Liquid Neural Networks

Ahmadieslamloo Elaheh

Ausilio Lorenzo

Jafarpour Farshad

Prodan George

University of Padova
Laboratory of Computational Physics
Group 2210

Table of contents

I. Introduction

- I.1 Liquid neural networks
- I.2 Secondary structure prediction
- I.3 Previous results

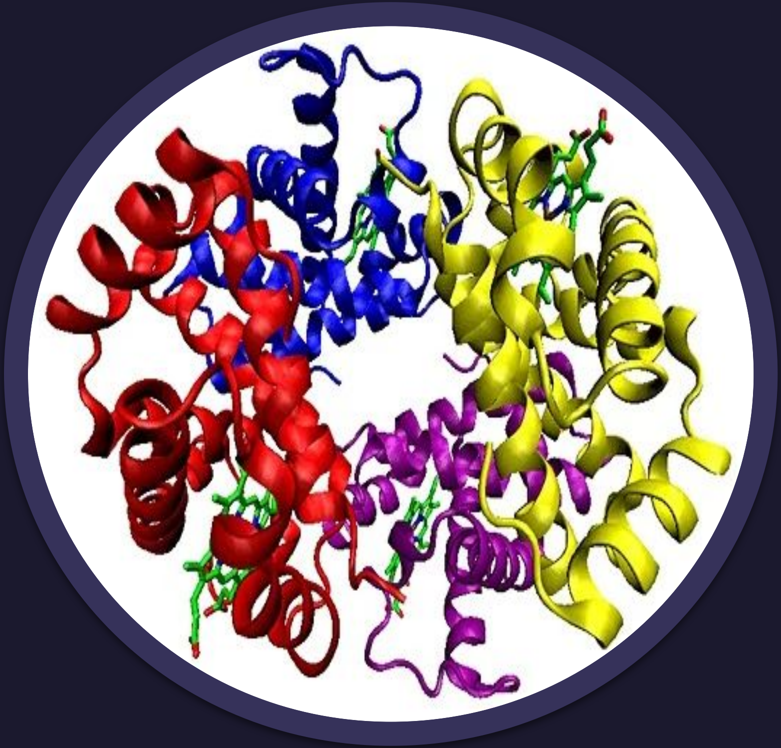
II. Methods

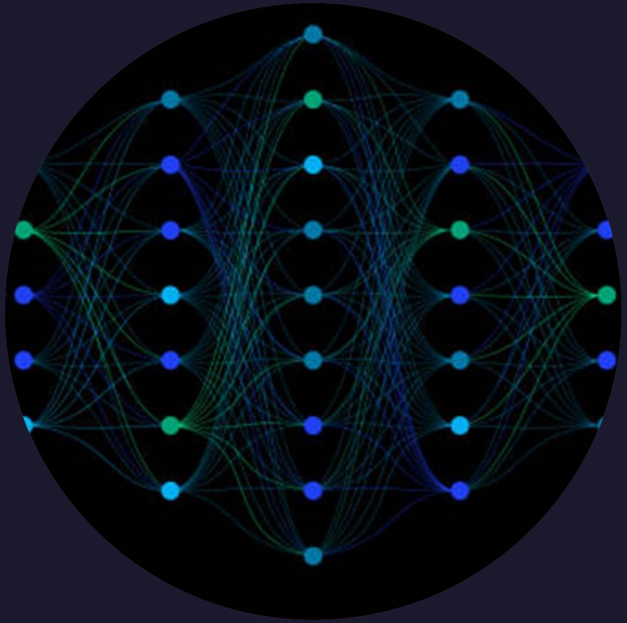
- II.1 LTC implementation
- II.2 DSSP

III. Results

- III.1 One-hot vector sequences
- III.2 Other methods

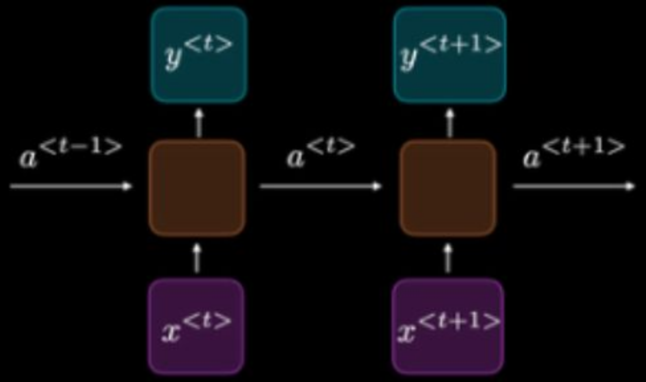
IV. Discussion and conclusion





Liquid Neural Networks

- Novel time-continuous RNN instance
- flow of hidden states is determined by ODEs
- inspired by neural dynamics in small species
- several possible application such as autonomous driving, recognition of hand gestures or human activities



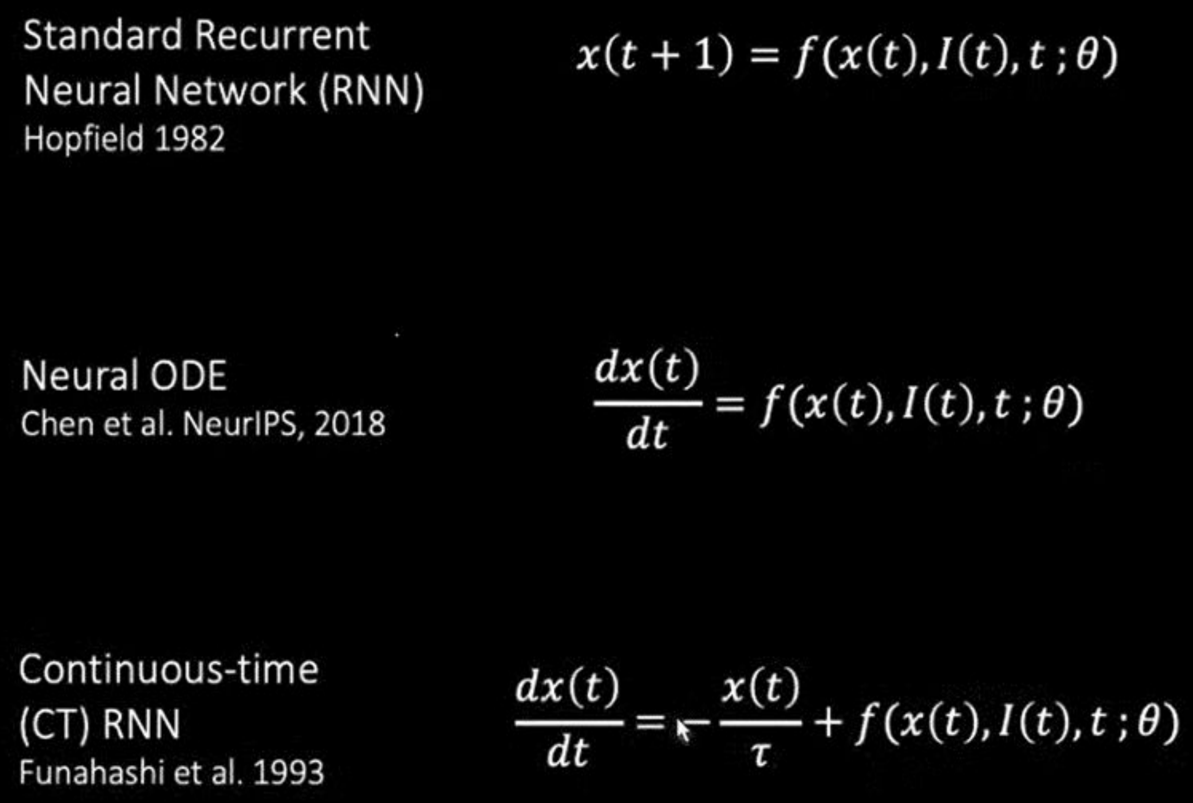
RNN Architecture

$$a^{(t+1)} = \sigma(w_a a^{(t)} + w_x x^{(t+1)} + b)$$

$$y^{(t)} = \sigma'(w' a^{(t)} + b'^{(t)})$$

Types of Recurrent Neural Networks

- Standard RNN : discretized version of continuous flow
- Neural ODE : time continuous network; the representation is a differential equation
- CT-RNN : a more stable version of Neural ODE



Standard Recurrent Neural Network (RNN)
Hopfield 1982

$$x(t+1) = f(x(t), I(t), t\,;\theta)$$

Neural ODE
Chen et al. NeurIPS, 2018

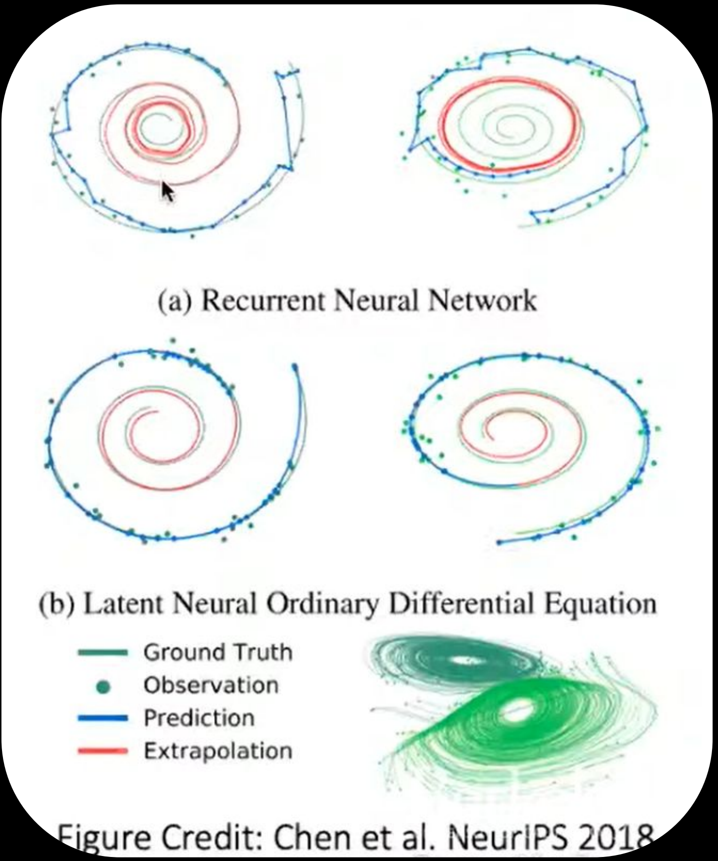
$$\frac{dx(t)}{dt} = f(x(t), I(t), t\,;\theta)$$

Continuous-time (CT) RNN
Funahashi et al. 1993

$$\frac{dx(t)}{dt} = -\frac{x(t)}{\tau} + f(x(t), I(t), t\,;\theta)$$

Standard RNNs vs ODE Networks

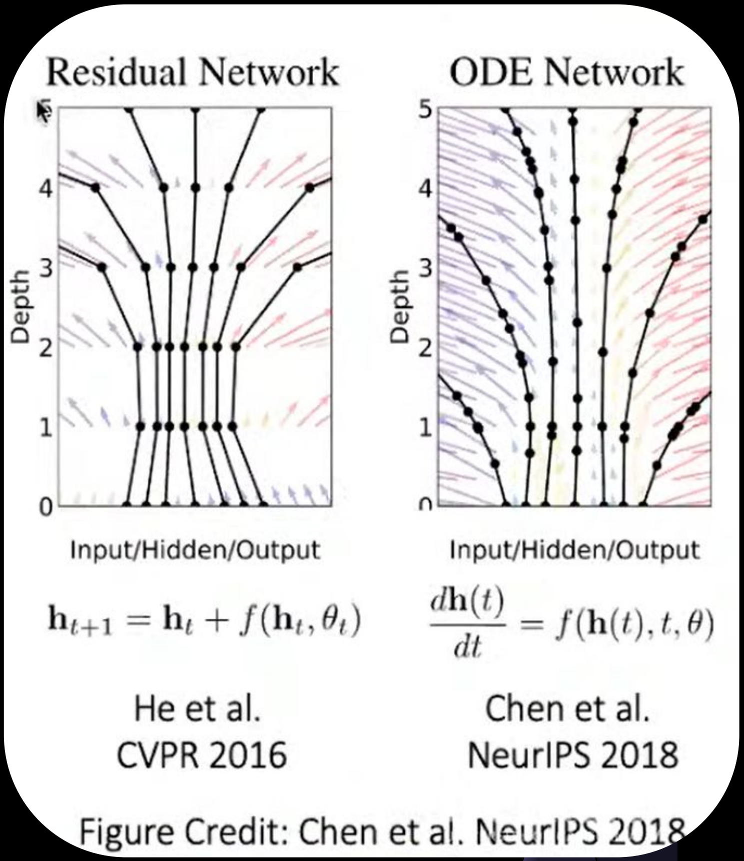
- ODE Networks : are continuous depth equivalent of a Residual Network
- Residual Networks : finite transformation of computation graphs
- Main difference : adaptive computations for ODE Networks
- transform space into a vector field



(a) Recurrent Neural Network

(b) Latent Neural Ordinary Differential Equation

Figure Credit: Chen et al. NeurIPS 2018



$$\mathbf{h}_{t+1} = \mathbf{h}_t + f(\mathbf{h}_t, \theta_t)$$

$$\frac{d\mathbf{h}(t)}{dt} = f(\mathbf{h}(t), t, \theta)$$

He et al.
CVPR 2016

Chen et al.
NeurIPS 2018

Figure Credit: Chen et al. NeurIPS 2018

LTCs : Implementation

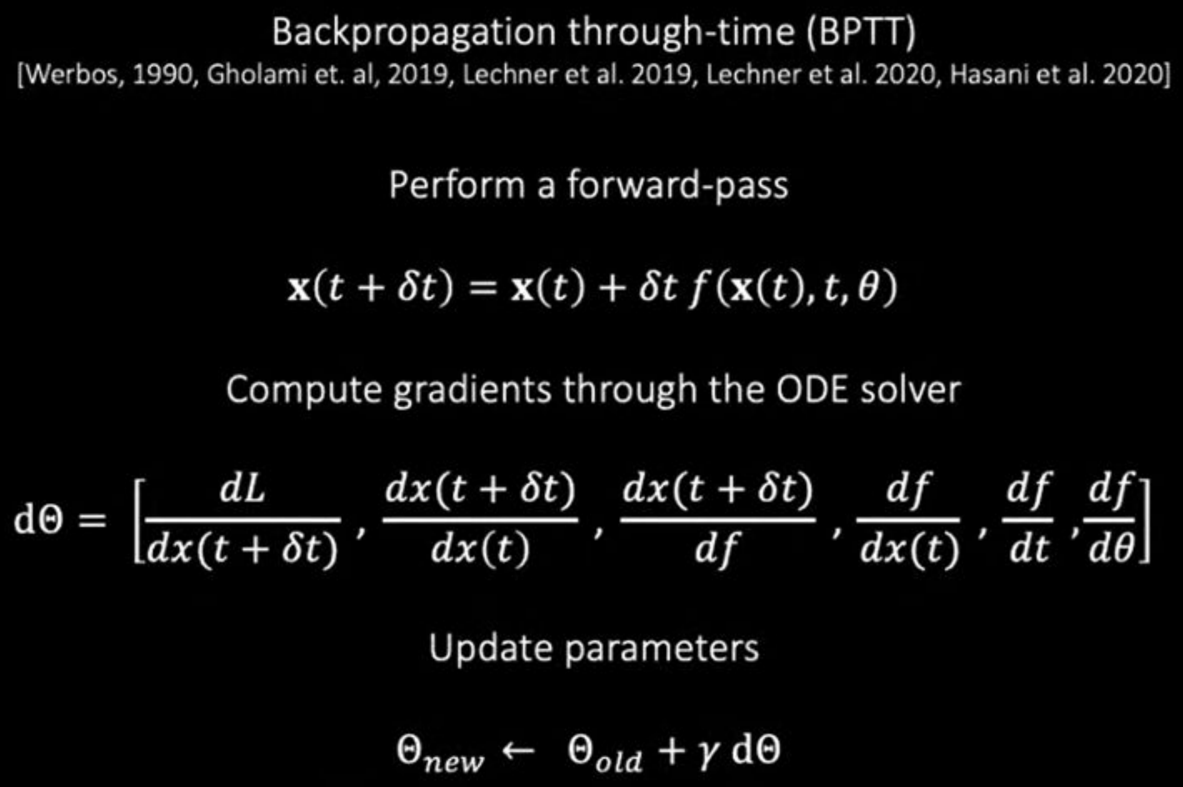
- RNN with the following hidden state equation
- Numerical ODE solvers to approximate the equation
- Forward pass complexity depends on choice of ODE solver

$$\frac{dx(t)}{dt} = f(x(t), t, \theta)$$

$$\frac{dx(t)}{dt} \approx \frac{x(t + \delta t) - x(t)}{\delta t} \approx f(x(t), t, \theta)$$
$$\Leftrightarrow \boxed{x(t + \delta t) = x(t) + \delta t\, f(x(t), t, \theta)}$$

LTCs : Training

- we use classical BPTT
- adjoint sensitivity method :
  - less computationally expensive
  - but can lead to errors



Backpropagation through-time (BPTT)

[Werbos, 1990, Gholami et. al, 2019, Lechner et al. 2019, Lechner et al. 2020, Hasani et al. 2020]

Perform a forward-pass

$$\mathbf{x}(t+\delta t) = \mathbf{x}(t) + \delta t\, f(\mathbf{x}(t), t, \theta)$$

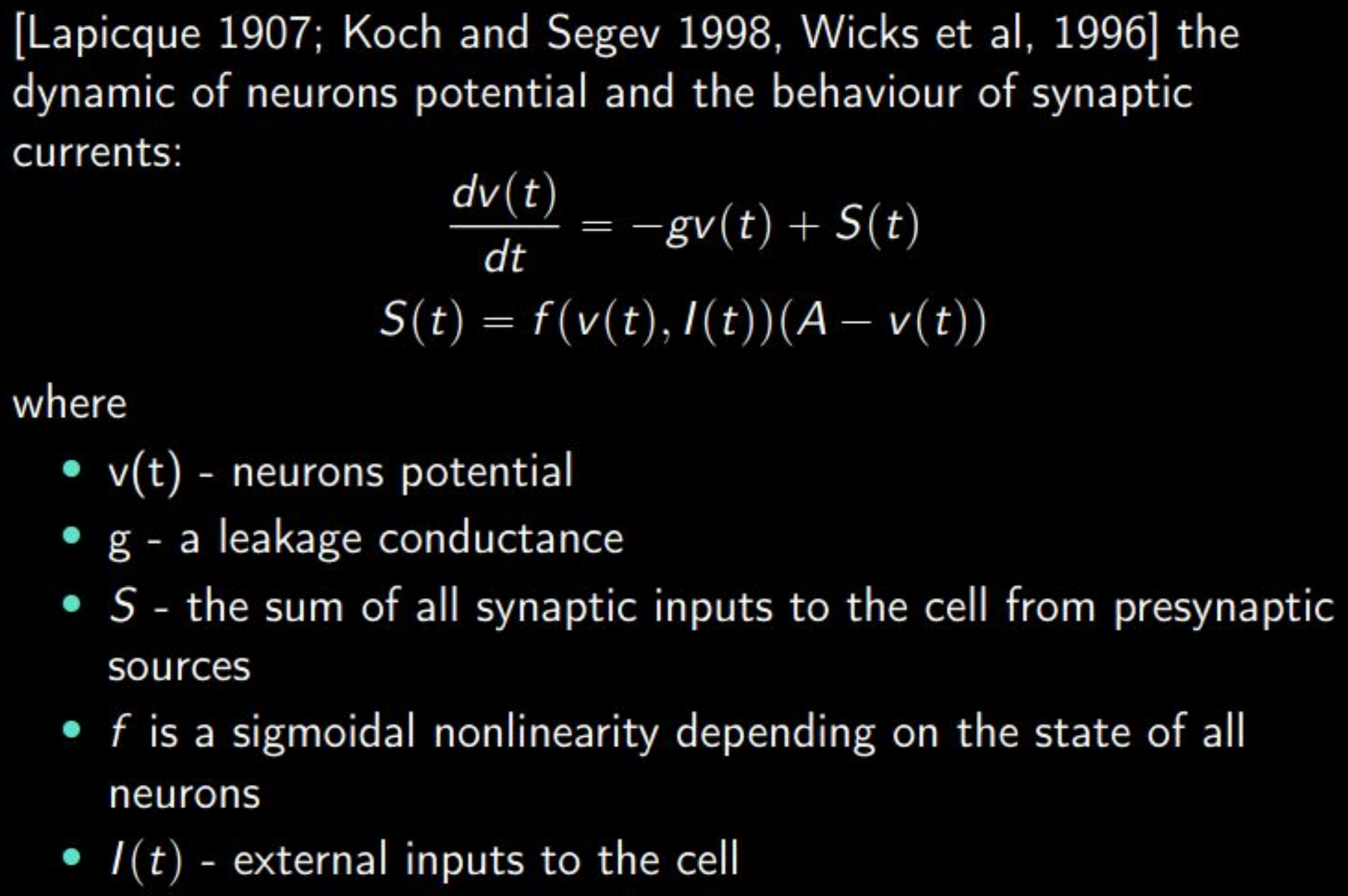
Compute gradients through the ODE solver

$$\mathrm{d}\Theta = \left[\frac{dL}{dx(t+\delta t)}, \frac{dx(t+\delta t)}{dx(t)}, \frac{dx(t+\delta t)}{df}, \frac{df}{dx(t)}, \frac{df}{dt}, \frac{df}{d\theta}\right]$$

Update parameters

$$\Theta_{new} \leftarrow \Theta_{old} + \gamma\, \mathrm{d}\Theta$$

Neural dynamics



[Lapicque 1907; Koch and Segev 1998, Wicks et al, 1996] the dynamic of neurons potential and the behaviour of synaptic currents:

$$\frac{dv(t)}{dt} = -gv(t) + S(t)$$

$$S(t) = f(v(t), I(t))(A - v(t))$$

where

- v(t) - neurons potential
- g - a leakage conductance
- S - the sum of all synaptic inputs to the cell from presynaptic sources
- f is a sigmoidal nonlinearity depending on the state of all neurons
- $I(t)$ - external inputs to the cell

Liquid Time-Constant Networks

$$d\mathbf{x}(t)/dt = -\mathbf{x}(t)/\tau + \mathbf{S}(t) \qquad \mathbf{S}(t) \in \mathbb{R}^M$$

$$\mathbf{S}(t) = f(\mathbf{x}(t), \mathbf{I}(t), t, \theta)(A - \mathbf{x}(t))$$

$$\frac{d\mathbf{x}(t)}{dt} = -\left[\frac{1}{\tau} + f(\mathbf{x}(t), \mathbf{I}(t), t, \theta)\right]\mathbf{x}(t) + f(\mathbf{x}(t), \mathbf{I}(t), t, \theta)A$$

“Liquid” = variable

$$\tau_{sys} = \frac{\tau}{1 + \tau f(\mathbf{x}(t), \mathbf{I}(t), t, \theta)}$$

System time-constant

Previous results using LNNs

Thank you!

Mathias Lechner Alexander Amini Daniela Rus Radu Grosu

Feel free to

Check out our latest repositories

https://github.com/raminmh/liquid_time_constant_networks

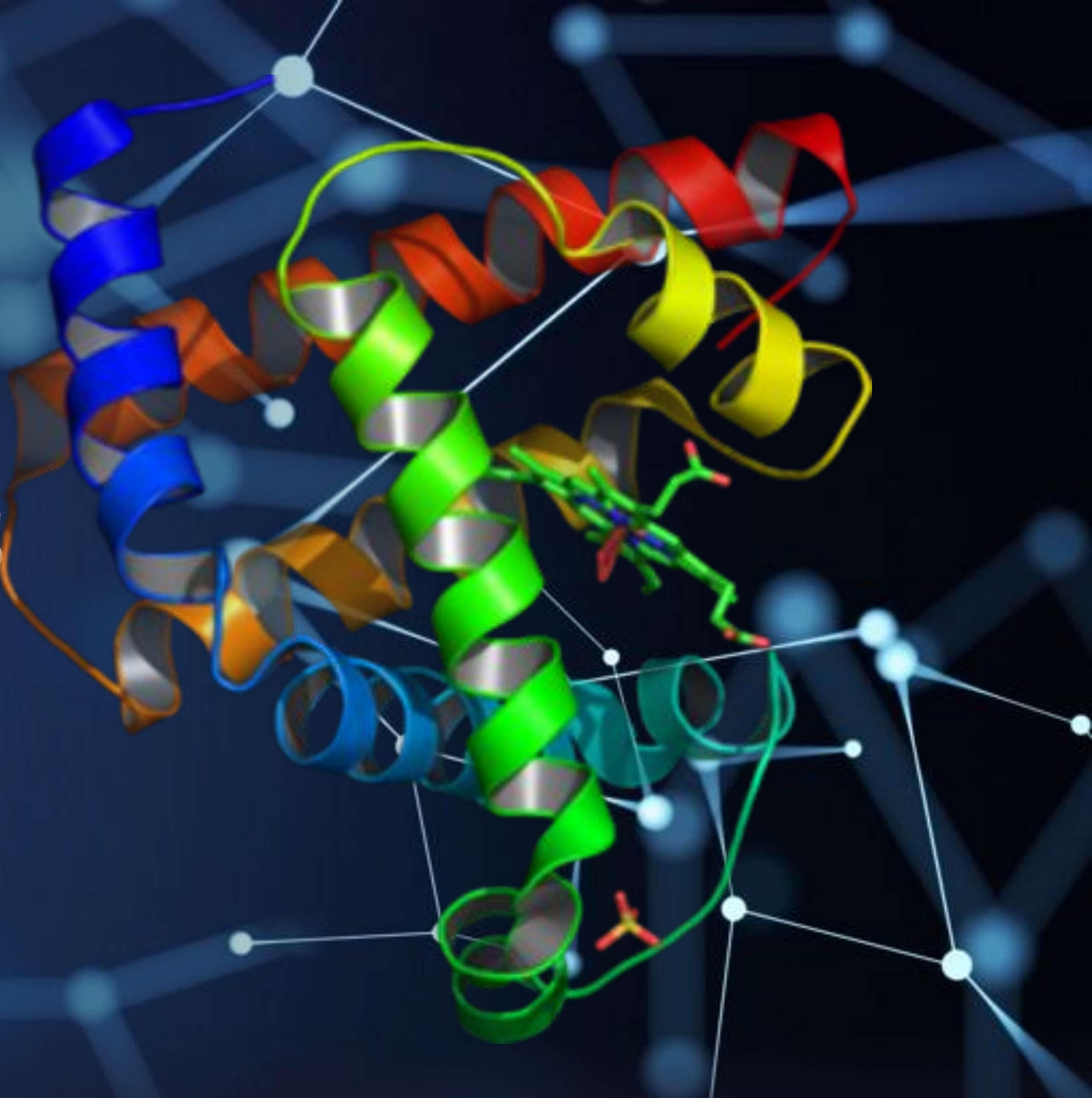
https://github.com/mlech261/keras-ncp

https://github.com/mlech261/ode-lstms

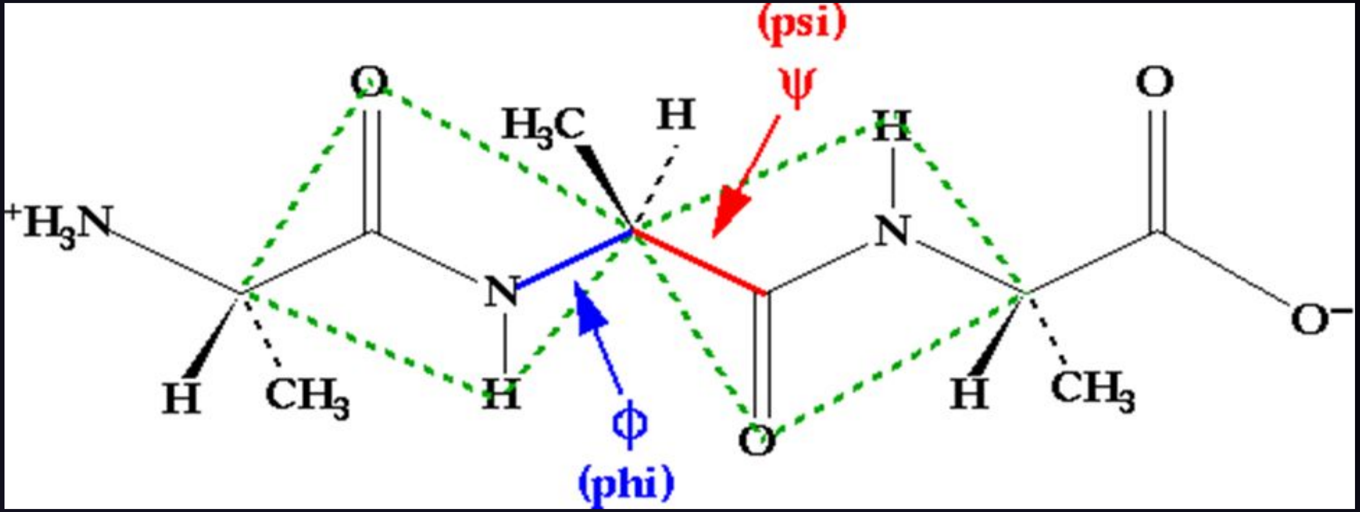
and to reach out:
rhasani@mit.edu

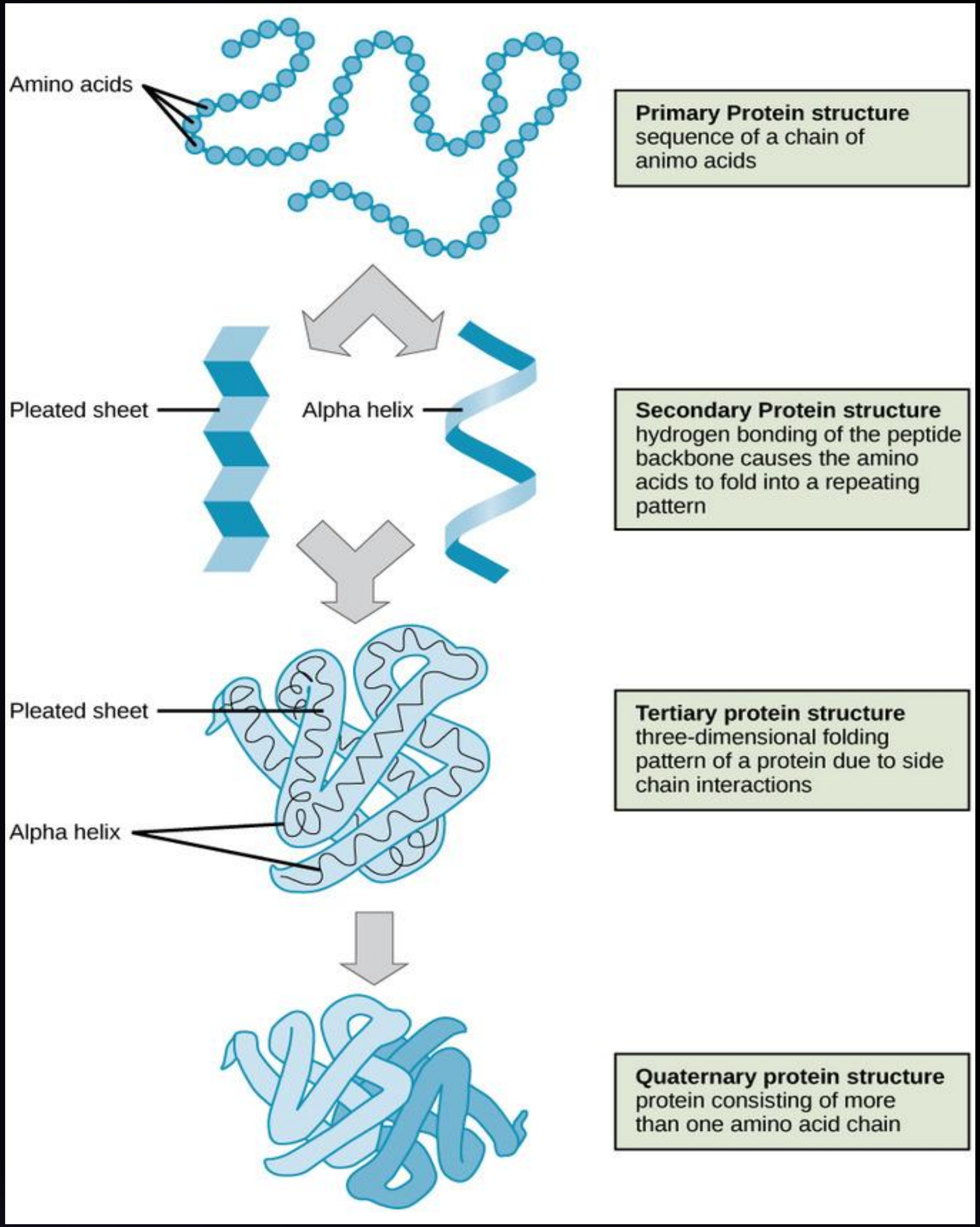
Liquid Time-Constant Networks (LTCs)
Hasani et al. AAAI 2021

Protein structures



Structural hierarchy of proteins





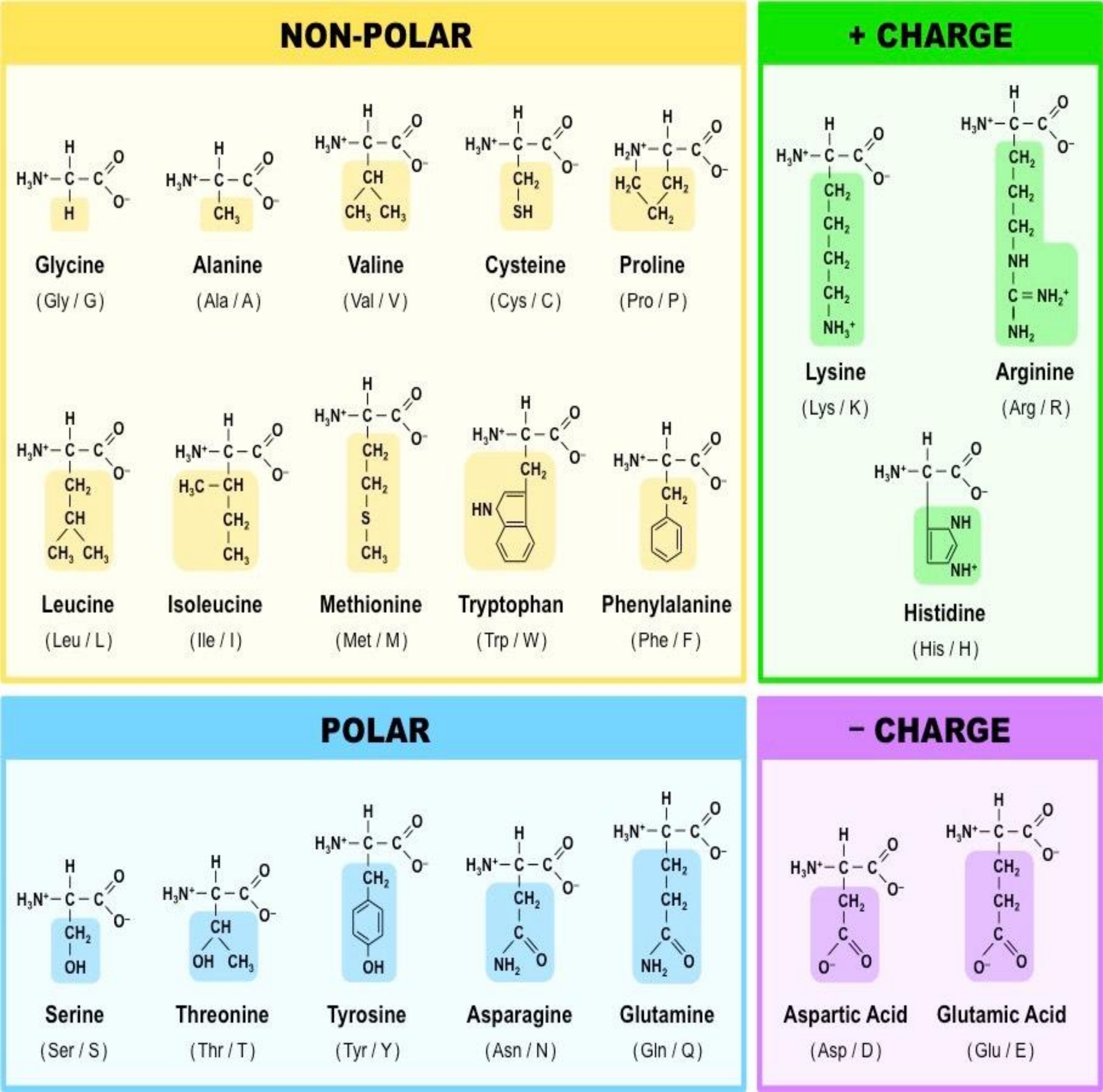
Primary Protein structure
sequence of a chain of animo acids

Secondary Protein structure
hydrogen bonding of the peptide backbone causes the amino acids to fold into a repeating pattern

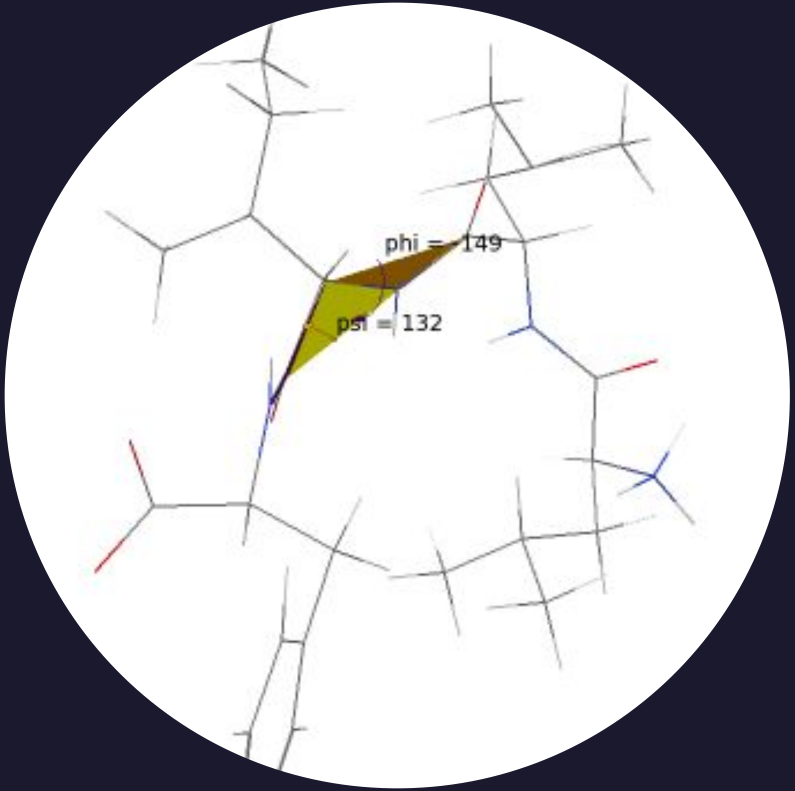
Tertiary protein structure
three-dimensional folding pattern of a protein due to side chain interactions

Quaternary protein structure
protein consisting of more than one amino acid chain

Types of amino acids



DSSP classification



G = 3-turn helix (310 helix). Min length 3 residues.

H = 4-turn helix (α helix). Minimum length 4 residues.

I = 5-turn helix (π helix). Minimum length 5 residues.

T = hydrogen bonded turn (3, 4 or 5 turn)

E = extended strand in parallel and/or anti-parallel β-sheet conformation. Min length 2 residues.

B = residue in isolated β-bridge (single pair β-sheet hydrogen bond formation)

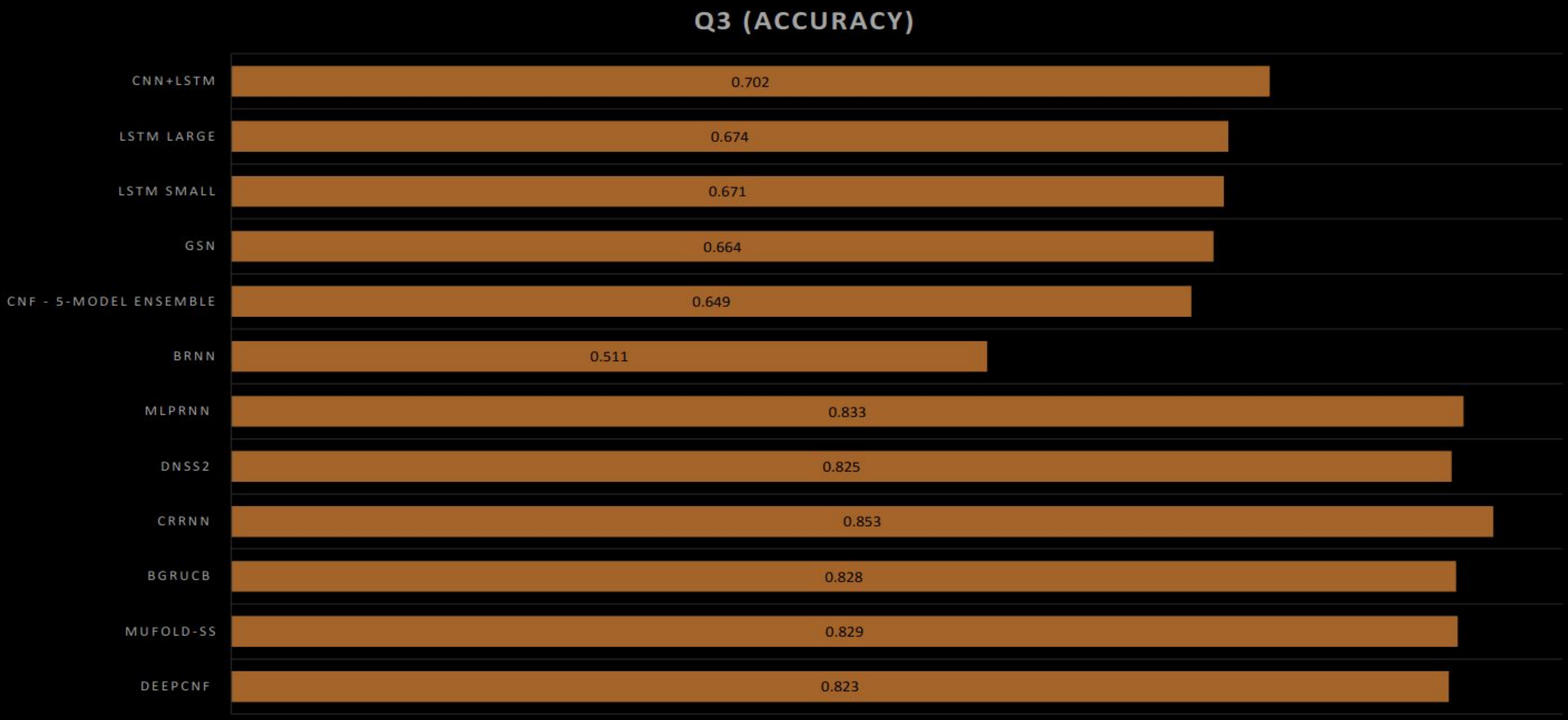
S = bend (the only non-hydrogen-bond based assignment).

C = coil (residues which are not in any of the above conformations).

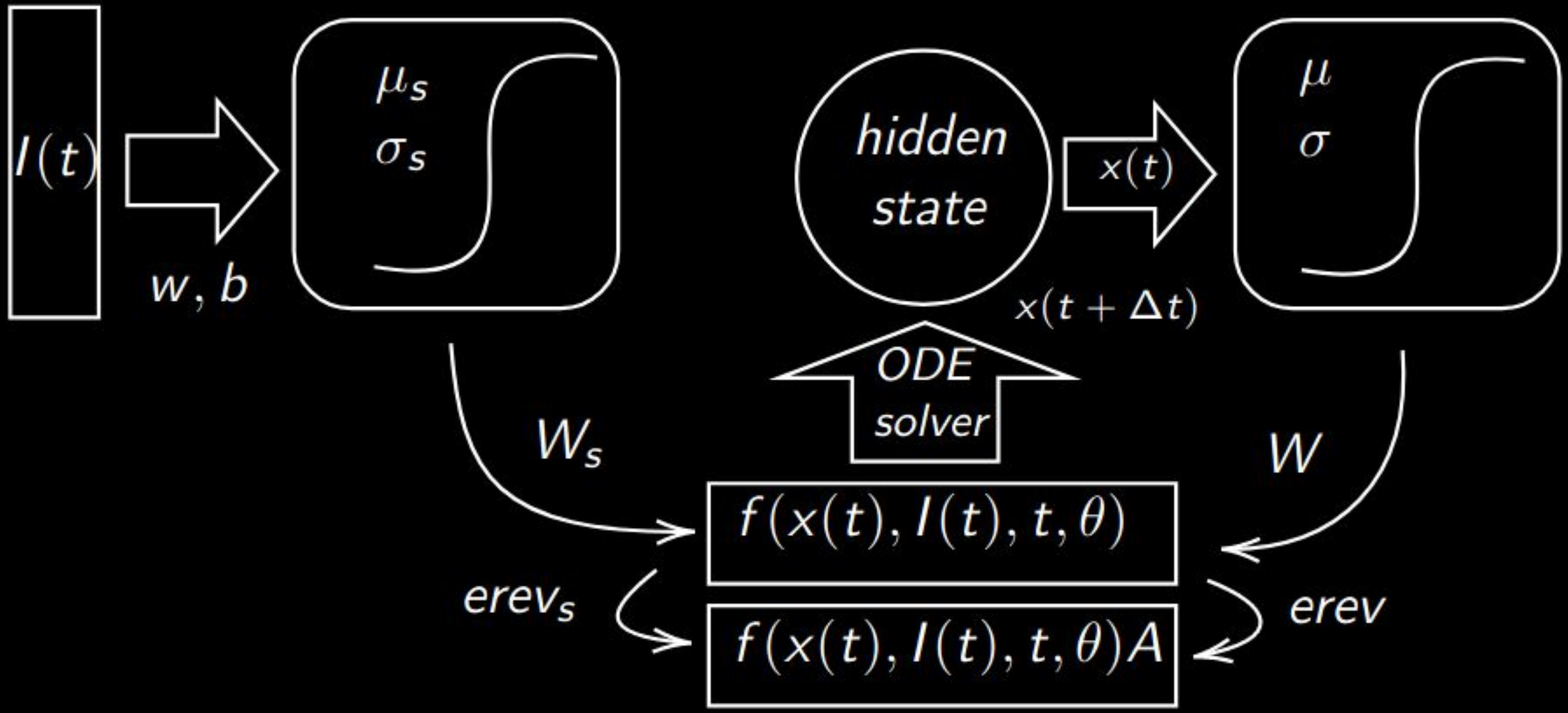
Other methods for prediction

- Statistical Analysis
- Information theory (GOR)
- Hydrophobicity profiles
- Multiple Sequence Alignment
- ML Technology
- Joined

Previous results on PSSP

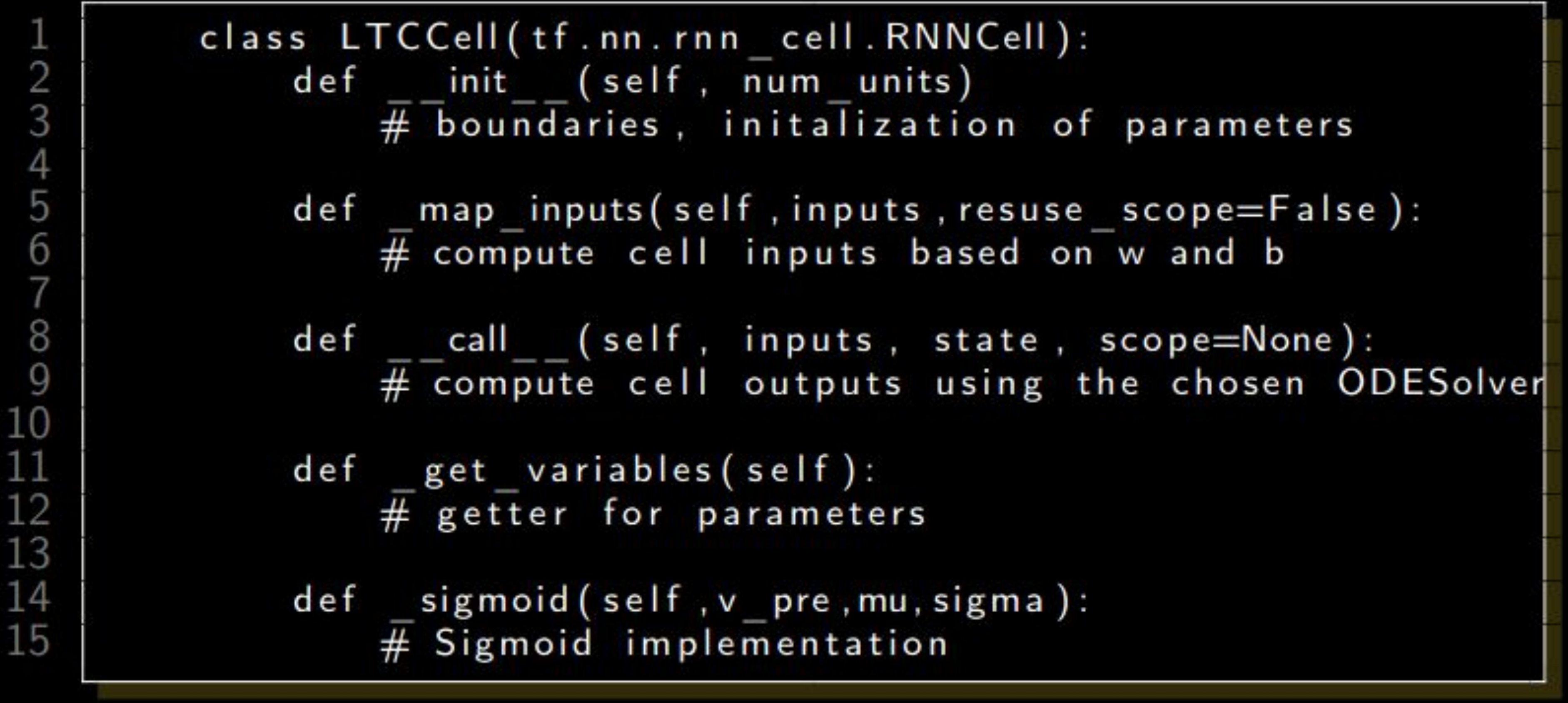


Network implementation



Network implementation

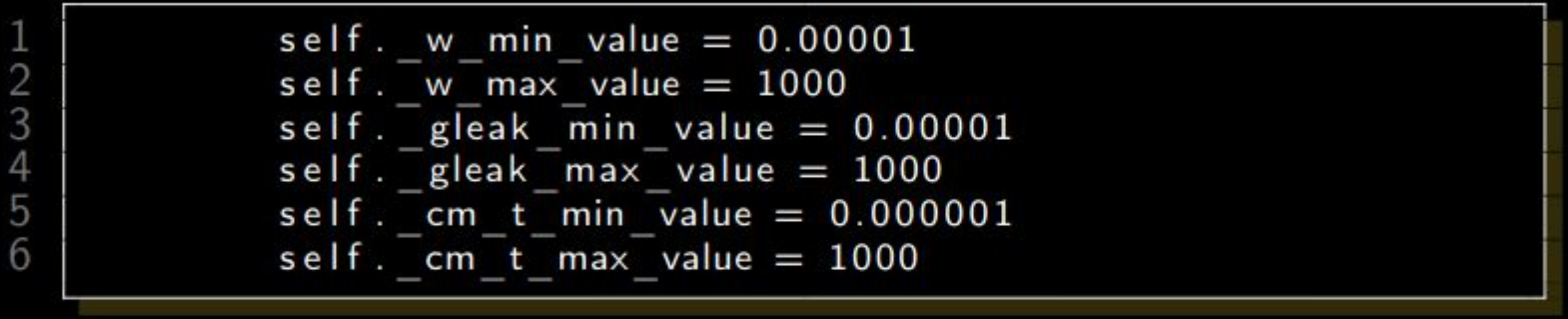
Liquid NN class (implemented as RNN subclass)



```
class LTCCell(tf.nn.rnn_cell.RNNCell):
    def __init__(self, num_units)
        # boundaries, initalization of parameters

    def _map_inputs(self,inputs,resuse_scope=False):
        # compute cell inputs based on w and b

    def __call__(self, inputs, state, scope=None):
        # compute cell outputs using the chosen ODESolver

    def _get_variables(self):
        # getter for parameters

    def _sigmoid(self,v_pre,mu,sigma):
        # Sigmoid implementation
```

Network implementation

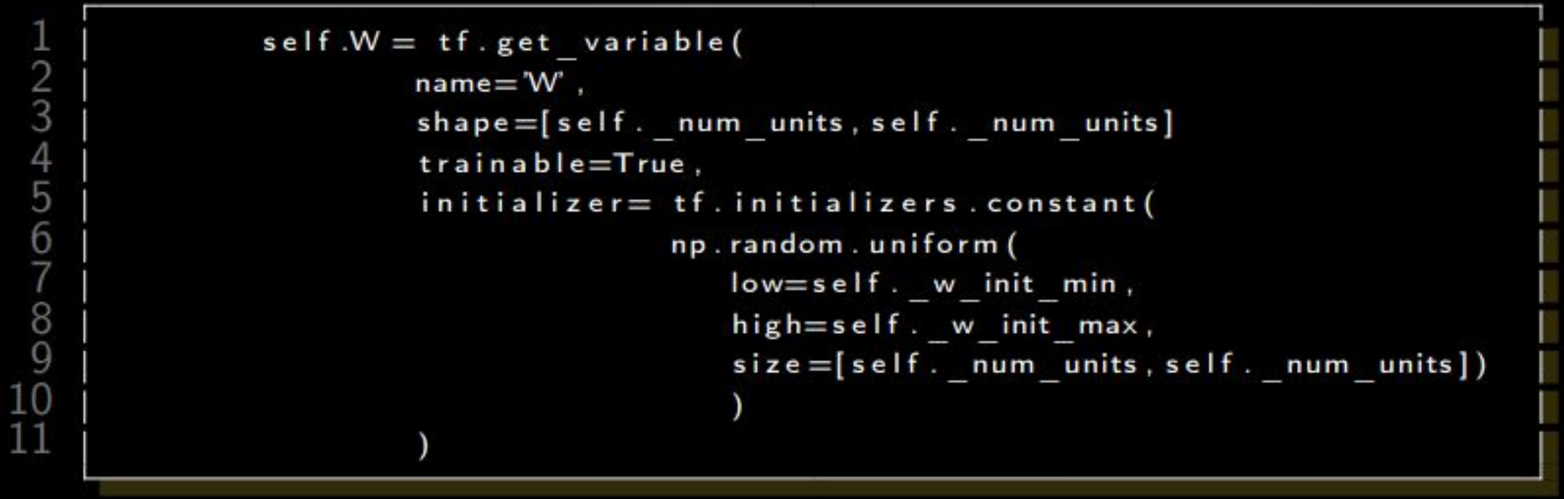
Boundaries of trainable parameters



```
self._w_min_value = 0.00001
self._w_max_value = 1000
self._gleak_min_value = 0.00001
self._gleak_max_value = 1000
self._cm_t_min_value = 0.000001
self._cm_t_max_value = 1000
```

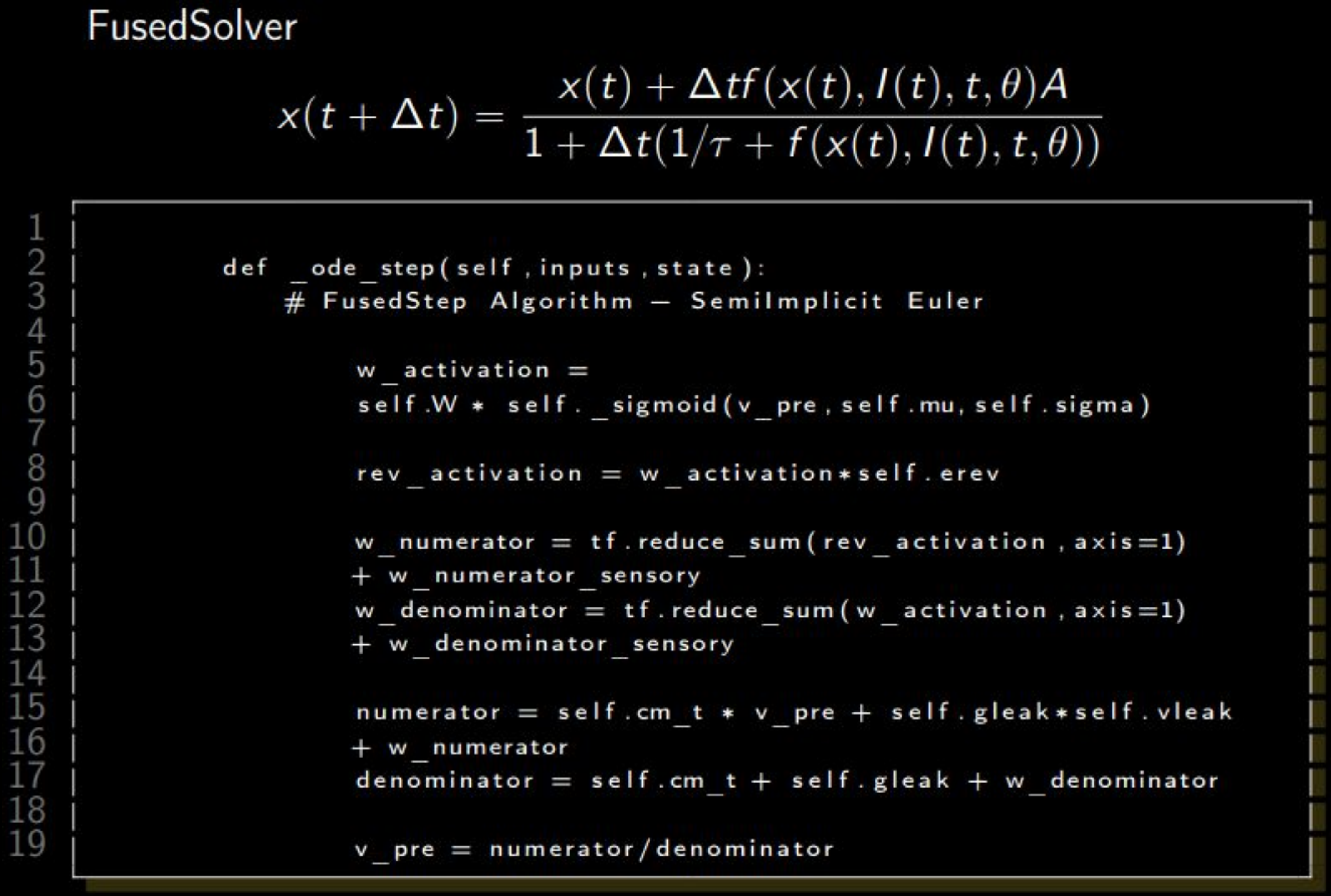
Define a trainable variable

e.g. W - the matrix of weights involved in the hidden state output



```
self.W = tf.get_variable(
    name='W',
    shape=[self._num_units, self._num_units]
    trainable=True,
    initializer= tf.initializers.constant(
            np.random.uniform(
              low=self._w_init_min,
              high=self._w_init_max,
              size=[self._num_units, self._num_units])
              )
    )
```

Network implementation



FusedSolver

$$x(t+\Delta t) = \frac{x(t) + \Delta t f(x(t), I(t), t, \theta)A}{1 + \Delta t(1/\tau + f(x(t), I(t), t, \theta))}$$

```
def _ode_step(self, inputs, state):
    # FusedStep Algorithm — SemiImplicit Euler

        w_activation =
        self.W * self._sigmoid(v_pre, self.mu, self.sigma)

        rev_activation = w_activation*self.erev

        w_numerator = tf.reduce_sum(rev_activation, axis=1)
        + w_numerator_sensory
        w_denominator = tf.reduce_sum(w_activation, axis=1)
        + w_denominator_sensory

        numerator = self.cm_t * v_pre + self.gleak*self.vleak
        + w_numerator
        denominator = self.cm_t + self.gleak + w_denominator

        v_pre = numerator/denominator
```

Datasets used for PSSP

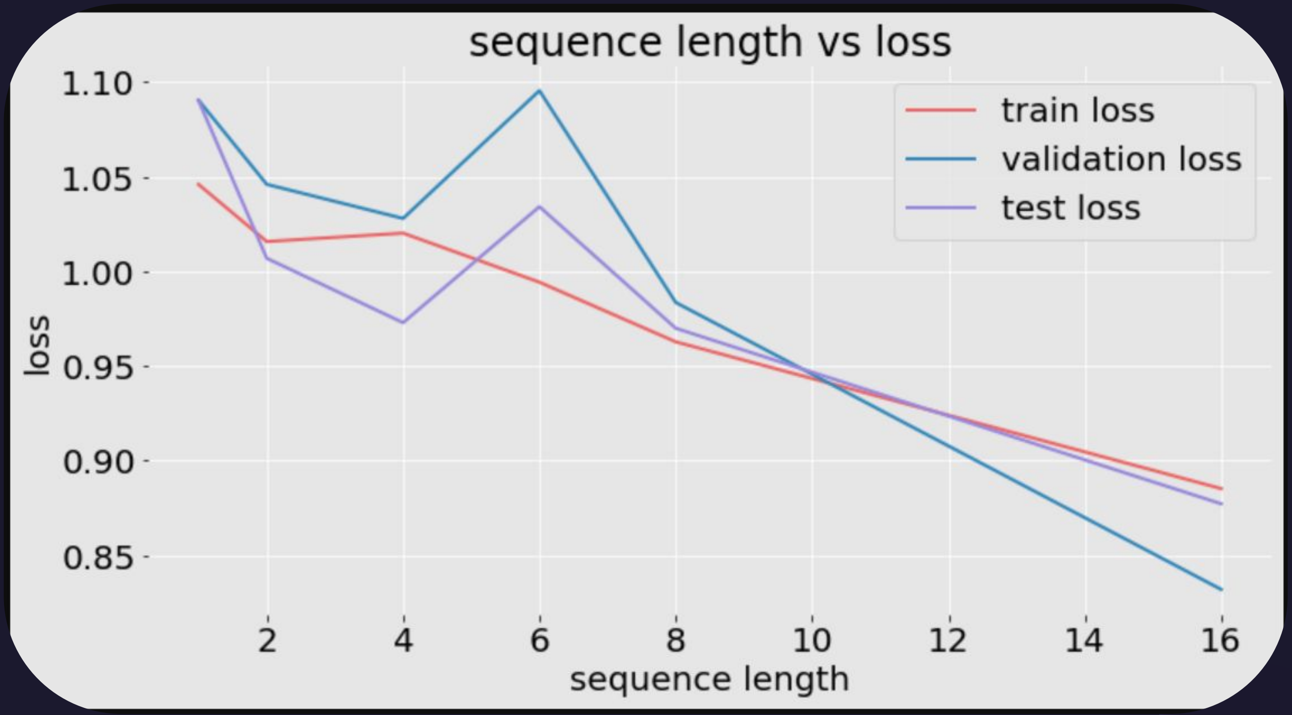
Name of dataset	Number of proteins	sequence lengths range
CB513	514	20 - 874
CASP12	21	76 - 1494
PDB	9079	20 - 1632
CATH40	31731	18 - 497

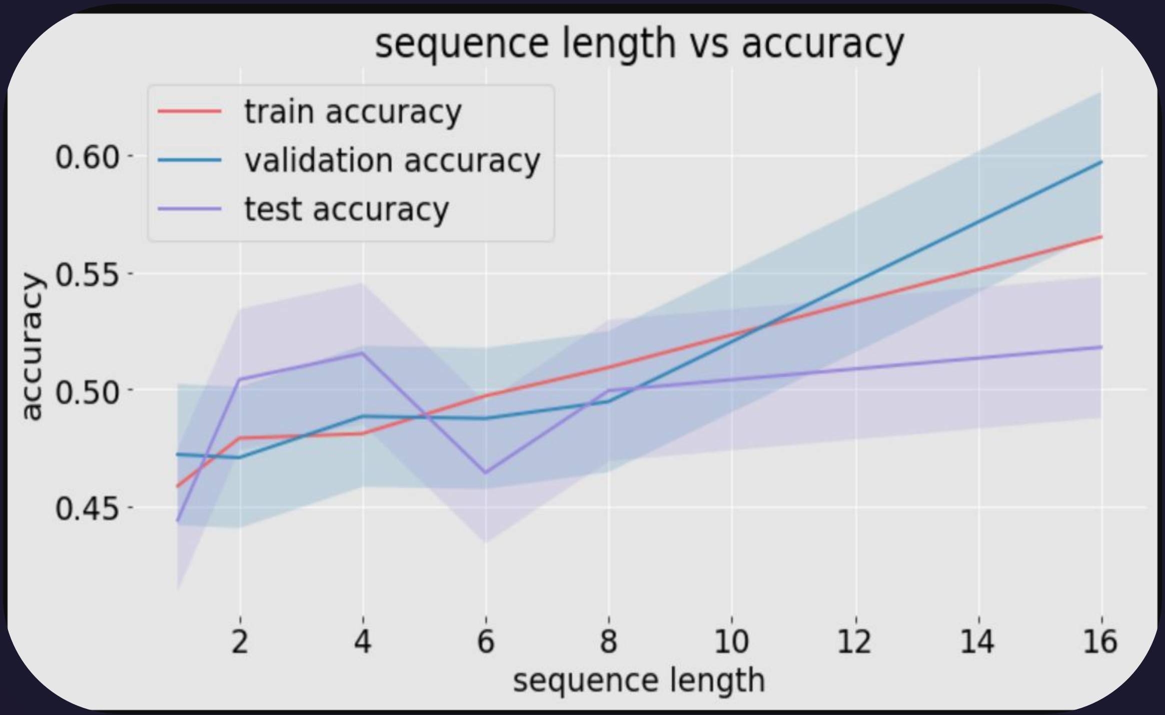
Results

- One-hot vector inputs
  - Varying sequence length
  - Full protein sequences
  - GridSearch

- Inputs based on other methods
  - NNI, NNIH methods
  - GridSearch
  - Testing on multiple datasets

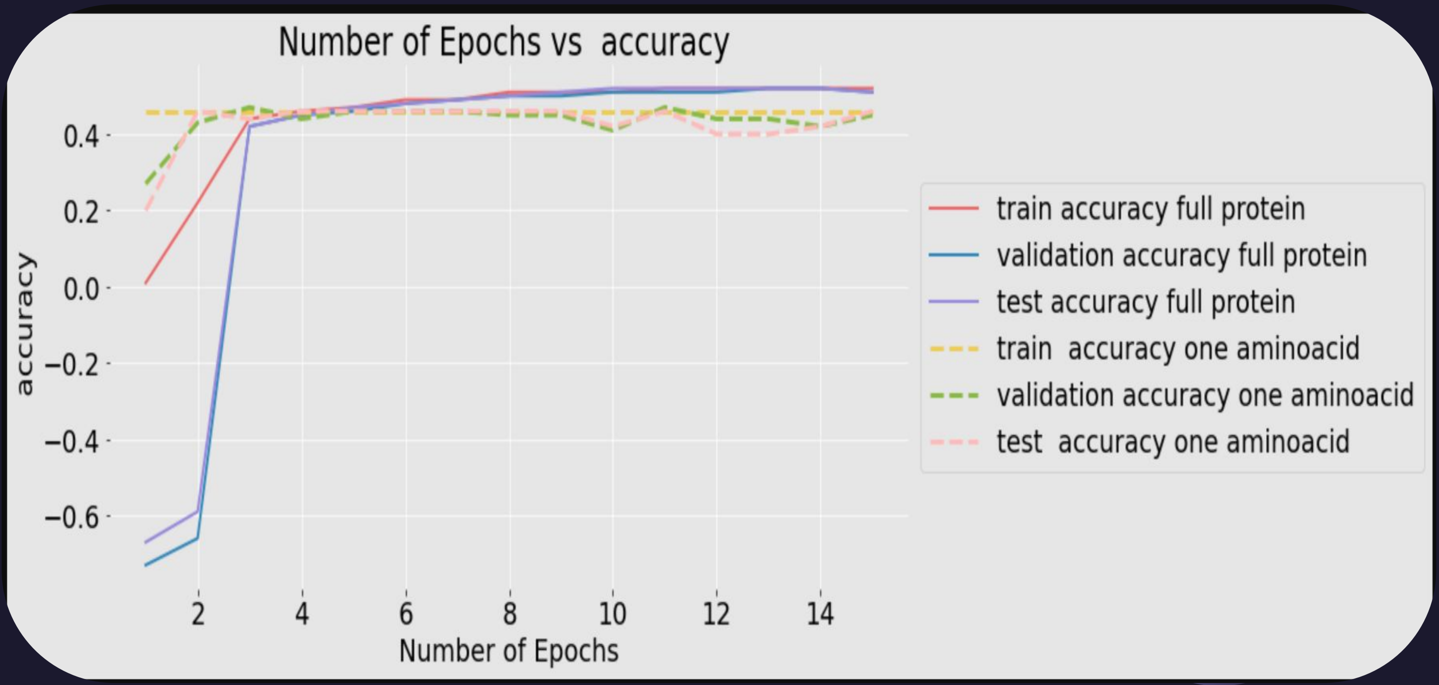
Varying sequence length





Full protein sequence

- Load data from JSON for each protein
- Create the sequences using delimiters between proteins



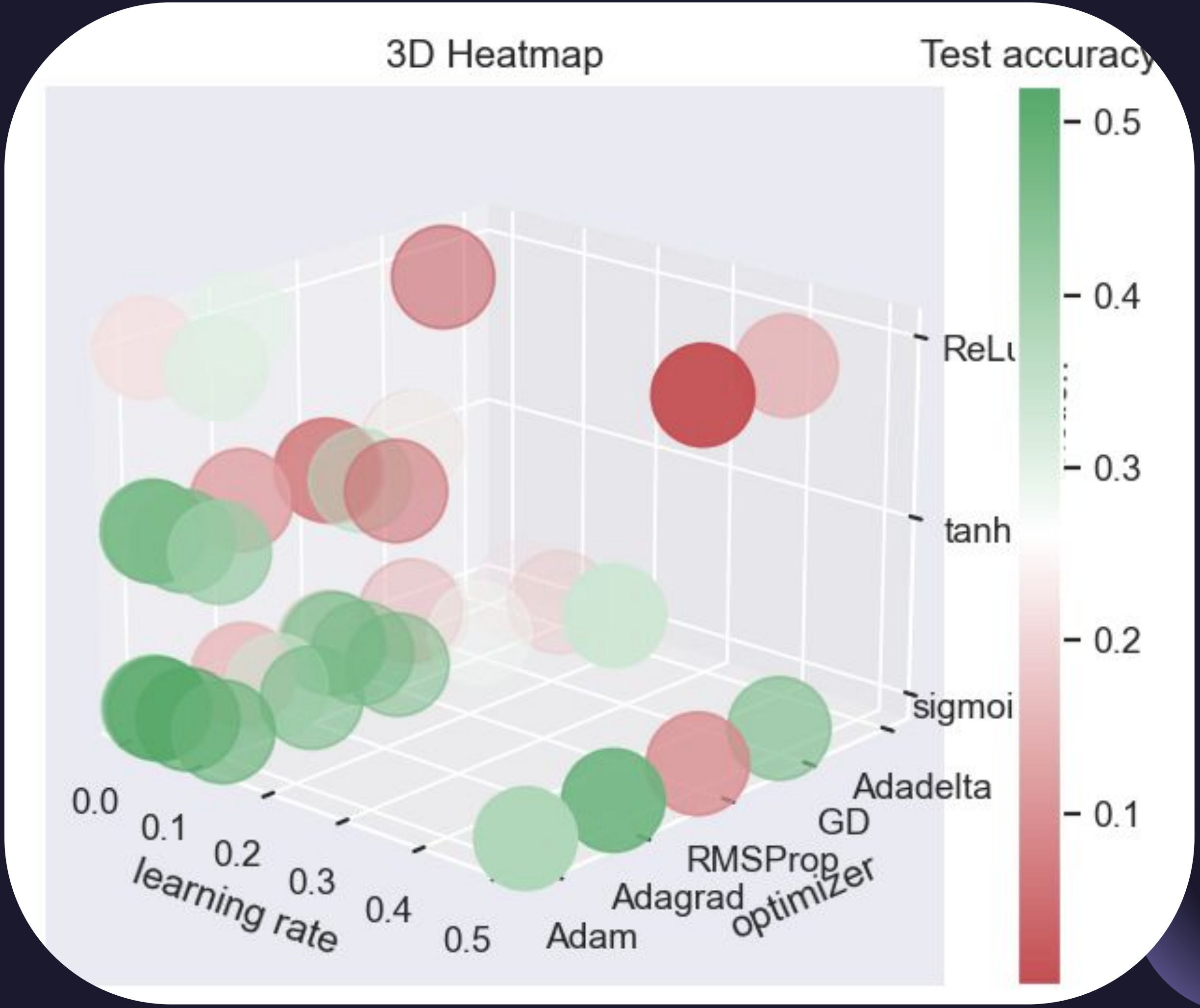
DSSP data

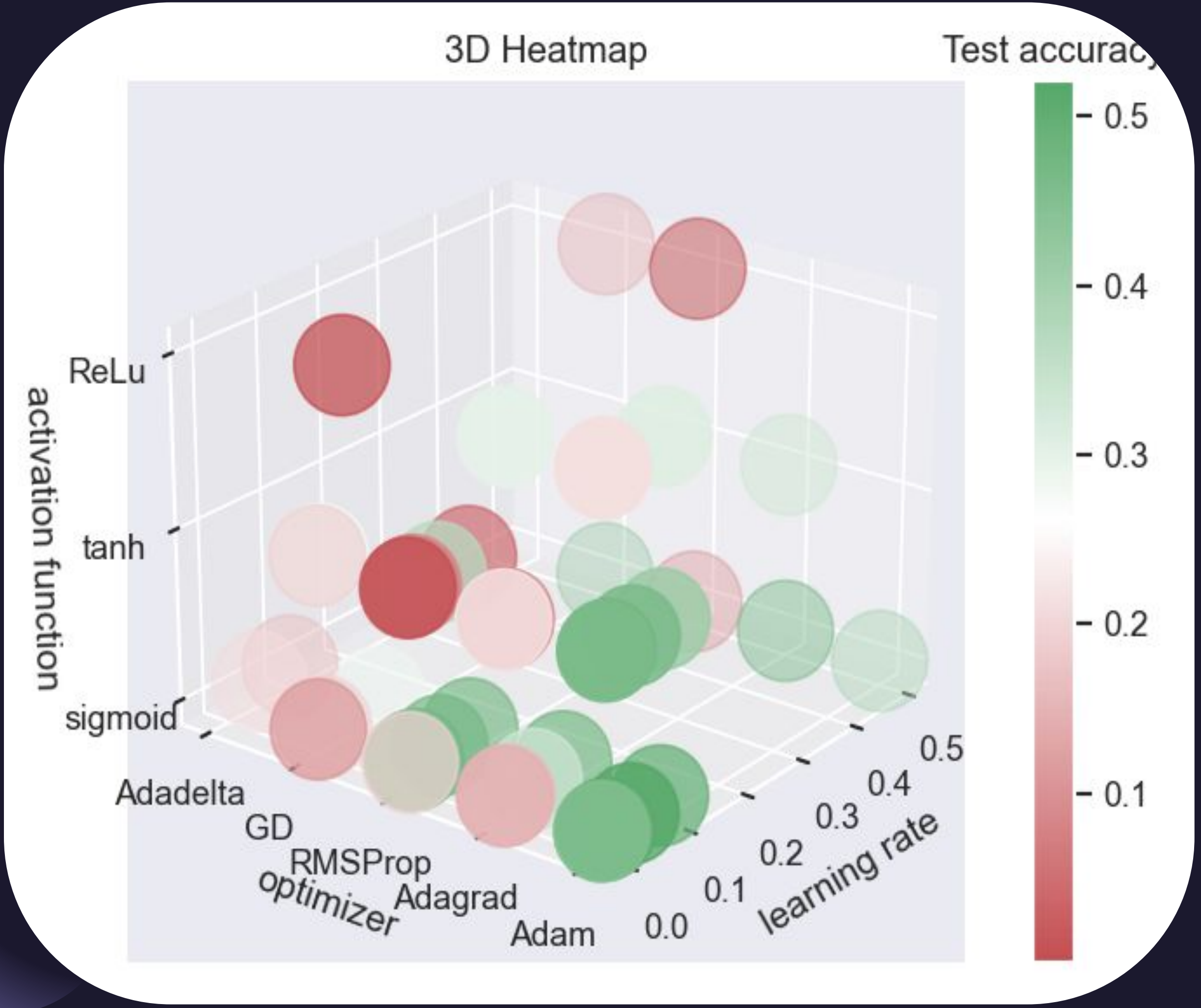
JSON format

```
[
        {
        "id": "# 123EX00.dssp",
        "labels": [ "H", "H", …
        "aminoacids": [ "A", "G", …
},

# next protein

{ … },

…]
```

GridSearch

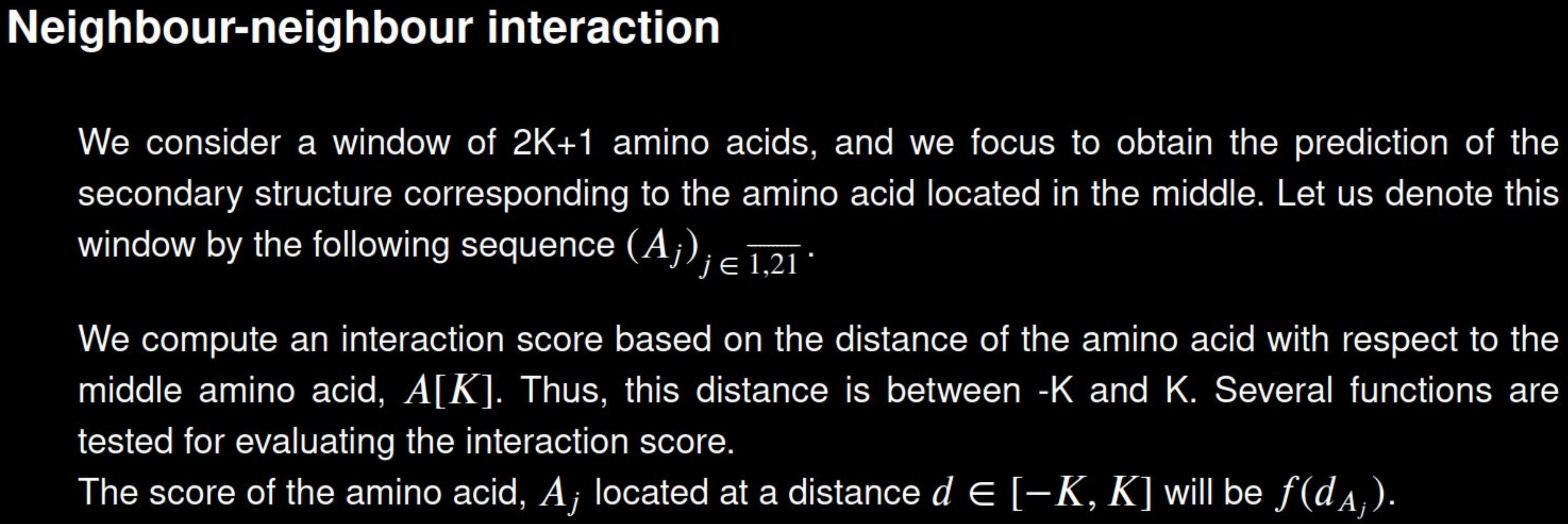
```
100%|██████████| 90/90
[7:17:47<00:00, 291.86s/it]
```





WINNER optimizer: Adam, LR: 0.05, activation: sigmoid

Developed methods



Neighbour-neighbour interaction

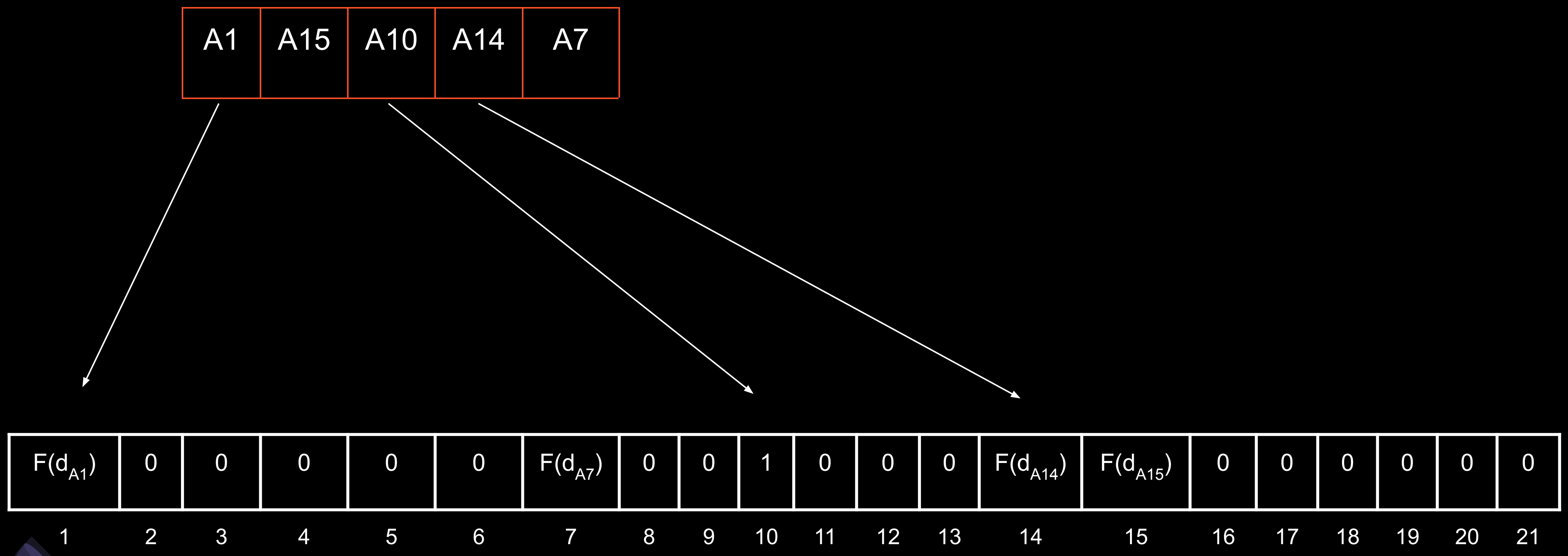
We consider a window of 2K+1 amino acids, and we focus to obtain the prediction of the secondary structure corresponding to the amino acid located in the middle. Let us denote this window by the following sequence $(A_j)_{j \in \overline{1,21}}$.

We compute an interaction score based on the distance of the amino acid with respect to the middle amino acid, $A[K]$. Thus, this distance is between -K and K. Several functions are tested for evaluating the interaction score.
The score of the amino acid, A_j located at a distance $d \in [-K, K]$ will be $f(d_{A_j})$.

A1	A15	A10	A14	A7

K=2 (window of 5 amino acids)
Middle amino acid: A10

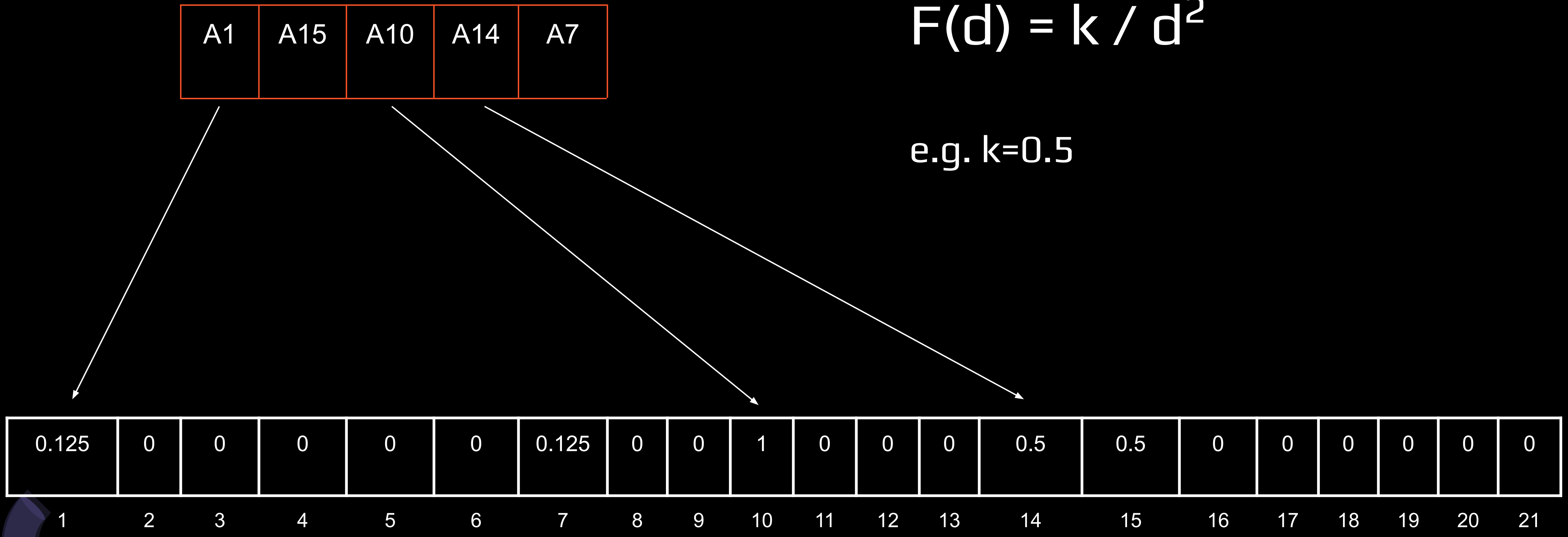
Neighbour-neighbour interaction



A1	A15	A10	A14	A7

F(d_{A1})	0	0	0	0	0	F(d_{A7})	0	0	1	0	0	0	F(d_{A14})	F(d_{A15})	0	0	0	0	0	0
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

Neighbour-neighbour interaction



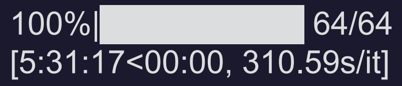
A1	A15	A10	A14	A7

0.125	0	0	0	0	0	0.125	0	0	1	0	0	0	0.5	0.5	0	0	0	0	0	0
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

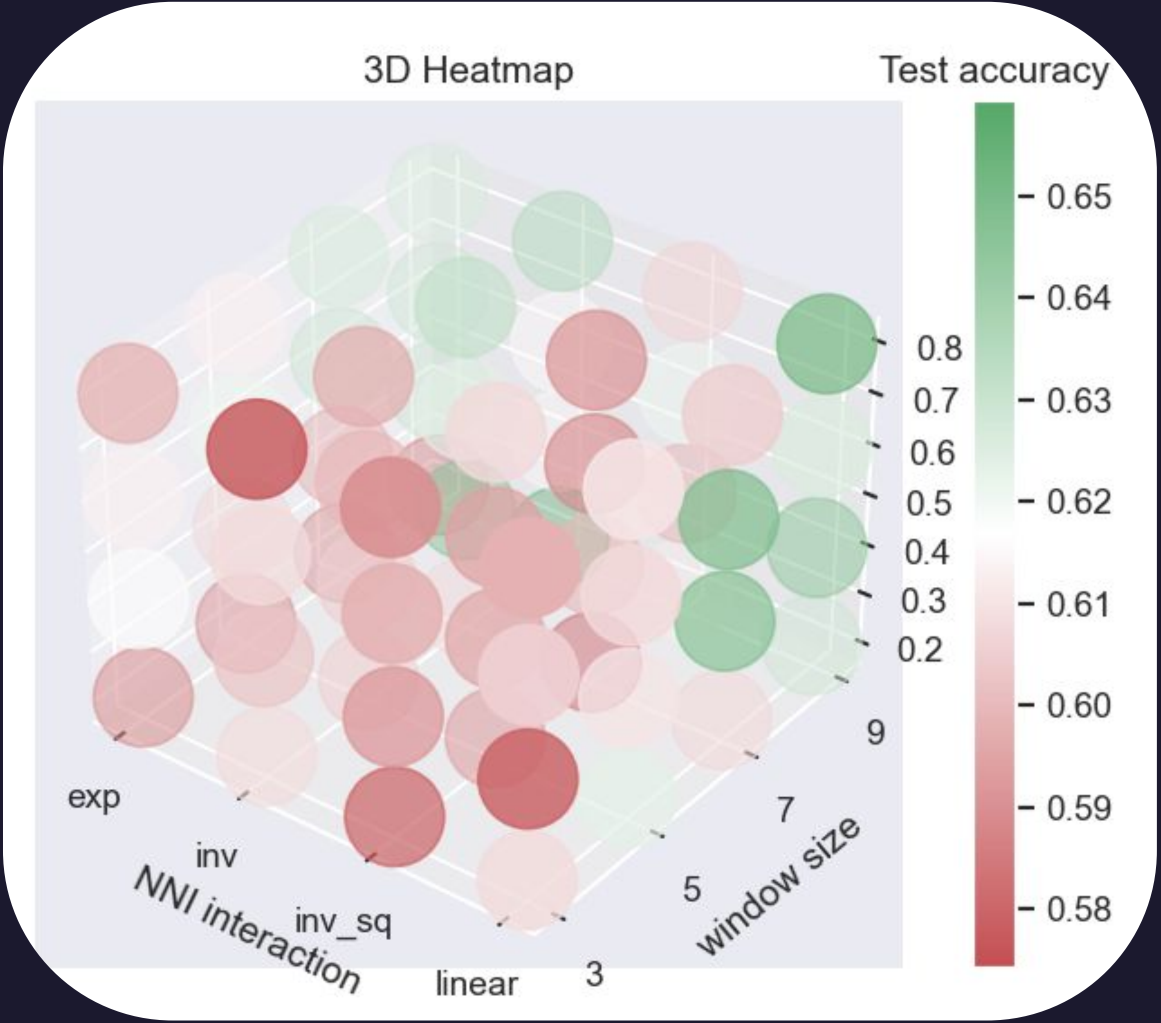
$$F(d) = k / d^2$$

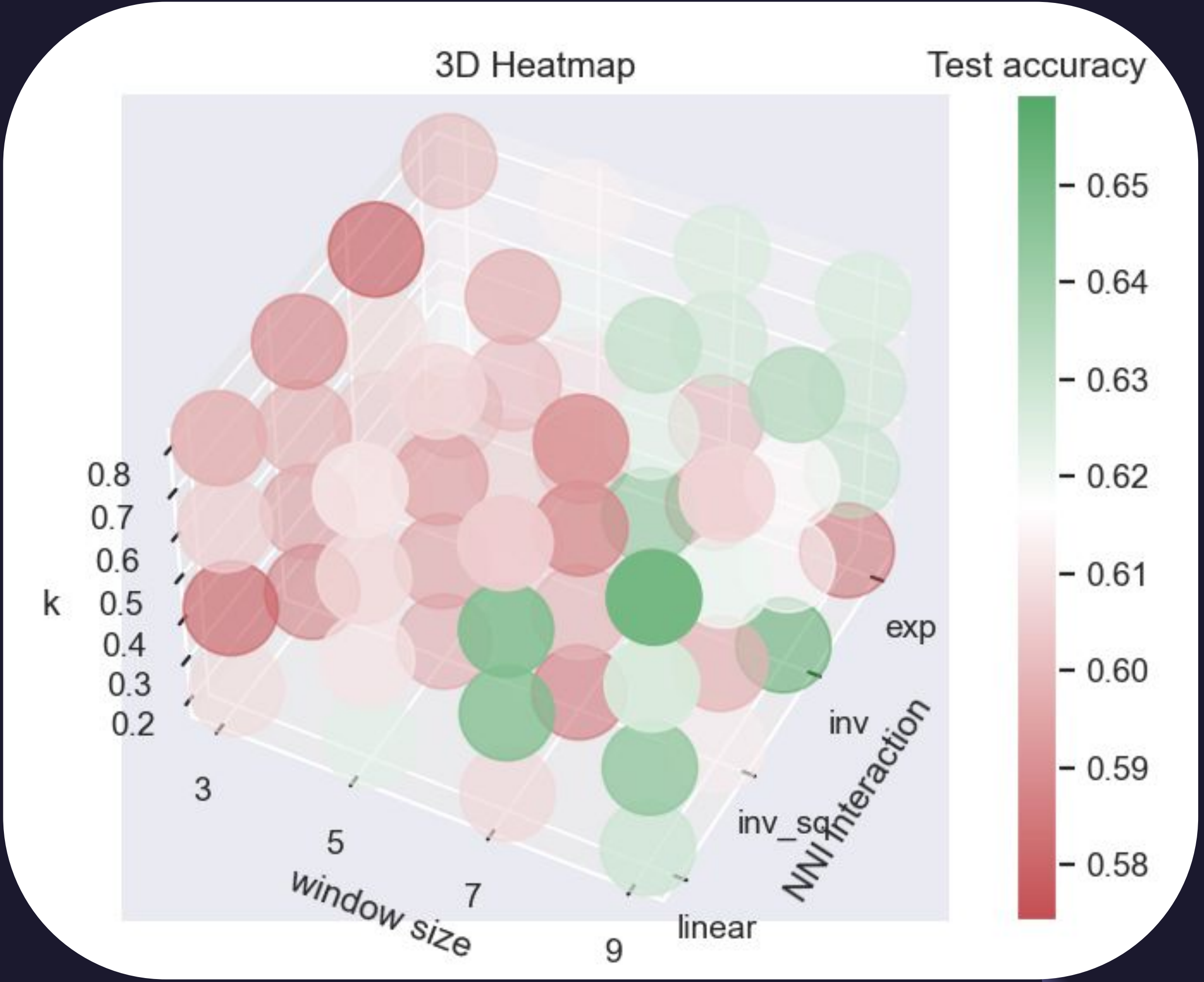
e.g. k=0.5

GridSearch NNI method

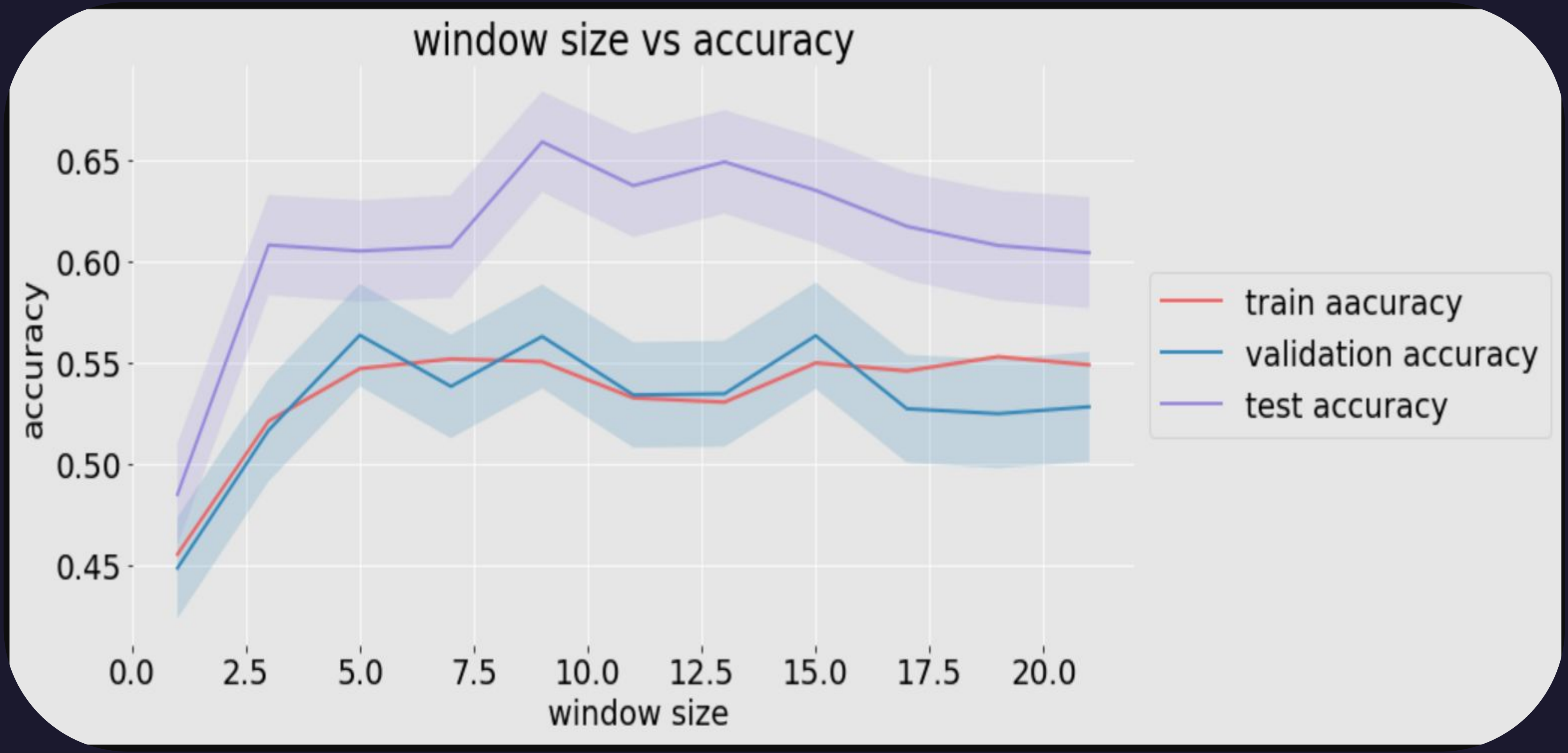


```
100%|██████████ 64/64
[5:31:17<00:00, 310.59s/it]
```

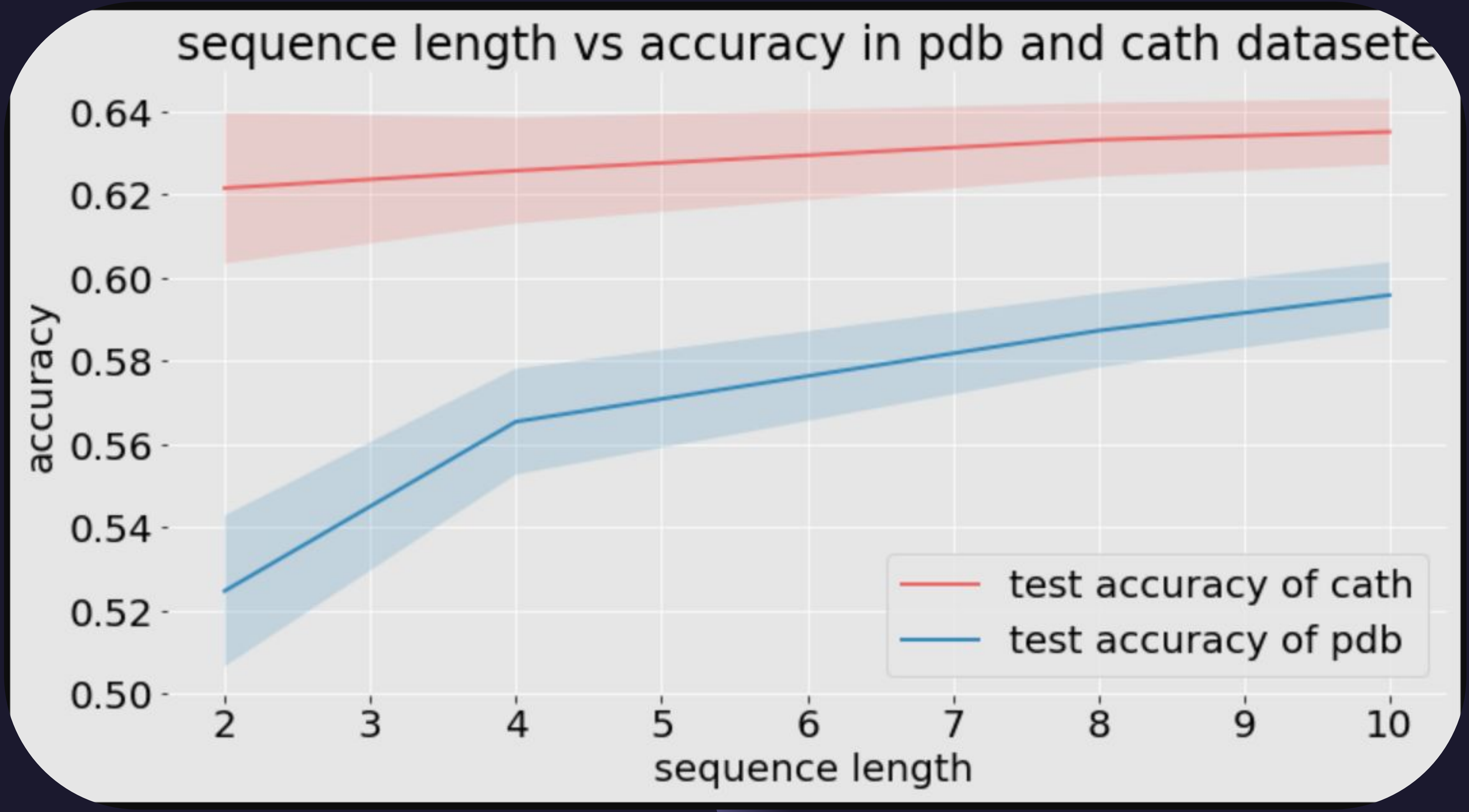




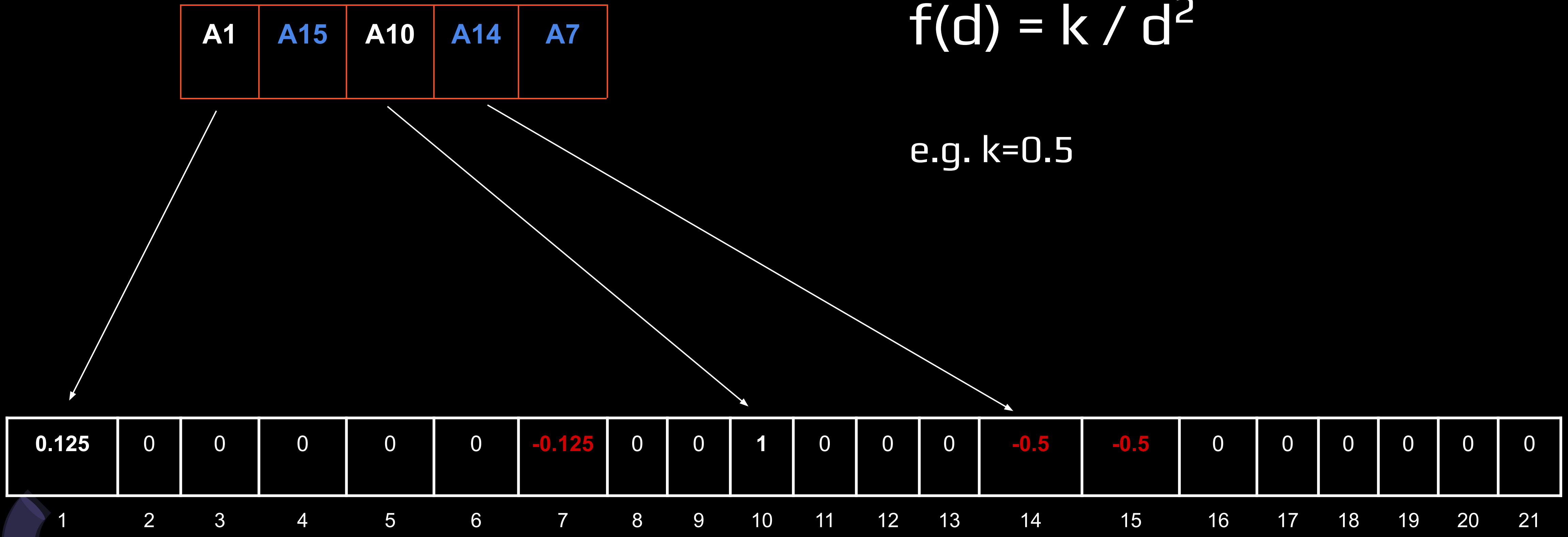
Amino acids windows



Sequence length improving NNI



About hydrophobicity



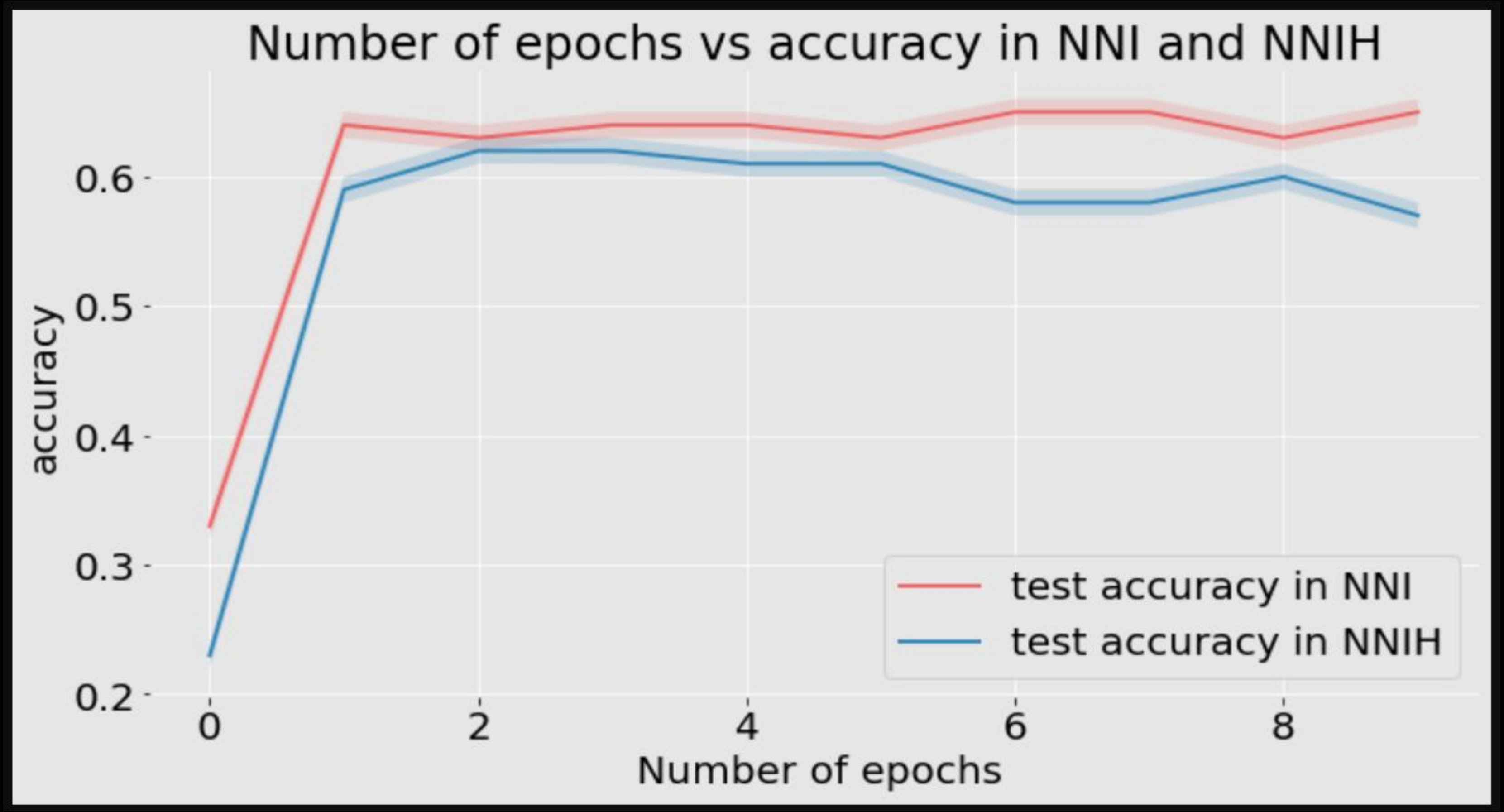
A1	A15	A10	A14	A7

0.125	0	0	0	0	0	-0.125	0	0	1	0	0	0	-0.5	-0.5	0	0	0	0	0	0
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

$$f(d) = k / d^2$$

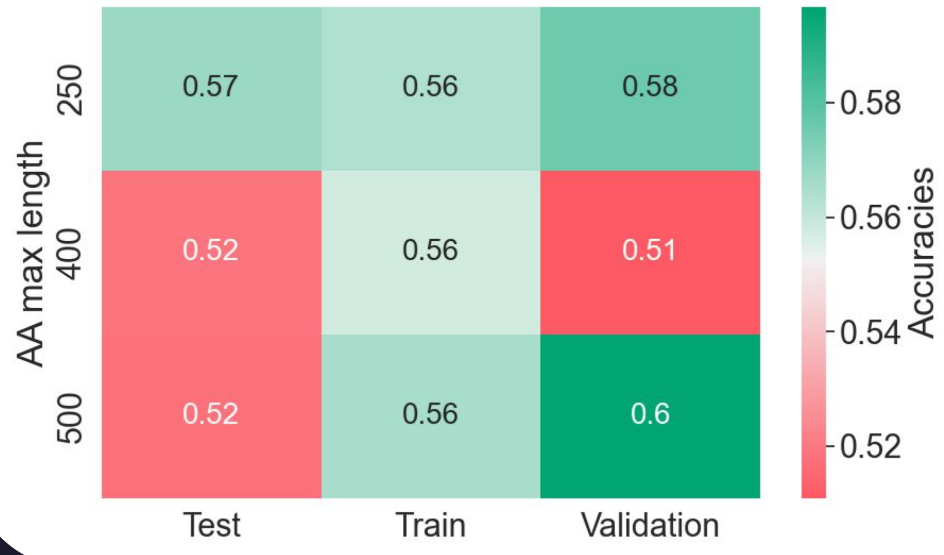
e.g. k=0.5

Results - hydrophobicity

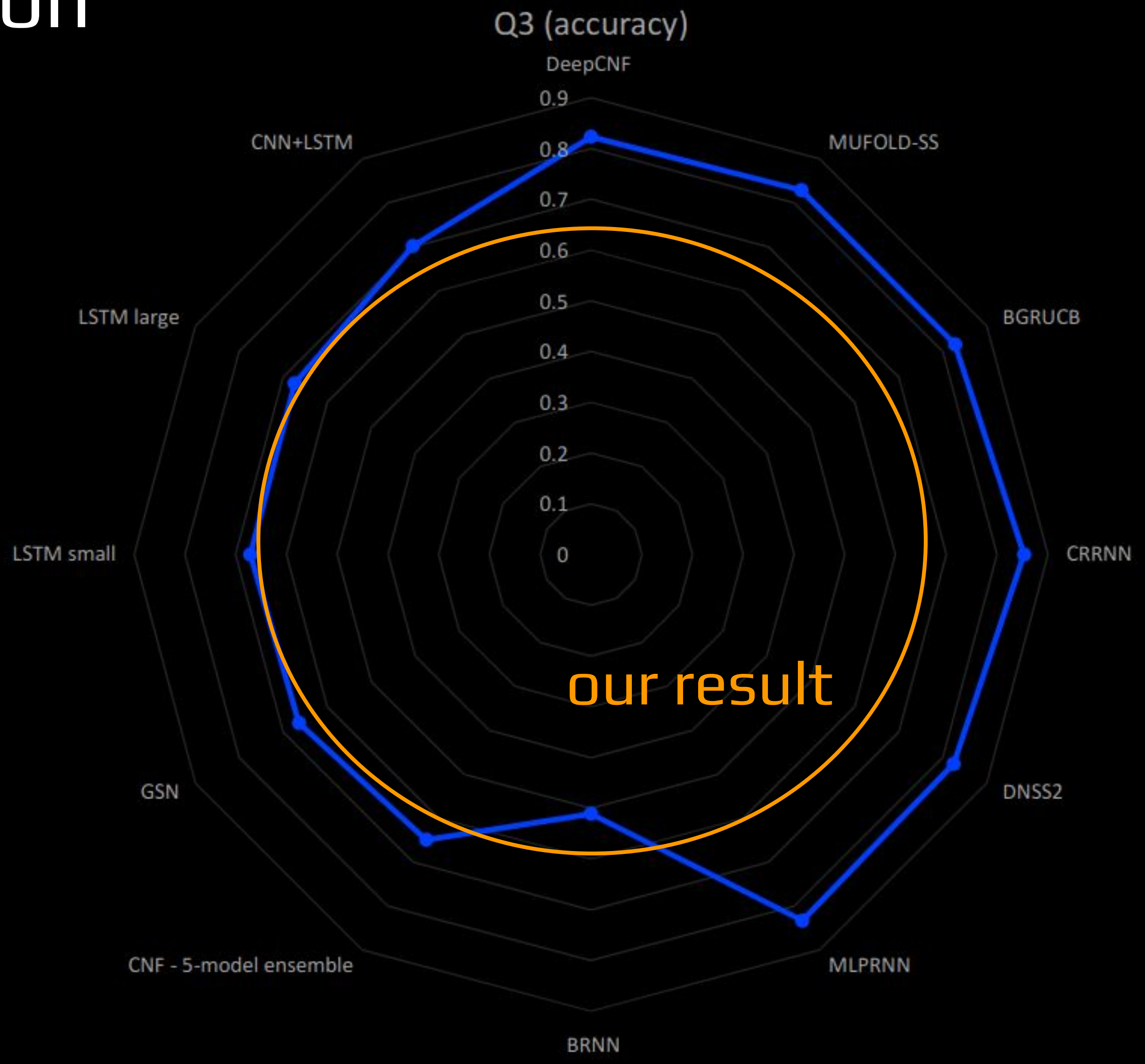


Testing on different datasets

Name of dataset	Number of proteins	sequence lengths range	test accuracy (+/-)
CB513	514	20 - 874	0.56 +/- 0.01
CASP12	21	76 - 1494	0.64 +/- 0.01
PDB	9079	20 - 1632	0.59 +/- 0.01
CATH40	31731	18 - 497	0.64 +/- 0.01



Literature comparison



Comparing with different networks

all the networks are trained on the best parameters found from the previous GridSearch on LTC
dataset: CATH40

Network	training acc (+/- std)	validation acc (+/- std)	test accuracy (+/- std)
LTC	55.56 +/- 0.03	54.73 +/- 0.91	63.31 +/- 0.88
LSTM	59.37 +/- 0.03	51.29 +/- 0.91	61.55 +/- 0.89
NODE	55.97 +/- 0.03	52.74 +/- 0.91	59.86 +/- 0.90
CTGRU	43.54 +/- 0.04	44.60 +/- 0.90	48.37 +/- 0.91
CTRNN	54.89 +/- 0.04	52.28 +/- 0.91	61.45 +/- 0.89

LIMITATIONS

- Computational resources

  NVIDIA GeForce GTX 1650 Ti, 4096MiB

- LNNs fail to learn from features related to physical properties?

  NNIH experiment

MAIN CONCLUSIONS

- Liquid neural network's performance increases when using better input representations

- Larger window size of amino acids and sequence length can increase the performance of the network

- The structure of larger proteins is more difficult to be predicted

FURTHER WORK

- Adding convolutional layers for mapping more representative inputs to the LTC network
- Implementing methods such as PSSM

Thank You!

https://github.com/prodangp/LCP-B