MODEL SUMMARY

Team background

Competition Name: CHAMPS (CHemistry And Mathematics in Phase Space)

Team Name: hybrid

Private Leaderboard Score: -3.23968

Private Leaderboard Place: #1

Name: Zico Kolter

Location: Pittsburgh PA

Email: Zico.Kolter@us.bosch.com

Name: Shaojie Bai

Location: Pittsburgh PA

Email: shaojieb@andrew.cmu.edu

Name: Devin Wilmott

Location: Pittsburgh PA

Email: Devin.Willmott@us.bosch.com

Name: Mordechai Kornbluth

Location: Cambridge MA

Email: [mckornbluth@gmail.com](mailto:mckornbluth@gmail.com)

Name: Jonathan Mailoa

Location: Cambridge MA

Email: [jpmailoa@gmail.com](mailto:jpmailoa@gmail.com)

Jonathan Mailoa and Mordechai Kornbluth are both Research Engineers working out of the Boston lab of Bosch Research. They are domain experts on DFT and ML approach to molecular simulation, and have worked a great deal on molecular modeling, including lately some work with GNNs.

Zico Kolter is a faculty member working in machine learning at CMU, but works in industry one day a week at the Bosch Center for Artificial Intelligence (BCAI) in Pittsburgh. Devin Willmott is a Research Scientist at BCAI, and Shaojie Bai is a graduate student at CMU, but did this while interning at BCAI. They were all coming to the competition from the ML side. Zico had actually done a bit of (pre-deep-learning, so ancient history) work in ML for molecular modeling, though we didn't end up using many of those methods.

One of the goals of the BCAI is to demonstrate world-class ML capabilities in conjunction with applications that are of substantial interest to Bosch and the scientific community.  These goals aligned well for this competition, and we jointly decided to enter. Team members were then chosen from interested parties within Bosch based upon their experience with ML and the domain.

We started participating “full time” with 4 weeks and one day remaining in the competition.  During this time, Shaojie and Devin spent approximately 20 hours/week on the competition. Jonathan and Mordechai spent on the order of 15 hours/week (amortized over the 4 weeks), and Zico spend 5 hours/week.

We have an existing collaboration (within Bosch) on ML-enhanced atomistic CMS modeling, and the competition’s theme perfectly fitted our team’s ML and domain expertise. Many of us spent about 50-70% of working hours starting from mid July until the end of August.

Jonathan and Mordechai preprocessed the molecule data and engineered features. Devin and Jonathan built the initial graph neural network (GNN) model. Shaojie and Zico built the final graph transformer (GT) model. Shaojie, Devin, and Jonathan trained the models. Shaojie fine-tuned the best GT models. Jonathan ensembled the models. Everyone debugged the code. Mordechai, Jonathan, Shaojie, and Devin prepared the code for the final submission.

Summary

We used a graph transformer neural network, with a meta-graph architecture. In contrast to most graph methods for molecules, where atoms are nodes and bonds are edges, in our graph each atom, bond (both chemical bonds, and non-chemical bonds, i.e., just pairs of atoms are included in the model), and even triplets or quads, if desired, all become nodes for the graph transformer to perform self-attention on. Following the standard transformer architectures, at each layer of the network, we use multi-head self-attention that mixes the embeddings between the nodes, with an attentional bias based on the Cartesian distance between the nodes. Key input features included embeddings of the atomic element(s)+chemical bond environment, partial charges and distances (which are represented as sinusoidal embeddings), among others. Libraries employed include: RDKit (with xyz2mol interface), OpenBabel (with pybel interface), PyTorch, Nvidia APEX (for distributed GPU training). The training takes 3-5 days for each model; the final submission was an ensembled combination of 13 models.

Features Selection / Engineering

Atomic features: Each atom type and its immediate environment (number of neighbors and bond orders), as well as partial charge and the angle of its two closest atoms.

Bond features: Atom types, distance, bond order, including chemical bonds and non-chemical bonds. We found that ignoring non-chemical bonds led to worse performance due to the neglect of hydrogen bonds.

Triplet features: Atom types of each member of the triplet, angle.

Features are selected by domain experts based on ease of extraction using open source libraries / manual coding, and then filtered out by ML experts based on how much they helped / harmed accuracy performance on the smaller scale models.

In our implementation, we enforced rotation invariance by taking only relative distances and angles for the geometric features. Atom permutation invariance is enforced by the usage of graph transformer architecture.

A one-hot-encoding maps some bond features onto the atom features: For example, instead of merely a 1JHC type, the subtypes are distinguished based on the number of neighbors and the chemical bond order. The motivation here was to capture much of the bond hybridization within the atom encoding.

We found that while including atomic partial charges in the atomic features (which adds some form of electrostatic interaction into the model) helped the model performance. However, directly encoding electrostatic interactions such as q1\*q2/r2 in the pairwise features harmed the model performance. Adding a Coulomb/multipole encoding to each bond 1/rn (n=1,...,4) was expected to help performance due to the physical equations, but turned out to not help.

We did not use any external data.

Training Method(s)

We initially trained our models using absolute loss instead of the custom CHAMPS average log MAE loss. Cosine anneal learning rate scheduler was used, and the PyTorch model training generally lasted for 200 epochs. The learning speed was strongly dependent on the learning rate scheduler that we used. For some of our best models, we fine-tune the model by additional training for approximately 40 epochs using CHAMPS average log MAE loss and cosine anneal learning rate scheduler to improve their single model scores by around -0.020 to -0.030. For each coupling subtype (including the element and chemical environment information), the targets were scaled to the mean and standard deviation, simplifying the learning process.

At the end of the competition we ensembled 13 models. The method outlined below enabled us to train all 13 of our models on the full training dataset, and do model selection based fully on the test dataset without having to worry about the varying degree of each model’s overfitting of the training set data.

We ensembled 13 graph transformer (GT) models. We employed a model selection algorithm using these 13 models to pre-select which 9 best models can participate in the final ensemble for each coupling type. We do this selection by looking at the scalar coupling prediction of each model for each coupling type for the test set, and determine how many times a model’s prediction is the median of all 13 models for a given type. The 9 best models with highest median count for a given type is selected as the final ensemble model for that type, while the remaining 4 are eliminated and do not participate in the final ensemble.

The 9 best models for each type are then used for final ensemble. For each scalar coupling entry, 9 values are predicted. We only take the center 5 median values out of these 9 predictions (the minimum and maximum 2 values are considered too noisy and eliminated). These 5 median values are then simply averaged to generate the final prediction. We found that for a limited number of models with differing accuracy / noise level, this method works better than straight mean or median of all / best few model predictions. This double elimination method generally gives us a score boost of -0.170 to -0.180 over our best single model.

We have hard-coded the 13 pre-trained models to be used in the ensembling script through the ‘model\_mask’ boolean array, in the script predictor.py.

Interesting findings

Our approach used the following four techniques that set us apart from the competition; some of these techniques were used by some competitors as well:

1. Transformer architecture: We tested both geometric models (using pytorch geometric) and transformer architectures, and found that the self-attention based transformer architectures surpassed the geometric models, to the extent that the final blending did not include any geometric models. Our transformer architecture modeled each molecule as a unit, rather than each atom.
2. Atom/bond/triplet meta-graph approach: The state of a molecule was described by the atomic, bond, triplet, and in some versions the quadruplets. The bonds included all pairs of atoms, including those that did not correspond to chemical bonds or magnetic couplings. Thus our final model was a meta-graph of the molecule, encoded as a single state of the transformer. The attention of the transformer used distances (or generalized distances) between each of these units.
3. Encoding by bond order subtypes: Preprocessing using the pybel and rdkit libraries enabled strong differentiation between subtypes, generalizing e.g. 1JHC into those corresponding to sp2, sp3, and sp4 hybridization. The subtypes are passed into the transformers via hierarchical embeddings (projected to the same dimension).
4. Blending techniques: Using specialized blending techniques, separate for each coupling type and identifying the median models, allowed us to reduce noise in the final predictions.

Simple Features and Methods

During the code design process, we found the biggest gains from the following:

* Features: Partial charges, embedding the element and bond subtypes (including bond order), physical distance, including all pairs even those without a chemical bond.
* Methods: Target scaling for each subtype was helpful. For model simplicity, some methods that could be dropped include: Fine-tuning to the CHAMPS loss (instead of the overall loss); ensembling various models.

Model Execution Time

Each model takes 3-5 days to train, depending on the available computational resources. The model prediction is relatively fast, usually ~ 15 minutes from a saved model (on the submission set), depending on computational resources and the size of the model.

References

Kaggle forum post describing the model: <https://www.kaggle.com/c/champs-scalar-coupling/discussion/106575>