# Chapter 5

# Graph neural network methods for protein structure prediction

## Abstract

A central problem in the field of biophysics is the limited understanding of the complex relationship between a biopolymer's sequence, structure, and function. Many researchers have developed models connecting protein sequence directly to function, the majority of which do not directly consider the protein's structure. This pattern results from the fact that sequence data are far easier to collect and therefore far more abundant. Computational methods based on physical principals, such as molecular dynamics simulations, attempt to address this problem by facilitating the prediction of structure from sequence and can provide more accurate methods upon which to make functional predictions. However, these methods fail to scale to the large sequence libraries necessary to support high-throughput functional screening assays. In recent years, significant strides have been made to apply deep learning methods to molecular structure prediction. These techniques dramatically reduce the computational requirements for this process and offer promising results in early applications. Unfortunately, many of the existing models predict structural features from amino acid identities, limiting their ability to account for post-translational modifications that may have a dramatic effect on the folded conformation. Here, I present a set of generalizable methods and preliminary results regarding the use of graph neural network models (GNNs) to perform protein structure prediction. These methods directly address the issues with existing techniques for rapid prediction of structural properties from primary sequence alone. I propose that the architecture of GNN models properly fits the format of available protein sequence and structure data, that these methods provide the flexibility to account for non-standard amino acid residues and post-translational modifications, and that using widely available computational resources, these methods can be applied at the scale necessary to perform high-throughput protein screening *in silico*.

## 5.1 Introduction

Predicting the relationship between a protein's primary amino acid sequence and its 3-dimensional folded conformation is one of the largest open challenges in the field of biophysics. The overall 3D structure of these macromolecules, which results from their unique folding patterns, dictates important properties such as stability and solubility as well as their functional activity [1]. However, the rate at which new protein structure datasets can be collected pales in comparison to the rates of novel sequence discovery and functional characterization [2]. Because of the inextricable link between a protein's 3D structure and its function, accurate predictive models of this process would be of immense value to the life science research community. Such models could be used to infer structural features and mechanistic functions of newly discovered or characterized protein sequences. To address this dire need, several techniques have been developed to predict and/or simulate the effects of sequence change on structure and, in turn, the relationship between that change in structure and the observed change in function.

Techniques for modeling protein structure have typically relied on two main, non-mutually exclusive approaches: pattern recognition and energy function-based simulation. Pattern recognition has seen widespread adoption in methods that aim to compare a query sequence to existing and known sequence or structure features. Some of these techniques predict the existence of contacts between residues using measures of sequence conservation, relying on the assumption that covarying residues are likely to be in physical proximity [3, 4, 5]. The results of these methods have inspired newer deep learning approaches that employ measures of sequence conservation to predict the full protein structures without the need for any structural template [6, 7]. These methods combine measures of sequence covariation and conservation with residue identity information to significantly boost the performance of the model. However, when structural data for a related proteins exists, far more accurate models can sometimes be built by methods combining pattern recognition with physics-based simulation. One example of such a technique is homology modeling [8], which recognizes known sequence features and uses these to align portions of a protein structure to previously characterized structures and yield an estimate of the query protein's conformation. Atomic force fields can then be used to simulate the overall energy of the structure and adjust its local conformation to move it to a lower energy, and therefore more probable, state. These newer deep learning techniques appear to have a significant advantage by balancing accuracy with the ability to perform these predictions in high-throughput. Unfortunately, many of the deep learning models developed to date suffer from the fact that their input and output data structures (2D pairwise contact or distance matrices) and underlying operations (2D convolutions) are highly rigid and fit the irregular chemical structures of proteins poorly.

Recently, neural network and deep learning methods have been developed that are suitable for use on irregular graphs where the connectivity between nodes is not fixed at a single values, unlike grid-like structure by 2D CNNs. Such methods are useful for learning representations both of individual nodes or edges within a graph [9] and for performing full graph-level classification [10]. Given that a molecular structure can be represented as a graph, with atoms as the nodes and bonds as the edges, it follows that these powerful new methods may be applicable to problems in protein structure and function prediction. Some early methods have been successfully applied to protein function problems such as paratope [11] and small molecule binding prediction [12]. However, these methods either consider only the effect of amino acid sequence without allowing for post-translational modifications [11] or use direct measurements of protein structures from crystallographic data as the input to the model [12]. In the latter case, they take advantage of known non-covalent interactions as measured by inter-atomic proximity to generate a more complete picture of the folded protein structure, and would have been blind to such features if using the polypeptide sequence (or covalent network of bonds) alone as inputs. Fully modeling the relationship between sequence, structure, and function requires models that take as input the covalent bond network as defined by the protein's primary amino acid sequence and post-translational modifications to predict the non-covalent contacts or distances present in the fully-folded structure.

Fortunately, the problem of structural contact prediction using graph neural networks fits nicely into existing frameworks and problem definitions in the language of graph theory. This problem can be formulated as a link prediction problem, defined as the task of predicting novel edges that may exist within a graph but are not present in the given graph structure [13]. This problem is perhaps most thoroughly explored in social network analysis, where the goal is to predict relationships between people that may not yet exist or be represented [14]. Due to its importance in social theory and other tasks, many techniques exist to predict links in network structures [15, 16, 17]. Applying these methods to the problem of predicting non-covalent links that exist in a protein structure could yield time- and computation-efficient models of protein structure that could be employed for the problem of protein engineering. Here, I present preliminary work, collectively known as Tessellate, to model the non-covalent contacts governing protein structure as a link prediction problem using graph neural network methods.

## 5.2 Methods

### 5.2.1 Raw data processing pipeline

No tool currently exists to rapidly transform protein data from sequence and structure files into the necessary graph representations for GNNs. To address this, I developed a pipeline capable of performing these conversions in parallel. This pipeline takes as input an mmCIF file retrieved from the Protein Data Bank and passes it through a series of operations performed by various programs to add missing, unmodeled atoms to the structure, remove unnecessary solvent atoms, and convert the structure into the graph format necessary for training the GNN models. Each structure is processed at the atom-level instead of the residue level to limit the number of unique embeddings necessary for the model input. Whereas the total number of atoms in the periodic table is relatively fixed, the number of chemically unique “residues”, when accounting for pot-translational modifications or non-standard amino acids, is nearly infinite making embedding generation inefficient and impractical.

The first step in this pipeline is to split each mmCIF file into its constituent bioassemblies. Each bioassembly represents a different model of the protein or complex structure, each of which contains unique conformational information that will be processed independently through the pipeline. These different assemblies often provide additional conformation data that can be used to train models that more accurately reflect the dynamics of the protein structures. Each bioassembly model then goes through three parallel processing pipelines to generate the data used for model training. The first of these pipelines returns all pairwise contacts between all atoms in the structure, the second identifies and extracts the covalent bond network based on the structure, and the third identifies, extracts, and integrates the covalent bond network based on the polymer sequences to fill any gaps left unmodeled in the structure file.

### Identifying contacts

First, all atom- and residue-level contacts are identified using the GetContacts toolkit [18]. This toolkit requires that hydrogen atoms, initially unmodeled in many of the PDB structures, be added to the structure file. PyMol is used to perform this addition and output a hydrogenated version of each initial structure files [19]. GetContacts can then be run on this modified file to return a list of all inter-atomic contacts. Based on this output file, inter-residue contacts can be calculated by mapping each atom to the residue to which it belongs. To reduce storage demands, this step is generally performed at the time of model training and only the inter-atomic contacts are stored in file form.

### Calculating covalent adjacency matrix based on structure

Much of the graph structure of a given polymer can be identified directly from the mmCIF file. To do this, the solvents are first removed from the mmCIF file and the OpenBabel format conversion tool [20] is used to convert the mmCIF file to a molreport file that lists each edge (covalent bond) present in the structure along with the identities of the nodes (atoms) present at the end of each edge. The atomic number is used to retrieve a trainable embedding that represents node features during the training of the model.

Unfortunately, this conversion process is not perfect. For many structures, flexible regions of the protein cannot be properly characterized by existing crystallographic techniques and are left unmodeled and omitted from the structure file entirely. Because graph neural networks employ the underlying adjacency matrix, in this case the covalent bond network, to propagate information, missing connections in this adjacency matrix (unmodeled atoms), could significantly hinder the method's ability to learn meaningful representations predictive of inter-residue contacts. To alleviate the issue of missing atoms, I then calculate the full adjacency matrix from the raw sequence and join this information to the graph resulting directly from the structural data.

### Calculating covalent adjacency matrix based on sequence

The mmCIF files from the PDB also contain the full sequences of all polymer chains in the structure. These sequences can be extracted into fasta files using a custom script built on Biopython [21], . The fasta files can then be used as input to OpenBabel to again convert them to molreport format, giving the node and edge features that define the covalent bond graph of the structure. While all atoms comprising the biopolymer chains will be present in the sequence graph, these sequence data from the mmCIF files give no indication of the presence of post-translational modifications or small-molecule ligands upon which the final structure of the protein may depend. To create the most accurate graph representation of the covalent bonds of a structure, it is therefore necessary to combine the graphs generated by both the structure- and sequence-processing pipelines.

### 5.2.2 Combining covalent graphs from structure and sequence

Given the limitations of the covalent bond graphs generated from either the structure or sequence of the biopolymer independently, it is necessary to combine the data from these sources to complement any gaps and to create the final covalent bond graph to be used for model training. Each group of nodes and edges from the two pipelines is treated as a set, and the union of these sets is taken to represent the final covalent bond structure of the biopolyer or biopolymer complex. Processing the data in this manner both closes gaps due to unmodeled atoms and fully accounts for chemical groups or disulfide connections that cannot be derived from sequence information alone.

The combined datasets from these two pipelines generate three files describing the structure of the covalent bond graph. The first file contains the identities of every atom in the structure, which are used to generate embeddings at the time of model training. The second file contains the covalent adjacency matrix for the given structure, which describes every pair of atoms for which there exists a covalent bond. The third and final file is a mask that indicates which atoms were not present (unmodeled) in the original structure file. This mask serves a critical role during model training as it is used to down-weight predictions of connections between pairs of residues that contain unmodeled atoms. All processing of the raw structure files was automated using the Snakemake tool [22].

### 5.2.3 Subgraph data processing

Loading graph representations of large structures into memory is inefficient and may have a negative effect on the training speed and convergence time of deep learning models. Leading models such as AlphaFold [6] sidestep this issue by performing predictions on patches of the contact or distance map that are of fixed size in both the input and output dimension. To convert existing Tessellate models to this strategy, I implemented a dataloader capable of outputting batches of fixed-size input and output pairs (equal dimension each) instead of loading all data from a single structure into memory at once. To fix output sizes, the dataloader will treat each potential interaction (e.g. the potential interaction between residue 3 and residue 4) as an independent example. A fixed number of these interactions can then be batched together (e.g. a batch size of 64 could support a contact map patch size of 8x8).

Fixing input sizes complicates the data loading procedure, as different residues have variable number of constituent atoms. To address this problem, I have built a dataloader that selects two atoms within the "focus residues" and traverses out from these along the covalent graph structure until a fixed number of atoms are added to a subgraph. This process can be performed greedily, accepting all atoms of neighboring residues until a final step where there potentially exist more possible atoms to include than space available in the subgraph. At this point, I perform random selection of the neighboring atoms to add to the subgraph and potentially weight them by the number of connections to atoms already added to the subgraph.

For this data loading procedure to be efficient (i.e. not bottlenecked by the preprocessing to be performed by the CPU), the data must be further processed to avoid exhausting the computational resources available to the dataloader. To minimize the number of dataloader-performed processing operations, I used a custom Python script to preprocess the necessary datasets and load them into a resource-efficient TileDB database [23]. Descriptions of the final processed datasets are given below.

`atom_ids`

Dense 1D array of with length = number of atoms in structure. The main attribute for each atom is the atomic number (accessible using the `atomic_number` attribute). Hydrogen atoms are omitted. Additional attributes include:

- `chain` - the alphabetic chain identifier
- `res_type` - the three letter encoding for each residue (fully capitalized)
- `res_num` - residue number within the given chain
- `atom_name` - Unique name for each atom within a given residue

Shape: `[number of atoms]`

`atom_adj`

Sparse 2D array of covalent contacts between all atoms. Value is 1 if a covalent contact exists, 0 otherwise.
Dense shape: `[number of atoms` $\times$ `number of atoms]`

`atom_dist`

Dense 2D array of hop distances and connectedness between every pair of atoms. Hop distance as calculated by NetworkX [24] implementation of Floyd-Warshall algorithm [25]. Value of `distance` per cell is distance if connected, -1 otherwise. Value of `connected` per cell is 1 if connected, 0 otherwise.
Shape: `[number of residues` $\times$ `number of residues]` with `distance` and `connected` attributes

`atom_contact`

Sparse 3D array of contacts of different types for each pair of atoms. Channels are:

1. `hp` - Hydrophobic interactions
2. `hb` - Hydrogen bonds
3. `vdw` - van der Waals interactions

4. `wb` - Water bridges
5. `sb` - Salt bridges
6. `ps` - Pi-stacking
7. `pc` - Pi-cation
8. `ts` - T-stacking

Dense shape: `[number of channels` $\times$ `number of atoms` $\times$ `number of atoms]`

`atom_mask`

Dense 2D tensor masking the atoms not modeled in the original structure. Cell value is 1 if the atom was modeled and 0 otherwise.
Shape: `[number of atoms` $\times$ `number of atoms]`

`res_ids`

1D array mapping atom number to information about chain, amino acid.
Shape: `[number of residues]`

`res_adj`

Sparse 2D array of covalent contacts between all atoms. Value is 1 if a covalent contact exists, 0 otherwise.
Dense shape: `[number of residues` $\times$ `number of residues]`

`res_dist`

Dense 2D array of hop distances and connectedness between every pair of residues. Hop distance as calculated by NetworkX [24] implementation of Floyd-Warshall algorithm [25]. Value of `distance` per cell is distance if connected, -1 otherwise. Value of `connected` per cell is 1 if connected, 0 otherwise.
Shape: `[number of residues` $\times$ `number of residues]` with `distance` and `connected` attributes

`res_contact`

Sparse 3D array of contacts of different types for each pair of atoms. Channels are:

1. `hp` - Hydrophobic interactions
2. `hb` - Hydrogen bonds

3. `vdw` - van der Waals interactions
4. `wb` - Water bridges
5. `sb` - Salt bridges
6. `ps` - Pi-stacking
7. `pc` - Pi-cation
8. `ts` - T-stacking

Dense shape: `[number of channels` $\times$ `number of residues` $\times$ `number of residues]`

`res_mask`

Dense 2D tensor masking the atoms not modeled in the original structure. Cell value is 1 if the atom was modeled and 0 otherwise.
Shape: `[number of residues` $\times$ `number of residues]`

`mem_mat`

Sparse 2D membership matrix assigning each atom to 1 and only 1 residue in the final structure.
Dense shape: `[number of residues` $\times$ `number of atoms]`

### 5.2.4 Model architectures

To date, I have tested the ability of several different architectures to predict protein graph structures, with an emphasis on models likely to provide interpretability. However, these architectures and the design choices behind them have not been thoroughly vetted and are likely extremely far from optimal. Consequently, these models should only be used as a starting point for further exploration.

#### Full-structure Graph Attention Network (GAT)

The graph attention network (GAT) [26] has previously demonstrated impressive performance in structural biology tasks [11]. This architecture uses a learned attention mechanism to calculate a weighted average of a node's neighbors to generate a new embedding at each stacked layer. GAT architectures have the advantage of increased interpretability over competing techniques due to the ability to visualize the scores output by each attention mechanism. As such, it stands to reason that a GAT model trained on the full covalent adjacency matrix would have the capacity to both learn patterns in protein biochemistry and relate these identified patterns to the user through the interrogation of these attention scores.

In Tessellate's implementation of a GAT-based contact map prediction model, the atoms from a given structure are first embedded to a fixed-length vector representation. These embeddings then pass through a batch-normalization layer [27]. After normalization, the normalized atom embeddings are passed through a fixed number of GAT layers each consisting of multiple attention heads, as in the initial GAT implementation [26]. Each of these layers is followed by a leaky ReLU activation [28] and a batch normalization. At this point, the vectors representing each atom in each residue are summed to generate residue-level embeddings. These embeddings are again passed through several GAT layers with activation and batch normalization as for atom-level embeddings. Finally, the dot product of each pair of embeddings is calculated and this vector representation is concatenated to the hop-distance for each pair of residues. The residue-pair embeddings then pass through several linear layers with ReLU activation and batch norm before generating an 8-dimensional output through sigmoid activation to predict a contact in each of the 8 non-covalent contact channels. Loss is calculated as the mean binary cross entropy between the predictions and the target for each potential contact in the upper triangle of each channel's contact matrix.

### Full-structure Position-aware Graph Neural Network (P-GNN)

A second promising GNN approach is the position-aware graph neural network (P-GNN) [29]. This approach addresses a common challenge of GNNs: that two subgraphs with identical structures, but residing in very different locations within a larger graph, will often generate identical or near-identical embeddings. This property is particularly problematic for modeling polymeric chains, as, by definition, these chains are comprised of identical subgraphs, amino acids in the case of proteins and nucleotides for DNA. P-GNNs solve this identical subgraph problem by generating embeddings based on randomly chosen sets of "anchor nodes". Information from these anchor node sets is only propagated to nodes within or adjacent to the set to enable embeddings that are "aware" of their global position within the graph. Through careful selection of a variable number of variable sized anchor sets, P-GNNs can perform provably better than competing GNN techniques.

For the P-GNN-based contact map prediction model, the architecture follows directly from the logic outlined for the GAT model described above. The initial embedding step is identical and batch normalization is again performed after every layer, but the multi-headed GAT mechanism is replaced with a P-GNN for each layer through the depth of the network. The main difference between the GAT and P-GNN models lies in the method by which the 8 channels of contacts are predicted. For the GAT model, further GAT steps were employed at the residue level before a fully-connected neural network performed link prediction baed on the embeddings generated by the GAT layers. By contrast, for the P-GNN network, a 2D CNN was used directly following the final atom-level graph convolution. Loss is again calculated as the mean binary cross entropy between the predictions and the target for each potential contact in the upper triangle of each channel's contact matrix.

**Patch Graph Attention Network (patchGAT)**

Taking inspiration from the success of the AlphaFold model [6], the proposed patchGAT architecture aims to balance the power and interpretability of the GAT architecture with the computational feasibility of fixed-input fixed-output techniques such as 2D convolutions. A major challenge of training GNNs on large sets of non-uniformly sized graphs is that the memory requirements to process each example are dynamic and scale poorly with increasing input size. The patchGAT architecture aims to solve this problem by employing the same underlying structure as the GAT, but processing only a fixed-sized subgraph of the covalent protein structure on each forward pass. Because the number of atoms and residue-residue contacts (input and output size, respectively) are fixed for each training example, multiple training examples can be batched together for more efficient processing. Moreover, with a fixed batch size, the memory requirements to process each batch become fixed and predictable, allowing the design of a model architecture that easily fits into GPU memory.

### 5.2.5 Data set selection

An initial dataset to train and validate the models was selected by performing a search across the entirety of the PDB using the following criteria:

- Chain Type: Protein chain but no DNA or RNA or Hybrid
- Experimental Method: X-RAY
- Ligand Search: Has no free ligands
- Number of Chains: $>0$, $\leq 1$
- Model Count: 1

The search results can be regenerated using the search string given below:

```
Chain Type: there is a Protein chain but not any DNA or RNA or Hybrid and
Experimental Method is X-RAY and Ligand Search : Has free ligands=no and
Stoichiometry in biological assembly: Stoichiometry is A and
Number of Chains Search : Min Number of Chains=0 Max Number of Chains=1 and
ModelCountQuery: mvStructure.modelCount.comparator==
    mvStructure.modelCount.value=1
```

These search parameters were chosen to test the initial implementations of the model for a variety of reasons. First, by having all structures be monomeric, issues with graph symmetry could be avoided. In addition, a monomeric constraint reduced the overall size of the structures that would be returned, as they would not be complexed with other chains or small molecules. Second, small sizes for the input dataset were important because GPU resources were limited for the initial implementation to be tested. Having large training examples would not be permissive of a high-capacity model that may be necessary to learn import contact-predicting features. Third, the lack of ligands would minimize the number of unique structural features that the network would observe, making it more likely to identify useful patterns in the data. Ultimately, a large proportion of the returned structures were still too large to fit into GPU memory for most models. Accordingly, the list was further reduced by a combination of thinning based on the pre-defined ProteinNet training and validation datasets [30] and random selection of examples that could be processed by the majority of the model implementations without raising an out of memory error.

The final structures used for model training and evaluation are given in Table 5.1.

| | |
|---|---|
| Training examples | `3CBN, 4MQ3, 2W2S, 3ZZP, 5DAZ, 4G29, 4IL7, 3PIW, 3EY6, 1JYH, 2WWE, 2F3L, 2A90, 2CXA, 3NZL, 1CEW, 4JQF, 1JHS, 2NRR, 2VGA, 3RLE, 3B7X, 3DOA, 1UOY, 2NYV, 2F68, 5CWJ, 1I60, 2JAY, 4HCS, 4PGR, 4PN7, 1BM8, 3U4K, 1MIJ, 4QFT, 3VBC, 1Z3X, 2NLS, 2X3M, 4GLP, 1VK4, 2R4Q, 4QWV, 2YMO, 1XKR, 5FD9, 1IPA, 1OPS, 3EOH, 5HY6, 1HU3, 1C25, 2AHE, 2VGE` |
| Validation examples | `3O48, 1SR8, 2OHE, 2PGE, 2UYO, 2QGO, 2PQ6, 1DVO, 1WER, 3K1H, 2GGO, 3TCQ, 3NXH, 3LJN, 1J72, 2B3O, 2HBJ, 2V75, 1VLO, 4FLE, 4ME2, 2FP3, 3SBG, 1L8F, 4JG5, 3LYW, 3WKV, 4CH7, 3LOD, 3C8U, 4TRK, 1RC9, 1HUS, 1GLV, 3PJW, 3Q6B, 2PFT, 2AP3, 3F8T, 4UVK, 3JXV, 3ILV, 3LH5, 4M9P, 1F32, 2A0J, 4OO3, 4DB6, 4UML, 1AOA, 3G87, 1O13` |

Table 5.1: Training and validation set structures for multiple-example training runs.

### 5.2.6 Code and data availability

All code is available at `https://github.com/FordyceLab/tessellate`. Protein structure files were retrieved from the Protein Data Bank at `https://www.rcsb.org`.

## 5.3 Results

### 5.3.1 Overfitting to a single training example

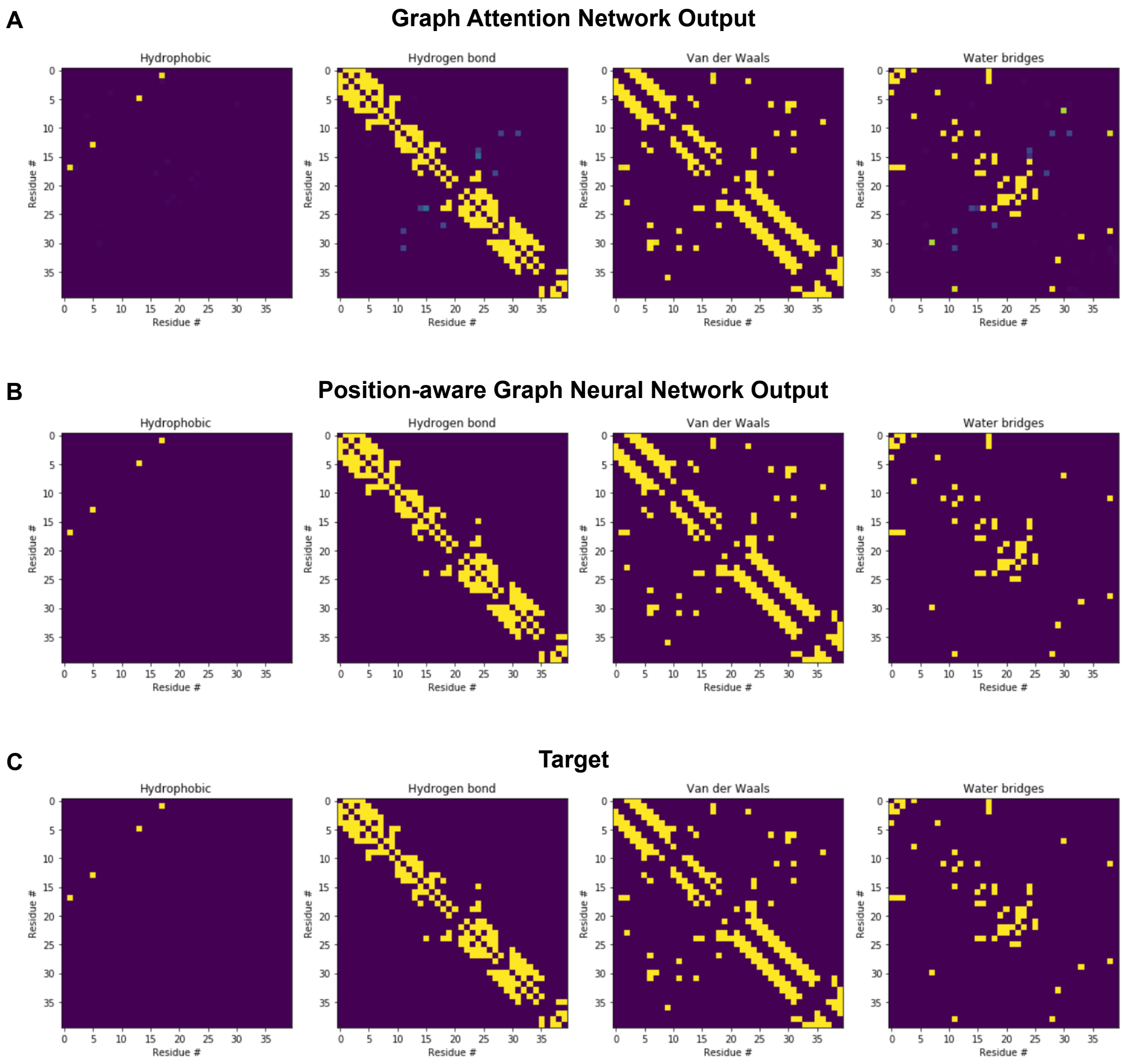

Figure 5.1: Overfit contact map predictions for two GNN architectures trained for 15000 epochs against the crystal structure 6E6O. Although all 8 contact channels were predicted, positive contacts were only present in the first four (hydrophobic, hydrogen bond, van der Waals, and water bridges). Only these four channels are shown, as predicted values were uniformly close to 0 for the other channels lacking contacts. Values near 0 are shown in purple and values near 1 are shown in yellow. **(A)** Predictions resulting from the full-structure GAT model. **(B)** Predictions resulting from the full-structure P-GNN model. **(C)** The target contact map used to calculate loss during training.

Each model architecture was explored using the “overfit, regularize, tune” paradigm set forth by Andrej Karpathy [31]. As such, the initial goal for each architecture style was to generate a model capable of overfitting to a single protein structure when the model is trained repeatedly and exclusively against that example. During overfitting, the model is trained to the point where it is able to predict with perfect or near-perfect accuracy against a given dataset at the expense of generalizing to other datasets. Through this process, a researcher can assess whether a model has the capacity to fit datasets similar to the initial target and discard models likely to perform poorly in more complex tasks. For example, an underpowered model may not have the complexity to accurately model a small dataset, resulting in a failure to overfit, even with indefinitely extended training. Such models would be therefore unlikely to work on a larger dataset with significantly more more diverse examples.

For each of the model architectures outlined above, I implemented several versions of the same general structure, varying hyperparameters with each new model. Each independent model was then trained on a single training example picked from a set of small, monomeric protein structures form the the PDB (PDB ID: 6E6O - Pheromone from *Euplotes raikovi*, Er-1). This example was selected for its high crystal structure resolution (0.7 Å) and its short chain length (40 amino acids). Each of the models was trained to predict the true contact map based only on the covalent bond graph and the identities of the atoms at every node. Training was allowed to continue for 15,000 epochs (passes over the training data point). At the conclusion of training, the predicted contact maps for the 6E6O structure from the trained model were examined and compared to the initial target dataset (Figure 5.1). While the P-GNN architecture classified each potential contact with perfect accuracy, the GAT network continued to assign relatively high probability to a small number of non-contacts in the hydrogen bond and water bridge contact channels at the end of training. Given that nearly all positive contacts were predicted accurately by the GAT model by the end of training, I accepted this result as evidence of overfitting and continued to scale both architectures to a larger dataset.

### 5.3.2 Scaling to a larger training set

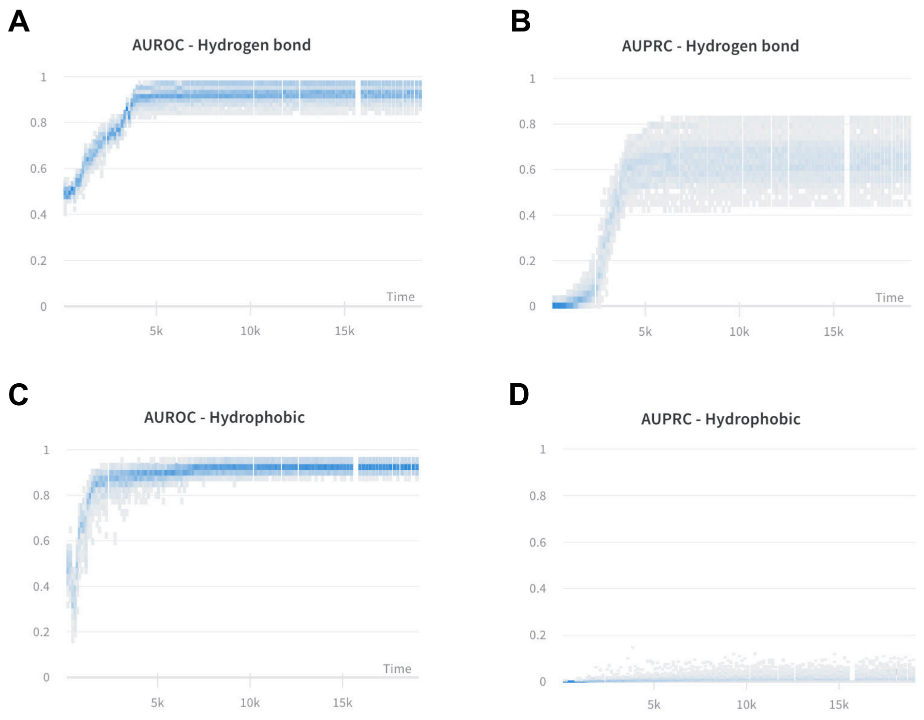

Figure 5.2: Distribution of the areas under the receiver operating characteristic (ROC) and precision-recall curves (PRC) for two contact channels, hydrogen bonds and hydrophobic interactions, are shown for all examples in the validation set over time (seconds) from the start of training. **(A)** Distributions of AUROC for the hydrogen bond channel. **(B)** Distributions of AUPRC for the hydrogen bond channel. **(C)** Distributions of AUROC for the hydrophobic channel. **(D)** Distributions of AUPRC for the hydrophobic channel.

Once overfitting had been demonstrated on a single-example training set, the next step was to scale to a larger and more diverse training set with a variety of sizes (numbers of atoms/residues) and contact patterns (alpha helices, beta sheets, etc.). For this task, I selected a set of 55 small, monomeric protein structures from the PDB. This training set was subset from the PDB search results described in Section 5.2.5 based on the ability of the model to process each example without running out of memory on the available GPU (NVIDIA GTX 1080Ti, 11GB RAM) and was complemented by a validation set of nearly equal size (52 example structures). The validation dataset was collected similarly to the training data, based on the initial culling of the PDB, then intersecting with ProteinNet's validation data, then selecting the structures that could fit into memory alongside the model architectures. For each pass, the model would train on the full training dataset and predict each example in the validation dataset.

A known issue with the PyTorch dataloader causes system memory use to increase when more than one thread is used for dataloading under certain circumstances (`https://github.com/pytorch/pytorch/issues/13246`). The dataloader used for the multiple-example training run suffered this issue and, as such, training time for multiple-example runs was limited by the time until system memory was exhausted and an out-of-memory exception was raised. As of writing, a solution for this issue has not been implemented. Consequently, all multiple-example runs were allowed to train until the point at which the exception was raised, causing training to terminate. Despite this limitation, relevant performance metrics of all tested models appear to have stabilized by the time an out of memory error ocurred (Figure 5.2).

Model performance was measured primarily by monitoring the area under both the receiver operating characteristic (AUROC) and the precision recall curve (AUPRC) for each of the 8 contact channels. As expected, performance for each of these metrics is quite poor at the beginning of training, but improves as the model is optimized. An example of the change in these metrics for a P-GNN model is shown in Figure 5.2, with the hydrogen bond (Figure 5.2A,B)and hydrophobic interaction (Figure 5.2C,D) channels shown as examples. Based on AUROC metrics alone, the models' performance appears quite good, indicating a high true positive rate and low false positive rate (Figure 5.2A,C). However, the contact map target datasets are significantly unbalanced, with many more negative examples than positive examples. Consequently, model performance can be more accurately judged based on the AUPRC results, which are significantly less optimistic about the model's performance. For the hydrogen bond channel, AUPRC averages roughly 0.6 (Figure 5.2B) while AUPRC for the hydophobic interaction channel is considerably worse, finishing at around 0.05 (Figure 5.2D). These results indicate poor general performance of the model across multiple contact channels and suggest that the positive results observed in the AUROC results may simply be a function of class imbalance, since the poor PRC performance reflects the model's inability to recall positive examples.

One interesting observation from comparing the AUROC and AUPRC performance metrics is that only the average AUPRC (and not the AUROC) differs dramatically between the two channels. One potential reason for this disparity is the difference between both the average proximity and average abundance of either type of interaction. Hydrogen bonds, as calculated by GetContacts, tend to be both more abundant and present in closer proximity along the 1D sequence and, therefore, within the covalent bond graph (see Figure 5.1C for an illustrative example). It may be easier for the GNN model to learn patterns indicative of these interactions due to both their abundance and linear proximity.

### 5.3.3 Computational resource demands

As alluded to above, a major downside of the training approach described here is the limitation of available GPU memory. This constraint enforces limits on the size and complexity of the models and datasets that can be used for the training process. Results from the 55 example training set suggest that the models may not have the correct capacity to identify patterns necessary to accurately predict inter-residue contacts. Unfortunately, given the memory constraint of the currently available GPU resource, I was unable to test larger and more complex models on the small 55-member dataset.

Further examination of GPU utilization and memory suggests a more general issue with training a model on full protein structures. While GPUs can greatly expedite training of neural network models, they are most efficient when processing large, uniform batches of data that consume as many of the available resources as possible without exhausting them. Based on the allocated memory of the GPU used to train the models described (Figure 5.3A), it is not possible to fit models or datasets much larger than those employed here into memory on the GPU. In this regard, the current training pipeline is able to maximize available memory usage for the training process and it is not possible to, for instance, train two models on the same GPU at a given time. However, the GPU utilization at any point in time is both far more variable and lower than the amount of allocated memory (Figure 5.3A). This result indicates that two independent, but equally-concerning, phenomena are likely occurring during the training of these models. The first of these CPU bottlenecking. In cases where the training data cannot be processed and transferred to the GPU fast enough, the GPU accumulates idle cycles, leading to an overall decrease in utilization per unit time. The second issue is that the variable sizes of the independent structure examples lead to drastically different compute and memory requirements per example. If each example were a fixed size, the size of the model and the batch size of training examples could be adjusted to consistently utilize the majority of GPU resources. However, when the datasets are variable in size, this adjustment cannot be performed and the size of the model must instead be adjusted to fit the largest of the training data examples. This adjustment leads to inefficient resource utilization for any example smaller than the largest and raises significant concerns for scaling this approach to larger datasets, such as those that consider a greater proportion of the PDB structures.

The small monomeric protein dataset upon which these models were trained is not necessarily representative of the PDB as a whole. In general, these protein structures tend to be smaller than a significant portion of the PDB (Figure 5.3B). This size difference is due to a number of factors including the ab sense of multiple protein chains and non-protein ligands such as small molecules or nucleic acids. For the GNN approach to be viable for the prediction of both intra- and inter-chain contacts, it must be able to train on and predict contacts for multi-chain structures far larger than those presented here. For this reason, the most promising strategy moving forward appears to be processing potential contacts in batches using a scheme like that proposed for patchGAT above.

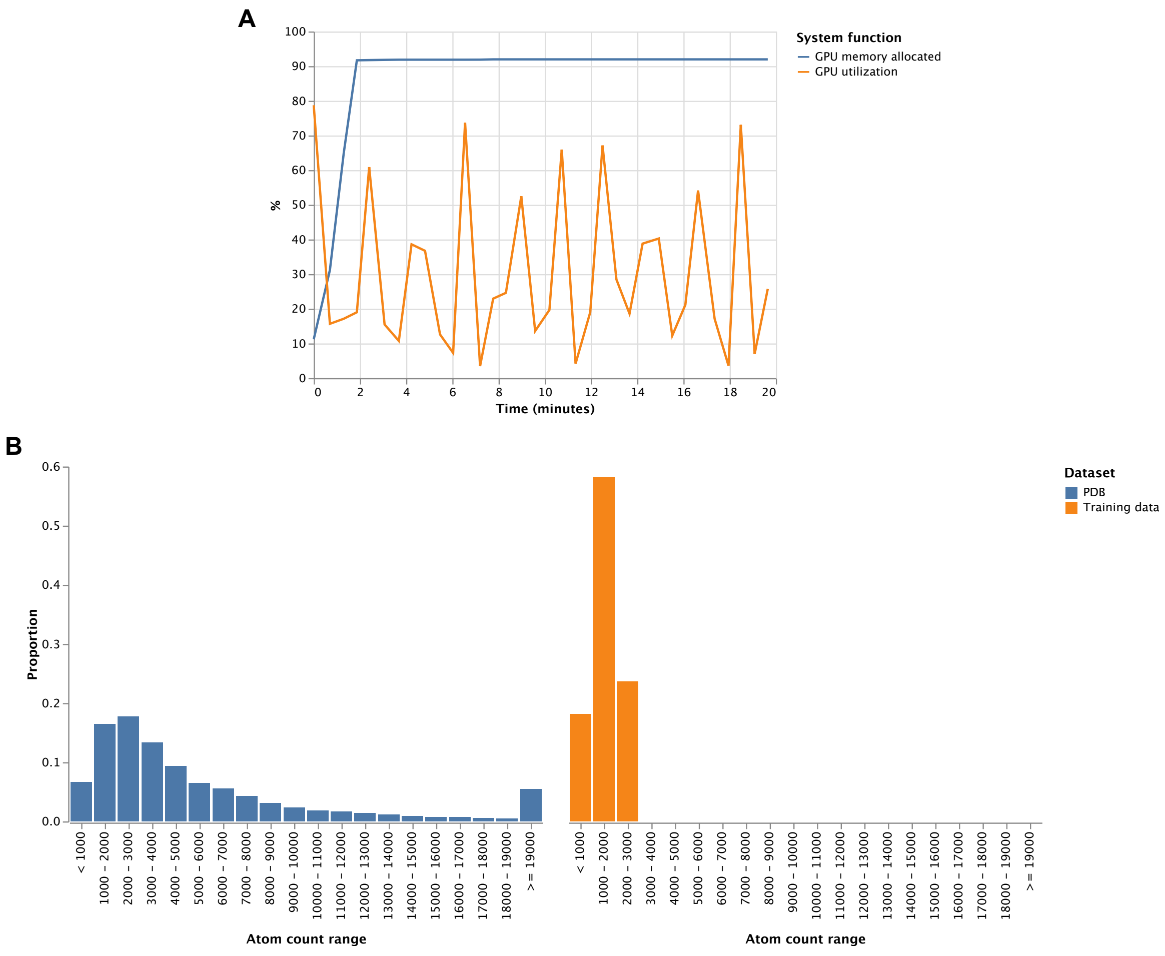

Figure 5.3: An example of GPU memory allocation and utilization during the first 20 minutes of training a P-GNN model on a set of 55 small, monomeric protein structures.

## 5.4 Discussion

As a result of technical challenges associated with loading data into available GPU memory, initial results of the various GNN architectures applied to the contact prediction problem are limited. However, a few variations of each model type have been implemented and tested on extremely small datasets with varying levels of success. All results and required scripts are present in the GitHub repository linked above and are available for further exploration by the reader, if desired. Despite limited results, these efforts revealed a number of important observations regarding the data processing pipeline and the computational efficiency of this project.

While GetContacts provides a fast, efficient tool to calculate the inter-atomic contacts present in a crystal structure or through MD trajectories, at the time of writing, it is not capable of calculating all contacts between all types of molecules in a structure. For example, GetContacts is explicitly not capable of reporting salt bridges, pi-cation, pi-stacking, and T-stacking interactions specifically between a protein component and a non-protein ligand. For this reason it may be particularly challenging to accurately predict the structures of ligand-bound proteins that rely on these types of interactions to be stable enough for crystallization, as these interactions will be missing from the dataset entirely. While GetContacts certainly has its limitations, competing tools, such as Arpeggio [32] come with their own set of issues regarding data format compatibility and similar blind spots when calculating contacts.

Beyond limitations imposed by the data processing pipeline, GNNs also have their own intrinsic limitations that pose significant challenges to their implementation for predicting protein or protein complex structures. The first of these challenges is intrinsic to the design of GNN models themselves. For instance, it is not immediately clear whether GNNs can be used to model complexes of proteins comprised of multiple identical molecular structures. GNN models are deterministic functions once they have been trained and their output relies only on the initial node embedding and graph structure given as input. As a consequence, if the same initial embedding vector is used for each atom of the same element and covalent graph structure is identical for two molecules, these molecules will receive identical embeddings during the training process. This perfect symmetry poses a challenge for the prediction of homooligomeric complex structures comprised of multiple identical protein chains. One potential solution is to introduce a stochastic operation, as in variational autoencoders [33], to break this symmetry and allow the chains to adopt unique embeddings during training and inference. However, the viability of this strategy remains to be tested.

The final, and perhaps most immediate challenge facing the implementation of a successful GNN for protein structure prediction is the design of a dataloading and model training pipeline that enables efficient use of computational resources. Because each structure, and therefore each graph fed to the model, is different in size, it is difficult to batch multiple examples together or to estimate the memory required to fit each example into GPU memory. Ultimately, this limitation leads to low overall utilization of GPU resources which limit training speed at best and out-of-memory issues that halt training at worst. One potential solution is to break the contact map up into small, fixed-sized patches and batch these together during the training process, as described in the proposed patchGAT architecture discussed above. While this pipeline would almost certainly result in better computational performance and far more predictable resource use, the ultimate effect on model accuracy remains unknown. One possibility is that, even for high-capacity models, the subgraph fed to the model would not include enough of the protein's covalent bond structure for the model to learn meaningful long-range dependencies necessary to predict distant contacts.

While GNN models offer a promising new way to model protein folding and intermolecular interactions, a significant amount of further research is required to validate their utility and characterize their strengths and weaknesses. The work presented here provides a robust, parallel data processing pipeline to generate the graph structures and contacts maps necessary to undertake this task as well critical observations to help guide these future inquiries.

# References

[1] C. N. Pace, S. Treviño, E. Prabhakaran, and J. M. Scholtz, “Protein structure, stability and solubility in water and other solvents,” *Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences*, vol. 359, pp. 1225–34; discussion 1234–5, Aug. 2004.

[2] E. C. Alley, G. Khimulya, S. Biswas, M. AlQuraishi, and G. M. Church, “Unified rational protein engineering with sequence-based deep representation learning,” *Nature Methods*, vol. 16, pp. 1315–1322, Dec. 2019.

[3] S. W. Lockless and R. Ranganathan, “Evolutionarily conserved pathways of energetic connectivity in protein families,” *Science*, vol. 286, pp. 295–299, Oct. 1999.

[4] D. S. Marks, L. J. Colwell, R. Sheridan, T. A. Hopf, A. Pagnani, R. Zecchina, and C. Sander, “Protein 3D structure computed from evolutionary sequence variation,” *PLOS One*, vol. 6, p. e28766, 2011.

[5] T. A. Hopf, A. G. Green, B. Schubert, S. Mersmann, C. P. I. Schärfe, J. B. Ingraham, A. Toth-Petroczy, K. Brock, A. J. Riesselman, P. Palmedo, C. Kang, R. Sheridan, E. J. Draizen, C. Dallago, C. Sander, and D. S. Marks, “The EVcouplings Python framework for coevolutionary sequence analysis,” *Bioinformatics*, vol. 35, pp. 1582–1584, May 2019.

[6] A. W. Senior, R. Evans, J. Jumper, J. Kirkpatrick, L. Sifre, T. Green, C. Qin, A. Žídek, A. W. R. Nelson, A. Bridgland, H. Penedones, S. Petersen, K. Simonyan, S. Crossan, P. Kohli, D. T. Jones, D. Silver, K. Kavukcuoglu, and D. Hassabis, “Improved protein structure prediction using potentials from deep learning,” *Nature*, vol. 577, pp. 706–710, Jan. 2020.

[7] M. AlQuraishi, “End-to-end differentiable learning of protein structure,” *Cell Systems*, vol. 8, pp. 292–301.e3, Apr. 2019.

[8] J. Yang, R. Yan, A. Roy, D. Xu, J. Poisson, and Y. Zhang, “The i-tasser suite: protein structure and function prediction,” *Nature Methods*, vol. 12, pp. 7–8, Jan. 2015.

[9] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” *ArXiv.*

[10] R. Ying, J. You, C. Morris, X. Ren, W. L. Hamilton, and J. Leskovec, “Hierarchical graph representation learning with differentiable pooling,” in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS’18, (Red Hook, NY, USA), pp. 4805–4815, Curran Associates Inc., 2018.

[11] A. Deac, P. VeliČković, and P. Sormanni, “Attentive cross-modal paratope prediction,” *Journal of Computational Biology*, vol. 26, pp. 536–545, June 2019.

[12] E. N. Feinberg, D. Sur, Z. Wu, B. E. Husic, H. Mai, Y. Li, S. Sun, J. Yang, B. Ramsundar, and V. S. Pande, “PotentialNet for molecular property prediction,” *ACS Central Science*, vol. 4, pp. 1520–1530, Nov. 2018.

[13] D. Liben-Nowell and J. Kleinberg, “The link prediction problem for social networks,” in *Proceedings of the Twelfth International Conference on Information and Knowledge Management*, CIKM ’03, (New York, NY, USA), pp. 556–559, Association for Computing Machinery, 2003.

[14] L. Backstrom and J. Leskovec, “Supervised random walks: predicting and recommending links in social networks,” in *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining*, WSDM ’11, (New York, NY, USA), pp. 635–644, Association for Computing Machinery, 2011.

[15] Y. Zhang, Q. Yao, W. Dai, and L. Chen, “AutoKGE: searching scoring functions for knowledge graph embedding,” *ArXiv*, vol. abs/1904.11682, 2019.

[16] L. Haonan, S. H. Huang, T. Ye, and G. Xiuyan, “Graph star net for generalized multi-task learning,” *ArXiv*, 2019.

[17] R. Wang, B. Li, S. Hu, W. Du, and M. Zhang, “Knowledge graph embedding via graph attenuated attention networks,” *IEEE Access*, vol. 8, pp. 5212–5224, 2020.

[18] A. J. Venkatakrishnan, R. Fonseca, A. K. Ma, S. A. Hollingsworth, A. Chemparathy, D. Hilger, A. J. Kooistra, R. Ahmari, M. M. Babu, B. K. Kobilka, and R. O. Dror, “Uncovering patterns of atomic interactions in static and dynamic structures of proteins,” *Biorxiv*, 2019.

[19] Schrödinger, LLC, “The PyMOL Molecular Graphics System, Version 1.8.” Nov. 2015.

[20] N. M. O’Boyle, M. Banck, C. A. James, C. Morley, T. Vandermeersch, and G. R. Hutchison, “Open Babel: An open chemical toolbox.,” *Journal of Cheminformatics*, vol. 3, p. 33, Oct. 2011.

[21] P. J. A. Cock, T. Antao, J. T. Chang, B. A. Chapman, C. J. Cox, A. Dalke, I. Friedberg, T. Hamelryck, F. Kauff, B. Wilczynski, and M. J. L. de Hoon, “Biopython: freely available Python tools for computational molecular biology and bioinformatics,” *Bioinformatics*, vol. 25, pp. 1422–1423, June 2009.

[22] J. Köster and S. Rahmann, "Snakemake - a scalable bioinformatics workflow engine," *Bioinformatics*, vol. 28, pp. 2520–2522, Oct. 2012.

[23] S. Papadopoulos, K. Datta, S. Madden, and T. Mattson, "The TileDB array data storage manager," *Proceedings of the VLDB Endowment*, vol. 10, p. 349–360, Nov. 2016.

[24] D. A. Schult, "Exploring network structure, dynamics, and function using NetworkX," in *In Proceedings of the 7th Python in Science Conference (SciPy)*, pp. 11–15, 2008.

[25] R. W. Floyd, "Algorithm 97: shortest path," *Communications of the ACM*, vol. 5, p. 345, June 1962.

[26] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph Attention Networks," *International Conference on Learning Representations*, 2018. accepted as poster.

[27] S. Ioffe and C. Szegedy, "Batch normalization: accelerating deep network training by reducing internal covariate shift," 2015.

[28] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *In ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, 2013.

[29] J. You, R. Ying, and J. Leskovec, "Position-aware graph neural networks," in *Proceedings of the 36th International Conference on Machine Learning* (K. Chaudhuri and R. Salakhutdinov, eds.), vol. 97 of *Proceedings of Machine Learning Research*, (Long Beach, California, USA), pp. 7134–7143, PMLR, 09–15 Jun 2019.

[30] M. AlQuraishi, "ProteinNet: a standardized data set for machine learning of protein structure," *BMC Bioinformatics*, vol. 20, p. 311, June 2019.

[31] A. Karpathy, "A recipe for training neural networks," Apr 2019.

[32] H. C. Jubb, A. P. Higueruelo, B. Ochoa-Montaño, W. R. Pitt, D. B. Ascher, and T. L. Blundell, "Arpeggio: a web server for calculating and visualising interatomic interactions in protein structures," *Journal of Molecular Biology*, vol. 429, pp. 365–371, Feb. 2017.

[33] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2014.