

# The Composition of Dense Neural Networks and Formal Grammars for Secondary Structure Analysis

Polina Lunina, **Semyon Grigorev**

JetBrains Research, Programming Languages and Tools Lab  
Saint Petersburg University

February 24, 2019

## Problem: high variability of data

- Mutations
- Different kinds of noise
- ...

**Problem: high variability of data**

- Mutations
- Different kinds of noise
- ...

**We need probabilistic approaches**

# Sequences Analysis: classification, search...

## Problem: high variability of data

- Mutations
- Different kinds of noise
- ...

## Level of abstraction

- Primary structure
- Secondary structure
- Tertiary structure

**We need probabilistic approaches**

# Sequences Analysis: classification, search...

## Problem: high variability of data

- Mutations
- Different kinds of noise
- ...

We need probabilistic approaches

## Level of abstraction

- Primary structure
- Secondary structure
- Tertiary structure

We should handle secondary structure

# Sequences Analysis: classification, search...

## Problem: high variability of data

- Mutations
- Different kinds of noise
- ...

We need probabilistic approaches

## Level of abstraction

- Primary structure
- Secondary structure
- Tertiary structure

We should handle secondary structure

Probabilistic approach which handles secondary structure

# Proposed Solution: Parsing + Artificial Neural Network

- Use parsing to extract features, not to model secondary structure
  - ▶ As compared to the classical way of probabilistic CF grammars utilization

# Proposed Solution: Parsing + Artificial Neural Network

- Use parsing to extract features, not to model secondary structure
  - ▶ As compared to the classical way of probabilistic CF grammars utilization
- Formal grammars as secondary structure description
- Parsing as features extraction
- Artificial neural network as probabilistic model for features processing



# Solution Structure

## Grammar

Fixed formal grammar (not necessarily context-free) describes features of secondary structure and can be tuned to increase the quality of result.

## Sequences

Each sequence is treated as a text in  $\{A, C, G, T\}$  alphabet.

## Result of classification

## Parser

Parser extracts features of the given sequence secondary structure. Implementation of parsing algorithm is based on matrix multiplications (Valiant, Okhotin) and utilizes GPGPU.

## Neural Network

Dense neural network with more than 10 dense layers. Aggressive dropout and batch normalization for learning process stabilization. Typical building block:

Dropout (75%)	input: 1024 output: 1024
------------------	-----------------------------

Dense	input: 1024 output: 1024
-------	-----------------------------

BatchNormalization	input: 1024 output: 1024
--------------------	-----------------------------

Activation (relu)	input: 1024 output: 1024
----------------------	-----------------------------

## Matrices

0	1	0	1
0	0	1	0
0	0	0	1
0	0	0	0

Parsing result is (0-1) matrix  $M$  which represents secondary structure features for sequence  $\omega$ :

$M[i, j] = 1 \iff s1 \xrightarrow{*} \omega[i, j]$ ,  
and 0 otherwise.

## Vectors

0	1	0	1
0	0	1	0
0	0	0	1
0	0	0	0

$\Rightarrow$

$[0, 1, 0, 1, 0, 1, 0, 0, 1, 0] \Rightarrow$   
 $[84, 128]$

Line-by-line compressed matrix representation: sequence of 8 cells (bits) is compressed into a byte. Bottom left triangle of the matrix is always empty, so can be ignored.

# Solution Structure

## Grammar

Fixed formal grammar (not necessarily context-free) describes features of secondary structure and can be tuned to increase the quality of result.

## Sequences

Each sequence is treated as a text in  $\{A, C, G, T\}$  alphabet.

## Result of classification

## Parser

Parser extracts features of the given sequence secondary structure. Implementation of parsing algorithm is based on matrix multiplications (Valiant, Okhotin) and utilizes GPGPU.

## Neural Network

Dense neural network with more than 10 dense layers. Aggressive dropout and batch normalization for learning process stabilization. Typical building block:

Dropout (75%)	input: 1024 output: 1024
------------------	-----------------------------

Dense	input: 1024 output: 1024
-------	-----------------------------

BatchNormalization	input: 1024 output: 1024
--------------------	-----------------------------

Activation (relu)	input: 1024 output: 1024
----------------------	-----------------------------

## Matrices

0	1	0	1
0	0	1	0
0	0	0	1
0	0	0	0

Parsing result is (0-1) matrix  $M$  which represents secondary structure features for sequence  $\omega$ :

$M[i, j] = 1 \iff s1 \xrightarrow{*} \omega[i, j]$ ,  
and 0 otherwise.

## Vectors

0	1	0	1
0	0	1	0
0	0	0	1
0	0	0	0

$\Rightarrow$

$[0, 1, 0, 1, 0, 1, 0, 0, 1, 0] \Rightarrow$   
 $[84, 128]$

Line-by-line compressed matrix representation: sequence of 8 cells (bits) is compressed into a byte. Bottom left triangle of the matrix is always empty, so can be ignored.

# Solution Structure

## Grammar

Fixed formal grammar (not necessarily context-free) describes features of secondary structure and can be tuned to increase the quality of result.

## Sequences

Each sequence is treated as a text in  $\{A, C, G, T\}$  alphabet.

## Result of classification

## Parser

Parser extracts features of the given sequence secondary structure. Implementation of parsing algorithm is based on matrix multiplications (Valiant, Okhotin) and utilizes GPGPU.

## Neural Network

Dense neural network with more than 10 dense layers. Aggressive dropout and batch normalization for learning process stabilization. Typical building block:

Dropout (75%)	input: 1024 output: 1024
------------------	-----------------------------

Dense	input: 1024 output: 1024
-------	-----------------------------

BatchNormalization	input: 1024 output: 1024
--------------------	-----------------------------

Activation (relu)	input: 1024 output: 1024
----------------------	-----------------------------

## Matrices

0	1	0	1
0	0	1	0
0	0	0	1
0	0	0	0

Parsing result is (0-1) matrix  $M$  which represents secondary structure features for sequence  $\omega$ :

$M[i, j] = 1 \iff s1 \xrightarrow{*} \omega[i, j]$ , and 0 otherwise.

## Vectors

0	1	0	1
0	0	1	0
0	0	0	1
0	0	0	0

$\Rightarrow$

$[0, 1, 0, 1, 0, 1, 0, 0, 1, 0] \Rightarrow [84, 128]$

Line-by-line compressed matrix representation: sequence of 8 cells (bits) is compressed into a byte. Bottom left triangle of the matrix is always empty, so can be ignored.

# Solution Structure

## Grammar

Fixed formal grammar (not necessarily context-free) describes features of secondary structure and can be tuned to increase the quality of result.

## Sequences

Each sequence is treated as a text in  $\{A, C, G, T\}$  alphabet.

## Result of classification

## Parser

Parser extracts features of the given sequence secondary structure. Implementation of parsing algorithm is based on matrix multiplications (Valiant, Okhotin) and utilizes GPGPU.

## Neural Network

Dense neural network with more than 10 dense layers. Aggressive dropout and batch normalization for learning process stabilization. Typical building block:

Dropout (75%)	input: 1024 output: 1024
------------------	-----------------------------

Dense	input: 1024 output: 1024
-------	-----------------------------

BatchNormalization	input: 1024 output: 1024
--------------------	-----------------------------

Activation (relu)	input: 1024 output: 1024
----------------------	-----------------------------

## Matrices

0	1	0	1
0	0	1	0
0	0	0	1
0	0	0	0

Parsing result is (0-1) matrix  $M$  which represents secondary structure features for sequence  $\omega$ :

$M[i, j] = 1 \iff s1 \xrightarrow{*} \omega[i, j]$ ,  
and 0 otherwise.

## Vectors

0	1	0	1
0	0	1	0
0	0	0	1
0	0	0	0

$\Rightarrow$

$[0, 1, 0, 1, 0, 1, 0, 0, 1, 0] \Rightarrow$   
 $[84, 128]$

Line-by-line compressed matrix representation: sequence of 8 cells (bits) is compressed into a byte. Bottom left triangle of the matrix is always empty, so can be ignored.

# Solution Structure

## Grammar

Fixed formal grammar (not necessarily context-free) describes features of secondary structure and can be tuned to increase the quality of result.

## Sequences

Each sequence is treated as a text in  $\{A, C, G, T\}$  alphabet.

## Result of classification

## Parser

Parser extracts features of the given sequence secondary structure. Implementation of parsing algorithm is based on matrix multiplications (Valiant, Okhotin) and utilizes GPGPU.

## Neural Network

Dense neural network with more than 10 dense layers. Aggressive dropout and batch normalization for learning process stabilization. Typical building block:

Dropout (75%)	input: 1024 output: 1024
------------------	-----------------------------

Dense	input: 1024 output: 1024
-------	-----------------------------

BatchNormalization	input: 1024 output: 1024
--------------------	-----------------------------

Activation (relu)	input: 1024 output: 1024
----------------------	-----------------------------

## Matrices

0	1	0	1
0	0	1	0
0	0	0	1
0	0	0	0

Parsing result is (0-1) matrix  $M$  which represents secondary structure features for sequence  $\omega$ :

$M[i, j] = 1 \iff s1 \xrightarrow{*} \omega[i, j]$ ,  
and 0 otherwise.

## Vectors

0	1	0	1
0	0	1	0
0	0	0	1
0	0	0	0

$\Rightarrow$

$[0, 1, 0, 1, 0, 1, 0, 0, 1, 0] \Rightarrow$   
 $[84, 128]$

Line-by-line compressed matrix representation: sequence of 8 cells (bits) is compressed into a byte. Bottom left triangle of the matrix is always empty, so can be ignored.

# Solution Structure

## Grammar

Fixed formal grammar (not necessarily context-free) describes features of secondary structure and can be tuned to increase the quality of result.

## Sequences

Each sequence is treated as a text in  $\{A, C, G, T\}$  alphabet.

## Result of classification

## Parser

Parser extracts features of the given sequence secondary structure. Implementation of parsing algorithm is based on matrix multiplications (Valiant, Okhotin) and utilizes GPGPU.

## Neural Network

Dense neural network with more than 10 dense layers. Aggressive dropout and batch normalization for learning process stabilization. Typical building block:

Dropout (75%)	input: 1024 output: 1024
------------------	-----------------------------

Dense	input: 1024 output: 1024
-------	-----------------------------

BatchNormalization	input: 1024 output: 1024
--------------------	-----------------------------

Activation (relu)	input: 1024 output: 1024
----------------------	-----------------------------

## Matrices

0	1	0	1
0	0	1	0
0	0	0	1
0	0	0	0

Parsing result is (0-1) matrix  $M$  which represents secondary structure features for sequence  $\omega$ :

$M[i, j] = 1 \iff s1 \xrightarrow{*} \omega[i, j]$ ,  
and 0 otherwise.

## Vectors

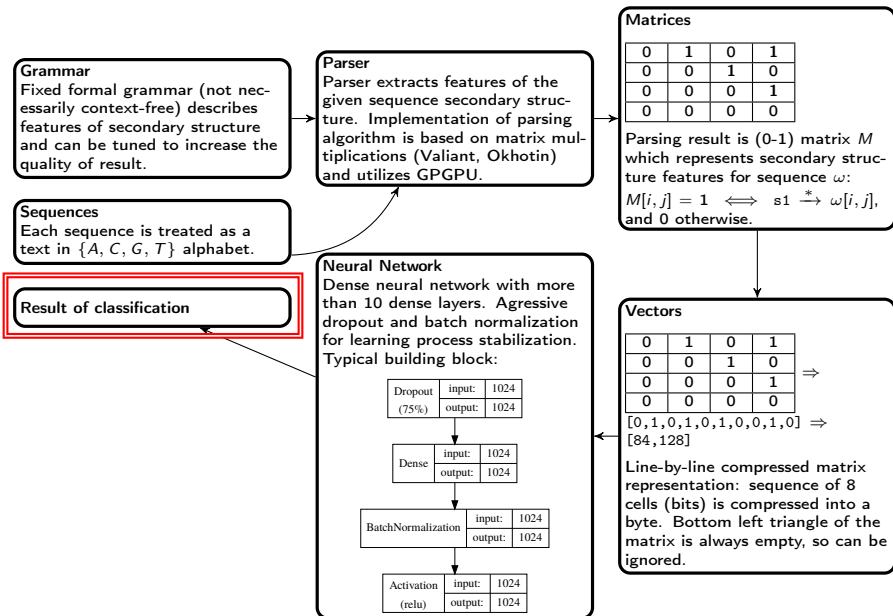
0	1	0	1
0	0	1	0
0	0	0	1
0	0	0	0

$\Rightarrow$

$[0, 1, 0, 1, 0, 1, 0, 0, 1, 0] \Rightarrow$   
 $[84, 128]$

Line-by-line compressed matrix representation: sequence of 8 cells (bits) is compressed into a byte. Bottom left triangle of the matrix is always empty, so can be ignored.

# Solution Structure



# Grammar

```
s1: stem<s0>
any_str: any_smb*[2..10]
any_smb: A | T | C | G
s0: any_str | any_str stem<s0> s0
stem1<s>:                \\ stem of height exactly 1
    A s T | G s C | T s A | C s G
stem2<s>:                \\ stem of height exactly 2
    stem1< stem1<s> >
stem<s>:                \\ stem of height 3 or more
    A stem<s> T
    | T stem<s> A
    | C stem<s> G
    | G stem<s> C
    | stem1< stem2<s> >
```



# Grammar

```
s1: stem<s0>
```

```
any_str: any_smb*[2..10]
```

```
any_smb: A | T | C | G
```

```
s0: any_str | any_str stem<s0> s0
```

```
stem1<s>:                \\ stem of height exactly 1
```

```
    A s T | G s C | T s A | C s G
```

```
stem2<s>:                \\ stem of height exactly 2
```

```
    stem1< stem1<s> >
```

```
stem<s>:                 \\ stem of height 3 or more
```

```
    A stem<s> T
```

```
    | T stem<s> A
```

```
    | C stem<s> G
```

```
    | G stem<s> C
```

```
    | stem1< stem2<s> >
```

# Grammar

s1: stem<s0>

any\_str: any\_smb\*[2..10]

any\_smb: A | T | C | G

s0: any\_str | any\_str stem<s0> s0

stem1<s>:                   \\ stem of height exactly 1

    A s T | G s C | T s A | C s G

stem2<s>:                   \\ stem of height exactly 2

    stem1< stem1<s> >

stem<s>:                   \\ stem of height 3 or more

    A stem<s> T

    | T stem<s> A

    | C stem<s> G

    | G stem<s> C

    | stem1< stem2<s> >

# Grammar

s1: stem<s0>

any\_str: any\_smb\*[2..10]

any\_smb: A | T | C | G

s0: any\_str | any\_str stem<s0> s0

stem1<s>:                   \\ stem of height exactly 1

    A s T | G s C | T s A | C s G

stem2<s>:                   \\ stem of height exactly 2

    stem1< stem1<s> >

stem<s>:                   \\ stem of height 3 or more

    A stem<s> T

    | T stem<s> A

    | C stem<s> G

    | G stem<s> C

    | stem1< stem2<s> >

# Grammar

```
s1: stem<s0>
any_str: any_smb*[2..10]
any_smb: A | T | C | G
s0: any_str | any_str stem<s0> s0
stem1<s>:                \\ stem of height exactly 1
    A s T | G s C | T s A | C s G
stem2<s>:                \\ stem of height exactly 2
    stem1< stem1<s> >
stem<s>:                \\ stem of height 3 or more
    A stem<s> T
    | T stem<s> A
    | C stem<s> G
    | G stem<s> C
    | stem1< stem2<s> >
```

# Grammar

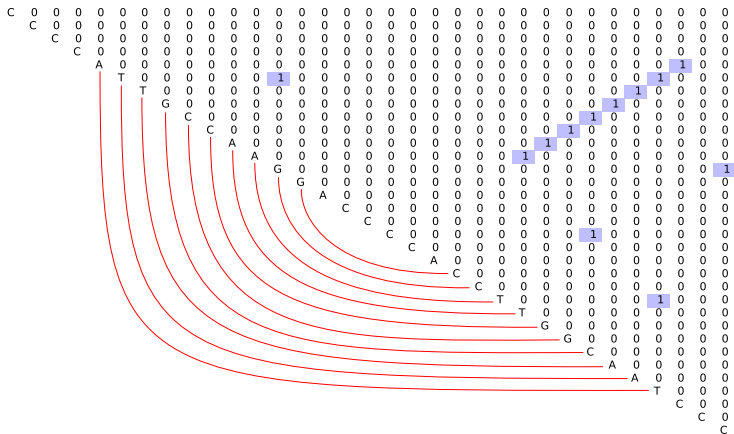
```
s1: stem<s0>
any_str: any_smb*[2..10]
any_smb: A | T | C | G
s0: any_str | any_str stem<s0> s0
stem1<s>:                \\ stem of height exactly 1
    A s T | G s C | T s A | C s G
stem2<s>:                \\ stem of height exactly 2
    stem1< stem1<s> >
stem<s>:                \\ stem of height 3 or more
    A stem<s> T
    | T stem<s> A
    | C stem<s> G
    | G stem<s> C
    | stem1< stem2<s> >
```

# Grammar

```
s1: stem<s0>
any_str: any_smb*[2..10]
any_smb: A | T | C | G
s0: any_str | any_str stem<s0> s0
stem1<s>:                \\ stem of height exactly 1
    A s T | G s C | T s A | C s G
stem2<s>:                \\ stem of height exactly 2
    stem1< stem1<s> >
stem<s>:                \\ stem of height 3 or more
    A stem<s> T
    | T stem<s> A
    | C stem<s> G
    | G stem<s> C
    | stem1< stem2<s> >
```

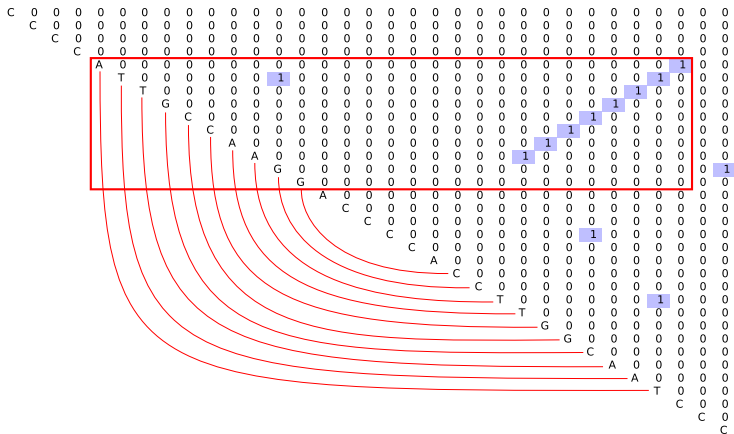
# Example 1: Stem

$\omega_1 = \text{CCCCATTGCCAAGGACCCCACCTTGGCAATCCC}$



# Example 1: Stem

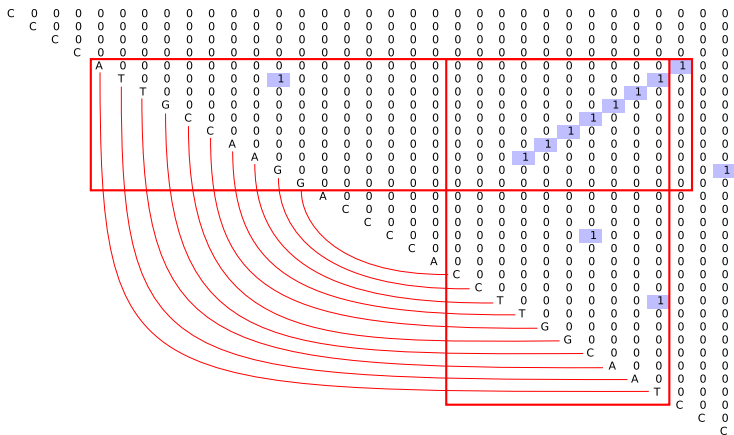
$\omega_1 = \text{CCCCATTGCCAAGGACCCACCTTGGCAATCCC}$





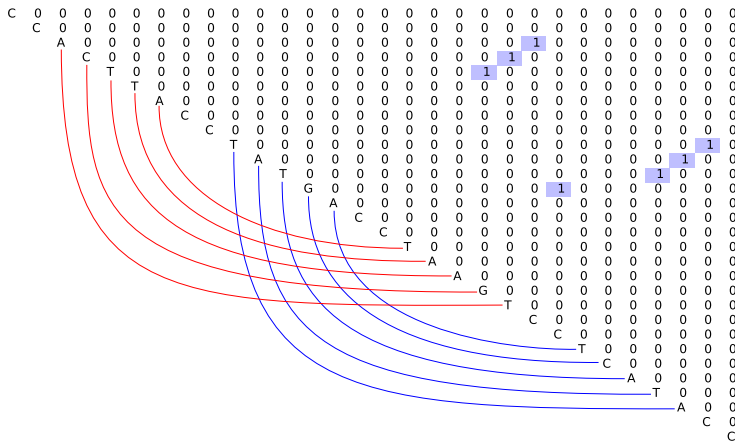
# Example 1: Stem

$\omega_1 = \text{CCCCATTGCCAAGGACCCACCTTGGCAATCCC}$



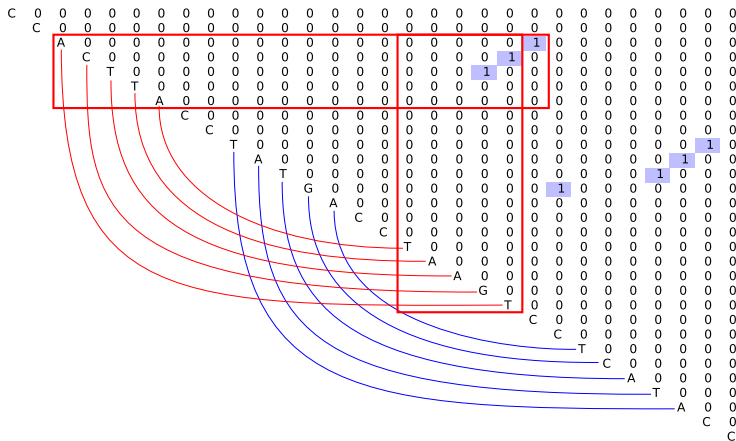
## Example 2: Pseudoknot

$\omega_2 = \text{C C A C T T A C C T A T G A C C T A A G T C C T C A T A C C}$



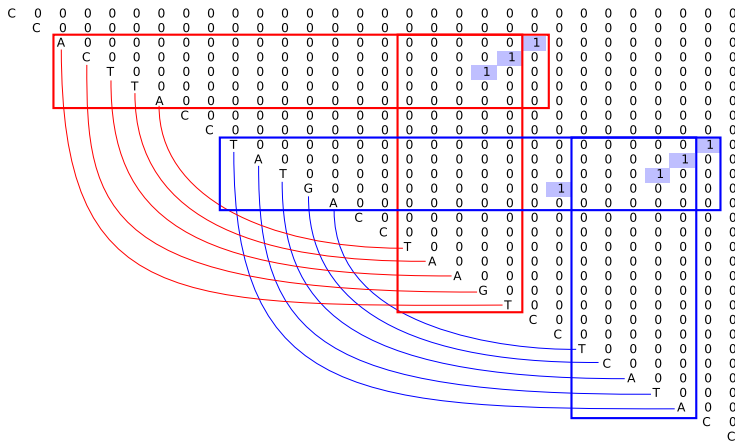
## Example 2: Pseudoknot

$\omega_2 = \text{C C A C T T A C C T A T G A C C T A A G T C C T C A T A C C}$



## Example 2: Pseudoknot

$\omega_2 = \text{CCACTTACCTATGACCTAAGTCCTCATACC}$

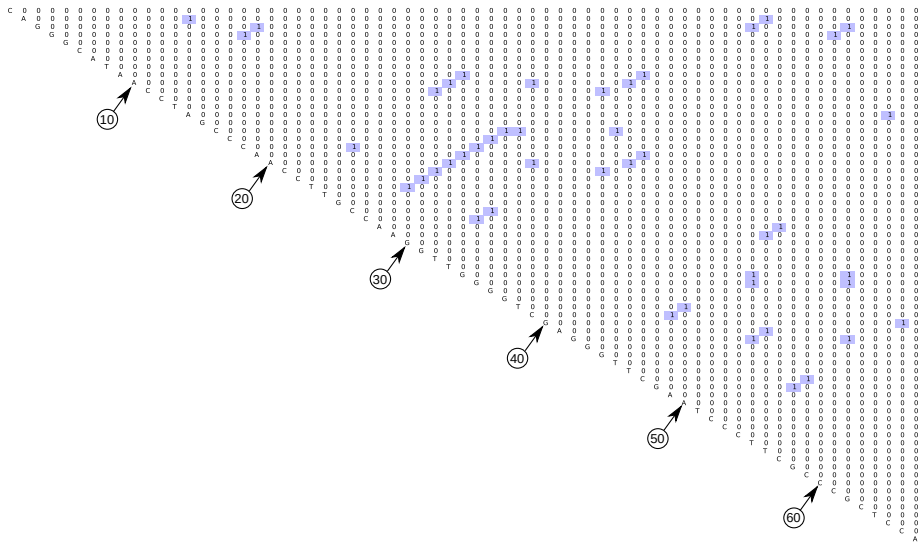


## Example 3: real tRNA

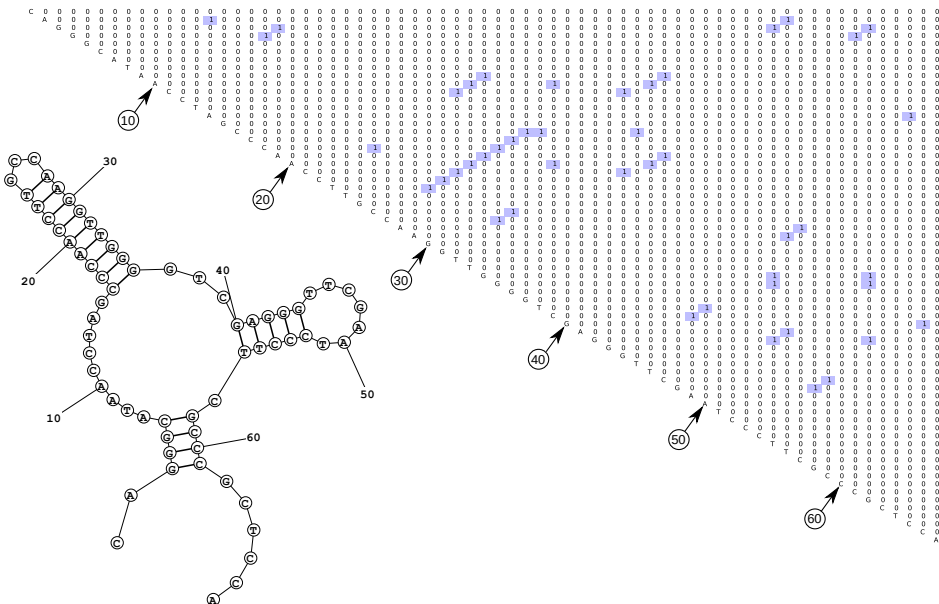
$\omega_3 =$  CAGGGCATAACCTAGCCCAACCTTGCCAAGG  
TTGGGGTCGAGGGTTCGAATCCCTTCGCCCGCTCCA

- *Novosphingobium aromaticivorans* DSM 12444  
chr.trna57-GlyGCC(268150-268084) Gly (GCC) 67 bp Sc: 22.9, from GtRNAb
- Predicted secondary structures are given by using the Fold Web Server with default settings

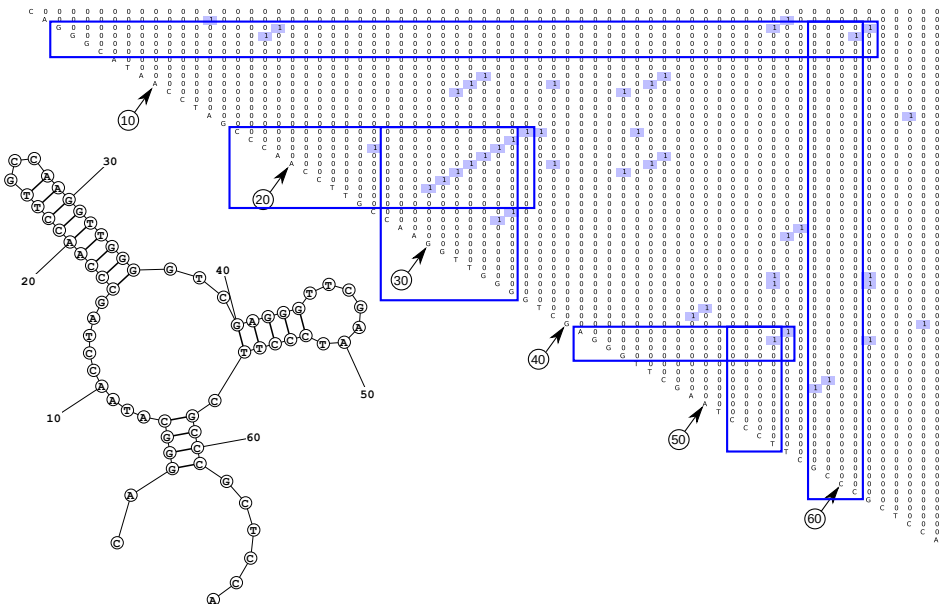
# Example 3: real tRNA



### Example 3: real tRNA

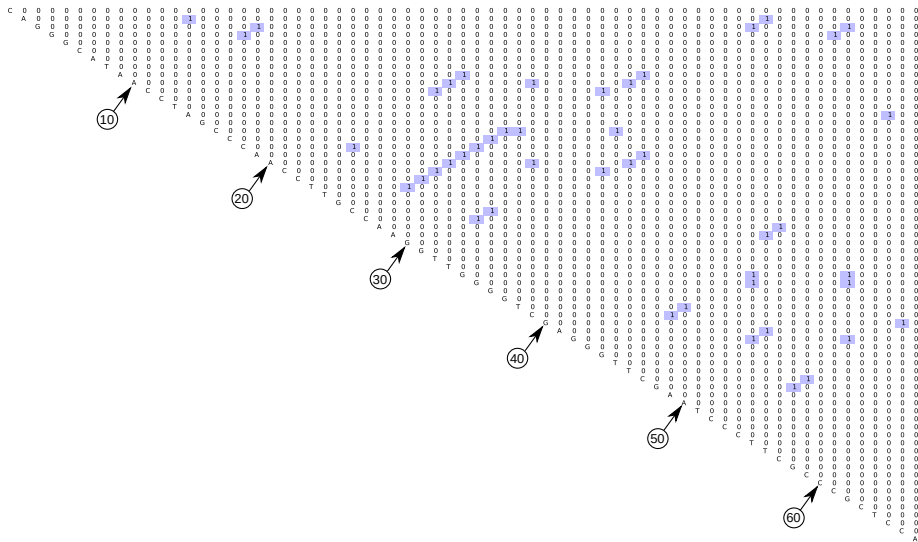


# Example 3: real tRNA

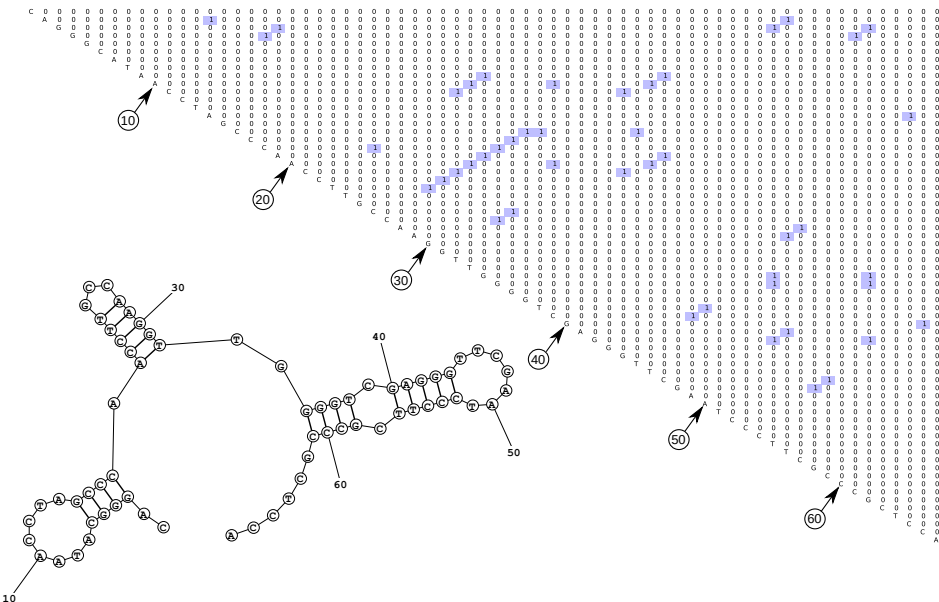




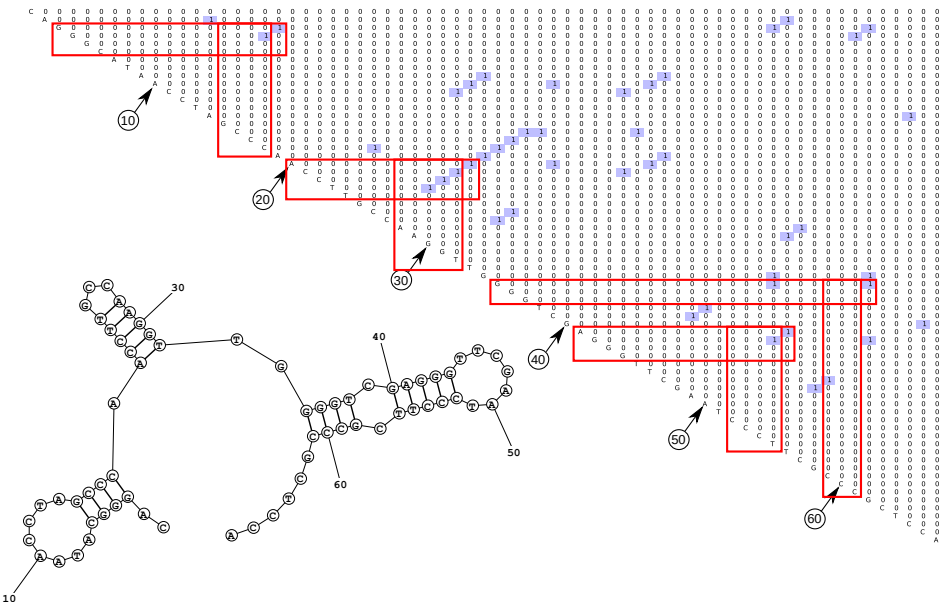
# Example 3: real tRNA



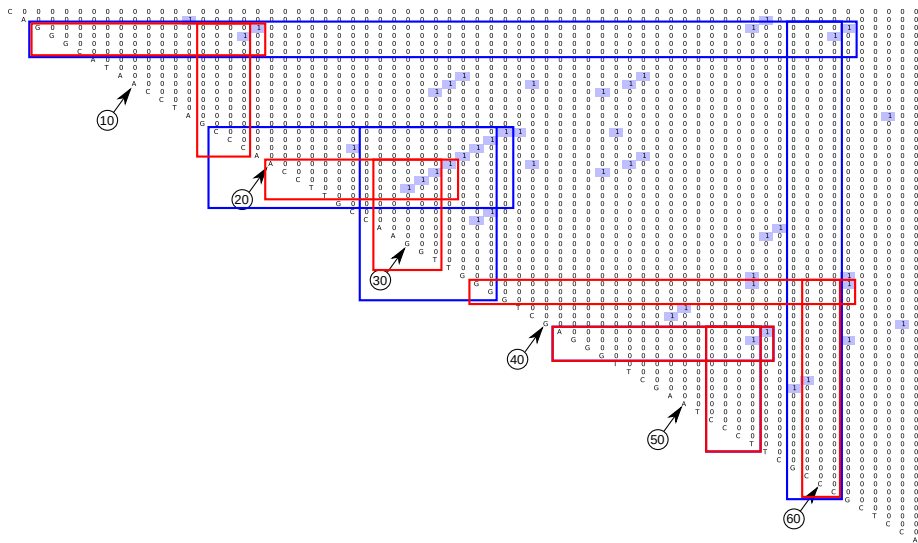
### Example 3: real tRNA



# Example 3: real tRNA



# Example 3: real tRNA



# Summary

- Parser is only a features extractor
- Parsing result contains all possible foldings (w.r.t. grammar) including impossible in practice
- Grammar is a parameter: one can add a G-T pair, change minimal height of the stem, etc
- It is possible to detect features which are not expressible in the language class in use

# Artificial Neural Networks

- Dense neural network
  - ▶ About 10 dense layers
  - ▶ ReLU activation function
- Problem: input size is fixed, but the length of sequences is variable

ACACAC		ACACAC\$\$
CGTACGCT	$\Rightarrow$	CGTACGCT
GCT		GCT\$\$\$\$\$

- Aggressive dropout (up to 90% after each layer) and batch normalization (after each layer) for learning stabilization

# Evaluation: 16s rRNA detection

- Training data
  - ▶ All sequences are 512 symbols in length
  - ▶ Totally up to 310000 sequences
  - ▶ Positive: random subsequences of 16s rRNA sequences from the Green Genes database
  - ▶ Negative: random subsequences of full genes from the NCBI database
- Validation set: up to 81000 sequences
- Accuracy is 90% after training

# Evaluation: tRNA classification

- Training data: 50000 sequences from GtRNADB
- Input data normalization
  - ▶ Set the upper bound of sequence length to 220
  - ▶ First  $k$  symbols of the input are tRNA and the rest  $220 - k$  symbols are filled by the special symbol
- Validation set: 217984 sequences for prokaryotes and 62656 sequences for eukaryotes from tRNADB-CE 3
- Accuracy is 97% after training
  - ▶ 3276 of eukaryotes (5.23% of all eukaryotes in the validation set) are classified as prokaryotes
  - ▶ 4373 of prokaryotes (2.01% of all prokaryotes in the validation set) are classified as eukaryotes



- DNN without parsing
  - ▶ Create a training set of matrices using parsing
  - ▶ Train the network  $NN_1$  which can handle vectorized matrices
  - ▶ Create network  $NN_2$  by extending  $NN_1$  with a set of layers which convert the sequence to input for  $NN_1$
  - ▶ Train  $NN_2$ , weights of layers from  $NN_1$  are fixed
- Other types of neural networks: bitwise networks, convolutional networks
- More evaluation
- Comparison with other tools

# Conclusion

- We propose the approach to handle secondary structure of sequences
- This approach can be applied for real data processing
- This approach can be extended to more expressive classes of formal languages
  - ▶ Conjunctive and boolean grammars
  - ▶ Multiple context-free grammars

- Semyon Grigorev:
  - ▶ [s.v.grigoriev@spbu.ru](mailto:s.v.grigoriev@spbu.ru)
  - ▶ [Semen.Grigorev@jetbrains.com](mailto:Semen.Grigorev@jetbrains.com)
- Polina Lunina:
  - ▶ [lunina\\_polina@mail.ru](mailto:lunina_polina@mail.ru)
- Trained models:

Thanks!