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

Febrary 24, 2019

Sequences Analysis: classification, search, etc

Problem: high variability of data

- Mutations
- Different kinds of noice
- . . .

Sequences Analysis: classification, search, etc

Problem: high variability of data

- Mutations
- Different kinds of noice
- . . .

We need probabilistic approaches

Sequences Analysis: classification, search, etc

Problem: high variability of data

- Mutations
- Different kinds of noice
- . . .

We need probabilistic approaches

Level of abstraction

- Plain text
- Secondary structure
- 3D structure

Sequences Analysis: classification, search, etc

Problem: high variability of data

- Mutations
- Different kinds of noice
- . . .

We need probabilistic approaches

Level of abstraction

- Plain text
- Secondary structure
- 3D structure

We should handle secondary structure

Sequences Analysis: classification, search, etc

Problem: high variability of data

- Mutations
- Different kinds of noice
- . . .

We need probabilistic approaches

Level of abstraction

- Plain text
- Secondary structure
- 3D structure

We should handle secondary structure

Probabilistic approach which can handle secondary structure information

Our Receip: Parsing + Artificial Neural Network

- Parsing is not for secondary structure modelling, but for features extraction
  - In the opposite to classical way of probabilistic CF grammars utilization

Our Receip: Parsing + Artificial Neural Network

- Parsing is not for secondary structure modelling, but for features extraction
  - ▸ In the opposite to classical way of probabilistic CF grammars utilization
- Formal grammars for secondary structure handling
- Parsing for features extraction
- Artificial neural network as probabilistic model for features processing
- Extracted features handling

Solution Structure

Grammar
Fixed formal grammar (not necessary context-free) describes features of secondary structure and can be tuned in order to increase result quality.

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

Result of classification

Parser
Parser extracts features of 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. Agressive dropout and batch normalization for learning process stabilization. Typical building block:

Dropout	input:	1024
(75%)	output:	1024

Dense	input:	1024
	output:	1024

BatchNormalization	input:	1024
	output:	1024

Activation	input:	1024
(relu)	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 ω: $M.[i, j] = 1 \iff \mathtt{s1} \to^* \omega.[i, j]$, and 0 in other case.
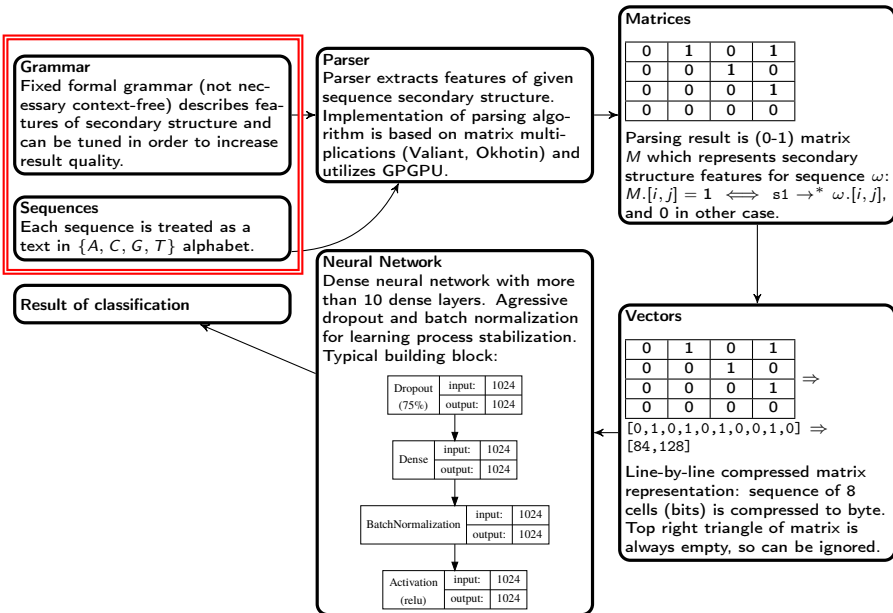
Vectors

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 to byte. Top right triangle of matrix is always empty, so can be ignored.

Solution Structure

Grammar
Fixed formal grammar (not necessary context-free) describes features of secondary structure and can be tuned in order to increase result quality.

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

Result of classification

Parser
Parser extracts features of 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. Agressive dropout and batch normalization for learning process stabilization. Typical building block:

Dropout	input:	1024
(75%)	output:	1024

Dense	input:	1024
	output:	1024

BatchNormalization	input:	1024
	output:	1024

Activation	input:	1024
(relu)	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 ω: $M.[i, j] = 1 \iff \mathtt{s1} \to^* \omega.[i, j]$, and 0 in other case.

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 to byte. Top right triangle of matrix is always empty, so can be ignored.

Solution Structure

Grammar
Fixed formal grammar (not necessary context-free) describes features of secondary structure and can be tuned in order to increase result quality.

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

Result of classification

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

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 ω: $M.[i, j] = 1 \iff \texttt{s1} \to^* \omega.[i, j]$, and 0 in other case.

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 to byte. Top right triangle of matrix is always empty, so can be ignored.

Neural Network
Dense neural network with more than 10 dense layers. Agressive 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

Solution Structure

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 ω: $M.[i, j] = 1 \iff \mathtt{s1} \to^* \omega.[i, j]$, and 0 in other case.

Grammar
Fixed formal grammar (not necessary context-free) describes features of secondary structure and can be tuned in order to increase result quality.

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

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

Result of classification

Neural Network
Dense neural network with more than 10 dense layers. Agressive 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

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 to byte. Top right triangle of matrix is always empty, so can be ignored.

Solution Structure

Grammar
Fixed formal grammar (not necessary context-free) describes features of secondary structure and can be tuned in order to increase result quality.

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

Result of classification

Parser
Parser extracts features of 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 ω: $M.[i, j] = 1 \iff \texttt{s1} \to^* \omega.[i, j]$, and 0 in other case.
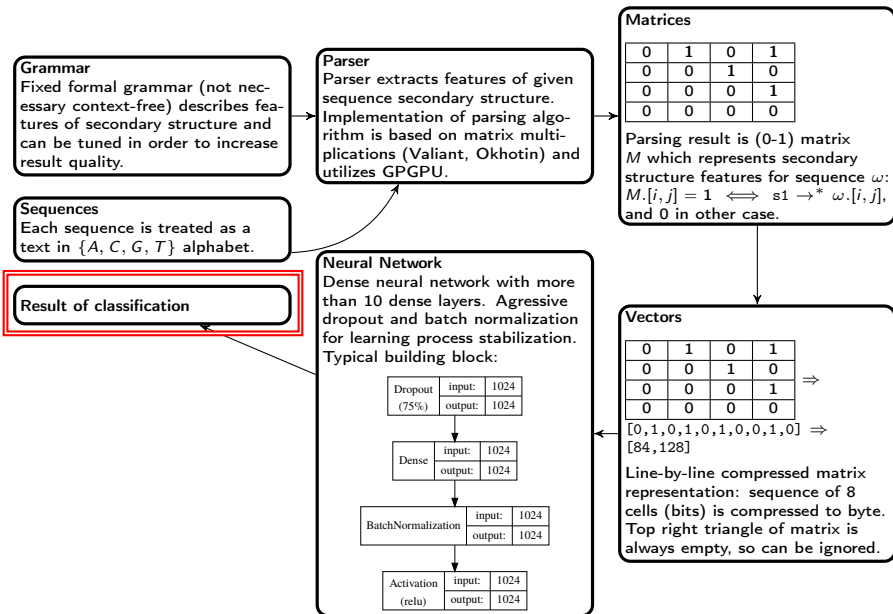
Vectors

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 to byte. Top right triangle of matrix is always empty, so can be ignored.

Solution Structure

Grammar
Fixed formal grammar (not necessary context-free) describes features of secondary structure and can be tuned in order to increase result quality.

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

Result of classification

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

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 ω: $M.[i, j] = 1 \iff \mathtt{s1} \to^* \omega.[i, j]$, and 0 in other case.

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

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 to byte. Top right triangle of matrix is always empty, so can be ignored.

Solution Structure

Grammar
Fixed formal grammar (not necessary context-free) describes features of secondary structure and can be tuned in order to increase result quality.

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

Result of classification

Parser
Parser extracts features of 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. Agressive dropout and batch normalization for learning process stabilization. Typical building block:

Dropout	input:	1024
(75%)	output:	1024

Dense	input:	1024
	output:	1024

BatchNormalization	input:	1024
	output:	1024

Activation	input:	1024
(relu)	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 ω: $M.[i, j] = 1 \iff \text{s1} \to^* \omega.[i, j]$, and 0 in other case.

Vectors

0	1	0	1
0	0	1	0	$\Rightarrow$
0	0	0	1
0	0	0	0

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

Line-by-line compressed matrix representation: sequence of 8 cells (bits) is compressed to byte. Top right triangle of matrix is always empty, so can be ignored.

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

$$\omega_1 = \texttt{CCCCATTGCCAAGGACCCCACCTTGGCAATCCC}$$
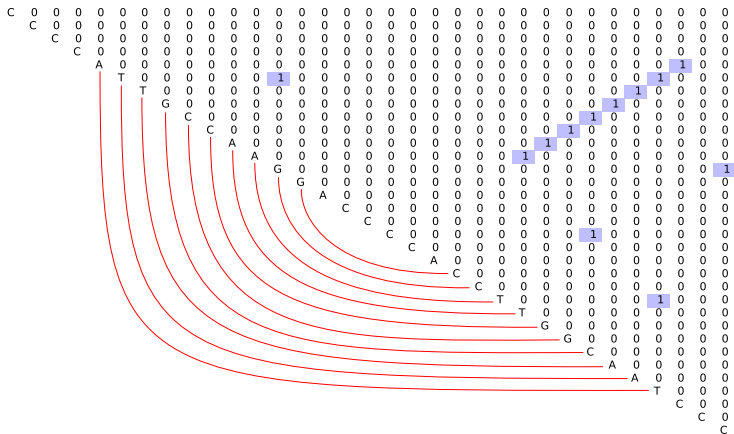
Example 1: Stem

$$\omega_1 = \texttt{CCCCATTGCCAAGGACCCCACCTTGGCAATCCC}$$

Example 1: Stem

$$\omega_1 = \texttt{CCCCATTGCCAAGGACCCCACCTTGGCAATCCC}$$

$$\omega_2 = \texttt{CCACTTACCTATGACCTAAGTCCTCATACC}$$
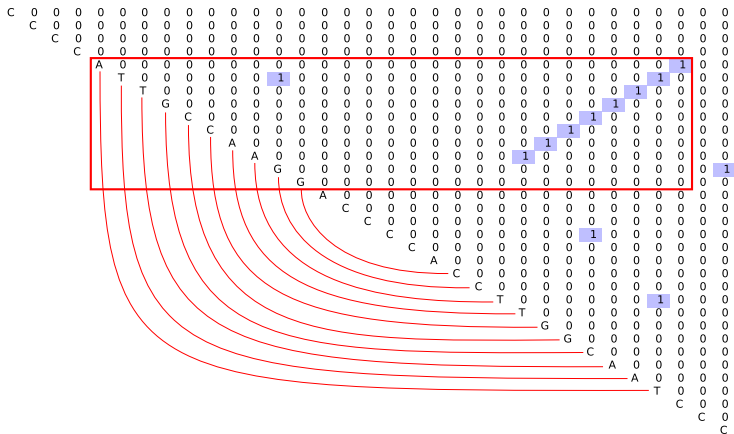
$$\omega_2 = \texttt{CCACTTACCTATGACCTAAGTCCTCATACC}$$
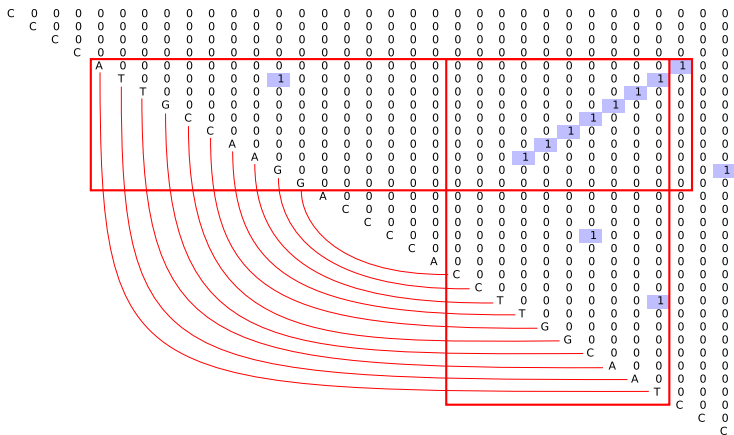
Example 2: Pseudoknot

$$\omega_2 = \texttt{CCACTTACCTATGACCTAAGTCCTCATACC}$$
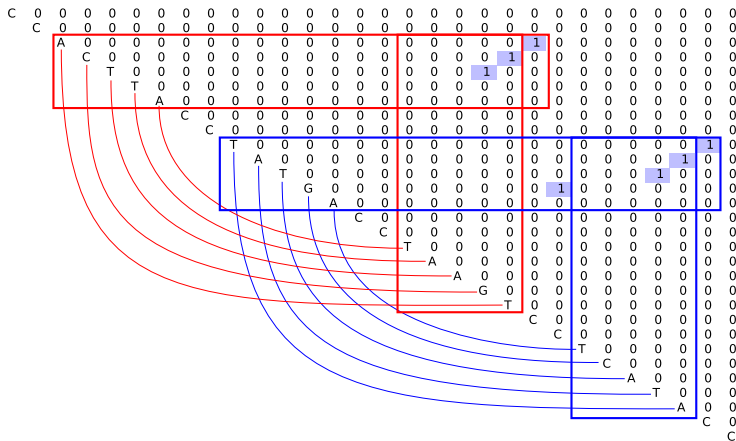
Example 3: real tRNA

$$\omega_3 = \texttt{CAGGGCATAACCTAGCCCAACCTTGCCAAGG}$$
$$\texttt{TTGGGGTCGAGGGTTCGAATCCCTTCGCCCGCTCCA}$$

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

Example 3: real tRNA
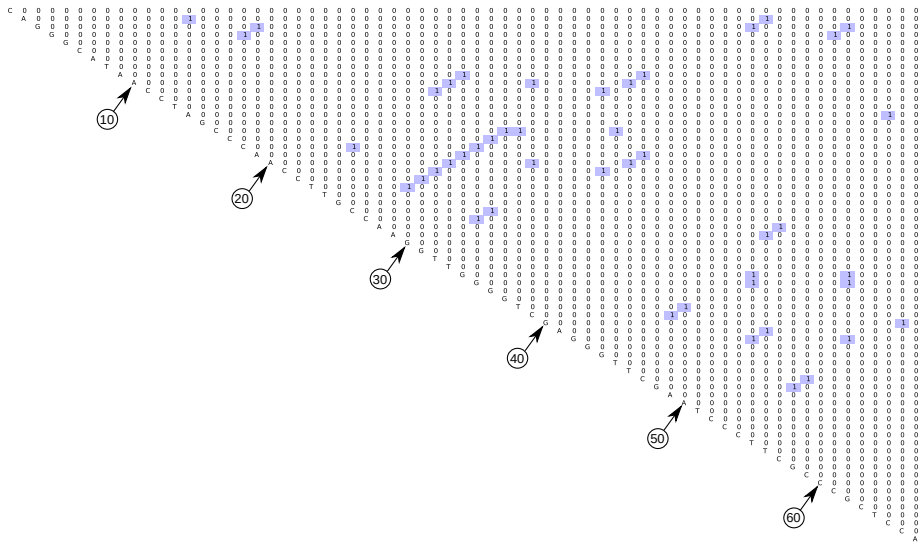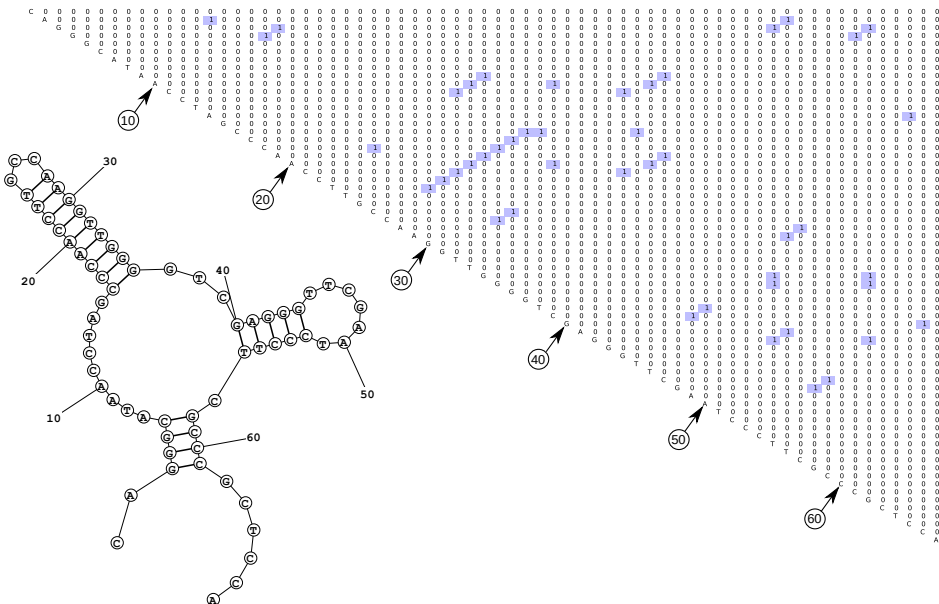
Example 3: real tRNA
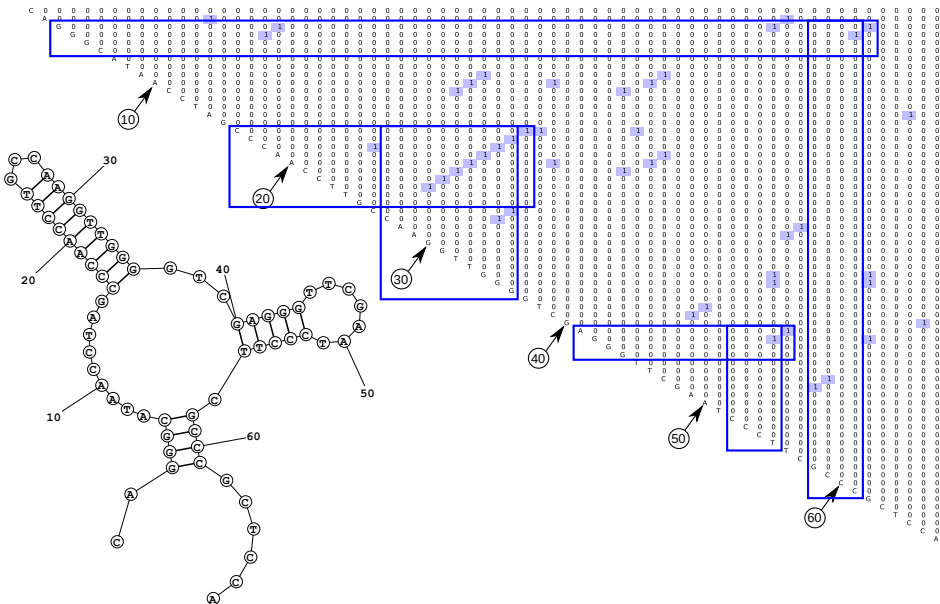
Example 3: real tRNA
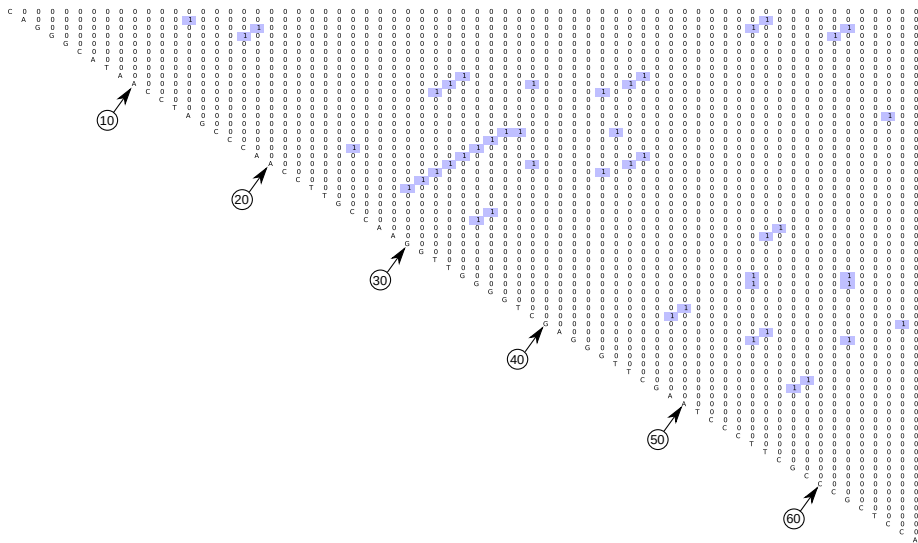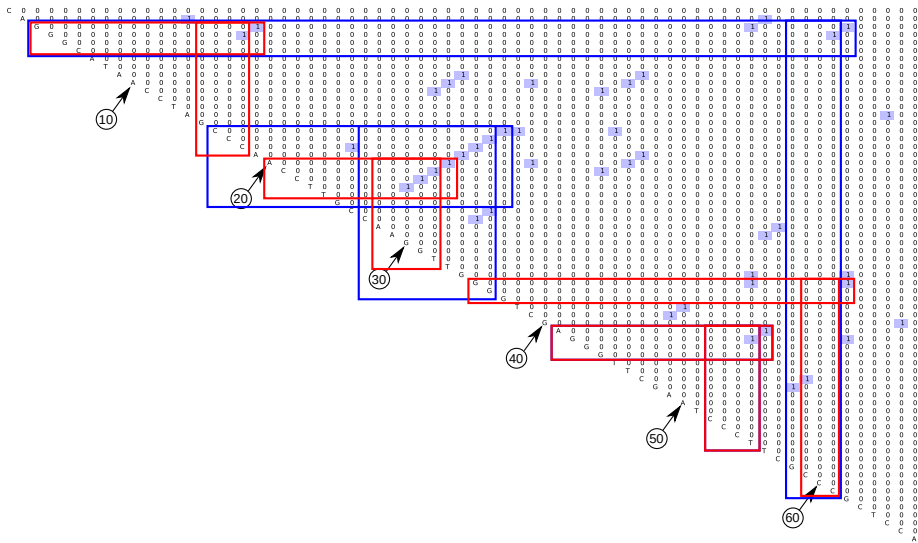
Example 3: real tRNA
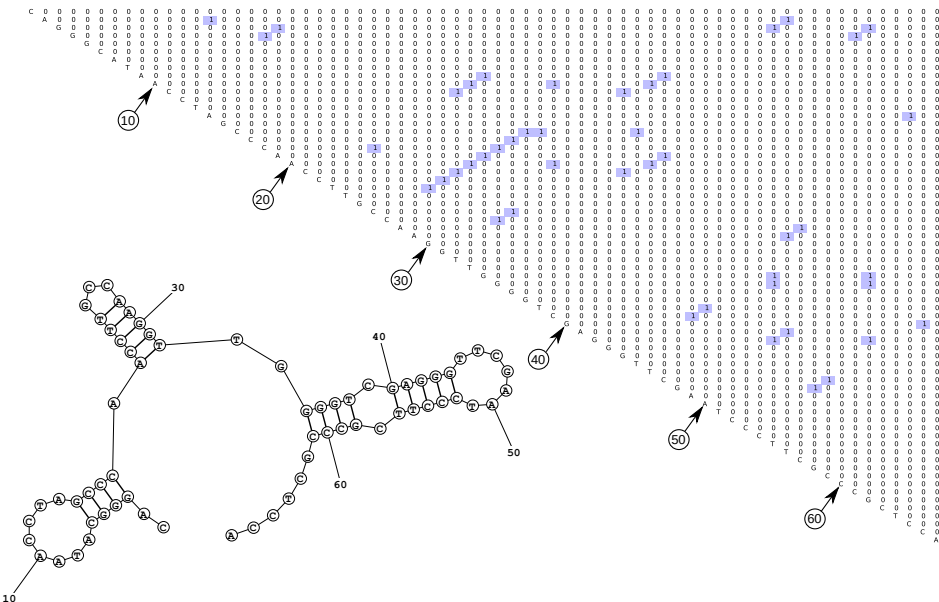
Example 3: real tRNA
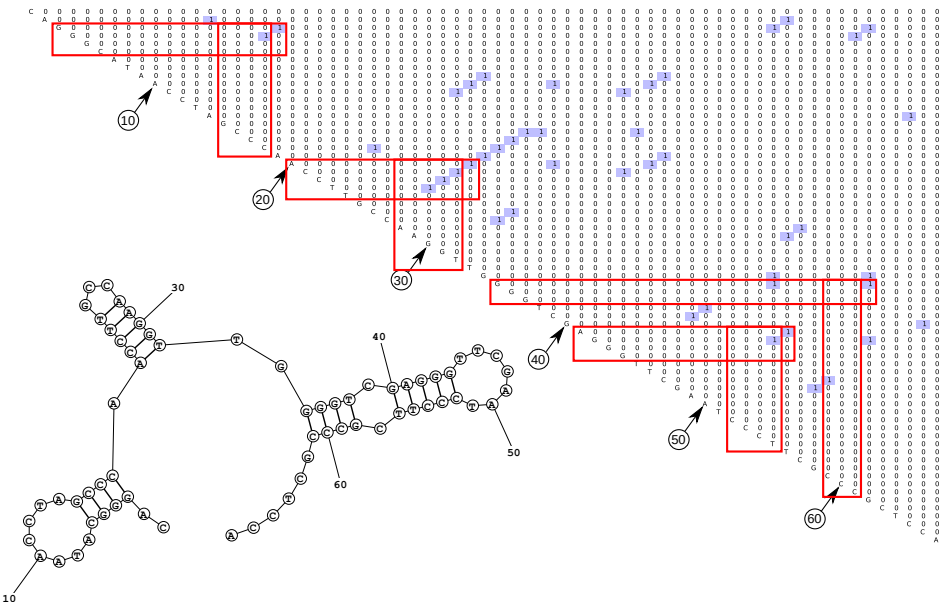
Example 3: real tRNA

Example 3: real tRNA

Summary

- Parser is a features extructor, not a way to model secondary structure of full sequence
- Parsing result contains all possible foldings (w.r.t. grammar) including practically unreal foldings
- Grammar is a varaible parameter: one can add G-T pair, change minimal height of stem, etc
- It is possible to detect features which is not expressable in language class which in use

Artificial Neural Networks

- Dence neural network
  - About 10 dence layers
  - `Relu` activation function
- Problem: size of input is fixed, but length of sequences is variable

```
ACACAC                                ACACAC$$
CGTACGCT          ⇒                   CGTACGCT
GCT                                   GCT$$$$$
```

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

Evaluation: 16s rRNA detection

- Training data
  - All sequences have the length of 512 symbols
  - 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
- After training, accuracy is 90%

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 a special symbol
- Validation set: 217984 sequences for prokaryotes and 62656 sequences for eukaryotes from tRNADB-CE 3
- After training, accuracy is 97%
  - 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

Future work

- 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
- Comprison with other tools

Conclusion

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

Contact Information

- Semyon Grigorev:
  - ► s.v.grigoriev@spbu.ru
  - ► Semen.Grigorev@jetbrains.com
- Polind Lunina:
  - ► lunina_polina@mail.ru
- Trained models:

Thanks!