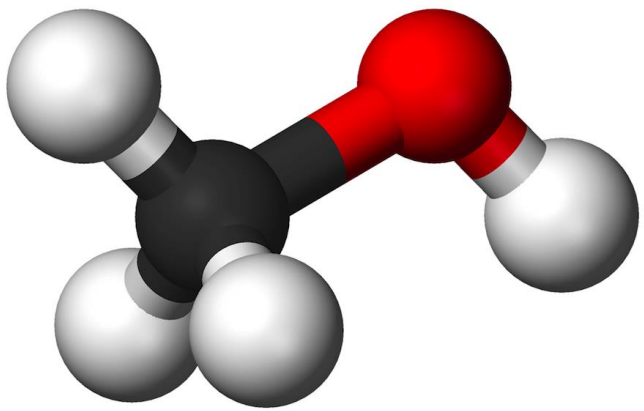
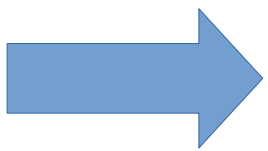
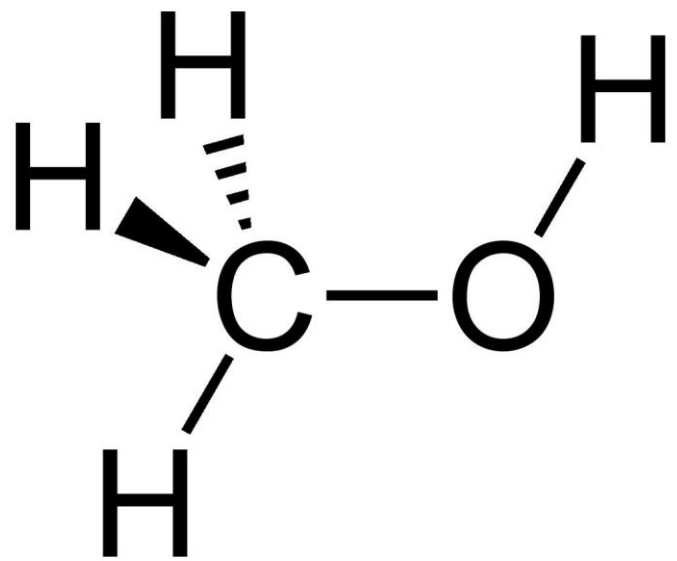
Предсказание свойств молекул и создание молекул с заданными свойствами

Курбанов Ринат, Шилов Валентин,
Висков Василий, Питанов Елисей

Как же построить граф из описания молекулы?







Что у нас есть:

1) матрица 3D структуры молекулы
2) частичные заряды
3) SMILES
4) число атомов(очевидно)
5) плотность
6) внутренняя энергия
7) доп. параметры типа объема формы, ароматичности, гибридизации


```
100-01-6
38.2721 1.73
30.528
385.365
104.5
15
O     2.7374    1.0976   -0.000   -0.52
O     2.7373   -1.0976    0.000   -0.52
N     2.1292    0.0000   -0.000   0.91
N    -3.4908   -0.0001    0.000   -0.9
C     0.7093    0.0000    0.000   0.13
C    -2.0804    0.0000   -0.000   0.1
C     0.0120    1.2080    0.000   -0.15
C     0.0119   -1.2080    0.000   -0.15
C    -1.3829    1.2080    0.000   -0.15
C    -1.3830   -1.2079   -0.000   -0.15
H     0.5219    2.1680    0.000   0.15
H     0.5218   -2.1680    0.000   0.15
H    -1.9154    2.1558    0.000   0.15
C1=CC(=CC=C1N)[N+](=O)[O-]
```

Алгоритм построения:

- Считываем свойства и инициализируем граф с помощью библиотеки networkx
- Считываем атомы, заполняем их свойства
- С помощью библиотеки rdkit из SMILES получаем представление молекулы в унифицированном формате
- Обращаемся к графу и добавляем узлы с конкретными свойствами для каждого атома
- Проставляем свойства донора и аксептора(для Рината: донор и аксептор это видимо как будет направлено подтюнивание/изменение свойств? Ну как я понял из химии)
- Проставляем связи между атомами

```
tag=f.readline().strip()
# Graph properties
properties = f.readline()
na=int(f.readline())
g,l=init_graph_our(tag, properties)
```

```
def init_graph_our(g_tag,prop):

    prop = prop.split()
    g_H=float(prop[0])
    g_p=float(prop[1])

    labels = [g_H, g_p]
    return nx.Graph(tag=g_tag,H=g_H, p=g_p), labels
```

```
# Atoms properties
for i in range(na):
    a_properties = f.readline()
    a_properties = a_properties.replace('.*^', 'e')
    a_properties = a_properties.replace('*^', 'e')
    a_properties = a_properties.split()
    atom_properties.append(a_properties)
# SMILES
smiles = f.readline()
smiles = smiles.split()
smiles = smiles[0]
m = Chem.MolFromSmiles(smiles)
m = Chem.AddHs(m)
```

```
# Create nodes
for i in range(0, m.GetNumAtoms()):
    atom_i = m.GetAtomWithIdx(i)

    g.add_node(i, a_type=atom_i.GetSymbol(), a_num=atom_i.GetAtomicNum(), acceptor=0, donor=0,
               aromatic=atom_i.GetIsAromatic(), hybridization=atom_i.GetHybridization(),
               num_h=atom_i.GetTotalNumHs(), coord=np.array(atom_properties[i][1:4]).astype(np.float),
               pc=float(atom_properties[i][4]))
```

```
#connecting to db and retrieving mol features
fdef_name = os.path.join(RDConfig.RDDataDir, 'BaseFeatures.fdef')
factory = ChemicalFeatures.BuildFeatureFactory(fdef_name)
feats = factory.GetFeaturesForMol(m)
for i in range(0, len(feats)):
    if feats[i].GetFamily() == 'Donor':
        node_list = feats[i].GetAtomIds()
        for i in node_list:
            g.node[i]['donor'] = 1
    elif feats[i].GetFamily() == 'Acceptor':
        node_list = feats[i].GetAtomIds()
        for i in node_list:
            g.node[i]['acceptor'] = 1
```

```
# Read Edges
for i in range(0, m.GetNumAtoms()):
    for j in range(0, m.GetNumAtoms()):
        e_ij = m.GetBondBetweenAtoms(i, j)
        if e_ij is not None:
            g.add_edge(i, j, b_type=e_ij.GetBondType(),
                       distance=np.linalg.norm(g.node[i]['coord']-g.node[j]['coord']))
        else:
            # Unbonded
            g.add_edge(i, j, b_type=None,
                       distance=np.linalg.norm(g.node[i]['coord'] - g.node[j]['coord']))
```

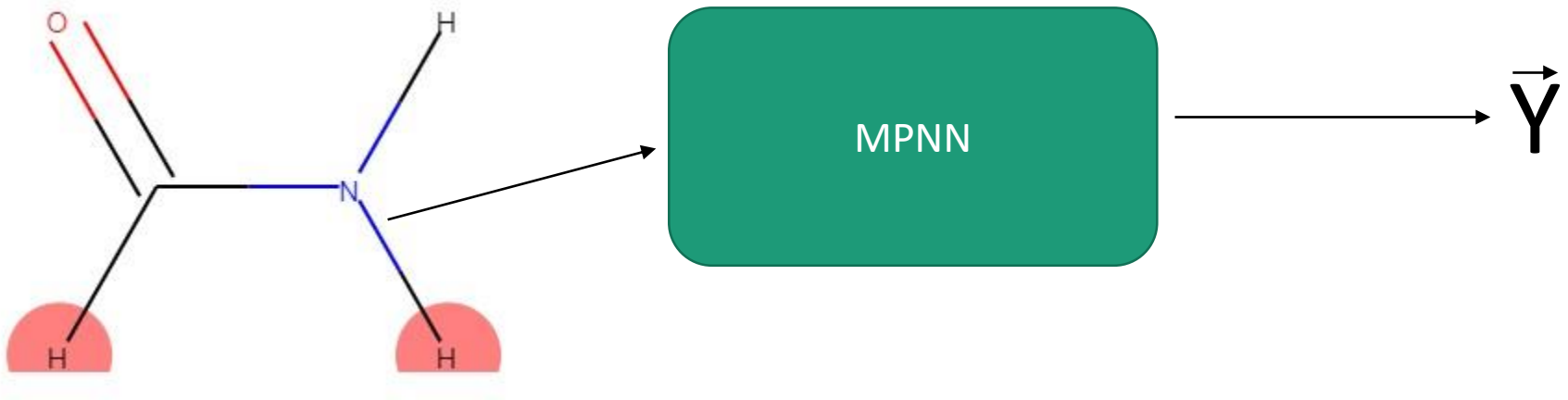
Message Passing Neural Networks

Вход:

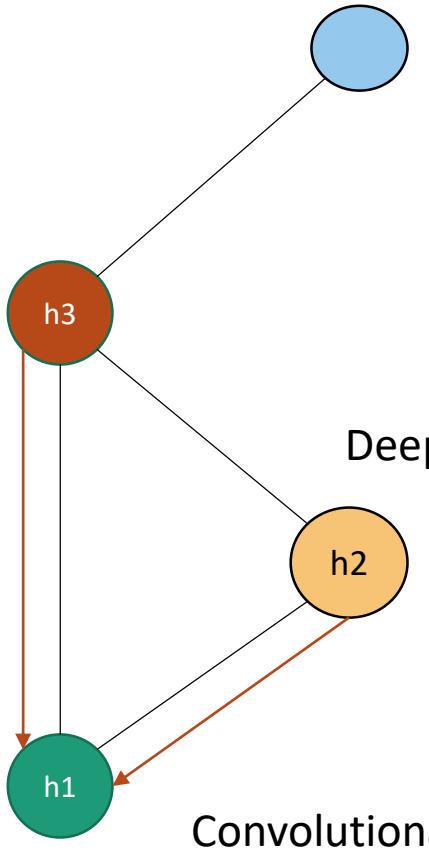
- Ненаправленный граф
- Признаки узлов и связей

Выход:

- Свойства: плотность, Энтальпия



Massage Passing этап. Сбор информации с соседних узлов



$$m_v^{t+1} = \sum_{w \in N(v)} M_t(h_v^t, h_w^t, e_{vw})$$

Gated Graph Neural Networks: $M_t(h_v^t, h_w^t, e_{vw}) = A_{e_{vw}} h_w^t$

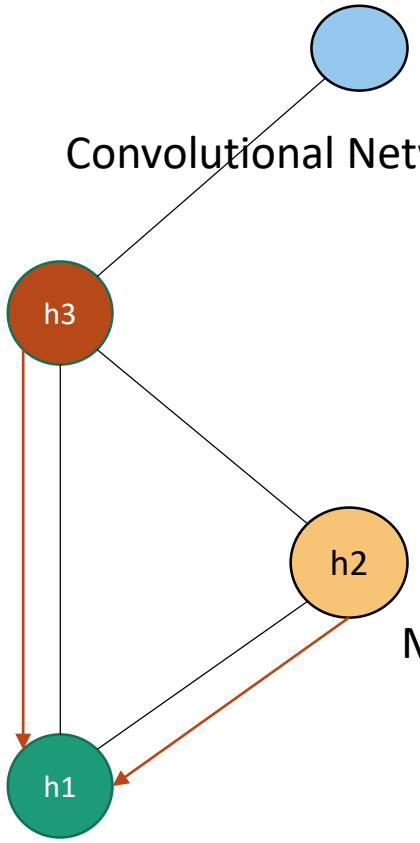
где A(e_{vw}) - нейронная сеть, которая отображает вектор ребер e_{vw} в матрицу $d^x d$

Deep Tensor Neural Networks: $M_t = \tanh\left(W^{fc}((W^{cf} h_w^t + b_1) \odot (W^{df} e_{vw} + b_2))\right)$

Molecular Graph Convolutions: $M(h_v^t, h_w^t, e_{vw}^t) = e_{vw}^t$

Convolutional Networks for Learning Molecular Fingerprints: $M(h_v, h_w, e_{vw}) = (h_w, e_{vw})$

Massage Passing этап. Обновление информации текущего узла



$$h_v^{t+1} = U_t(h_v^t, m_v^{t+1})$$

Convolutional Networks for Learning Molecular Fingerprints: $U_t(h_v^t, m_v^{t+1}) = \sigma(H_t^{\deg(v)} m_v^{t+1})$

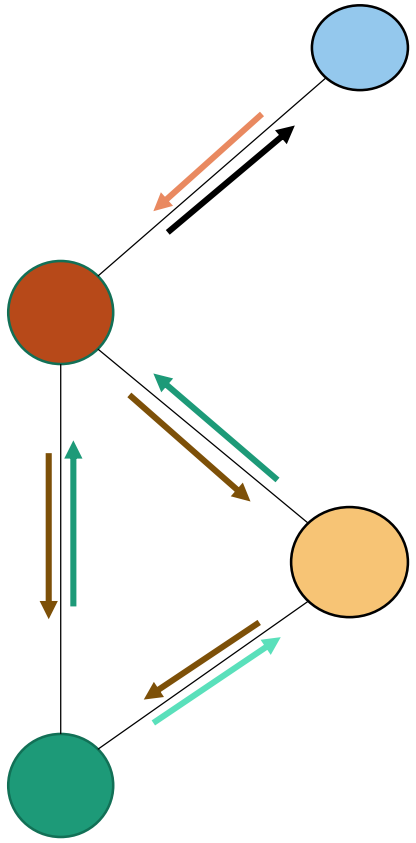
Gated Graph Neural Networks: $U_t = \mathrm{GRU}(h_v^t, m_v^{t+1})$

Deep Tensor Neural Networks: $U_t(h_v^t, m_v^{t+1}) = h_v^t + m_v^{t+1}$

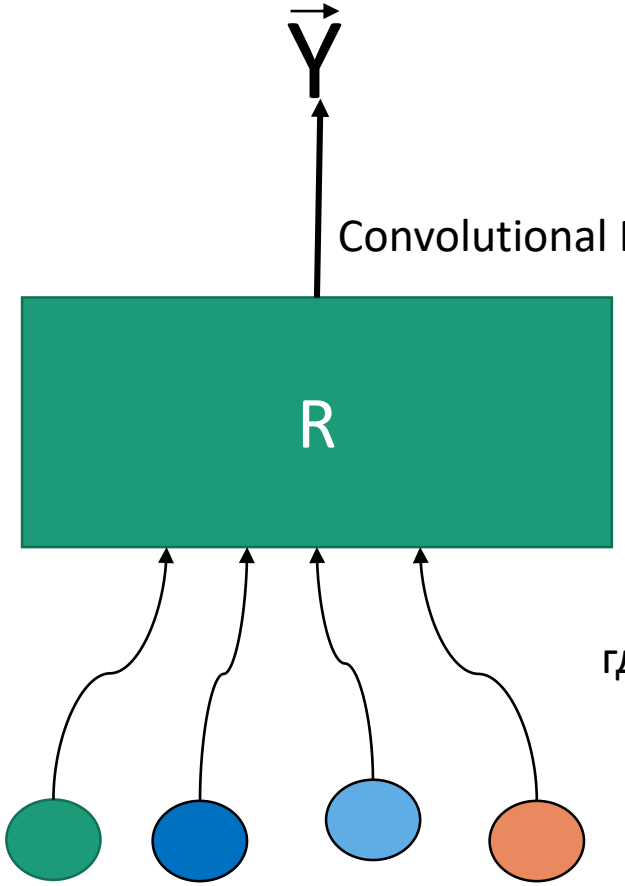
Molecular Graph Convolutions: $U_t(h_v^t, m_v^{t+1}) = \alpha(W_1(\alpha(W_0 h_v^t), m_v^{t+1}))$

$$e_{vw}^{t+1} = U_t'(e_{vw}^t, h_v^t, h_w^t) = \alpha(W_4(\alpha(W_2, e_{vw}^t), \alpha(W_3(h_v^t, h_w^t))))$$

Massage Passing этап. Обновление происходит до гиперпараметра T



ReadOut этап.



$$\hat{y} = R(\{h_v^T | \, v \in G\})$$

Convolutional Networks for Learning Molecular Fingerprints: R= $f\left(\sum_{v,t} \mathrm{softmax}(W_t h_v^t)\right)$

где f - нейронная сеть, а Wt – обучаемые матрицы, по одной на каждый временной шаг t

Gated Graph Neural Networks: $R = \sum_{v \in V} \sigma\left(i(h_v^{(T)}, h_v^0)\right) \odot \left(j(h_v^{(T)})\right)$

где i и j - нейронные сети, и обозначает поэлементное умножение

Deep Tensor Neural Networks: $R = \sum_v \mathrm{NN}(h_v^T)$

Average Error Ratio. Average Loss

```
MPNN(
  (m): ModuleList(
    (0): MessageFunction(
      (learn_args): ParameterList()
      (learn_modules): ModuleList(
        (0): NNet(
          (fcs): ModuleList(
            (0): Linear(in_features=5, out_features=128, bias=True)
            (1): Linear(in_features=128, out_features=256, bias=True)
            (2): Linear(in_features=256, out_features=128, bias=True)
            (3): Linear(in_features=128, out_features=5329, bias=True)
          )
        )
      )
    )
  )
  (u): ModuleList(
    (0): UpdateFunction(
      (learn_args): ParameterList()
      (learn_modules): ModuleList(
        (0): GRU(73, 73)
      )
    )
  )
  (r): ReadoutFunction(
    (learn_args): ParameterList()
    (learn_modules): ModuleList(
      (0): NNet(
        (fcs): ModuleList(
          (0): Linear(in_features=146, out_features=128, bias=True)
          (1): Linear(in_features=128, out_features=256, bias=True)
          (2): Linear(in_features=256, out_features=128, bias=True)
          (3): Linear(in_features=128, out_features=2, bias=True)
        )
      )
      (1): NNet(
        (fcs): ModuleList(
          (0): Linear(in_features=73, out_features=128, bias=True)
          (1): Linear(in_features=128, out_features=256, bias=True)
          (2): Linear(in_features=256, out_features=128, bias=True)
          (3): Linear(in_features=128, out_features=2, bias=True)
        )
      )
    )
  )
)
```

epoch 44

criterion = nn.MSELoss()

optimizer = optim.Adam(model.parameters(), lr=1e-3)

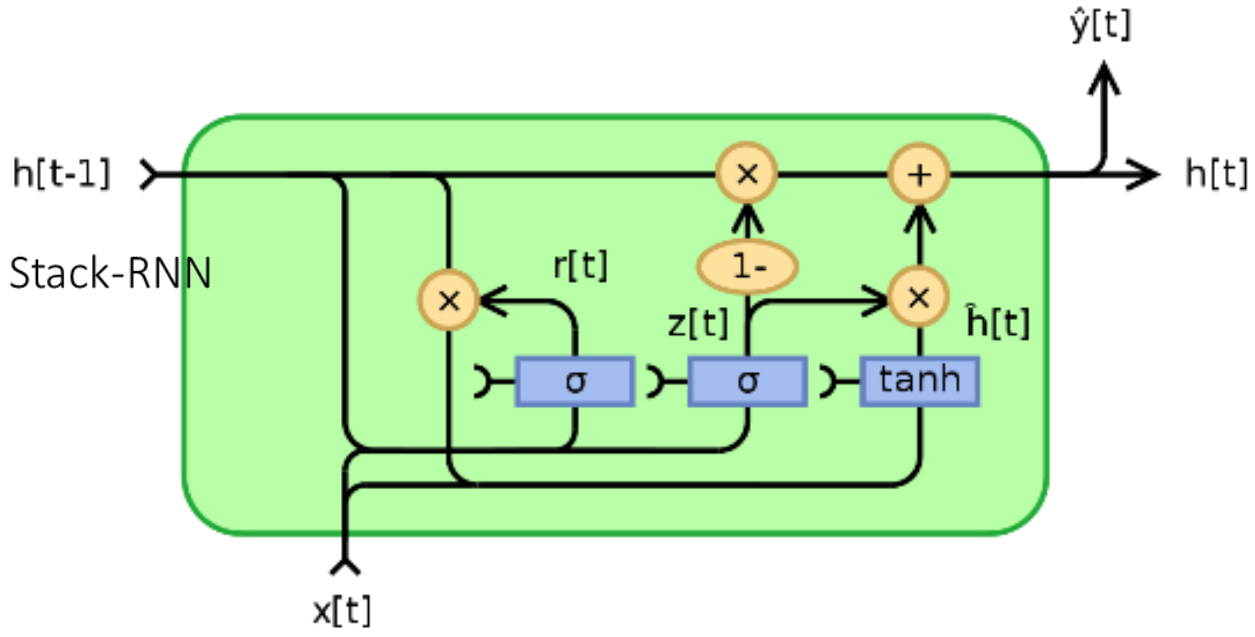
Average Error Ratio 1.5121180399656295

Average Loss 0.817

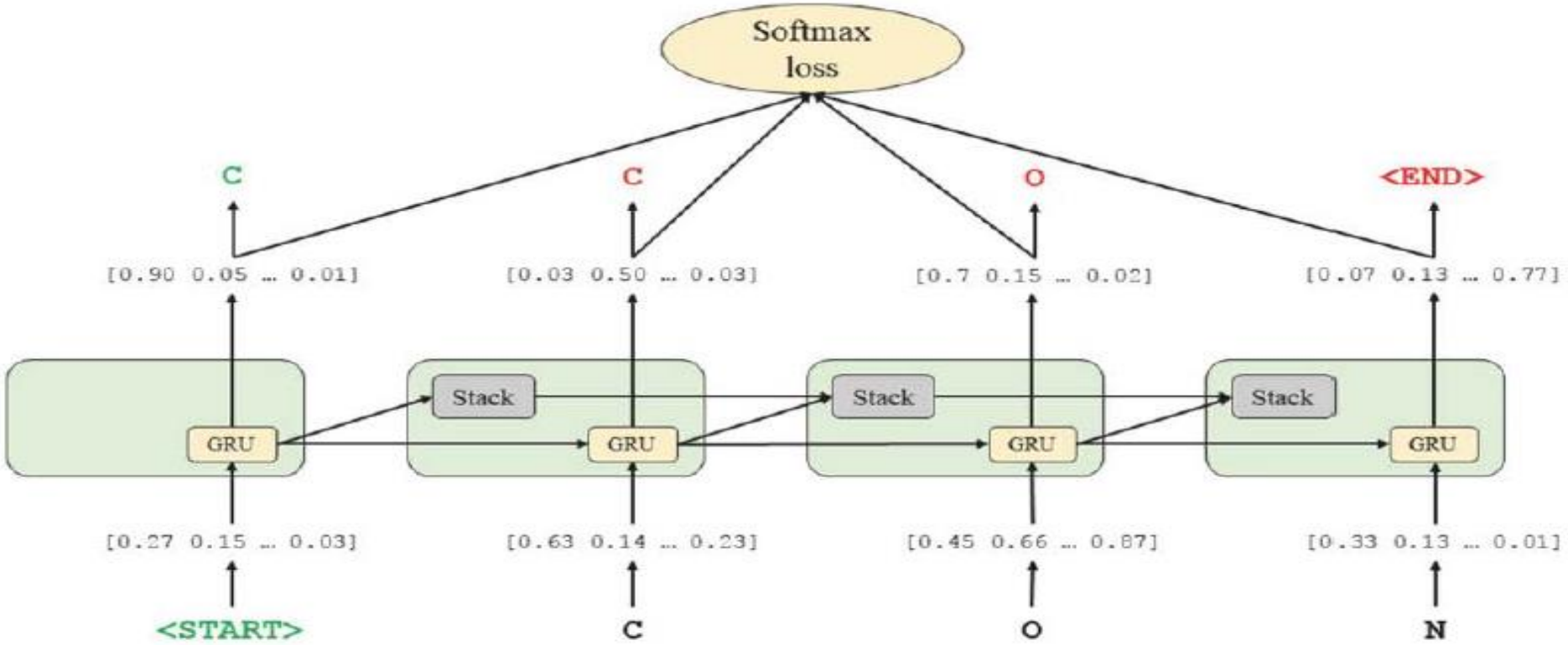
Генерация молекул(пока в 2d формате)

На входе и выходе: текстовое представление молекул – SMILES (например, C1=CC=CC=C1), преобразованное в вектор из номеров символов в алфавите:
“<>#%)(+-/.1032547698=A@CBFIHONPS[]\ceilonpsr”

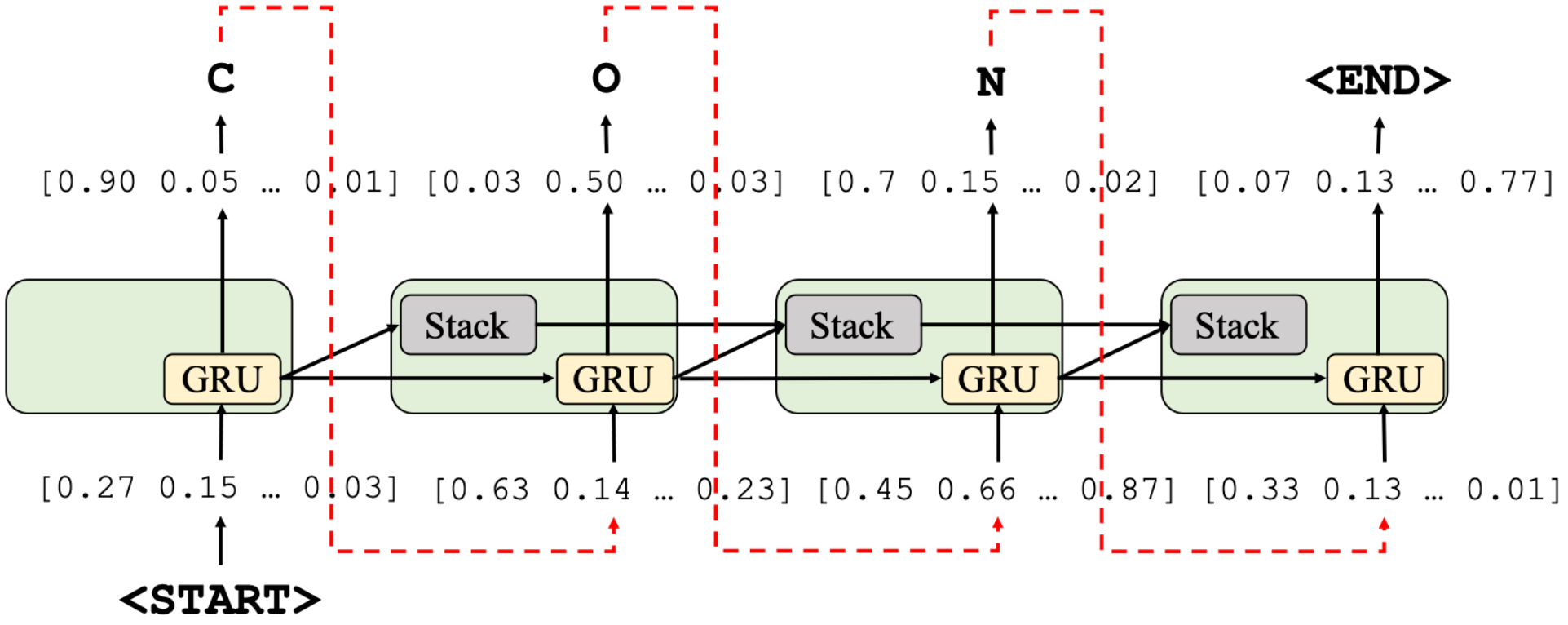
В основе сети – GRU (Gated recurrent units) – упрощённый вариант LSTM - ячейки



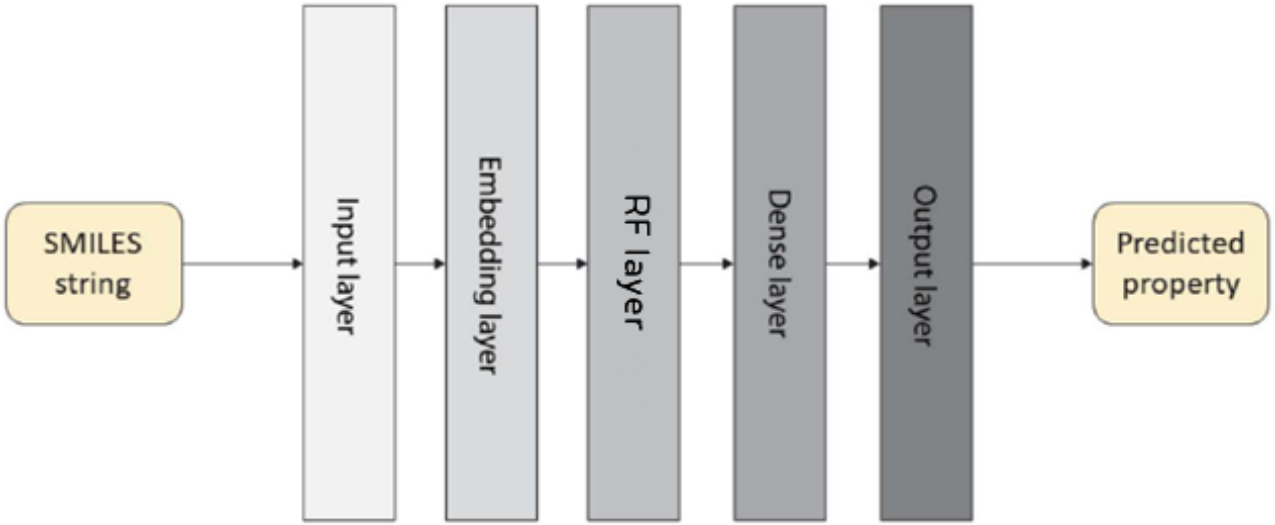
Stack-RNN (Обучение)

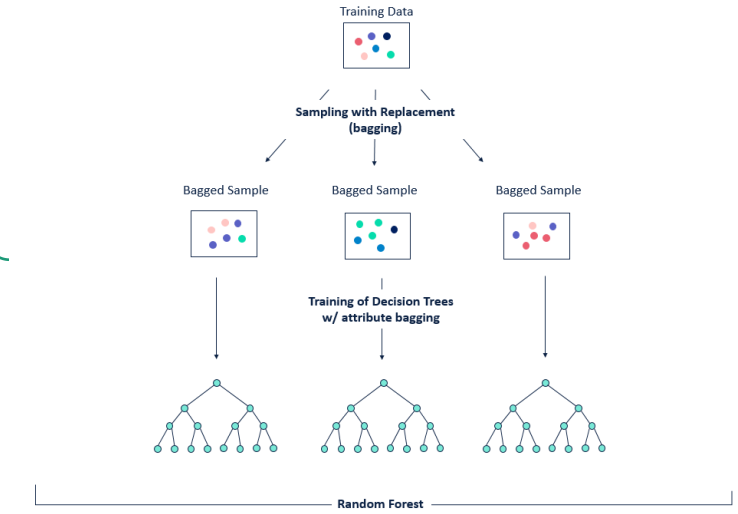


Stack-RNN (Генерация)



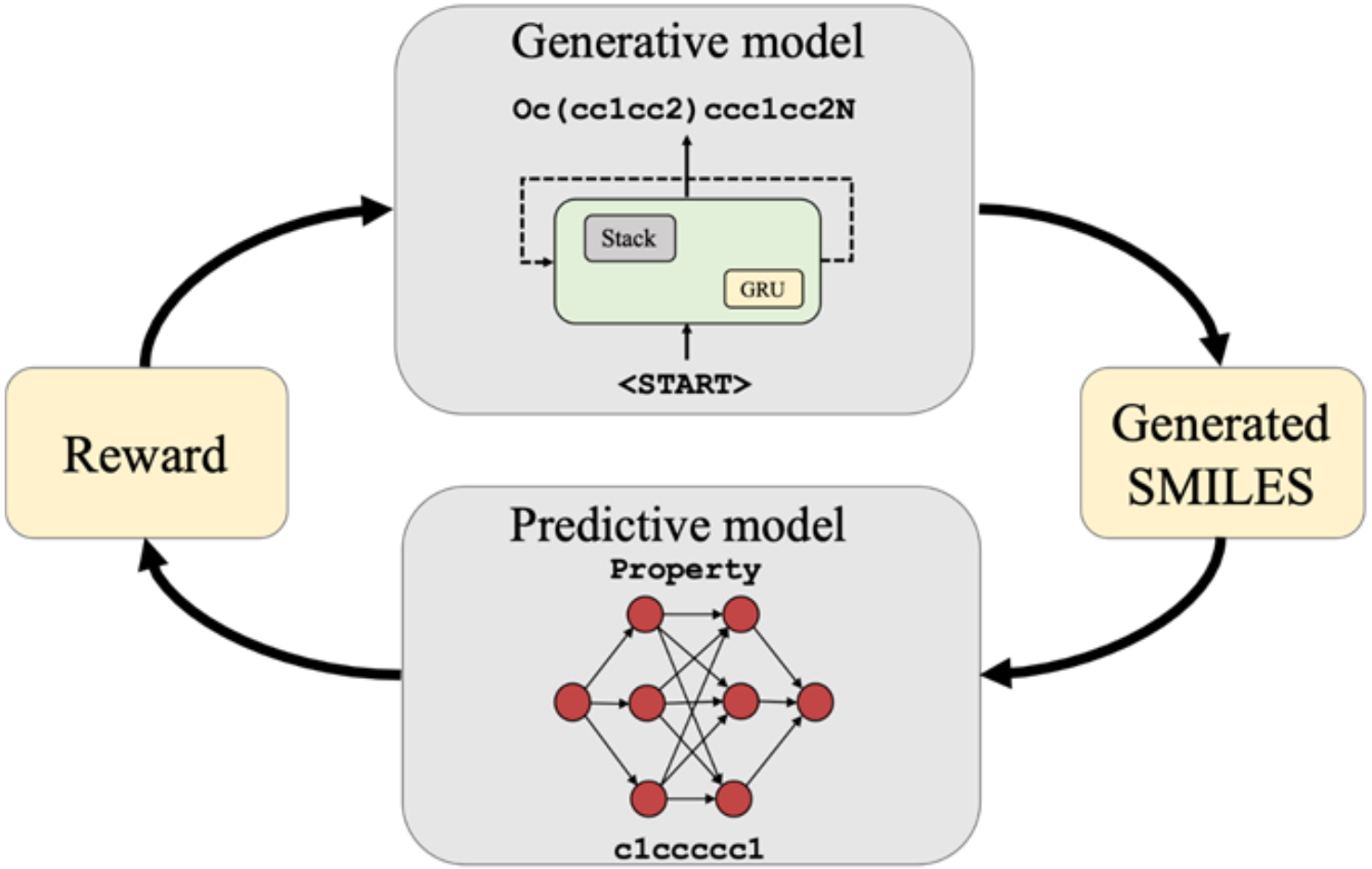
Предсказание свойств молекул (на основе только SMILES)





На входе и выходе: текстовое представление молекул – SMILES (например, C1=CC=CC=C1), преобразованное в вектор из номеров символов в алфавите:
“<>#%)(+-/.1032547698=A@CBFIHONPS[]\ceilonpsr”
Дальнейшее преобразование входа делается уже внутри сети (Embedding layer)

Объединяем генератор и предиктор для получения молекул с заданным свойством:



Валидация генерируемых сетью смайлов и их преобразование в 2D схему молекулы производится при помощи библиотеки RDKit

Добавляя штрафы, зависящие от свойства, вычисляемого при помощи этой сети, можно генерировать молекулы с нужными значениями этого свойства

Результаты:
сгенерированные молекулы и их энтальпия

H = -336.56554204305013

H = -438.1163493041992

H = -388.665784761556

H = -138.5582912597656

H = -363.78130212081317

H = -110.22050649414061