Machine learning Project

Weining Lv

April 2018

1 Introduction

Recently, deep learning become a hot topic in the world. It gets a state of the art results on many fields, such as computer vision and natural language processing. Compare with problems in computer vision, it still has a long way for people to catch on programming language and biological information problem in deep learning. I did two small work on this term project. First of all, I predict protein sequence via deep learning,(especially Long Short Term Memory model). Secondly, I try to use different model did code to tensor translate and put it to seq2seq model, and try to find common error on c programming.

2 Related Work

2.1 Some Famous Models

Convolutional neural networks [11] gets a prominent result, especially on image classification. Recurrent models (also referred to as sequence models) [15] deal successfully with time-series data like speech or handwriting recognition. In contrast to that, Recursive neural models [21] have been proposed and applied to sentiment analysis,question-answering,relation classification and discourse.

Capsule network also becomes a hot topic this year. A capsule is a group of neurons whose activity vector represents the instantiation parameters of a specific type of entity such as an object or an object part. [20]

2.2 Biological Information Part

Travers Ching et al.think biology and medicine are data rich and ill-understood. Deep learning will help to solve a variety of bio-medical problems, such as patient classification,fundamental biological process and treatment of patients.[4]

2.3 Programming Language Part

Neel [10] talks recent advances in neural program synthesis. In addition, patch generation code [13] or de-obfuscates code[19] give us a whole new perspective that combine machine learning with traditional programming language problem together, but both of problems don't involve to deep learning.

3 Bio-information Part

3.1 Introduction

Sequence prediction is a difficult problem in many research areas. Prediction of the next protein sequence is an important component of bioinformatics and in this project, I focus on predicting next amino acid component in a sequence. Specifically, I use Recurrent Neural Network(RNN) model in sequence modeling and did two experiments:(1) Training RNN model on small sequence data set and test its performance on this specific training set. (2) Training RNN model on relatively large data set of amino acid sequence, and test its performance. My implementation is based on torch7 deep learning tool[5].

3.2 Related Work

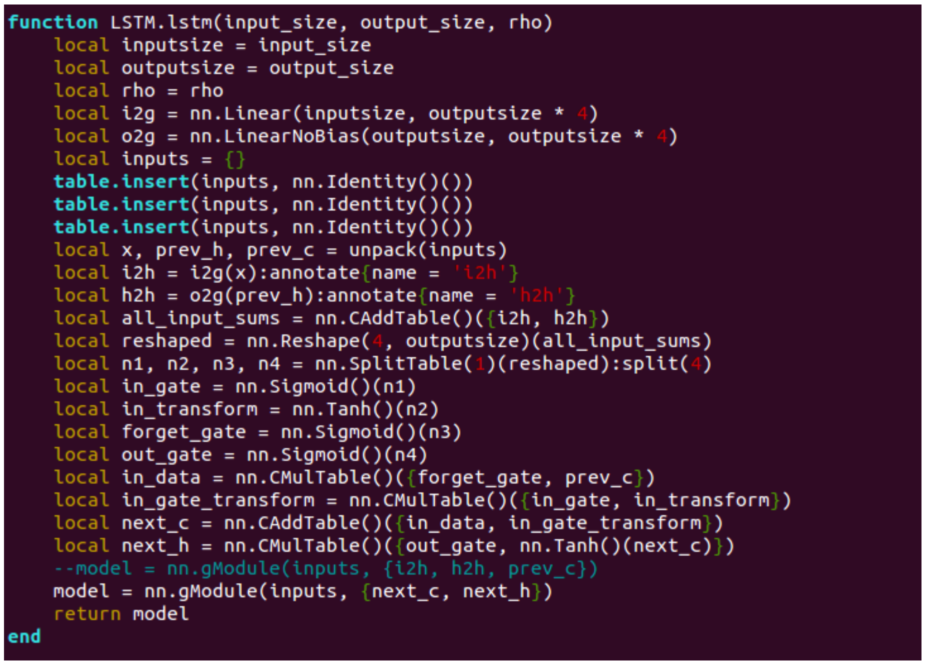
Hidden Markov Model(HMM) is a traditional statistical model for sequential modeling. They are also widely used in bioinformatics [14]. However, traditional HMM suffer from the fact that states should be drawn from a fixed size of states space. It is also not good in modeling long term dependencies. Recently, recurrent neural network(RNN) [12] has achieved widely success in sequential data modeling. It has been demonstrated to perform well in speech recognition[6], machine translation[3], as well as handwritten recognition[7]. Traditional RNN has hidden state which is represented as a vector. At each time step, the hidden state is updated based on the hidden state in the previous time step and the input in the current time step. It is defined as follow:

$$h^t = g(W_h * h^{t-1} + W_x * x^t + b)$$

where h represents the hidden state and t represent the current time step, and x represent the input. U is a matrix represent the linear transformation from previous time step t-1, and W is another matrix representing the transformation from input x. g is a nonlinear function.

Traditional recurrent neural network suffers from the problem that long term dependency is hard to achieve. This problem is solved by Long-short term memory, a special type of RNN which is designed specifically to solve this problem[9]. It provides a solution by adding additional information about when to forget the previous information and when to remember the previous information. It has a memory cell encoding the memory of each time step. It has three gate, namely, forget gate f, input gate i, and output gate O. LSTM unit is represented as the following formula:

$$i^t = \sigma(W_{ix} * x^t + W_{ih} * h^{t-1} + W_{ic} * c^{t-1})$$



```
function LSTM.lstm(input_size, output_size, rho)
    local inputsize = input_size
    local outputsize = output_size
    local rho = rho
    local i2g = nn.Linear(inputsize, outputsize * 4)
    local o2g = nn.LinearNoBias(outputsize, outputsize * 4)
    local inputs = {}
    table.insert(inputs, nn.Identity()())
    table.insert(inputs, nn.Identity()())
    table.insert(inputs, nn.Identity()())
    local x, prev_h, prev_c = unpack(inputs)
    local i2h = i2g(x):annotate{name = 'i2h'}
    local h2h = o2g(prev_h):annotate{name = 'h2h'}
    local all_input_sums = nn.CAddTable()({i2h, h2h})
    local reshaped = nn.Reshape(4, outputsize)(all_input_sums)
    local n1, n2, n3, n4 = nn.SplitTable(1)(reshaped):split(4)
    local in_gate = nn.Sigmoid()(n1)
    local in_transform = nn.Tanh()(n2)
    local forget_gate = nn.Sigmoid()(n3)
    local out_gate = nn.Sigmoid()(n4)
    local in_data = nn.CMulTable()({forget_gate, prev_c})
    local in_gate_transform = nn.CMulTable()({in_gate, in_transform})
    local next_c = nn.CAddTable()({in_data, in_gate_transform})
    local next_h = nn.CMulTable()({out_gate, nn.Tanh()(next_c)})
    --model = nn.gModule(inputs, {i2h, h2h, prev_c})
    model = nn.gModule(inputs, {next_c, next_h})
    return model
end
```

Figure 1: The standard lstm unit using torch7. It uses nngraph module. It computes all the gates(forget gates, *in_gate*, and *out_gate* as well as transformation) in a single linear model

$$f^t = \sigma(W_{fx} * x^t + W_{fh} * h^{t-1} + W_{fc} * c^{t-1})$$

$$c^t = f^t \odot c^{t-1} + i^t \odot \phi(W_{cx} * X^t + W_{ch} * h^{t-1})$$

$$o^t = \sigma(W_{vx} * x^t + W_{oh} * ht - 1 + W_{oc} * c^t)$$

$$h^t = o^t \odot \phi(c^t)$$

Here σ represents sigmoid nonlinear function. $\odot$ represents element-wise multiplication. ϕ represents tanh nonlinear function.

In addition, Yi Shi et al present protein contact order can be effectively predicted from the primary sequence, at the absence of three-dimensional structure.[23] This is another way to predict the next protein sequence.

3.3 Data set and Data Preprocess

I use amino acid sequence data from Cytoplasm(C), Mitochondrion(M), Secretory(S) for sequence learning.

My data set select from release 48 of the SWISS-PROT database (Bairoch et al, 2005). In my data set, it has four class: Cytoplasm(C), Mitochondrion(M), Necleus(N) and Secretory(S). I pick three of four class as training set. The attach example is just one of training cases in cytoplasm, which represents part of amino acid on rat. Even in cytoplasm class, I test rat, human and drome with different component of amino acid sequences.

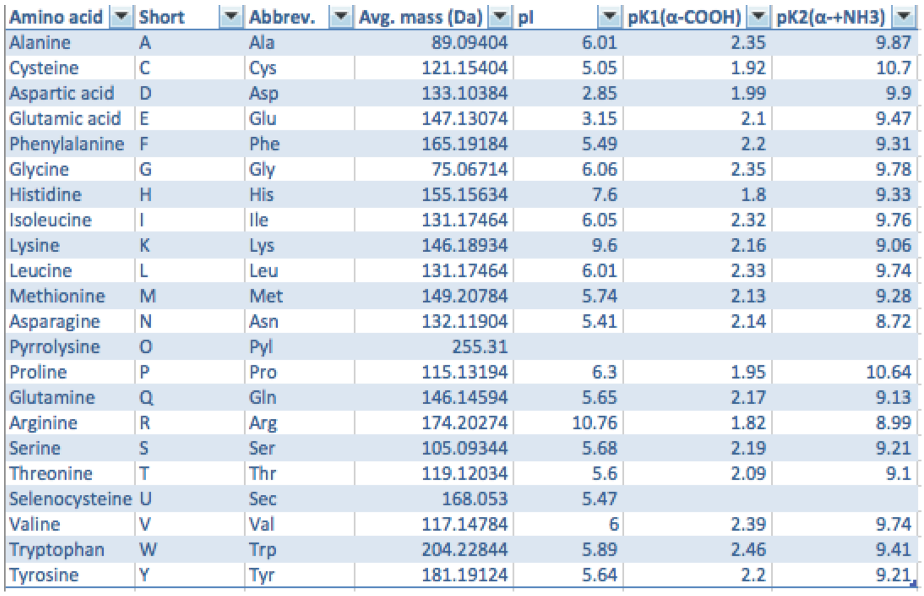
$> 2ABD\_RAT:$

$MAGAGGGGCPTGGNDFQWCFSQVKGAVDEDVAEADIISTVEFNYSGDLL$
$ATGDKGGRVVIFQREQENKGRAHSRGEYNVYSTFQSHEPEFDYLKSLEI$
$EEKINKIRWLPQQNAAHFLLSTNDKTIKLWKISERDKRAEGYNLKDEDG$
$RLRDPFRITALRVPILKPMDLMVEASPRRIFANAHTYHINSISVNSDHE$
$TYLSADDLRINLWHLEITDRSFNIVDIKPANMEELTEVITAAEFHPHQC$
$NVFVYSSSKGTIRLCDMRSSALCDRHAKFFEEPEDPSSRSFFSEIISSI$
$SDVKFSHSGRYMMTRDYLSVKVWDLNMEGRPVETHHVHEYLRSKLCSLY$
$ENDCIFDKFECCWNGSDSAIMTGSYNNFFRMFDRNTRRDVTLEASRENS$
$KPRASLKPRKVCSGGKRKKDEISVDSLDFNKKILHTAWHPMESIIAVAA$
$TNNLYIFQDKIN$

In this test case, each character means one kind of amino acids, the figure 1 show us the total standard amino acid.

3.4 Experiment

In this section, I will describe experiment. I did two categories of experiments. (1) I train traditional RNN model on a small amino acid sequence, namely $COA1\_BOVIN$. I want to learn the ability of the model to learn from a small



Amino acid	Short	Abbrev.	Avg. mass (Da)	pI	pK1(α-COOH)	pK2(α-+NH3)
Alanine	A	Ala	89.09404	6.01	2.35	9.87
Cysteine	C	Cys	121.15404	5.05	1.92	10.7
Aspartic acid	D	Asp	133.10384	2.85	1.99	9.9
Glutamic acid	E	Glu	147.13074	3.15	2.1	9.47
Phenylalanine	F	Phe	165.19184	5.49	2.2	9.31
Glycine	G	Gly	75.06714	6.06	2.35	9.78
Histidine	H	His	155.15634	7.6	1.8	9.33
Isoleucine	I	Ile	131.17464	6.05	2.32	9.76
Lysine	K	Lys	146.18934	9.6	2.16	9.06
Leucine	L	Leu	131.17464	6.01	2.33	9.74
Methionine	M	Met	149.20784	5.74	2.13	9.28
Asparagine	N	Asn	132.11904	5.41	2.14	8.72
Pyrrolysine	O	Pyl	255.31			
Proline	P	Pro	115.13194	6.3	1.95	10.64
Glutamine	Q	Gln	146.14594	5.65	2.17	9.13
Arginine	R	Arg	174.20274	10.76	1.82	8.99
Serine	S	Ser	105.09344	5.68	2.19	9.21
Threonine	T	Thr	119.12034	5.6	2.09	9.1
Selenocysteine	U	Sec	168.053	5.47		
Valine	V	Val	117.14784	6	2.39	9.74
Tryptophan	W	Trp	204.22844	5.89	2.46	9.41
Tyrosine	Y	Tyr	181.19124	5.64	2.2	9.21

Figure 2: It is a table listing the one-letter symbols, the three-letter symbols, and the chemical properties of the standard amino acids. [1]

dataset and generate sequence that is very close to that of the training set. (2) I train on a relatively large sequence data set, and test the ability of RNN in learning the sequence and generate new sequence.

3.5 Experiment 1

3.5.1 training

Part of the training sequence is shown below:

$MDEPSSLAKPLELNQHSRFIIGS$

$VSEDNSEDEISNLVKLDLLEEKE$

$GSLSPASVSSDTLSDLGISSLQD$

$GLALHMRSSMSGLHLVKQGRDRK$

$KIDSQRDFTVASPAEFVTRFGGN$

$KVIEKVLIANNGIAAVKCMRSIR$

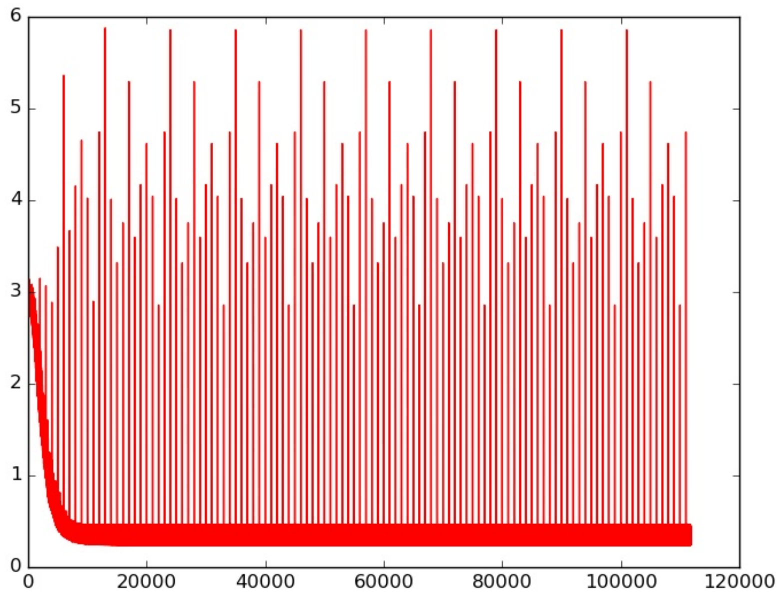


Figure 3: The training loss of lstm model on the specific sequence. The x axis is the time of training, and the number represent number of batch that has been trained. y axis is the loss.

$RWSYEMFRNERAIRFVVMVTPED$

$LKANAEYIKMADHYVPVPGGPNN$

$NNYANVELILDIAKRIPVQAVWA$

$GWGHASENPKLPELLLKNGIAFM$

$GPPSQAMWALGDKIASSIVAQTA$

$GIPTLPWSGSGLCVDWHENDFSK$

$RILNVPQELYEKGYVKDVDDGLK$

$AAEEVGYPV...$

The training loss of the experiment is shown in figure

3.5.2 testing

I use the first 10 elements(MDEPSSLAKP) in the sequence to initialize my model. The predicted sequence is shown below:

$MDEPSSLAKPLEHLGRKIIEMNV$

$DSTTITIEQPFGEAQINLLLRAA$

$QTTSLPSPYLLNQLMNDKIIRTS$

$NATLGIFGEQEVQGMSSVVFLRL$

$ADGKIIRTIAVVTPNLRRHKNVE$

$EVEEAFRYYVVNGKLNKLQDGSR$

$VFFGHPELLMRMSSSAMPLYESS$

$RIIIHFRRFSTTVEELIRRYVII$

$ERLGGRRRVAKKRERAGRDVDVE$

$LDGEAIIEIHVKNSLTGAGPIIE$

$ESMRLMMAGLCPQQQQARYMVHH$

$LDFLGTGYNGLTMPEFQTYVVSR$

$ASNAGKAKVEVEAEDGYYVLAEE$

$GKKKKMRMRMKSSVPSPWNHDPS$

$KYKMELEMDSKRIGTMVNENFYE$

$VFRNNGCIMDRMRVAAAVFSEAA$

$RILERFRRHHWVVAVPPRAILEG$

$PIIMKMAVSDDSGAAEEVFFHHY$

$VYPPDLPFGKKYVAAREMSPGSN$

I can compare the difference between the original sequence and the generated sequence.

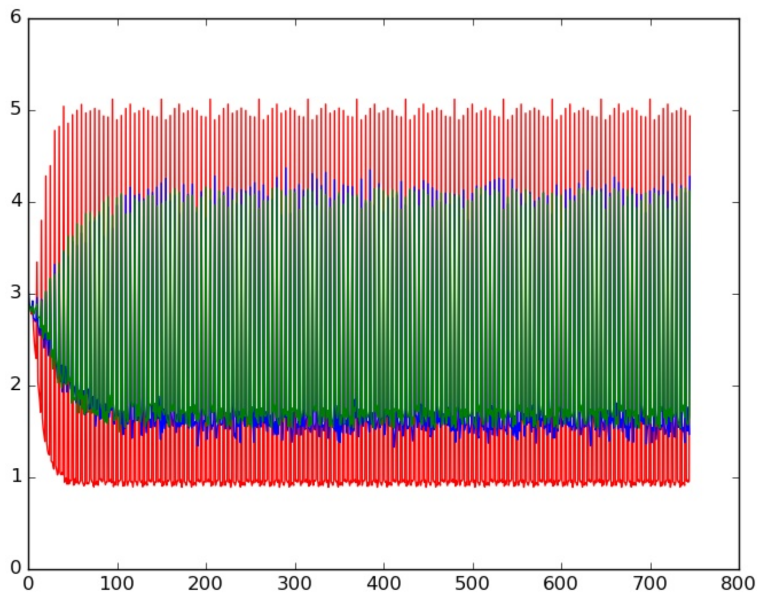


Figure 4: training loss of three different amino acid sequence datasets. Red: Training loss for Mitochondrion dataset. Blue: Training loss Secretory for Scretory dataset. Green: Training loss for Cytoplasm dataset. The x axis is the time of training, and the number represent number of batch that has been trained. y axis is the loss. The lower the loss means the model is getting better. However, there is some jump in loss during training. The model is not stable

3.6 Experiment 2

3.6.1 training

In this experiment, I tested generating amino acid sequence by LSTM with several different, relatively large animal amino acid sequence data set. The lstm parameters I use is shown in table. I train the lstm on amino sequence data set from different animal. In the training I used a two layer lstm model with 256 size of hidden state. The training loss curve with C,S,M data set is shown in figure 11.

I could see that there are still some problem in the training as all the curve some sudden jump of its value. I am currently checking the training process. There might be two reasons. (1) There are is no certain pattern in the training data set. (2) Since I merged several different amino sequences for all three datasets, the pattern might be ruined.

3.6.2 testing

In the testing, I tested by generating sequence from the trained model.

Sequence generated by model trained in Cytoplasm sequence:

$SQDKERPQSSPVVEPQPEPPKSLPPKGPIPVSKP$

$KASTPKSKSPSSSHSGLSSPAKPTSPFSNFPIQI$

$SIKPHSSSTSSSSAGSGSSPVSFGRSVLPRMPGP$

$SQLHQEHAPPPQPPPPPPPEMTPPQPQPGIQMHL$

Sequence generated by model trained on Mitochondrion sequence:

$QLEKFNQLLSIRPHNPIGDLDMRILSCMMKNRGL$

$GGSIRFRMTQMSLCPQPRFHRSHVSTGLLCGLPQ$

$QDPVKRYTWGRVPNADLRRQLRAFSEWKKLNECL$

$EKDDRDKADLAKTSVLLDHATYGHKDVHHHYREE$

Sequence generated by model trained on Secretory sequence:

$QLEKFNQLLSIRPHNPIGDLDMRILSCMMKNRGL$

$GGSIRFRMTQMSLCPQPRFHRSHVSTGLLCGLPQ$

$QDPVKRYTWGRVPNADLRRQLRAFSEWKKLNECL$

$EKDDRDKADLAKTSVLLDHATYGHKDVHHHYREE$

3.7 Conclusion

In this part, I explore the use of standard deep learning method, recurrent neural network to generate amino acid sequence. I did two experiment. I first trained the model on a small data set and try to predict the sequence on the trained model. Second, I trained our model on 3 relatively large, different amino acid data set and test them on generating sequence. I also compared the training loss of the three different data set. The training loss could be used to show whether there is certain pattern in the data set. Data set with clear pattern should be able generate smaller training loss. This is just an initial work of using deep neural network in generating amino acid sequences. More work can be done in this area, such as sequence to sequence model in transforming DNA to protein sequence.

4 Programming Language Part

4.1 Introduction

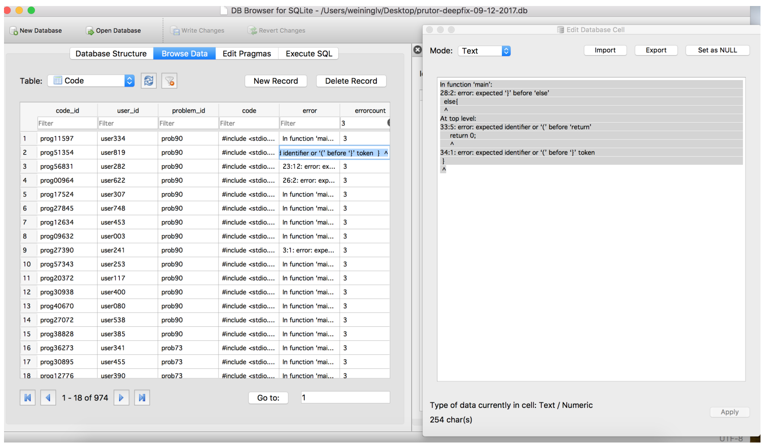
It still a new territory that try to combine programming language and machine learning together. Many researchers did some heuristic and conception work so far. We could think this part as neural program embedding. It could focus on many problem, such as program synthesis, program repair, code completion,fault localization and similarity detection. The key problem is how could we design a suitable program which could represent for the networks to exploit. Based on the idea of deepfix[8], I also use partial of codes, data set and same metric on my module, check more details on method part. Furthermore, instead of using part of algorithm on Word2Vec, I use GloVe to did the step about transfer token to tensors. My implementation is based on tensorflow.

4.2 Related Work

Mou et al.[16] did the first work that use convolutional neural network and abstract syntax trees together in order to classify programs. Piech et al[18] supposed a novel idea that focus on input-output pairs and linear mapping method try to simulate some feedback information. DeepFix (Gupta et al, 2017), SynFix (Bhatia Singh, 2016), and *skp* (Pu et al, 2016) are recent neural program repair techniques for correcting errors in student programs for MOOC assignments, all of them put code to tokens and use some deep learning model to classify and recognize error.

4.3 Data set and Data Preprocess

I directly use the dataset from paper deepfix. It has two classes of programs, either the compile programs or the erroneous programs. It totally has 53478 code problem and stored by sqlite3. It has code ID, user ID, code, error message and number of errors. Fig.5 shows the structure of database.



	code_id	user_id	problem_id	code	error	errorcount
	Filter	Filter	Filter	Filter	Filter	3
1	prog11597	user334	prob90	#include <stdio....	In function 'mai...	3
2	prog51354	user819	prob90	#include <stdio....	identifier or '(' before '}' token } ^	
3	prog56831	user282	prob90	#include <stdio....	23:12: error: ex...	3
4	prog00964	user622	prob90	#include <stdio....	26:2: error: exp...	3
5	prog17524	user307	prob90	#include <stdio....	In function 'mai...	3
6	prog27845	user748	prob90	#include <stdio....	In function 'mai...	3
7	prog12634	user453	prob90	#include <stdio....	In function 'mai...	3
8	prog09632	user003	prob90	#include <stdio....	In function 'mai...	3
9	prog27390	user241	prob90	#include <stdio....	3:1: error: expe...	3
10	prog57343	user253	prob90	#include <stdio....	In function 'mai...	3
11	prog20372	user117	prob90	#include <stdio....	In function 'mai...	3
12	prog30938	user400	prob90	#include <stdio....	In function 'mai...	3
13	prog40670	user080	prob90	#include <stdio....	In function 'mai...	3
14	prog27072	user538	prob90	#include <stdio....	In function 'mai...	3
15	prog38828	user385	prob90	#include <stdio....	In function 'mai...	3
16	prog36273	user341	prob73	#include <stdio....	In function 'mai...	3
17	prog30895	user455	prob73	#include <stdio....	In function 'mai...	3
18	prog12776	user390	prob73	#include <stdio....	In function 'mai...	3

1 - 18 of 974   Go to: 1

```
In function 'main':
28:2: error: expected '}' before 'else'
 else{
 ^
At top level:
33:5: error: expected identifier or '(' before 'return'
    return 0;
    ^
34:1: error: expected identifier or '(' before '}' token
 }
 ^
```

Type of data currently in cell: Text / Numeric
254 char(s)

Figure 5: Database

4.4 Method and Result

4.4.1 Tokenization

For the compiler phases, we could separate to front end and back end. In my project, I'm interested in front end, which include lexical analysis, syntax analysis, and semantic analysis. In my project, the first step is that I need to translate programming code to token. In simple terms, we need to use some regular expression to reformat and structure it to token. It already has a lot of mature tool help researchers work on that, such as JavaCC, Yacc, Lex and pycparser. The architecture of compiler is shown in Fig.6 and the core part about lexical analysis is shown in Fig.7.

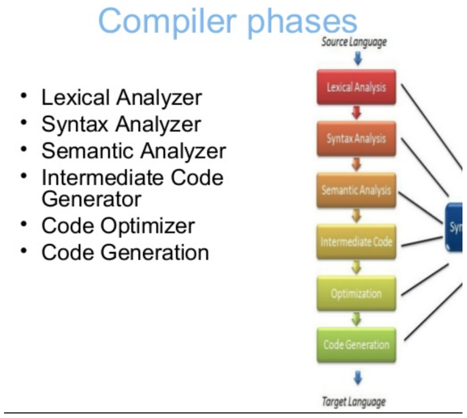
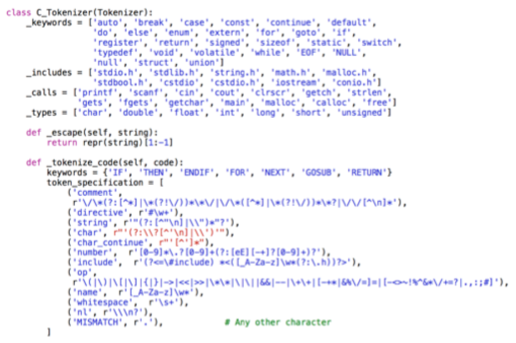


Figure 6: Compiler Phases



```
class C_Tokenizer(Tokenizer):
    _keywords = ['auto', 'break', 'case', 'const', 'continue', 'default',
                 'do', 'else', 'enum', 'extern', 'for', 'goto', 'if',
                 'register', 'return', 'signed', 'sizeof', 'static', 'switch',
                 'typedef', 'void', 'volatile', 'while', 'EOF', 'NULL',
                 'null', 'struct', 'union']
    _includes = ['stdio.h', 'stdlib.h', 'string.h', 'math.h', 'malloc.h',
                 'stdbool.h', 'cstdio', 'cstdio.h', 'iostream', 'conio.h']
    _calls = ['printf', 'scanf', 'cin', 'cout', 'clrscr', 'getch', 'strlen',
              'gets', 'fgets', 'getchar', 'main', 'malloc', 'calloc', 'free']
    _types = ['char', 'double', 'float', 'int', 'long', 'short', 'unsigned']

    def _escape(self, string):
        return repr(string)[1:-1]

    def _tokenize_code(self, code):
        keywords = {'IF', 'THEN', 'ENDIF', 'FOR', 'NEXT', 'GOSUB', 'RETURN'}
        token_specification = [
            ('comment',
             r'\/\*(?:[^*]|\*(?!\/))*\*\/|\/\*([^*]|\*(?!\/))*\*?|\/\/[^\n]*'),
            ('directive', r'#\w+'),
            ('string', r'"(?:[^"\n]|\\")*"?'),
            ('char', r"'(?:\\?[^'\n]|\\')'"),
            ('char_continue', r"'[^']*"),
            ('number',  r'[0-9]*\.?[0-9]+(?:[eE][-+]?[0-9]+)?'),
            ('include',  r'(?<=\#include) *<([_A-Za-z]\w*(?:\.h))?>'),
            ('op',
             r'\(|\)|\[|\]|{|}|->|<<|>>|\*\*|\|\||&&|--|\+\+|[-+*|&%\/=]=|[-<>~!%^&*\/+=?|.,:;#]'),
            ('name',  r'[_A-Za-z]\w*'),
            ('whitespace',  r'\s+'),
            ('nl', r'\\\n?'),
            ('MISMATCH', r'.'),           # Any other character
        ]
```

Figure 7: Lexical Analysis

4.4.2 GloVe

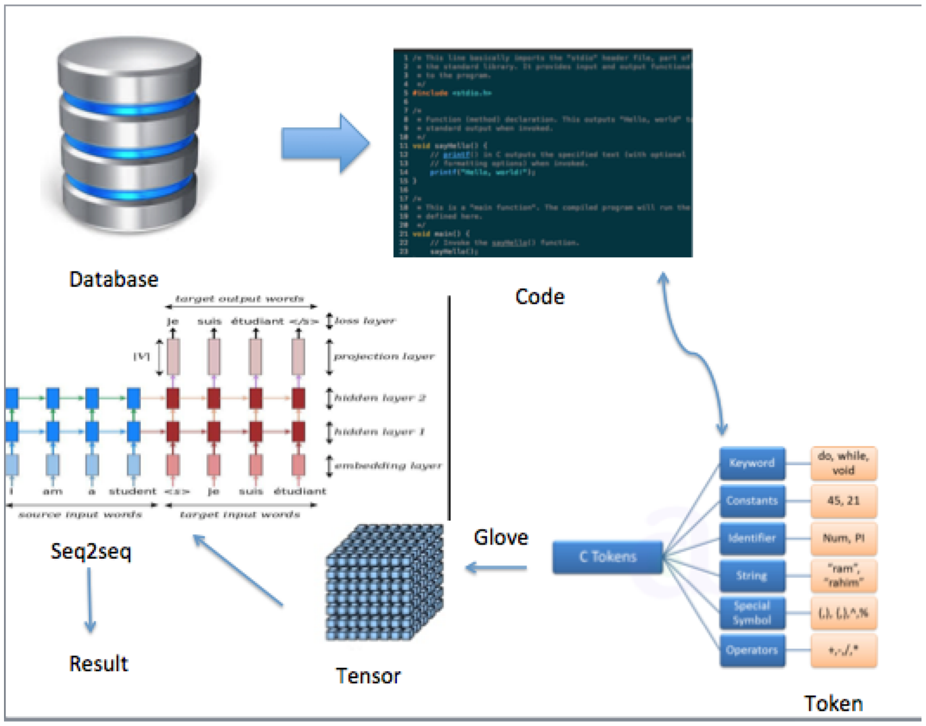
GloVe [17] is a fancy method translate word to vector. It has two core section, global matrix factorization and local context window methods. Compare with most word vector which rely on the distance or angle between pairs of word, GloVe and Word2Vec focus on various dimensions of difference. Furthermore, it focus on ratios of co-occurrence probabilities rather than the probabilities themselves. It has a couple of version about GloVe, I use and modify GradySimon's tf-GloVe[2] and transfer token to tensor.

4.4.3 Seq2Seq

Sutskever et al. [22] suppose a novel model which enjoyed great success in a variety of tasks such as machine translation, speech recognition, and text summarization. Based on the idea of LSTM,these method use a multi-layered LSTM to map the input sequence to a vector of a fixed dimensionality, and then another deep LSTM to decode the target sequence from the vector.

4.4.4 Structure

First of all, I extract code from database then transfer code to tokens. Second, I use GloVe to do a code2Vec steps, which means change tokens to tensors. Third, I use the seq2seq model training and testing. Finally, I could automatically repair some of common c error. The architecture of this system shown in Fig.8.



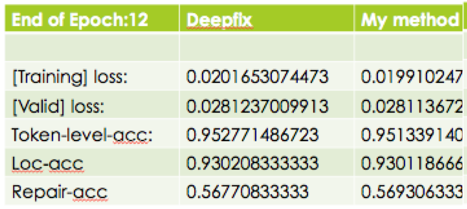
```
/* This line basically imports the "stdio" header file, part of
 * the standard library. It provides input and output functional
 * to the program.
 */
#include <stdio.h>

/*
 * Function (method) declaration. This outputs "Hello, world" t
 * standard output when invoked.
 */
void sayHello() {
    // printf() in C outputs the specified text (with optional
    // formatting options) when invoked.
    printf("Hello, world!");
}

/*
 * This is a "main function". The compiled program will run the
 * defined here.
 */
void main() {
    // Invoke the sayHello() function.
    sayHello();
```

Figure 8: The structure of find common c error system

4.4.5 Result

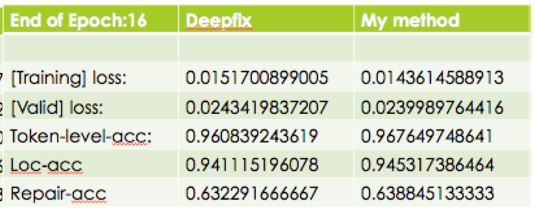
In this section, I will describe experiment. I did two categories of experiments. (1) I train and test 1-fold sequence data set with method deepfix. (2) I train and test 1-fold sequence data set with my method. I also try to train the whole sequence data set and when the steps come to 31000 with more than five days on my server, the model is over-fitting.

Here is some of results about training:



End of Epoch:12	Deepfix	My method
[Training] loss:	0.0201653074473	0.019910247
[Valid] loss:	0.0281237009913	0.028113672
Token-level-acc:	0.952771486723	0.951339140
Loc-acc	0.930208333333	0.930118666
Repair-acc	0.56770833333	0.569306333

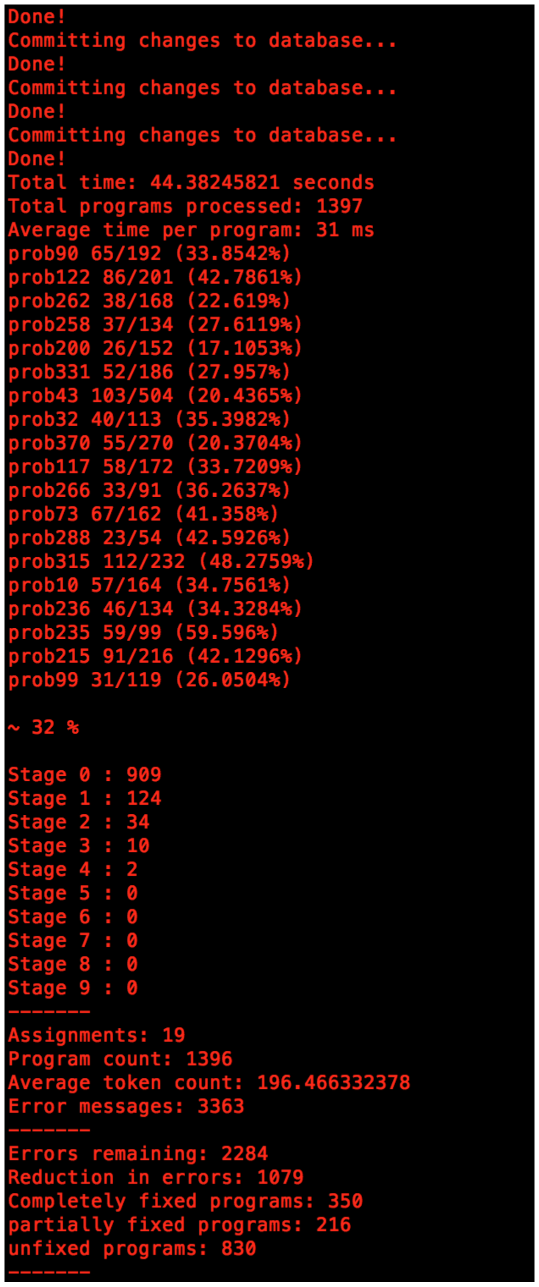
Figure 9: End of epoch: 12



End of Epoch:16	Deepfix	My method
[Training] loss:	0.0151700899005	0.0143614588913
[Valid] loss:	0.0243419837207	0.0239989764416
Token-level-acc:	0.960839243619	0.967649748641
Loc-acc	0.941115196078	0.945317386464
Repair-acc	0.632291666667	0.638845133333

Figure 10: End of epoch: 16

My testing results shown in Fig.11. Compare different results about training and testing, it looks no huge difference either use Word2Vec or GloVe method when we figure out this problem, I still have strong intuition that should has more structure and meaningful way work for code2Vec. In the testing part, it shows us the problem Id and the percentage of fix each programming bug. The results are not good enough and these data set is not fitting for real industry, most of error are typo error, such as try to fix a semicolon or equal mark miss.



```
Done!
Committing changes to database...
Done!
Committing changes to database...
Done!
Committing changes to database...
Done!
Total time: 44.38245821 seconds
Total programs processed: 1397
Average time per program: 31 ms
prob90 65/192 (33.8542%)
prob122 86/201 (42.7861%)
prob262 38/168 (22.619%)
prob258 37/134 (27.6119%)
prob200 26/152 (17.1053%)
prob331 52/186 (27.957%)
prob43 103/504 (20.4365%)
prob32 40/113 (35.3982%)
prob370 55/270 (20.3704%)
prob117 58/172 (33.7209%)
prob266 33/91 (36.2637%)
prob73 67/162 (41.358%)
prob288 23/54 (42.5926%)
prob315 112/232 (48.2759%)
prob10 57/164 (34.7561%)
prob236 46/134 (34.3284%)
prob235 59/99 (59.596%)
prob215 91/216 (42.1296%)
prob99 31/119 (26.0504%)

~ 32 %

Stage 0 : 909
Stage 1 : 124
Stage 2 : 34
Stage 3 : 10
Stage 4 : 2
Stage 5 : 0
Stage 6 : 0
Stage 7 : 0
Stage 8 : 0
Stage 9 : 0
-------
Assignments: 19
Program count: 1396
Average token count: 196.466332378
Error messages: 3363
-------
Errors remaining: 2284
Reduction in errors: 1079
Completely fixed programs: 350
partially fixed programs: 216
unfixed programs: 830
-------
```

Figure 11: Testing results

4.5 Conclusion

In this part, I explore the use of another standard deep learning method, seq2seq with GloVe and get meaningful result.This is just an initial work of using deep neural network in programming language. Many work can be done in the future, in my experience, it still has lot of things to do, either focus on static analysis or dynamic analysis with machine learning. For example, how could we design a fancy method, translate more structure programming code to tensor is still a big issue. In addition, how could we use some semantic analysis, such as data dependence and control dependencies and use some statistics way to tackle some of traditional problem is another issue.

References

[1] Amino Sequence kernel description. https://en.wikipedia.org/wiki/Proteinogenic$_amino_acid.Accessed$: $2010-09-30$.

[2] Tf-glove. https://github.com/GradySimon/tensorflow-glove.

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[4] Travers Ching, Daniel S Himmelstein, Brett K Beaulieu-Jones, Alexandr A Kalinin, Brian T Do, Gregory P Way, Enrico Ferrero, Paul-Michael Agapow, Michael Zietz, Michael M Hoffman, et al. Opportunities and obstacles for deep learning in biology and medicine. *bioRxiv*, page 142760, 2018.

[5] Ronan Collobert, Koray Kavukcuoglu, and Clément Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, number EPFL-CONF-192376, 2011.

[6] Alan Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 6645–6649. IEEE, 2013.

[7] Alex Graves, Marcus Liwicki, Santiago Fernández, Roman Bertolami, Horst Bunke, and Jürgen Schmidhuber. A novel connectionist system for unconstrained handwriting recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 31(5):855–868, 2009.

[8] Rahul Gupta, Soham Pal, Aditya Kanade, and Shirish Shevade. Deepfix:

Fixing common c language errors by deep learning. In *AAAI*, pages 1345–1351, 2017.

[9] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[10] Neel Kant. Recent advances in neural program synthesis. *arXiv preprint arXiv:1802.02353*, 2018.

[11] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[12] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[13] Fan Long and Martin Rinard. Automatic patch generation by learning correct code. *ACM SIGPLAN Notices*, 51(1):298–312, 2016.

[14] Andrea Mesa, Sebastián Basterrech, Gustavo Guerberoff, and Fernando Alvarez-Valin. Hidden markov models for gene sequence classification. *Pattern Analysis and Applications*, pages 1–13, 2015.

[15] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Eleventh Annual Conference of the International Speech Communication Association*, 2010.

[16] Lili Mou, Ge Li, Lu Zhang, Tao Wang, and Zhi Jin. Convolutional neural networks over tree structures for programming language processing. In *AAAI*, volume 2, page 4, 2016.

[17] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[18] Chris Piech, Jonathan Huang, Andy Nguyen, Mike Phulsuksombati, Mehran Sahami, and Leonidas Guibas. Learning program embeddings to propagate feedback on student code. *arXiv preprint arXiv:1505.05969*, 2015.

[19] Veselin Raychev, Martin Vechev, and Andreas Krause. Predicting program properties from big code. In *ACM SIGPLAN Notices*, volume 50, pages 111–124. ACM, 2015.

[20] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic routing between capsules. In *Advances in Neural Information Processing Systems*, pages 3859–3869, 2017.

[21] Richard Socher, Cliff C Lin, Chris Manning, and Andrew Y Ng. Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 129–136, 2011.

[22] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[23] David Arndt David S WishartEmail author Yi Shi, Jianjun Zhou and Guohui LinEmail author. Protein contact order prediction from primary sequences. *BMC Bioinformatics*, 22(14), 2008.