Compound-protein interaction prediction

Project report, spring 2022

Rodrigo Castellano Ontiveros
roco@kth.se

Course DD2402. Advanced Individual Course in Computational Biology
MSc Machine Learning

Contents

1 Introduction

Compound-protein interaction (CPI) plays an important role in drug discovery. The reason is that it is useful to predict possible side effects and find new uses for already existing drugs. Experimental identification of these interactions can be expensive and time-consuming. There are some methods used to aid experimental drug discovery, such as virtual screening (VS). The problem with VS is that it needs the 3D structure of the protein, therefore in cases where this 3D structure is not available other methods have to be applied.

Deep learning models have been successfully implemented to solve this task, leading to very good performance in CPI prediction. Several models can be applied, depending on the data available. Compounds can be either one-dimensional sequences or graphs, and proteins are treated as one-dimensional sequences (their 3D structure is disregarded). Different deep learning architectures have been applied so far, namely graph neural networks [Elbasani et al., 2021] to treat compounds as graphs, convolutional neural networks (CNN) to obtain the representation for different compounds, or recurrent neural networks [Karimi et al., 2019] to extract features from compounds and proteins. For this project, the focus was on the model applied in [Chen et al., 2020], based on transformers. Transformers perform successfully seq2seq tasks, and compounds and protein can be regarded as sequences. The authors have shown that their model improves previous work done in this area. The main goal is to investigate how they implemented the model and whether it is sensitive to different configurations of hyperparameters, as well as understand how certain changes in the architecture of the transformer can lead to changes in the behaviour of the model.

2 Methods

[Chen et al., 2020] focused on three main mistakes done in deep learning research, according to a research done by Google [Riley, 2019]. The first one was the use of datasets that are not proper for the goal set by the researchers, then the second problem was related to hidden compound bias, and the last one was splitting wrongly the dataset.

A dataset that suits the task of predicting the interaction between different compounds and proteins should let the model learn based on the interaction instead of focusing only on the compound or protein sequence. To address this, the dataset used by the authors has samples where, for the same compound, there is interaction and there is not. This is a way to make sure that the model does not focus only on the compound, but also in the protein. In some public datasets such as BindingDB, for some compounds only one class is present, and negative samples are introduced by using different algorithms, which may introduce noise in the dataset.

For the graph neural network model [Tsubaki et al., 2019] that was trained on Human dataset, it is shown in figure 1 the weight distribution of proteins and compounds. The weight distribution of proteins is focused around 0, which means that most of the information used by the model was the compound information. On the other side, experiments were done with only compound data, and the performance of both models with compound and protein and with only compound information was compared. The results for 10-fold cross validation with a two sample t-test for the resulting difference of AUC distribution was greater than 0.05, which led to the conclusion that using protein information does not bring significant improvement in results.

The dataset used by the authors in [Chen et al., 2020] was built in a way that label reversal is present. Label reversal refers to having compounds belonging to one class in the training set and with a different class in the test set.

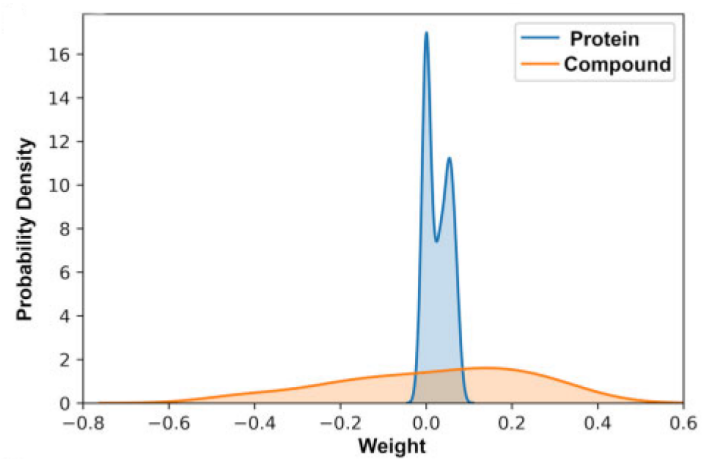

Figure 1: Results of the distribution of the weights for the model based on graph neural network. Blue represents the weight distribution of proteins with CNN and orange represents the weight distribution with graph neural networks [Chen et al., 2020].

2.1 Dataset

The dataset used in this project is called GPCR, created by [Chen et al., 2020]. The GPCR dataset comes from the GLASS database, where compound protein interactions are validated experimentally. GPCR is composed of information related to compounds, proteins and their interaction. In the dataset, each compound has samples where there is interaction as well as no interaction. The negative samples were validated experimentally, whereas in other public datasets the negative samples were generated based on similarity techniques or randomly doing cross combination of different compound and protein pairs. The GPCR dataset consists of 15343 CPI, with 5359 compounds and 356 proteins. For each compound, the frequency it is labelled as interacting and not interacting is approximately equal.

2.1.1 Compound information

For each sample, the information related to the compound is a list of all its atoms, and for each atom a different number of features, shown in table 1. Then, for each compound, there is also information about its adjacency matrix. The adjacency matrix is used to represent a graph, in which it is indicated when the different pair of vertices are adjacent or not. For our case, it reflects the relative position of each atom, and whether it is bonded with the other atoms (value 1) or not (value 0). Consequently, the spatial relation between atoms and the covalent bonds is specified. This is obtained

Feature	**Possible values**
Atom type	C, N, O, F, P, S, Cl, Br, I, other
Degree of atom	0, 1, 2, 3, 4, 5, 6
Formal charge	0,1
Number of radical electrons	0,1
Hybridization type	sp, sp2, sp3, sp3d, sp3d2, other
Aromatic	0,1
Number of hydrogen atoms attached	0,1,2,3,4
Chirality	0 (False) or 1 (True)
Configuration	R,S

Table 1: Features of the atoms included in the dataset.

with the python package RDKit, a library that takes the information of the compound in the Simplified Molecular Input Line Entry System format, SMILES, and converts it to the information described above. SMILES is commonly used to translate from a 3D structure to a sequence of ASCII symbols. Therefore, for each compound there is a tensor of dimension (number of atoms, atoms features), where the dimension of atoms features is 34, due to one-hot encoding of the variables.

2.1.2 Protein information

The **proteins** are given as amino acid sequence input to word2vec, applied to obtain embeddings. Word2Vec uses techniques such as Skip-Gram and Bag of Words. The process is first to divide the amino-acid sequence into 3-gram sequence and then a pretrained word2vec converts it to tensor representation. The dataset used by [Chen et al., 2020] to train word2vec is Uniprot, a database that contains protein sequence and functional information. The hidden dimension of the embedding was set to 100 . This results in a tensor of dimensions (number of words, 100) for each protein representation.

2.2 Architecture of the model

The architecture of the model implemented by [Chen et al., 2020] is shown in 2. Transformers play the main role in this architecture, by using multi-heads and self-attention layer to predict seq2seq tasks, where the input and output of the model are sequences. They had to modify the architecture of the original paper [Vaswani et al., 2017] to apply it to classification tasks. The main difference, as shown later, are the last layers and the encoder.

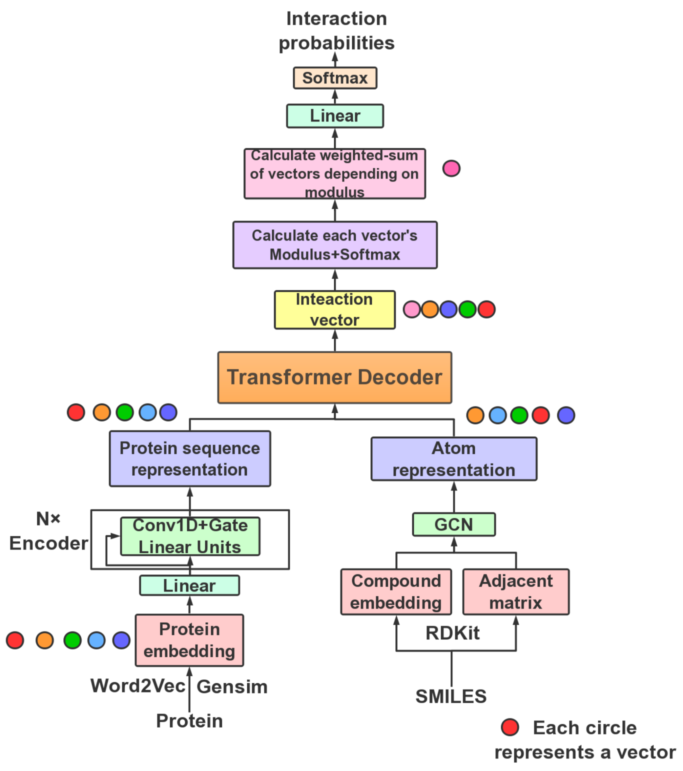

Figure 2: Architecture of the implemented model [Chen et al., 2020].

The embedding of the protein goes through an encoder of 1D-CNN and Gate Linear Unit (GLU). This encoder is applied three times and then the protein sequence representation is obtained. For the atom representations, the compound in SMILES format and the adjacent matrix are used as input of a graph convolutional network (GCN) and the atom representation is obtained. The GCN applies a graph for each compound, where the atoms in each compound are the vertices, represented by the features described in table 1, and the edges are the covalent bonds in the form of adjacency matrix.

Both atom and protein sequence are used as input to the transformer decoder. The transformer decoder mainly consists of eight heads and two self-attention layers. A clear image of the self attention layer is seen in figure 3, where the relationship between query, key and vector is shown.

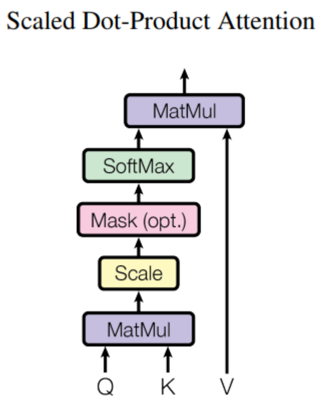

Figure 3: Self-attention layer from [Vaswani et al., 2017].

In the first self-attention layer, the query, key and value tensors are the atom representation. Then, after the first attention layer there is a second attention layer. In the second self-attention layer the query tensor is the atom representation, but the key and value tensors are the protein sequence representation. The attention tensor is calculated thanks to the query, key and value tensors. The query and key are used to compute the weights by which the value is scaled. The expression applied is:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_K}})V$$

where d_k is the dimension of the query and the key.

In the experimental part a variation is built in which there are two more attention layers with the query tensor as the atom representation and the key and value tensors as the protein representation. In this variation there are a total of 4 self-attention layers.

The multihead-attention plays a relevant role in the sense that it allows for paralellization by projecting several times the query, key and value and apply the attention mechanism to each projection. The expresion of multihead-attention is

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O$$

where $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$. The projections are $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$. d_{model} is the dimension of the tokens. A visualization is shown in figure 4. The original model has 8 heads, but in the experimental part there is a model with only one head. A sketch of the transformer decoder is shown in figure 5.

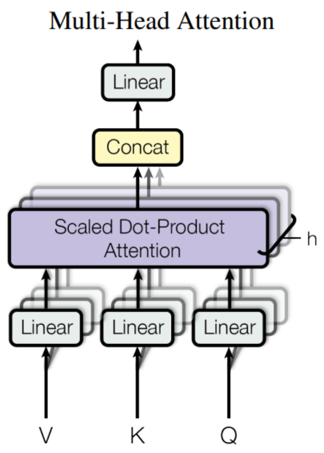

Figure 4: MultiHead-attention layer from [Vaswani et al., 2017].

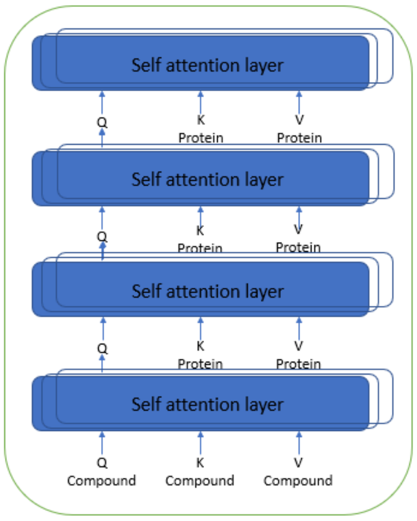

Figure 5: Sketch of the transformer decoder of the model.

The output of the transformer fed to a positional feed forward layer of dimension 512 and then the modulus of the resulting interaction vector is calculated, followed by a softmax layer. Finally, the weight sum of vectors is calculated by using as weights the values obtained from the softmax layer. This is followed by a linear and softmax layer, resulting in the probability that a certain compound and protein interact.

The loss function is binary cross entropy, and the optimizers are a combination of rectified Adam (RAdam) [Liu et al., 2019] and LookAhead [Zhang et al., 2019]. RAdam is a version of Adam where the variance of the adaptive learning rate is rectified thus avoiding convergence problems. At the beginning of the training there is a large variance due to the number of training samples used, and RAdam uses smaller learning rates at the beginning of the training. On the other side, LookAhead optimizer uses two sort of weights, slow and fast. The search direction the algorithm chooses is established by looking ahead the sequence of fast weights produced by another algorithm.

3 Results

One goal was to find out how sensitive the model is to certain hyperparameters. Different settings were investigated. The main hyperparameters that were tested were the learning rate together with L_2 regularization and dropout.

Fine tuning was done with 5-fold cross validation. Given that the original dataset is computationally too expensive to train, a subset consisting of 10% of the dataset was used. The total number of samples used for training was 1042, whereas the validation and test set consisted of 260 samples. The results of the validation set had a high variance when performing 5-fold cross validation, therefore, since part of the dataset was still available, a larger subset of samples was chosen, 2000 samples in total for validation and test. In this way more consistent results were obtained.

To evaluate the model, the evaluation metric chosen by the authors of [Chen et al., 2020] was Area Under Receiver Operating Characteristic Curve (AUC) and Area Under Precision Recall Curve (PRC), but the authors mostly focus on AUC to compare results. Consequently, I used AUC as a metric to compare results.

3.1 Hyperparameter tuning

The initial hyperparameter configuration that the authors used is shown in table 2. As mentioned before, the optimizer used was RAdam in combination with LookAhead, with an initial learning rate of 1e-4. The AUC for the training and validation data is shown in figure 6 and, for a total number of 20 epochs, the average AUC for the test set was 0.6954. The average AUC of the train set was 0.8431, which leads to the conclusion that there's a gap between train and validation set, probably due to bad generalization capabilities of the model and overfitting. The model was also trained with a batch size 64 and a learning rate 0.001, but the results of the model were worse. Different learning rate settings were applied, but the performance of the model did not greatly vary. For a learning rate decay of 0.9 every epoch after the epoch number 6, the test AUC was 0.7010, which is very similar to the previous configuration. Results are shown in figure 7.

In order to investigate the problem of generalization, different values for regularization and dropout were tested. The initial values were 1e-4 for regularization and 0.2 for dropout. The best results were given by a regularization of 0.01 and dropout 0.01. Although the test AUC improved up to 0.7172, the drawback of this combination is that the regularization term is higher, dropout is smaller and the gap between train and validation thus is bigger. The AUC of the training set was 0.9167. The results are shown in figure 8. After all the trainings were done, a new training was conducted, with a regularization of 0.01 and dropout 0.025 to tackle overfitting. The test AUC was 0.7174 and the training AUC was reduced to 0.8442, leading to a smaller gap between train and validation, shown in figure 9, but still noticeable.

Hyperparameter	**Value**
Number of encoder layers	3
Number of decoder layers	3
Dimension of atom representation	64
Number of attention heads	8
FFN inner hidden size	512
Hidden size	64
Patch size	7
Learning rate	1e-4
Weight decay	1e-4
Batch size	8
Dropout	0.2

Table 2: Initial setting of the model.

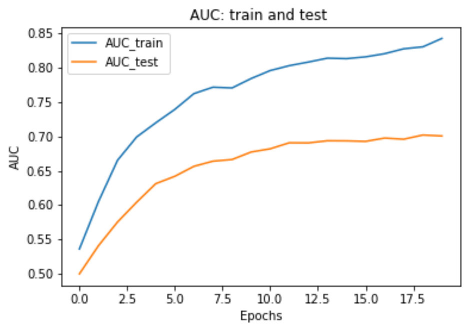

Figure 6: Initial configuration.

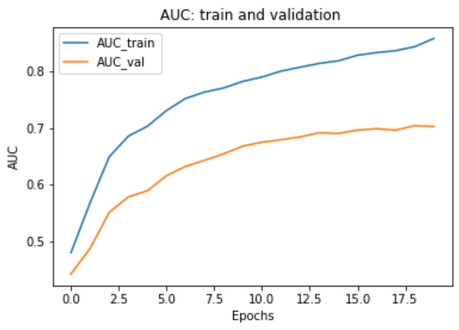

Figure 7: Learning rate decay 0.9.

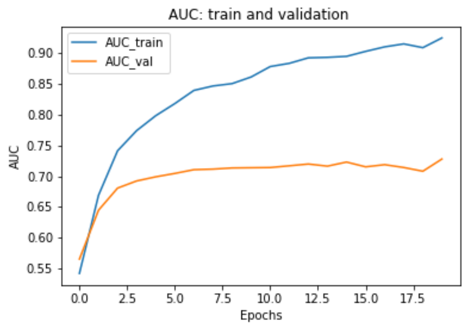

Figure 8: Dropout 0.01, L_2 regularization 0.01.

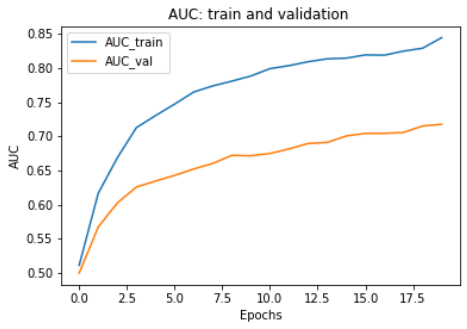

Figure 9: Dropout 0.25, L_2 regularization 0.01.

Figure 10: AUC for the train and validation sets.

3.2 Model architecture

Several modifications were done to the model architecture, mainly the transformer. One modification was the number of heads; originally the number of heads was 8, but the model was also trained with one head. Besides, the number of self-attention layers was modified. Originally there were two self-attention layers,and the change implemented was that the last self-attention layer was repeated three times, which makes a total of 4 self-attention layers.

For the configuration with one head and two self-attention layers, the results were very similar. As mentioned, for 8 heads and two self-attention layers the AUC was 0.7172 and for 1 head and two self-attention layers the obtained AUC was 0.7095. The plots of the train and validation AUCs are shown in figures 11, 12.

The best results were obtained for 8 heads and 4 self-attention layer, but the improvement was barely noticeable, the AUC was 0.7180. The results are shown in 13. In all cases the train AUC was very similar, and the PRC followed the same trend as the AUC.

As the results show, the model was not too sensitive to changes in the hyperparameters, and the only case where the AUC improved was at the expense of generalization capabilities, given that the model had a high AUC in train test but then in the validation test the AUC was about 0.2 lower. When it comes to transformer architecture, the results did not improve greatly. Statistical tests introduced in the next section were applied to confirm the hypothesis of some models performing better than others.

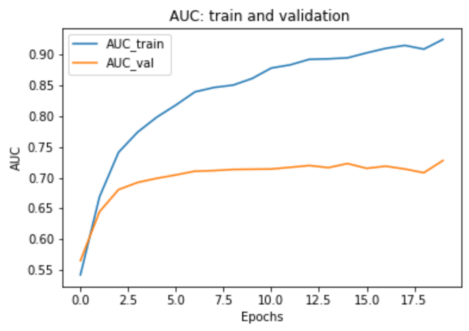

Figure 11: 8 heads and 2 self-attention layers.

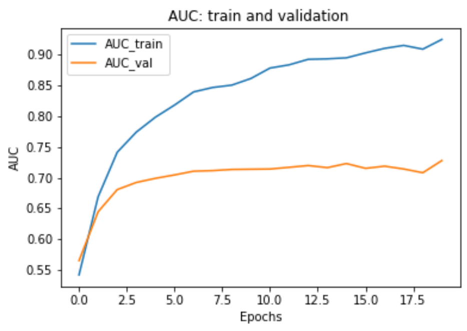

Figure 12: 1 head and 2 self-attention layers.

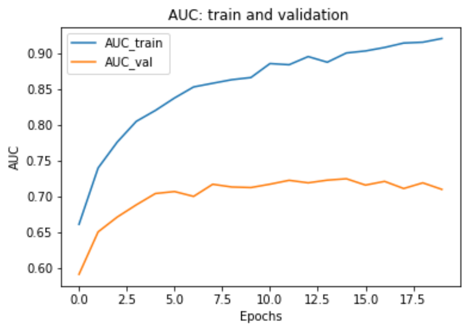

Figure 13: 8 heads and 4 self-attention layers.

Figure 14: AUC for the train and validation sets.

3.3 Statistical Test

The AUC of the following models was compared:

1. Initial configuration.
2. Learning rate decay.
3. Different regularization and dropout.
4. One head and two self-attention layers.
5. Eight heads and four self-attention layers.

To evaluate if the AUC resulting of doing 5-fold cross validation for each model was significantly different, a non-parametric statistical hypothesis test was applied. Since there were more than two models, the Friedman test was performed. The goal of this test is to know if the AUCs of all the models are drawn from the same distribution. For this test, real data is converted to ranked data and then the test is performed. For a significance level of 0.05, the p-value obtained was 0.003, which means that the AUCs come from different distributions.

The Wilcoxon signed-rank test can be applied to compare paired data samples. For models 1 and 3 the p-value was 0.043, meaning that AUCs came from different distributions, i.e. the third model performed better than the first model. Nonetheless, the p-value of models 1 and 2 was greater than 0.05, and it can be confirmed that the difference in AUC was not significant and they were not drawn from the same distribution.

For the models related to the transformer architecture, all the pair-wise comparisons returned a p-value greater than 0.05, except for the comparison of model 4 and 5, in which the p-value was 0.043. Thus, the model with 8 heads and 5 self-attention layers performed better than the model with one head and 2 self-attention layers.

4 Conclusions

To conclude, there are two points to analyse, how sensitive the model is with respect to its key hyperparameters as well as the transformer architecture.

For the hyperparameters, the decay of the learning rate did not change the performance of the model, as shown by the statistical test. On the other side, the change of parameters that play an important role in generalization and overfitting was statistically significant. The problem with this configuration is that even leading to better results, the generalization of the model was worse. However, with a more powerful device, further search could have been done. I tried for other set of parameters but probably dropout and L_2 regularization values were too high and it affected the final results. I believe that there should be a setting were generalization can be improved by not affecting the AUC of the model. Having said that, the change in AUC of the model was slightly better, with no great improvements and therefore it can be said that the model is robust.

The robustness of the model is also verified with the models that differ in the transformer architecture. The number of heads was not relevant for the AUC in the test set, but the number of attention layers did have an impact. Moreover, the model performed better with 8 heads and four attention layers than with 1 head and two attention layers. In any case, the model can be said to be robust with respect to the transformer architecture given that the change in AUC was smaller than 0.01.

References

Lifan Chen, Xiaoqin Tan, Dingyan Wang, Feisheng Zhong, Xiaohong Liu, Tianbiao Yang, Xiaomin Luo, Kaixian Chen, Hualiang Jiang, and Mingyue Zheng. Transformercpi: improving compound–protein interaction prediction by sequence-based deep learning with self-attention mechanism and label reversal experiments. *Bioinformatics*, 36(16): 4406–4414, 2020.

Ermal Elbasani, Soualihou Ngnamsie Njimbouom, Tae-Jin Oh, Eung-Hee Kim, Hyun Lee, and Jeong-Dong Kim. Gcrnn: graph convolutional recurrent neural network for compound–protein interaction prediction. *BMC bioinformatics*, 22(5):1–14, 2021.

Mostafa Karimi, Di Wu, Zhangyang Wang, and Yang Shen. Deepaffinity: interpretable deep learning of compound–protein affinity through unified recurrent and convolutional neural networks. *Bioinformatics*, 35(18):3329–3338, 2019.

Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. *arXiv preprint arXiv:1908.03265*, 2019.

Patrick Riley. Three pitfalls to avoid in machine learning, 2019.

Masashi Tsubaki, Kentaro Tomii, and Jun Sese. Compound–protein interaction prediction with end-to-end learning of neural networks for graphs and sequences. *Bioinformatics*, 35(2):309–318, 2019.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Michael Zhang, James Lucas, Jimmy Ba, and Geoffrey E Hinton. Lookahead optimizer: k steps forward, 1 step back. *Advances in Neural Information Processing Systems*, 32, 2019.