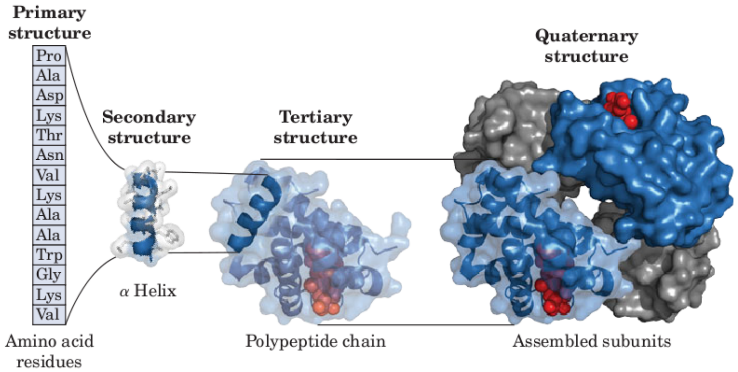
A Deep-Reinforcement Learning Approach to Protein Folding

Conrad Li, Benjamin Beal, Tyler Miller

Proteins: The Building Blocks of Life

- Proteins are macromolecules in organisms that provide a variety of essential roles for life (i.e. enzyme, structure, transport, messengers, antibodies…)
- Proteins are composed of a sequence of the 20 amino acids
- Proteins have structural hierarchy (primary, secondary, tertiary, quaternary)
- A protein's function is highly dependent on its 3D structure



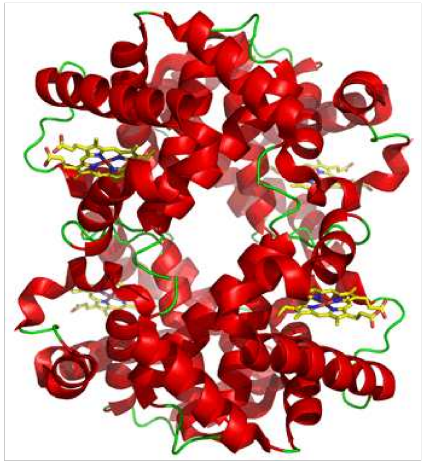
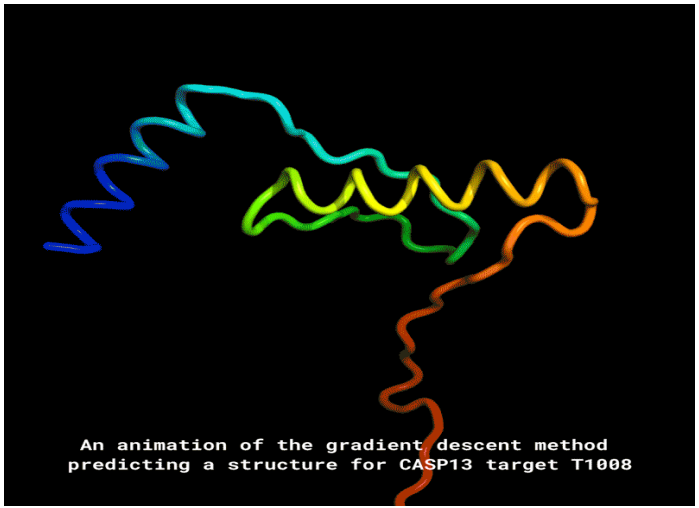


Image Source

The Protein Folding Problem

- Predict the 3D structure of a protein given its amino acid sequence
- One may interested in not only the final 3D structure, but also gaining insights into the process of folding itself
- Can be used to speed up drug discovery and discovering new proteins



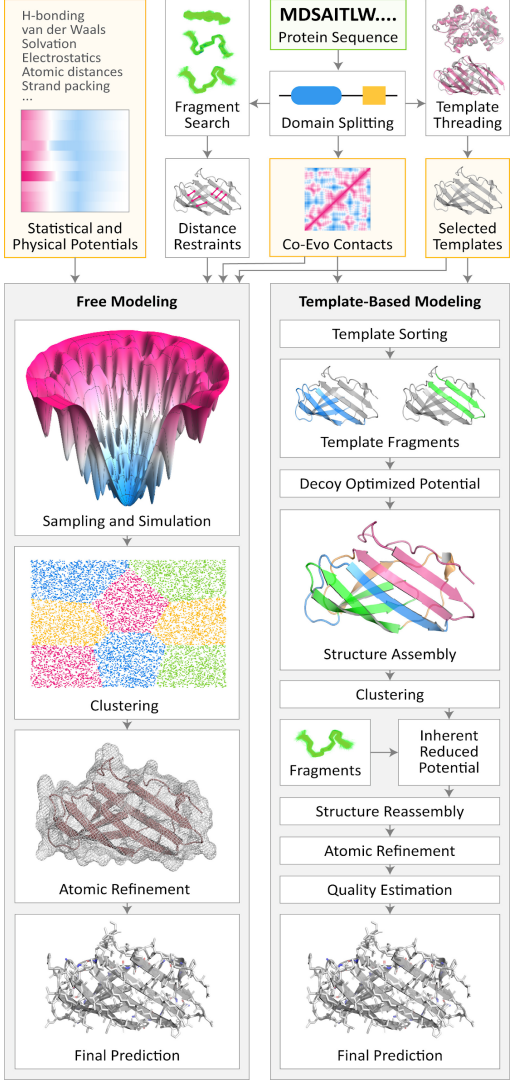
Conventional Methods

AlQuraishi, 2019

- Computational protein structure prediction methods can be separated into two categories
- Methods in the first category build explicit sequences to structure maps
  - Molecular dynamics simulations from first principles which are computationally costly)
  - Template-based methods which rely on similarities to previously discovered structures
- Methods in the second category use co-evolution techniques (necessitating a large amount of homologs) to predict distance and/or contact maps

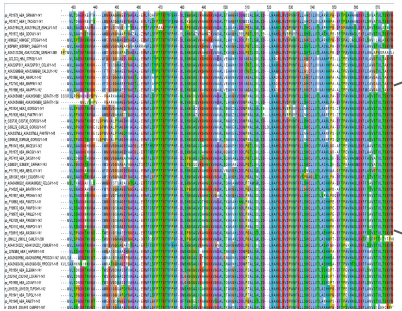
Conventional Pipelines for Protein Structure Prediction

(AlQuraishi, 2019)



RaptorX-Contact

MSA



Sequential Features

$L \times 26$

Pairwise Features

$L \times L \times 3$

Loss

Negative Log-Likelihood of Native Contacts

RaptorX

CNS

$$\begin{bmatrix} 0 & d_{12} & d_{13} & \dots & d_{1L} \\ d_{21} & 0 & \ddots & \ddots & \vdots \\ d_{31} & \ddots & 0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 & \vdots \\ d_{L1} & \dots & \dots & \dots & 0 \end{bmatrix}$$

L = Sequence Length

$$R_\theta : \mathbb{R}^{L \times 26}, \mathbb{R}^{L \times L} \rightarrow \mathbb{R}^{3 \times L}$$

$$\begin{bmatrix} \overbrace{x_1\ x_2\ x_3}^{\text{Residue 1}}\ \overbrace{x_4\ x_5\ x_6}^{\text{Residue 2}}\ \cdots \\ y_1\ y_2\ y_3\ y_4\ y_5\ y_6\ \cdots \\ z_1\ z_2\ z_3\ z_4\ z_5\ z_6\ \cdots \end{bmatrix}$$

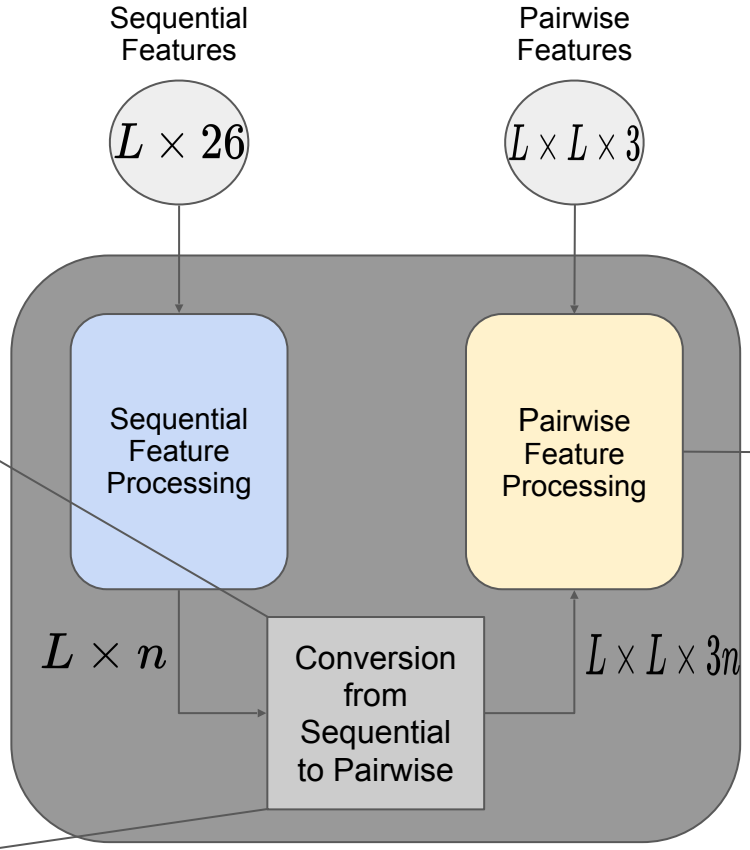
Figure based off Jinbo Xu's presentation J. Xu, 2019

RaptorX-Contact

$$X_\theta : \mathbb{R}^{L\times 26}, \mathbb{R}^{L\times L} \rightarrow \mathbb{R}^{L\times L}$$

v_i = Sequential Output Feature for the ith residue

$$w_{ij} = \begin{bmatrix} v_i \\ v_{\frac{i+j}{2}} \\ v_j \end{bmatrix}$$



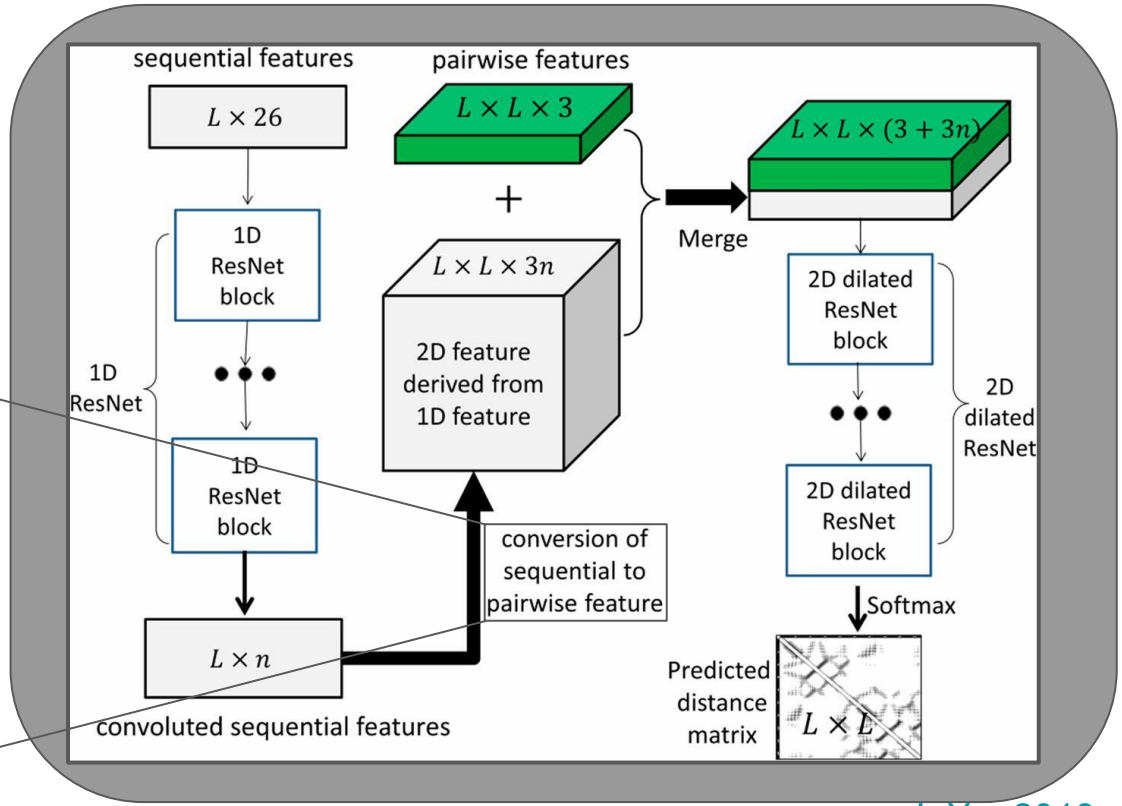
$$\begin{bmatrix} 0 & d_{12} & d_{13} & \dots & d_{1L} \\ d_{21} & 0 & \ddots & \ddots & \vdots \\ d_{31} & \ddots & 0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 & \vdots \\ d_{L1} & \dots & \dots & \dots & 0 \end{bmatrix}$$

A more detailed examination (See here for ResNets)

$$X_\theta : \mathbb{R}^{L\times 26}, \mathbb{R}^{L\times L} \rightarrow \mathbb{R}^{L\times L}$$

v_i = Sequential Output Feature for the ith residue

$$w_{ij} = \begin{bmatrix} v_i \\ v_{\frac{i+j}{2}} \\ v_j \end{bmatrix}$$



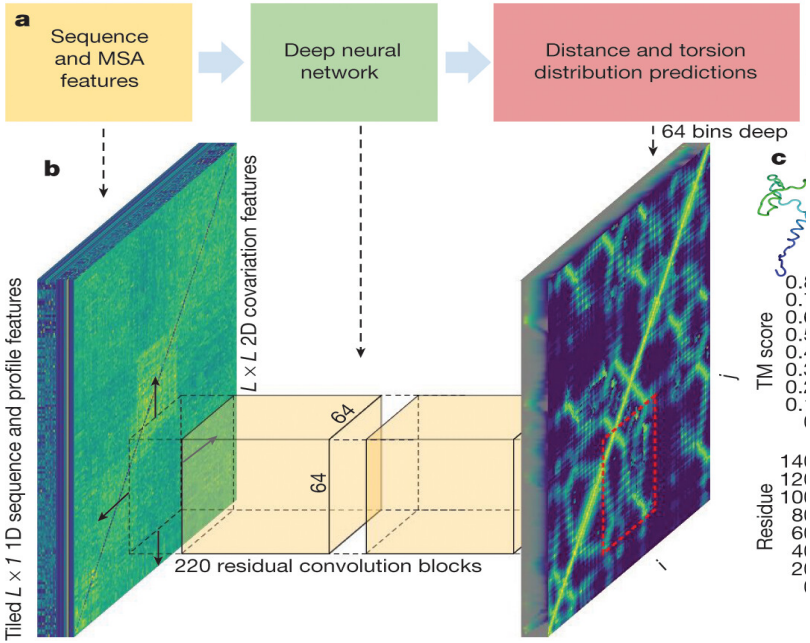
J. Xu, 2019

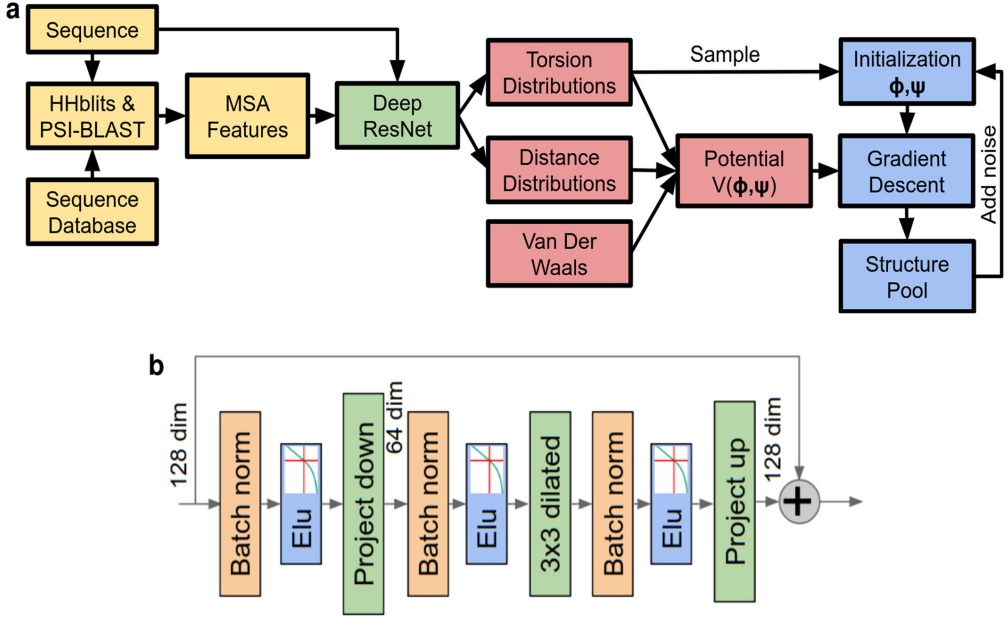
RaptorX-Contact Observations

- Outperforms existing DCA and template based approaches
- Does not directly yield 3D coordinates/model of protein - but can still construct 3D models without involving extensive conformational sampling using CNS
- Model quality directly correlated with MSA depth, logarithm of *Meff* ($R \approx 0.6$)
- Heavy reliance on co-evolution data prevents it from exploring the full protein design space
- Assumed probability of having a distance >16 Å is 0 which leads to errors

J. Xu, 2019

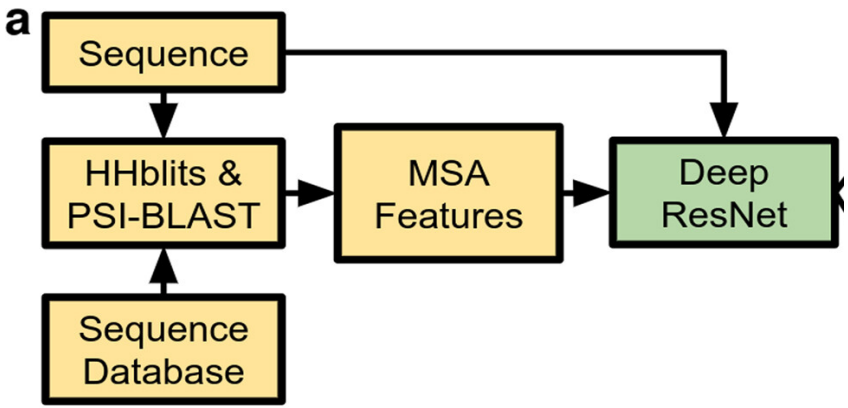
AlphaFold





Senior *et al.*, 2020

Input Representation

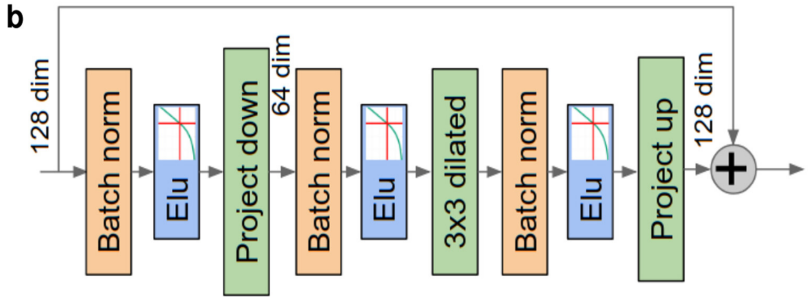


- HHblits alignments (L x 1)
- One hot AA encoding (L x 21)
- PSI-Blast profile (L x 21)
- HHblits profile (L x 22)
- Non-gapped profile (L x 21)
- HMM profile (L x 30)
- Potts model bias (L x 22)
- Deletion probability (L x 1)
- Residue indices (L x 1)
- Potts model parameters (L x L x 484)
- Norm of potts parameters (L x L)
- Gap matrix (L x L)

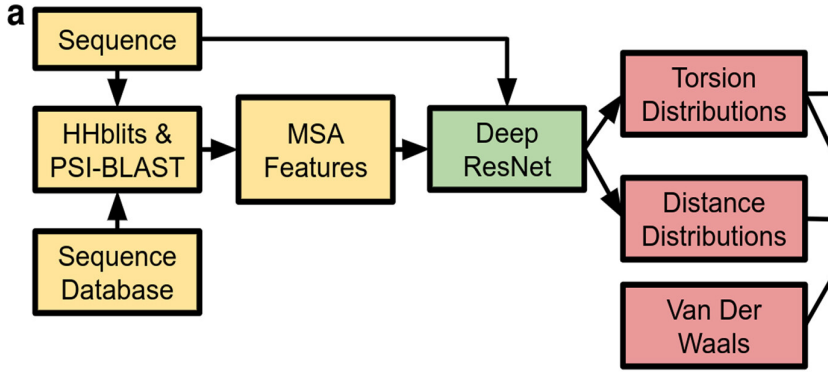
Senior *et al.*, 2020

Computation/Function

- The network takes the concatenated MSA features and sequence encoding as inputs
- Deep ResNet consists of 220 blocks such as the one shown below



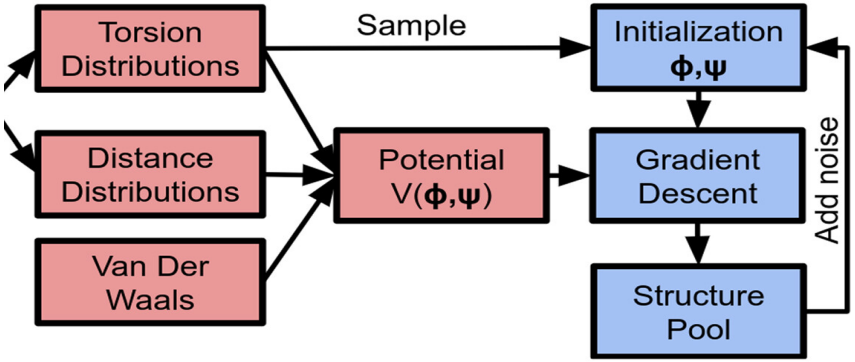
- The network learns a probability distribution of distances between residue pairs (L x L x 64)
- Trained/tested on 64 x 64 regions of entire L x L distogram



Senior *et al.*, 2020

Outputs and Learning

- The torsion and distance distributions, along with a Van Der Waals term, were used to construct a distance potential (shown below)



Distance potentials The basic distance potential is computed as a sum over all residue pairs of the likelihood of the inter-residue distances:

$$V_{\text{distance}}(\mathbf{x}) = -\sum_{i,j,\ i\neq j} \log P(d_{ij} \mid \mathcal{S}, \text{MSA}(\mathcal{S})). \quad (1)$$

The distance potential with a reference state becomes:

$$V_{\text{distance}}(\mathbf{x}) = -\sum_{i,j,\ i\neq j} \log P(d_{ij} \mid \mathcal{S}, \text{MSA}(\mathcal{S})) - \log P(d_{ij} \mid \text{length}, \delta_{\alpha\beta}). \quad (2)$$

The torsions are modelled with a von Mises distribution for each residue:

$$V_{\text{torsion}}(\boldsymbol{\phi}, \boldsymbol{\psi}) = -\sum_{i} \log p_{\text{vonMises}}(\phi_i, \psi_i \mid \mathcal{S}, \text{MSA}(\mathcal{S})). \quad (3)$$

The total potential that we optimise is thus:

$$V_{\text{total}}(\boldsymbol{\phi}, \boldsymbol{\psi}) = V_{\text{distance}}(G(\boldsymbol{\phi}, \boldsymbol{\psi})) + V_{\text{torsion}}(\boldsymbol{\phi}, \boldsymbol{\psi}) + V_{\texttt{score2\_smooth}}(G(\boldsymbol{\phi}, \boldsymbol{\psi})). \quad (4)$$

The terms are weighted equally as determined by cross-validation.

- Stochastic Gradient Descent is then used to find a set of torsion angles that minimizes this potential (shown above)

Senior *et al.*, 2020

AlphaFold (Pros and Cons)

Pros:

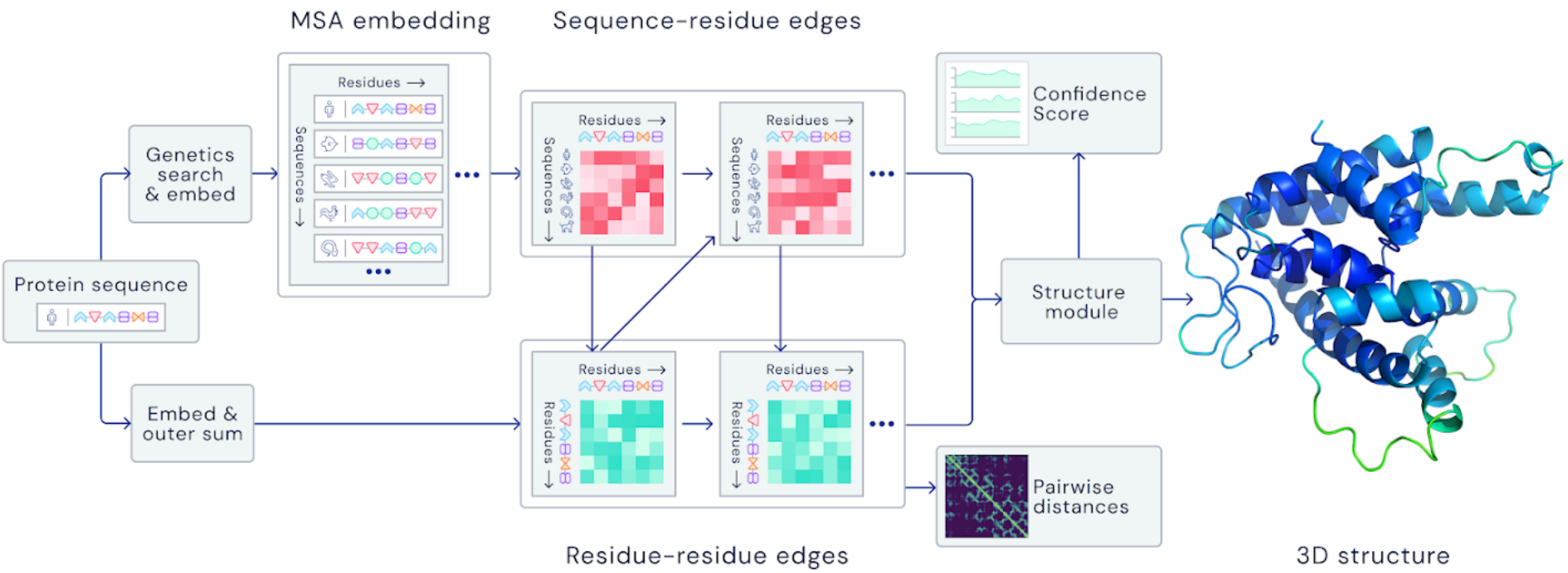
- Achieves high accuracy even with few sequence homologs
- Unprecedented FM accuracy
- Matches performance of template-based approaches without using a template

Cons:

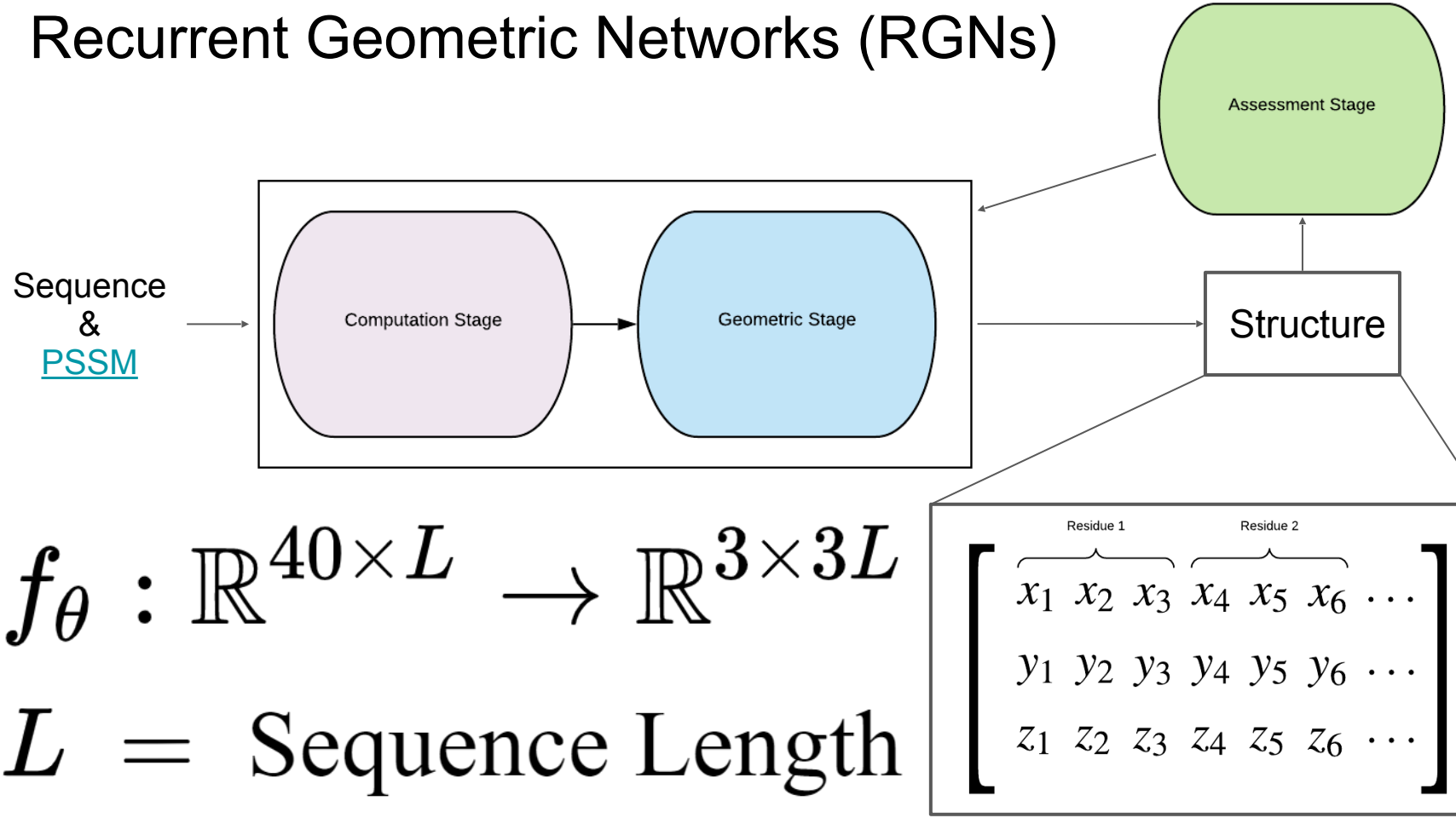
- Relies on a complex structure prediction pipeline that must be carefully designed
- Deep learning is applied to only a portion of the pipeline (e.g. predicting the distances) so the entire model is not strictly differentiable and relies on careful feature representation

Senior *et al.*, 2020

AlphaFold 2.0



Recurrent Geometric Networks (RGNs)

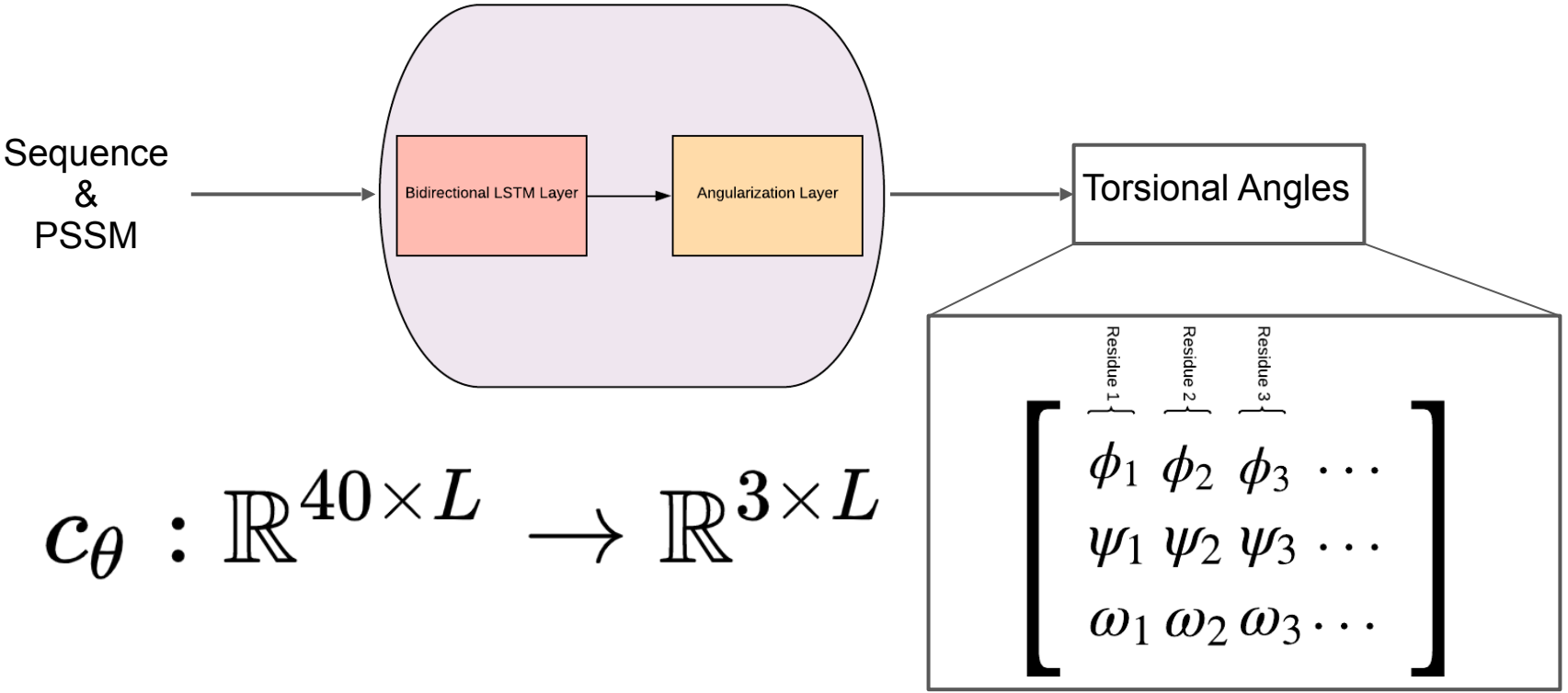


$$f_\theta : \mathbb{R}^{40 \times L} \rightarrow \mathbb{R}^{3 \times 3L}$$

$$L = \text{Sequence Length}$$

AlQuraishi, 2019

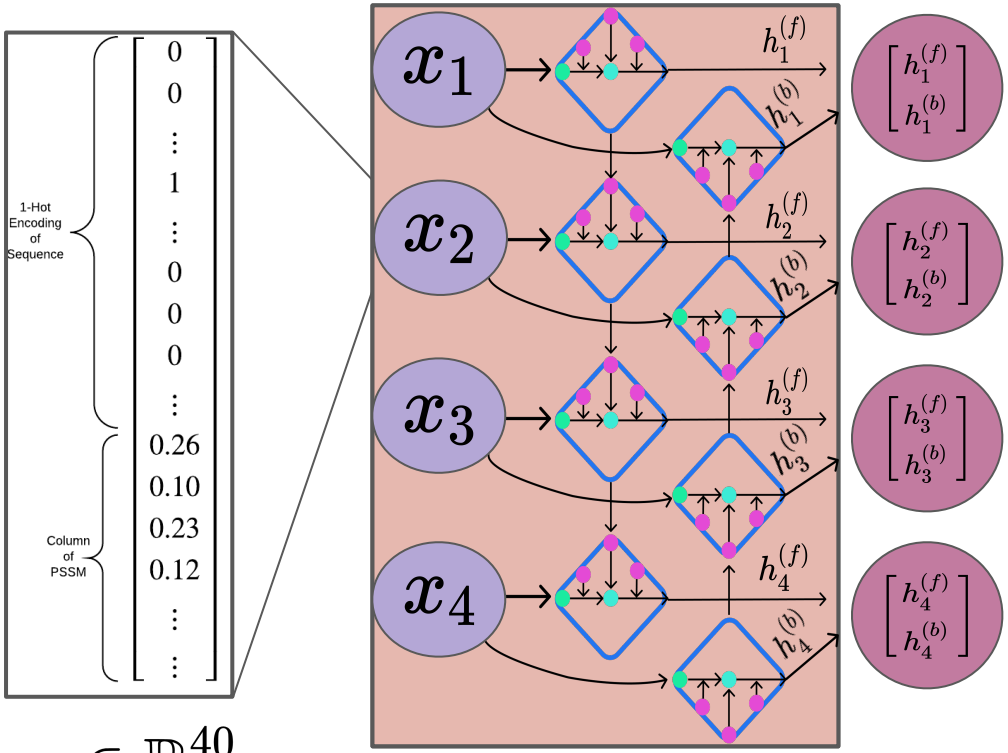
Computation Stage



$$c_\theta : \mathbb{R}^{40 \times L} \rightarrow \mathbb{R}^{3 \times L}$$

AlQuraishi, 2019

Bidirectional LSTM Layer



$$x_i \in \mathbb{R}^{40}$$

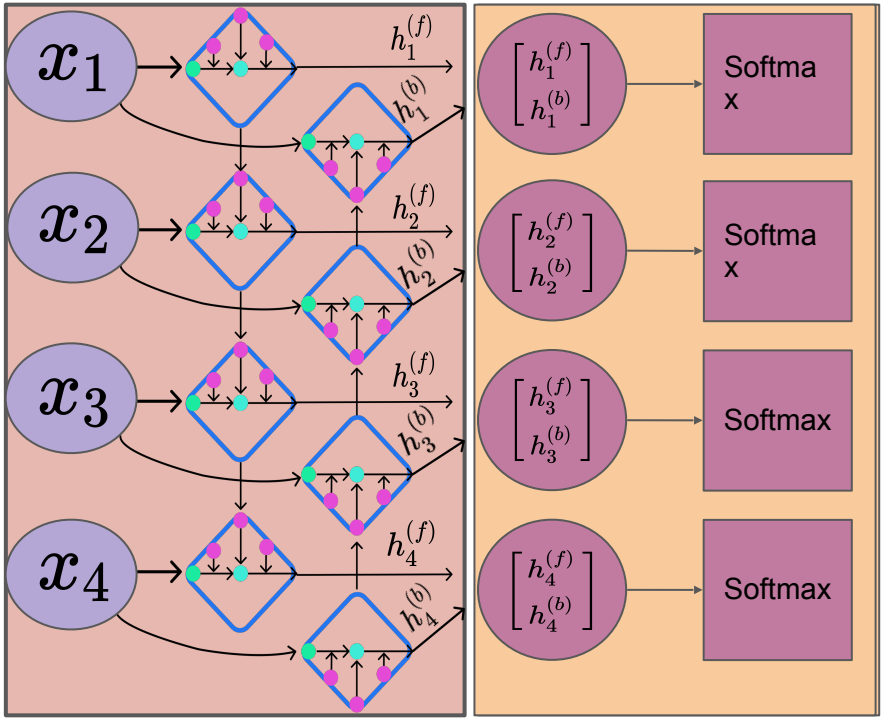
$$x_i = \text{ith residue}$$

$$h_i^{(f)}, h_i^{(b)} \in \mathbb{R}^{20} \quad \begin{bmatrix} h_i^{(f)} \\ h_i^{(b)} \end{bmatrix} \in \mathbb{R}^{40}$$

$$b_\theta^{(i)} : \mathbb{R}^{40} \rightarrow \mathbb{R}^{40}$$

AlQuraishi, 2019

Angularization Layer: Torsion Alphabet Probability Weight Generation



$$p_i = softmax(W_\alpha[h_i^{(f)}, h_i^{(b)}] + b_\alpha)$$

$$p_i \in \mathbb{R}^M \quad \Phi \in \mathbb{R}^{M \times 3}$$

Φ = Alphabet of sets of torsion angles

M = Number of "letters" (sets of torsion angles) in Φ

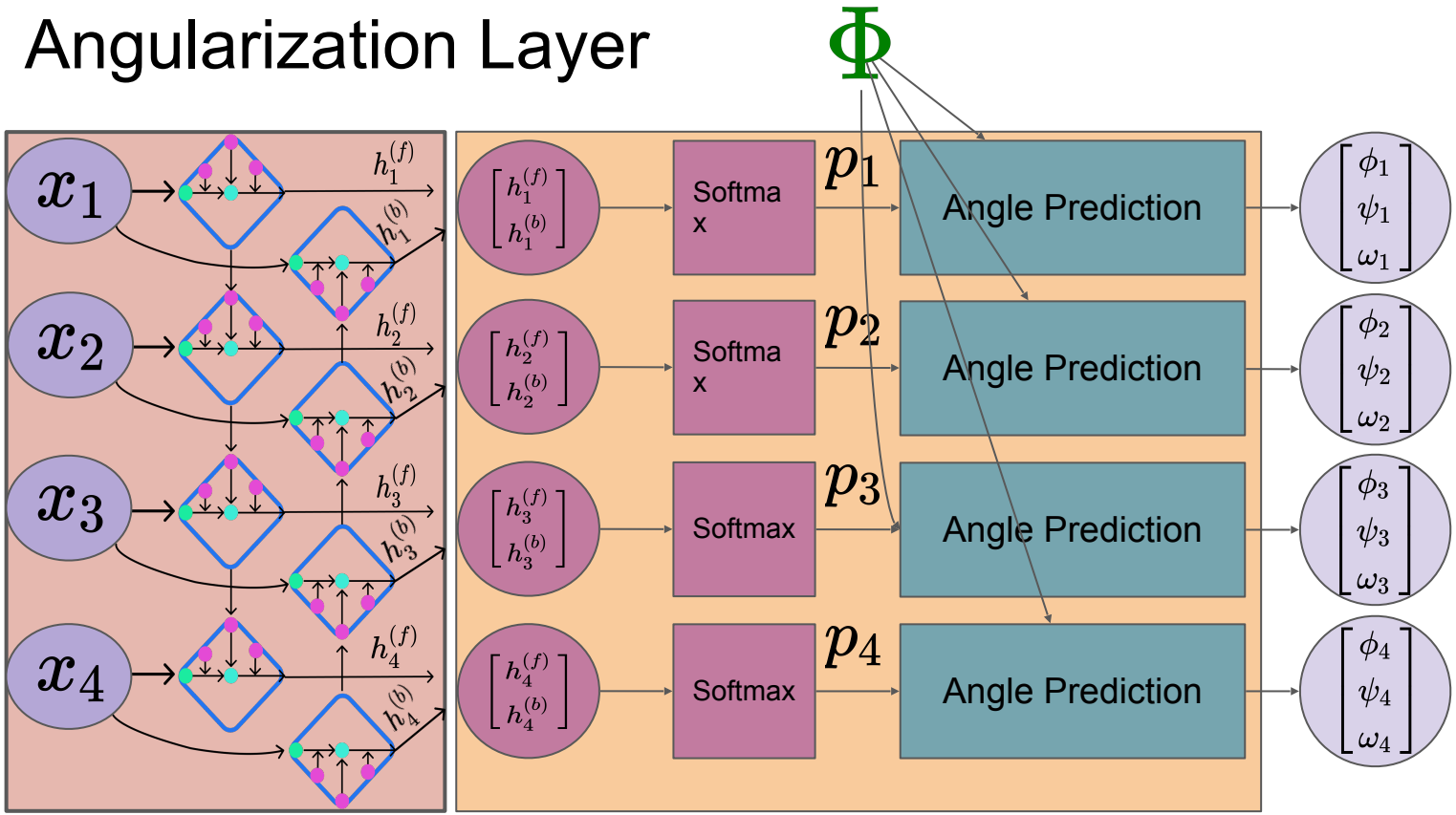
$$\begin{bmatrix} \phi_1 & \psi_1 & \omega_1 \\ \phi_2 & \ddots & \vdots \\ \phi_3 & \ddots & \vdots \\ \vdots & \cdots & \cdots \end{bmatrix}$$

*hyperparameter

*learned

AlQuraishi, 2019

Angularization Layer

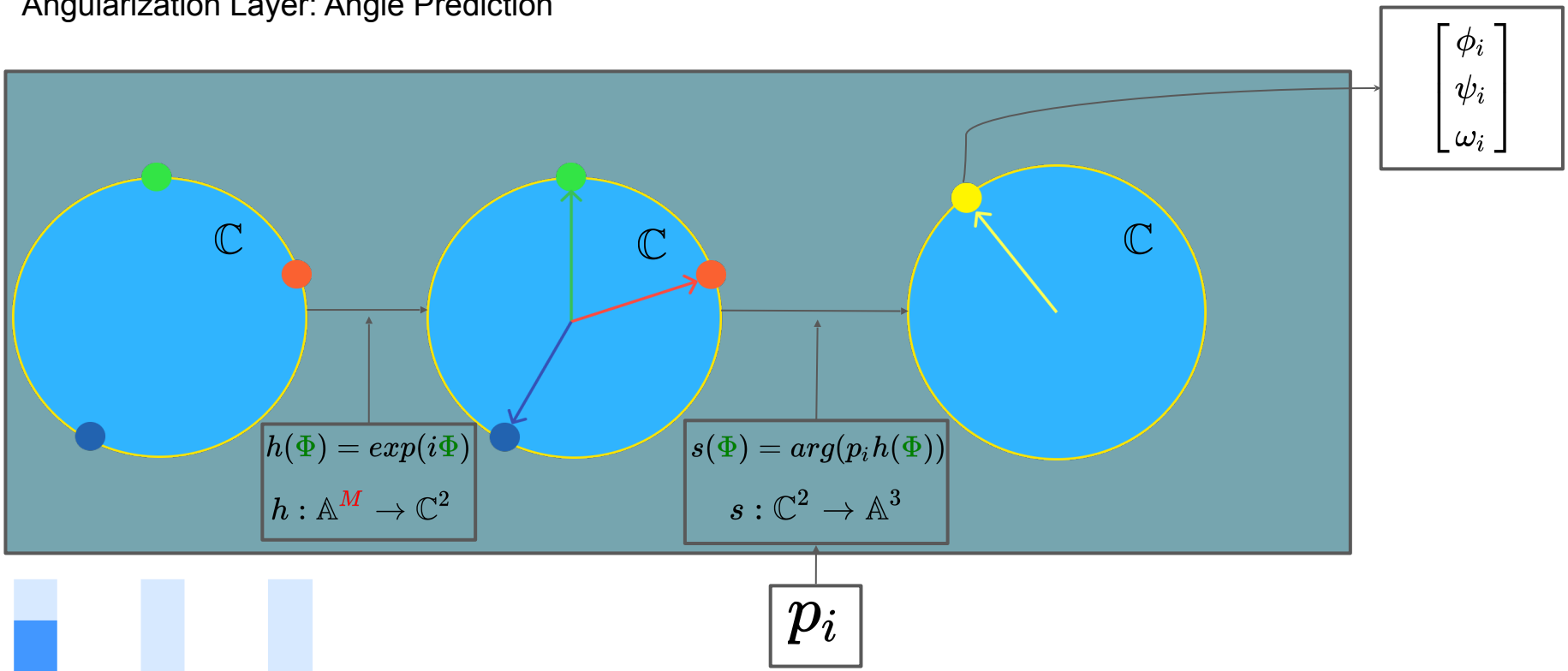


$$a_\theta^{(i)} : \mathbb{R}^{40} \rightarrow \mathbb{A}^3$$

*learned

AlQuraishi, 2019

Angularization Layer: Angle Prediction

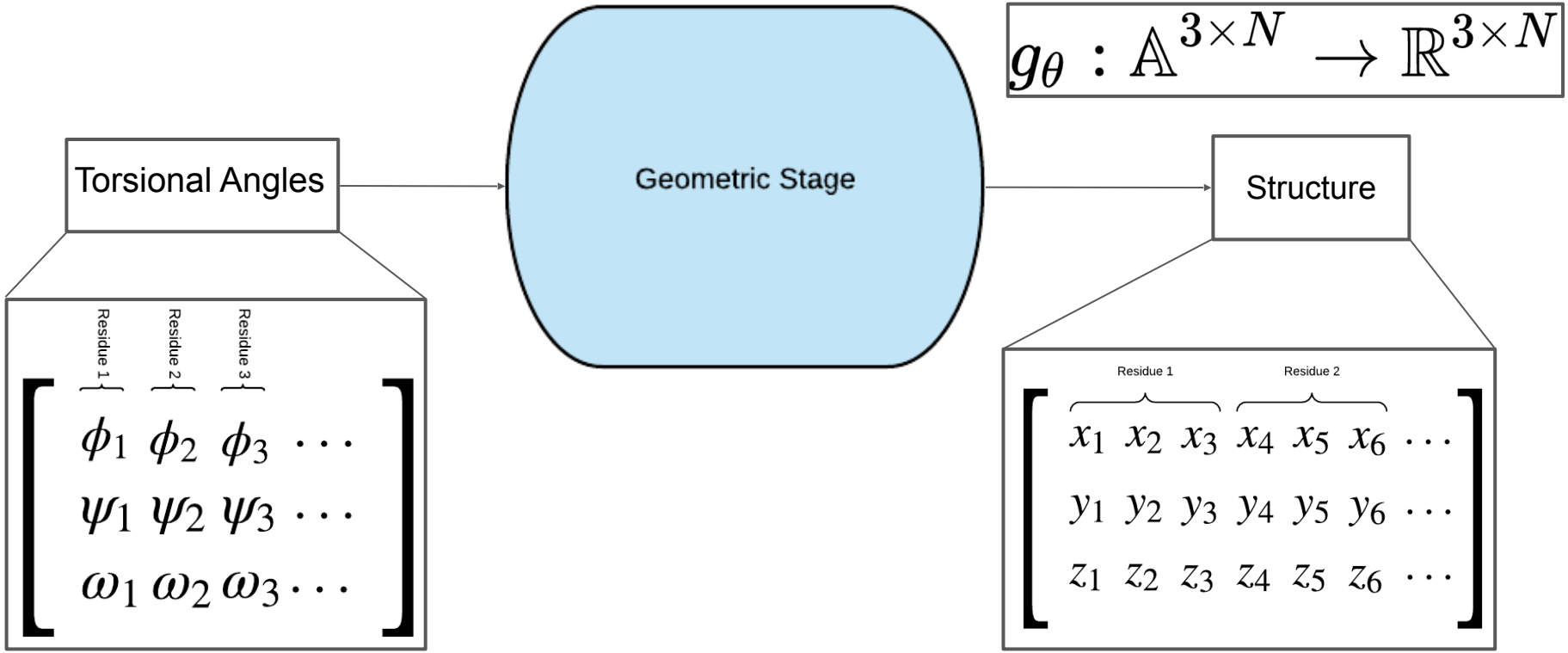


Note that for the sake of visual clarity, only 1 Cartesian vector is shown for each “letter”. In reality, there would be 3 Cartesian vectors for each letter (one for each torsional angle)

*hyperparameter
*learned

AlQuraishi, 2019

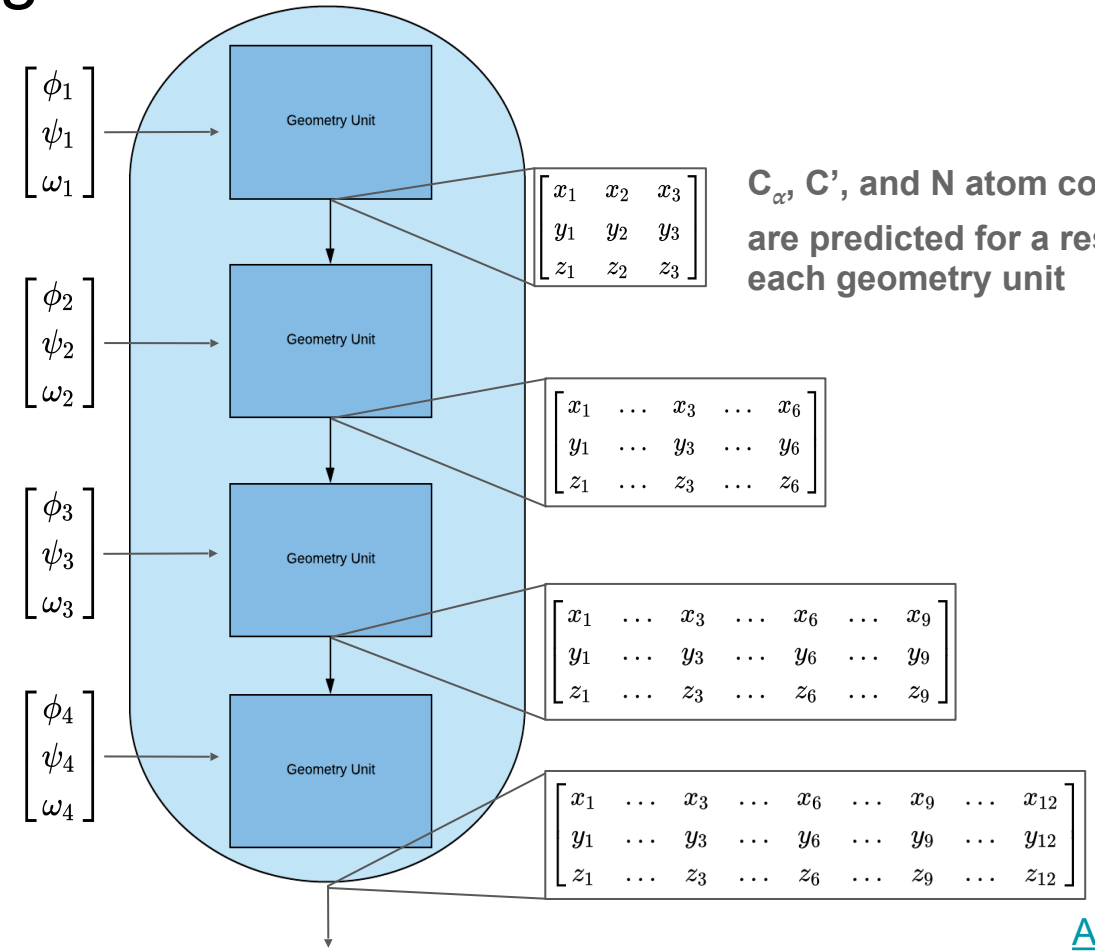
Geometric Stage



AlQuraishi, 2019

Geometric Stage

No Learning Occurs Here



C_α, C', and N atom coordinates are predicted for a residue at each geometry unit

AlQuraishi, 2019

Geometric Unit

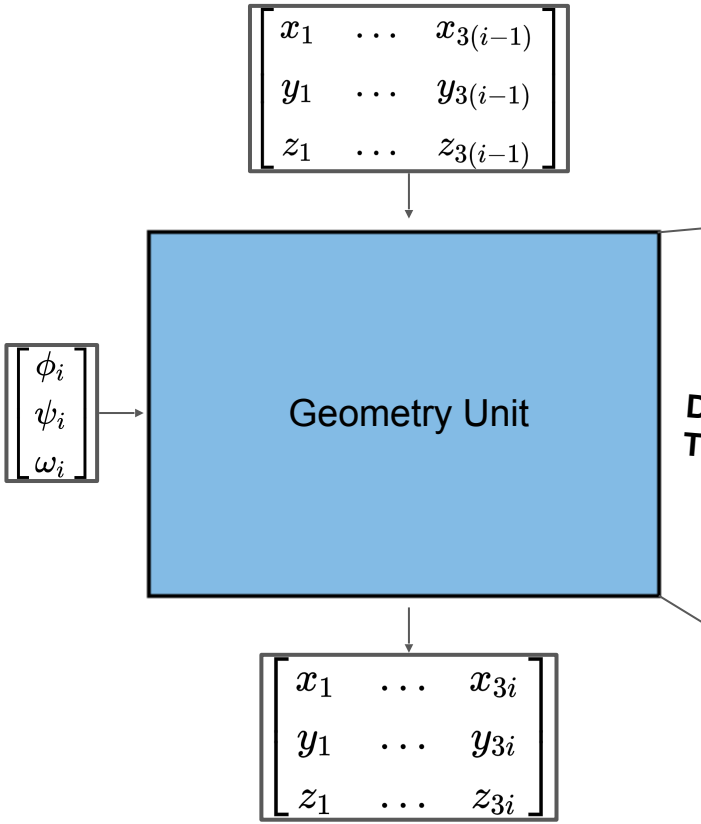
*hyperparameter

$[r_0, r_1, r_2]$: **fixed** bond lengths

$[\theta_1, \theta_2, \theta_3]$: **fixed** bond angles

$[\psi_1, \psi_2, \psi_3]$: **predicted** dihedral angles

c_k: 3D coordinates of the ***k***th atom in the protein



Denavit-Hartenberg Transformation

$$\tilde{c}_k = r_{kmod3} \begin{bmatrix} cos\left(\theta_{k \bmod 3}\right) \\ cos\left(\varphi_{k/3,k \bmod 3}\right) sin\left(\theta_{k \bmod 3}\right) \\ sin\left(\varphi_{k/3,k \bmod 3}\right) sin\left(\theta_{k \bmod 3}\right) \end{bmatrix}$$

$$m_k = c_{k-1} - c_{k-2}$$

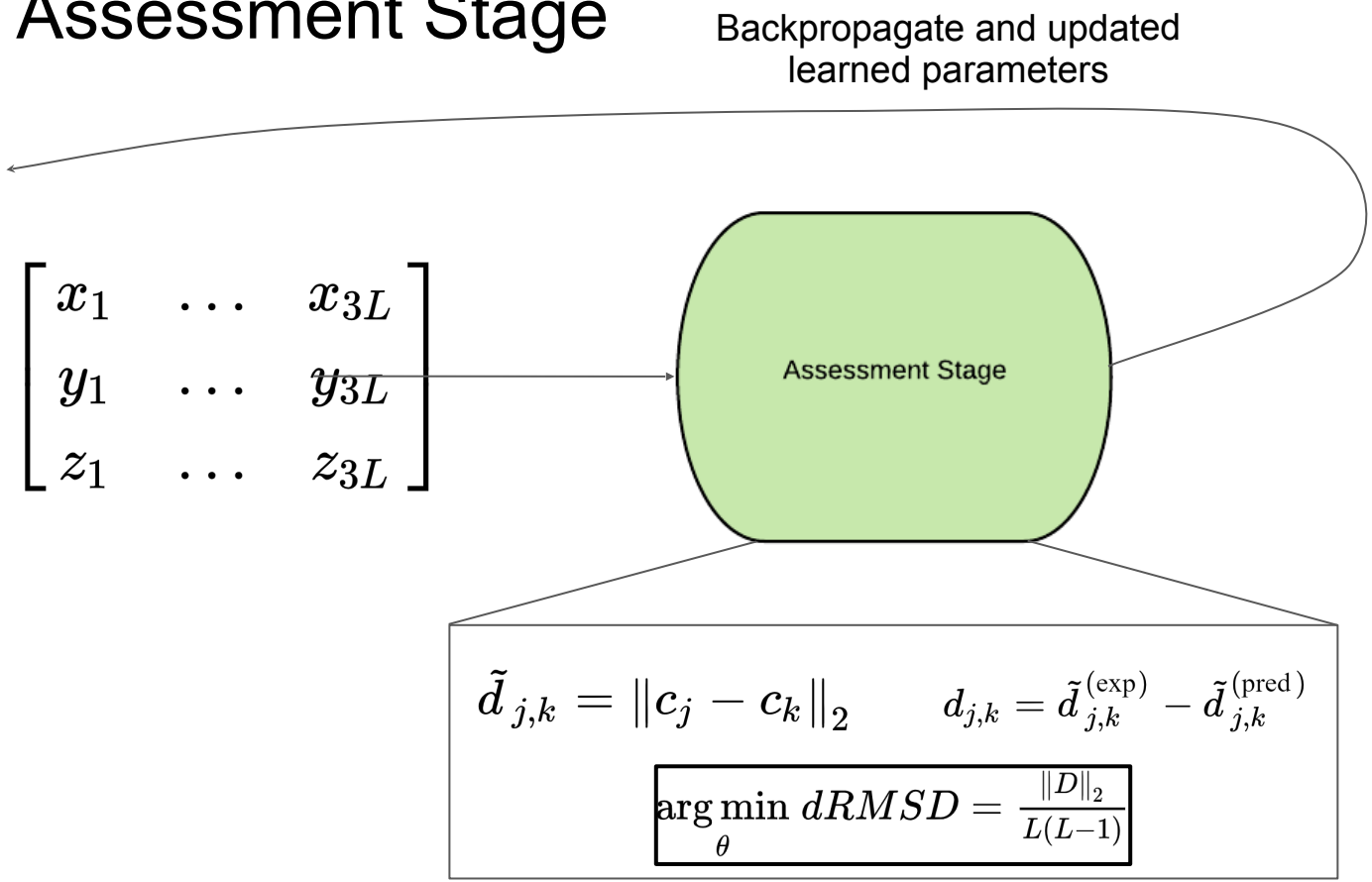
$$n_k = m_{k-1} \times \widehat{m_k}$$

$$M_k = [\widehat{m_k}, \widehat{n_k} \times \widehat{m_k}, \widehat{n_k}]$$

$$c_k = M_k \tilde{c}_k + c_{k-1}$$

AlQuraishi, 2019

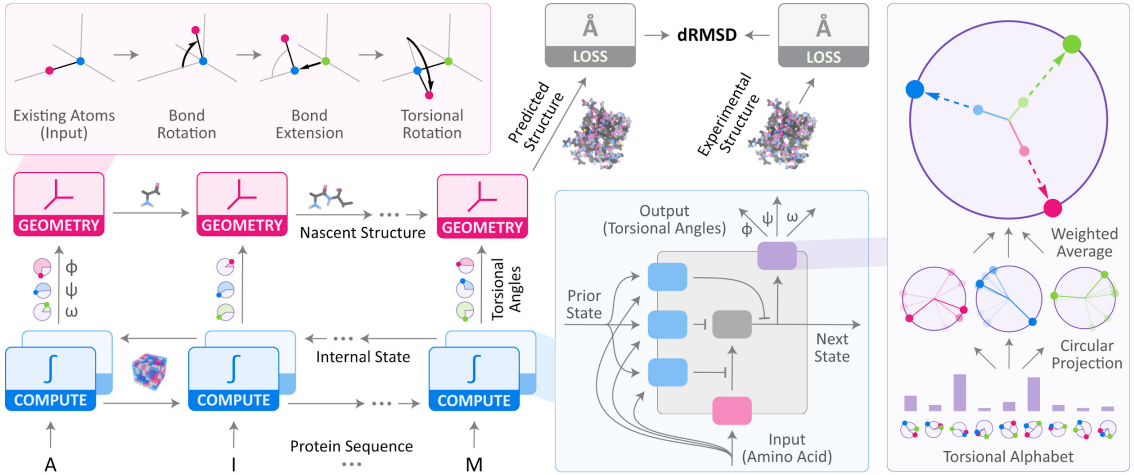
Assessment Stage



AlQuraishi, 2019

RGN Full Architecture

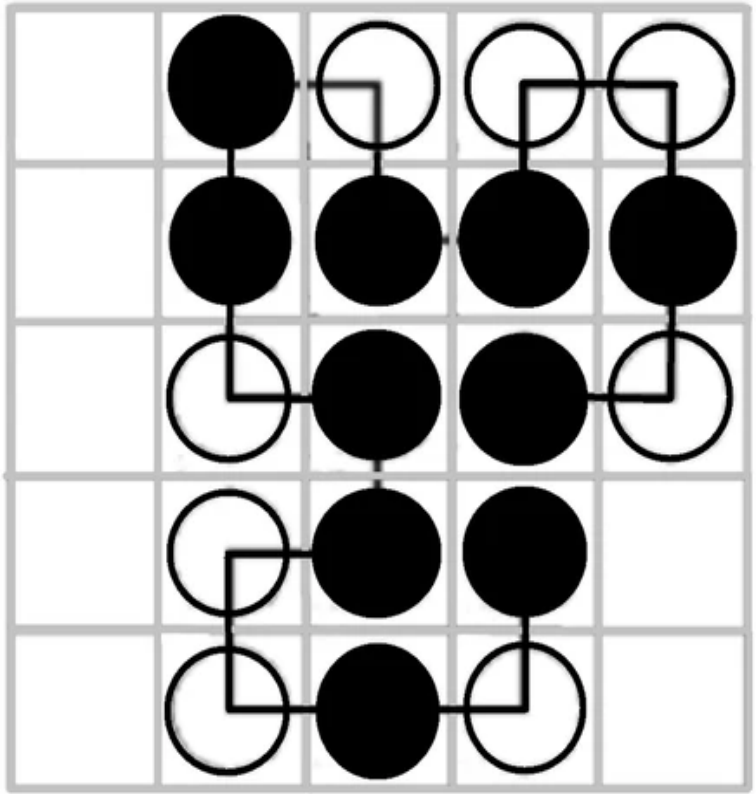
- 2 Bidirectional LSTM layers with 800 Compute Units per direction
- 0.5 input dropout rate
- Torsional Alphabet Size of 60
- LSTM and angularization layers both initialized with a uniform distribution
- ADAM optimizer with learning rate 0.001, β_1 = 095, β_2 = 0.99, and batch size of 32
- Gradient clipping with threshold of 5.0



AlQuraishi, 2019

DRL for 2D HP model

- Differs from the other methods in that the protein folding problem is modeled as a Deep Reinforcement Learning problem
- Uses 2D HP model, which treats every residue as either hydrophobic (H) or hydrophilic (P) and maps protein residues onto a 2D lattice
- Algorithm places residues one by one in lattice following a self-avoiding walk



Jafari, R., Javidi, M.M., 2020

DRL for 2D HP model cont.

- Goal is to minimize the free energy (the reward)
- Since proteins tend to form a hydrophobic core, a -1 reward (lowering the free energy) is given for each pair of non-consecutive H-residue pairs that end up as neighbors in the lattice
- Uses an actor-critic algorithm to learn the best policy
- Network uses three LSTM layers (each followed by a dropout layer) to approximate the Q-value function using a dueling architecture (the critic)
- Trained with an experience replay mechanism
- Goal is to place the current residue adjacent to the previous residue where the action space is {Up, Down, Left, Right}

Jafari, R., Javidi, M.M., 2020

DRL for 2D HP model

Pros:

- Novel use of DRL in protein folding
- Does not require labeled data unlike supervised learning
- Outperforms similar methods in less time

Cons:

- Reward function is very simplistic
- Only in two dimensions using a simplified lattice model
- Hydrophobicity has been binarized

CNR = Could not reach

Seq. #	ACO	Folding Zero	GA	EMC	ENLS	PERM	DRL
1	-9	-9	-9	-9	-9	-9	-9
2	-9	-8	-9	-9	-9	-9	-9
3	-8	-7	-8	-8	-8	-8	-8
4	-14	-13	-14	-14	-14	-14	-14
5	-23	-18	-23	-23	-23	-23	-23
6	-21	-18	-21	-21	-21	-21	-21
7	-36	-32	-34	-35	-36	-36	-36
8	-42	N/A	-37	-39	-39	-38	-42
9	-53	-48	CNR	-52	CNR	-53	-53
10	-50	N/A	CNR	CNR	CNR	-50	-50
11	-47	N/A	CNR	CNR	CNR	-48	-48

Jafari, R., Javidi, M.M., 2020

Comparison of Current SOA Methods

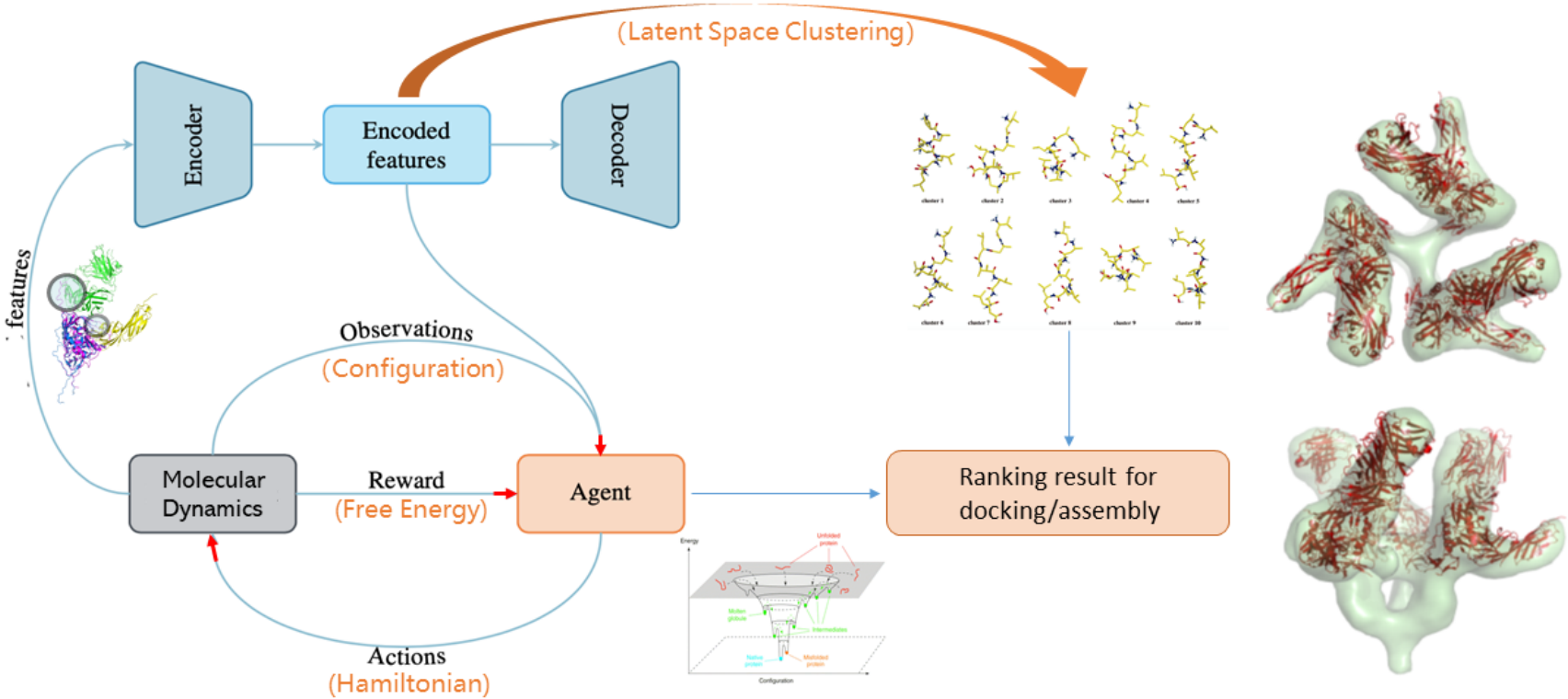
	RaptorX	AlphaFold	RGNs
Inputs	PSSM, MI, Predicted Secondary Structures, Predicted Solvent Accessibility, Pairwise Potential, CCMPred Coevolutionary Info	PSSMs (each from different software) and Potts model parameters	PSSM
Outputs	Distance Matrix	L x L x 64 distance distributions	Cartesian Coordinates
3D Model Determination	CNS	L-BFGS	Implicit
Energy Function	None	Constructed their own mean force potential	None
End-to-End Differentiable	No	No	Yes
Supervised/ Unsupervised	Yes	Yes	Yes
Architecture	1D & 2D ResNets	2D ResNets	LSTMs
Folding Time	10 minutes to few hours on 20 CPUs	Not provided	Milliseconds with 1 GPU
Training Time	Hours	About 5 days	Weeks to months with 1 GPU
Secondary Structure Prediction	Predicted with a separate ResNet	Implicit	Implicit

TEMP OVERVIEW OF APPROACH SLIDE

- Our basic overview of approach
  - Part 1 (ODE + amber trajectory)
  - Part 2 (DRL)
  - Part 3 (VAE for MD)
- Challenges of approach and our proposed solutions to them

Our Approach

General Diagram of Our Approach



Challenges for our approach

Appendix

Multiple Sequence Alignment (MSA)

- A sequence alignment of 3 or more proteins such that homologous nucleotides or amino acids are located in the same column.
- Achieved by inserting gaps of varying length within the sequences, allowing homologous positions to be aligned with each other

Clustal Omega

Position-Specific Scoring Matrix (PSSM)

- Specifies the scores for observing particular amino acids at specific positions, based on the amino acid frequencies at every position of a MSA.
- $20 \times L$ Matrix. 20 corresponds to the amino acid alphabet and L corresponds to the sequence length
- The probability of seeing amino acid i in position j of the sequence is: $$PPM_{ij} = \frac{c_{ij}+b}{k+20(b)}$$

  where c_{ij} is frequency of amino acid i at position j. b is a pseudocount added to account for possible examples of the pattern that are not represented in the training data. k is the number of sequences in the MSA.

- The matrix is then given by calculating the log odds ratios score of the observed given amino-acid at given position: $$PSSM_{ij} = \log \frac{PPM_{ij}}{p_i}$$ where p_i is the background distribution of amino acid i.

A Simple Sequence to PSSM Example

	Sequences
$k = 3$	$NIAGECC$
$b = 1$	$NTEHEWI$
$\forall i,\ p_i = 0.05$	$NITRGEW$

$$\Longrightarrow \quad PPM_{ij} = \frac{c_{ij}+b}{k+20(b)}$$

$$PPM = \begin{array}{c} N \\ T \\ \vdots \\ E \\ G \end{array} \begin{bmatrix} 0.174 & 0.044 & 0.044 & 0.044 & 0.044 & 0.044 & 0.044 \\ 0.044 & 0.087 & 0.087 & 0.044 & 0.044 & 0.044 & 0.044 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0.044 & 0.044 & 0.087 & 0.044 & 0.130 & 0.087 & 0.044 \\ 0.044 & 0.044 & 0.044 & 0.130 & 0.087 & 0.044 & 0.044 \end{bmatrix}$$

$$\Downarrow \quad PSSM_{ij} = \log \frac{PPM_{ij}}{p_i}$$

$$PSSM = \begin{array}{c} N \\ T \\ \vdots \\ E \\ G \end{array} \begin{bmatrix} 0.541 & -0.061 & -0.061 & -0.061 & -0.061 & -0.061 & -0.061 \\ -0.061 & 0.240 & 0.240 & -0.061 & -0.061 & -0.061 & -0.061 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ -0.061 & -0.061 & 0.240 & -0.061 & 0.416 & 0.240 & -0.061 \\ -0.061 & -0.061 & -0.061 & 0.416 & 0.240 & -0.061 & -0.061 \end{bmatrix}$$

Neural ODEs (Chen, et al. 2019)

We can represent Residual Networks in the form

$$h_t = h_{t-1} + f(h_{t-1}, \theta_t)$$

Viewing ResNets through a lense of differential equations, we can see that it resembles a discretization of Euler's Method, which is used to approximate continuous functions. Therefore, as we increase the number of layers in our ResNet $t \to \infty$ (i.e.), the network approximates a continuous function governed by the differential equation

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

We can then frame finding the output as an initial value problem where $h(0)$ is the input layer and $h(T)$ is the output "layer" at some time T. We can then use an ODESolver to solve the network. Our loss is can thus be generally defined as

$$L(z(t_1)) = L(z(t_0) + \int_{t_0}^{t_1} f(\mathbf{z}(t), t, \theta)dt = L(ODESolve(z(t_0), f, t_0, t_1, \theta))$$

Rather than backpropagate through the ODESolver (which would require a copious amount of memory), we can compute gradients with the *adjoint sensitivity* method where we solve the adjoint equation which results in constant space and linear time complexity.

Adjoint Sensitivity Method to Find the Gradient of Loss

In order to reverse auto-differentiate, we must first find the gradient of loss with respect to the hidden states which we will define as the *adjoint*, $\mathbf{a}(t) = \frac{d\mathbf{L}}{d\mathbf{z}(t)}$. Its dynamics are given by

$$\frac{d\mathbf{a}(t)}{dt} = -a(t)^T \frac{\partial f(\mathbf{z}(t),t,\theta)}{\partial z}$$

which we can obtain with a call to an ODESolver. In order to find $\frac{d\mathbf{L}}{d\mathbf{z}(t_0)}$, the ODESolver can be called again and must run backwards starting from $\frac{d\mathbf{L}}{d\mathbf{z}(t_1)}$. Note that in order to find $\frac{d\mathbf{L}}{d\mathbf{z}(t_0)}$, we need to recompute each hidden state together with the adjoints. Finally, we calculate the gradient of the loss with respect to the parameters with another integration.

$$\frac{d\mathbf{L}}{d\theta} = -\int_{t_1}^{t_0} a(t)^T \frac{\partial f(\mathbf{z}(t),t,\theta)}{\partial \theta} dt$$

In summary, we can calculate $a^T \frac{df}{dz}$ and $a^T \frac{df}{d\theta}$ with automatic differentiation. We can then calculate the integrals necessary for $\mathbf{z}$, $\mathbf{a}$, and $\frac{dL}{d\theta}$ with a single ODESolve.

Control for bias Hamiltonian (Wang, et al. 2020)

Simulating the positions of particles with conserved energy requires integrating the Hamiltonian equations of motion:

$$\frac{\partial p_i}{dt} = -\frac{\partial H}{\partial q_i}, \quad \frac{\partial q_i}{dt} = -\frac{\partial H}{\partial p_i}$$

where p_i and q_i are the momentum and position of the i-th particle, respectively. H is the Hamiltonian of the Systems; The Hamiltonian of a system specifies its total energy. For conservative systems it is given by:

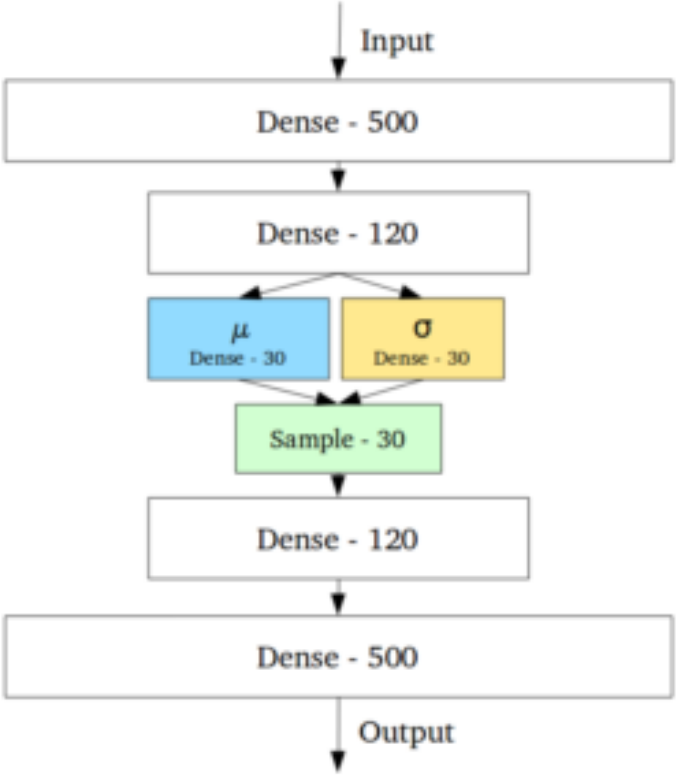
$$H(p, q) = U(q) + \sum_i \frac{||pi||^2}{2m_i}$$

Where $U(q)$, is the potential energy for all the particles, and $||p_i||^2/(2m_i)$ is the kinetic energy of the i-th particle.

Simulating an entire system explicitly tracking all relevant degrees of freedom is computationally intractable in most cases. Typically, one is interested in a small subset of a system, such as a molecule or protein, and concerned only with the influence of the environment, also known as *bath*, on the system but not the details of the environment itself. For this reason, one usually incorporates an environment Hamiltonian H_b with coarse-grained macroscopic variables of interest into the original Hamiltonian: $H_{tot} = H + H_b$. The environment, and therefore H_b, can also be explicitly controlled and optimized in many complex physical processes.

Variational Autoencoder

- Unsupervised learning method
- Generally, autoencoders are to learn parameters such that the output is identical to the input
- However, input is passes through information bottleneck so network must learn the most salient information
- VAE's are stochastic versions of traditional autoencoders; it learns a distribution for each latent state attribute which is then sampled for the output



LSTMs (Long-Short-Term-Memory Networks)

- LSTMs allows the recurrent network to learn how to control the data flow with various gates and prevents vanishing gradient
- The **learned parameters** are the weight matrices, $W_f, W_i, W_o,$ and W_c, and the bias vectors, $b_f, b_i, b_o,$ and b_c

$$f_t = \sigma\left(W_f\left[x_t, h_{t-1}\right] + b_f\right)$$

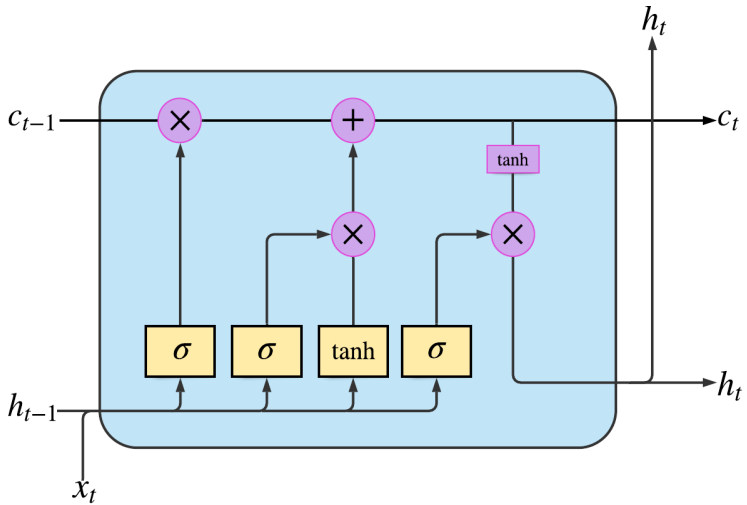
$$i_t = \sigma\left(W_i\left[x_t, h_{t-1}\right] + b_i\right)$$

$$o_t = \sigma\left(W_o\left[x_t, h_{t-1}\right] + b_o\right)$$

$$\tilde{c}_t = tanh\left(W_c\left[x_t, h_{t-1}\right] + b_c\right)$$

$$c_t = i_t \odot \tilde{c}_t + f_t \odot c_{t-1}$$

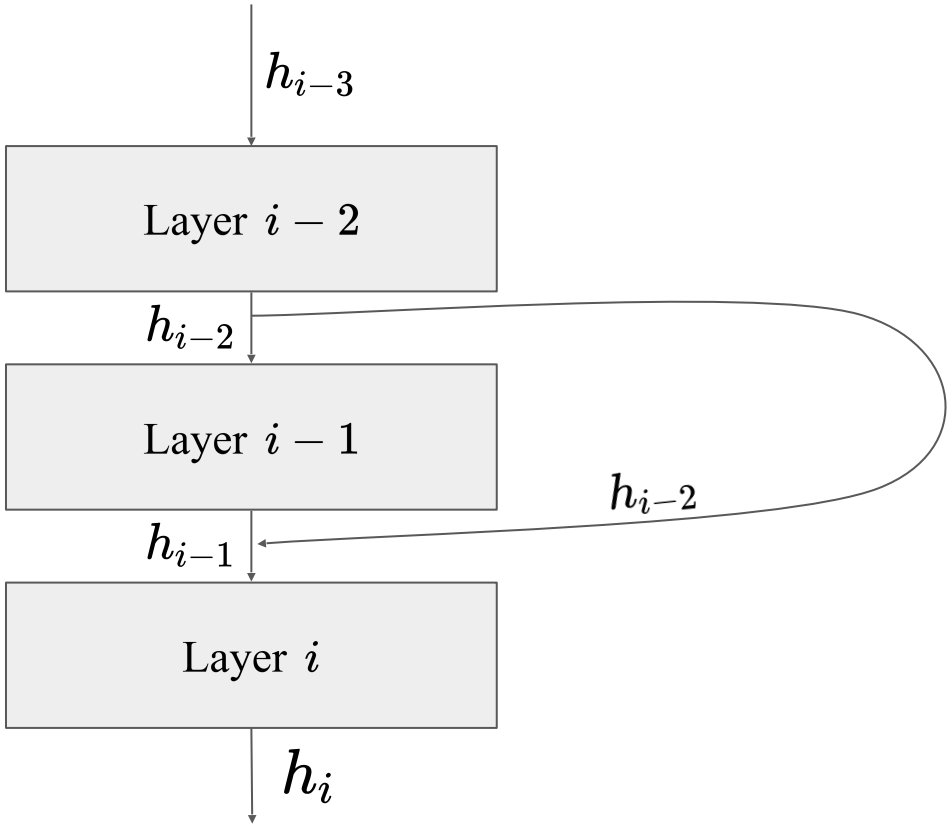
$$h_t = o_t \odot tanh\left(c_t\right)$$



AlQuraishi, 2019
Colah's Blog

Residual Networks

$$h_i = \alpha(W(h_{i-1}) + b_i + h_{i-2})$$



*learned

Wikipedia

References

- Aspuru-Guzik, A. (2018). Automatic Chemical Design Using a Data-Driven Continuous Representation of Molecules. ACS Central Science, 4, 268-276. doi: 10.1021/acscentsci.7b0057
- Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M.A. (2013). Playing Atari with Deep Reinforcement Learning. ArXiv, abs/1312.5602
- Wang, W., Axelrod, S., & Gómez-Bombarelli, R. (2020). Differentiable Molecular Simulations for Control and Learning. ArXiv, abs/2003.00868
- Rashid, M.A., Khatib, F., & Sattar, A. (2015). Protein preliminaries and structure prediction fundamentals for computer scientists. ArXiv, abs/1510.02775.
- Xu, J. (2019). Distance-based protein folding powered by deep learning. Proceedings of the National Academy of Sciences, 116(34), 16856–16865. https://doi.org/10.1073/pnas.1821309116
- Ching, T., Himmelstein, D. S., Beaulieu-Jones, B. K., Kalinin, A. A., Do, B. T., Way, G. P., Ferrero, E., Agapow, P. M., Zietz, M., Hoffman, M. M., Xie, W., Rosen, G. L., Lengerich, B. J., Israeli, J., Lanchantin, J., Woloszynek, S., Carpenter, A. E., Shrikumar, A., Xu, J., Cofer, E. M., … Greene, C. S. (2018). Opportunities and obstacles for deep learning in biology and medicine. Journal of the Royal Society, Interface, 15(141), 20170387. https://doi.org/10.1098/rsif.2017.0387
- Jafari, R., Javidi, M.M. Solving the protein folding problem in hydrophobic-polar model using deep reinforcement learning. SN Appl. Sci. 2, 259 (2020). https://doi.org/10.1007/s42452-020-2012-0
- AlQuraishi, M. (2019). End-to-end differentiable learning of protein structure. Cell Systems, 8(4), 292-301.e3. https://doi.org/10.1016/j.cels.2019.03.006
- Chen, T.Q., Rubanova, Y., Bettencourt, J., & Duvenaud, D.K. (2018). Neural Ordinary Differential Equations. ArXiv, abs/1806.07366.
- Chowdhury, R., Beglov, D., Moghadasi, M., Paschalidis, I. Ch., Vakili, P., Vajda, S., Bajaj, C., & Kozakov, D. (2014). Efficient maintenance and update of nonbonded lists in macromolecular simulations. Journal of Chemical Theory and Computation, 10(10), 4449–4454. https://doi.org/10.1021/ct400474w
- Chowdhury, R., & Bajaj, C. (2010). Multi-level grid algorithms for faster molecular energetics. Proceedings of the 14th ACM Symposium on Solid and Physical Modeling, 147–152. https://doi.org/10.1145/1839778.1839799
- Senior, A. W., Evans, R., Jumper, J., Kirkpatrick, J., Sifre, L., Green, T., Qin, C., Žídek, A., Nelson, A. W. R., Bridgland, A., Penedones, H., Petersen, S., Simonyan, K., Crossan, S., Kohli, P., Jones, D. T., Silver, D., Kavukcuoglu, K., & Hassabis, D. (2020). Improved protein structure prediction using potentials from deep learning. Nature, 577(7792), 706–710. https://doi.org/10.1038/s41586-019-1923-7