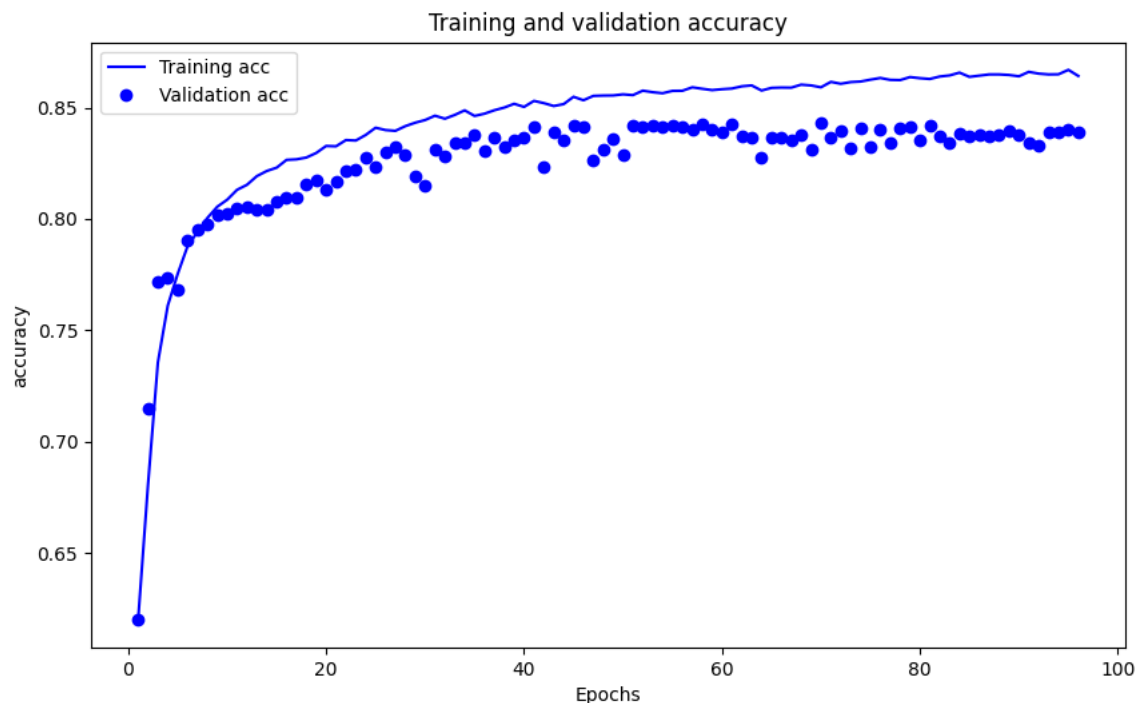
- 57 problems with 90029 solutions
- Training: 20000 samples
 - 9970 similar solutions; 10030 dissimilar solutions
- Validation: 5000 samples
 - 2474 similar solutions; 2526 dissimilar solutions
- 3 Layer Neural network
 - Fully connected layers: 34x48, 48x8, 8x1, RELU and sigmoid
- 86.6% accuracy at epoch 91

	Prb 1	Prb 2	Prb 3	Prb 4	Prb 5
Train 20000 samples	Sol 1.1	Sol 2.1	Sol 3.1	Sol 4.1	Sol 5.1
	Sol 1.2	Sol 2.2	Sol 3.2	Sol 4.2	Sol 5.2
	Sol 1.3	Sol 2.3	Sol 3.3	Sol 4.3	Sol 5.3
	Sol 1.4	Sol 2.4	Sol 3.4	Sol 4.4	Sol 5.4
	Sol 1.5	Sol 2.5	Sol 3.5	Sol 4.5	Sol 5.5
	Sol 1.6	Sol 2.6	Sol 3.6	Sol 4.6	Sol 5.6
Validate 5000 samples	Sol 1.4	Sol 2.4	Sol 3.4	Sol 4.4	Sol 5.4
	Sol 1.5	Sol 2.5	Sol 3.5	Sol 4.5	Sol 5.5
	Sol 1.6	Sol 2.6	Sol 3.6	Sol 4.6	Sol 5.6
	Sol 1.7	Sol 2.7	Sol 3.7	Sol 4.7	Sol 5.7
	Sol 1.8	Sol 2.8	Sol 3.8	Sol 4.8	Sol 5.8
	Sol 1.9	Sol 2.9	Sol 3.9	Sol 4.9	Sol 5.9



Similarity of solutions for problems *not seen* at training

- 57 problems with 90029 solutions
- Training: 20000 samples
 - 9977 similar solutions; 10023 dissimilar solutions
- Validation: 5000 samples
 - 2496 similar solutions; 2504 dissimilar solutions
- 3 Layer Neural network
 - Fully connected layers: 34x32, 32x6, 6x1, RELU and sigmoid
- 83.24% accuracy at epoch 75



Prb 1	Prb 2	Prb 3	Prb 4	Prb 5
Sol 1.1	Sol 2.1	Sol 3.1	Sol 4.1	Sol 5.1
Sol 1.2	Sol 2.2	Sol 3.2	Sol 4.2	Sol 5.2
Sol 1.3	Sol 2.3	Sol 3.3	Sol 4.3	Sol 5.3
Sol 1.4	Sol 2.4	Sol 3.4	Sol 4.4	Sol 5.3
Sol 1.5	Sol 2.5	Sol 3.5	Sol 4.5	Sol 5.5
Train 20000 samples			Validate 5000 samples	