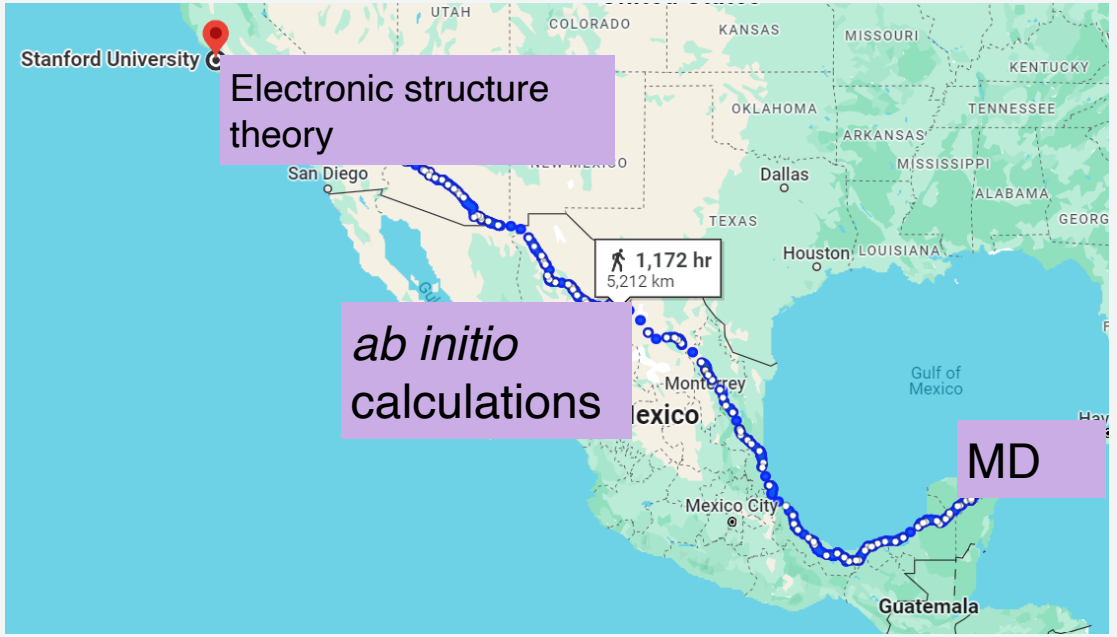
Using Neural Networks (Schnet) for Quantum MD Simulations

Grace Tully’s Rotation Project in
Rotskoff Lab with Clay Baton

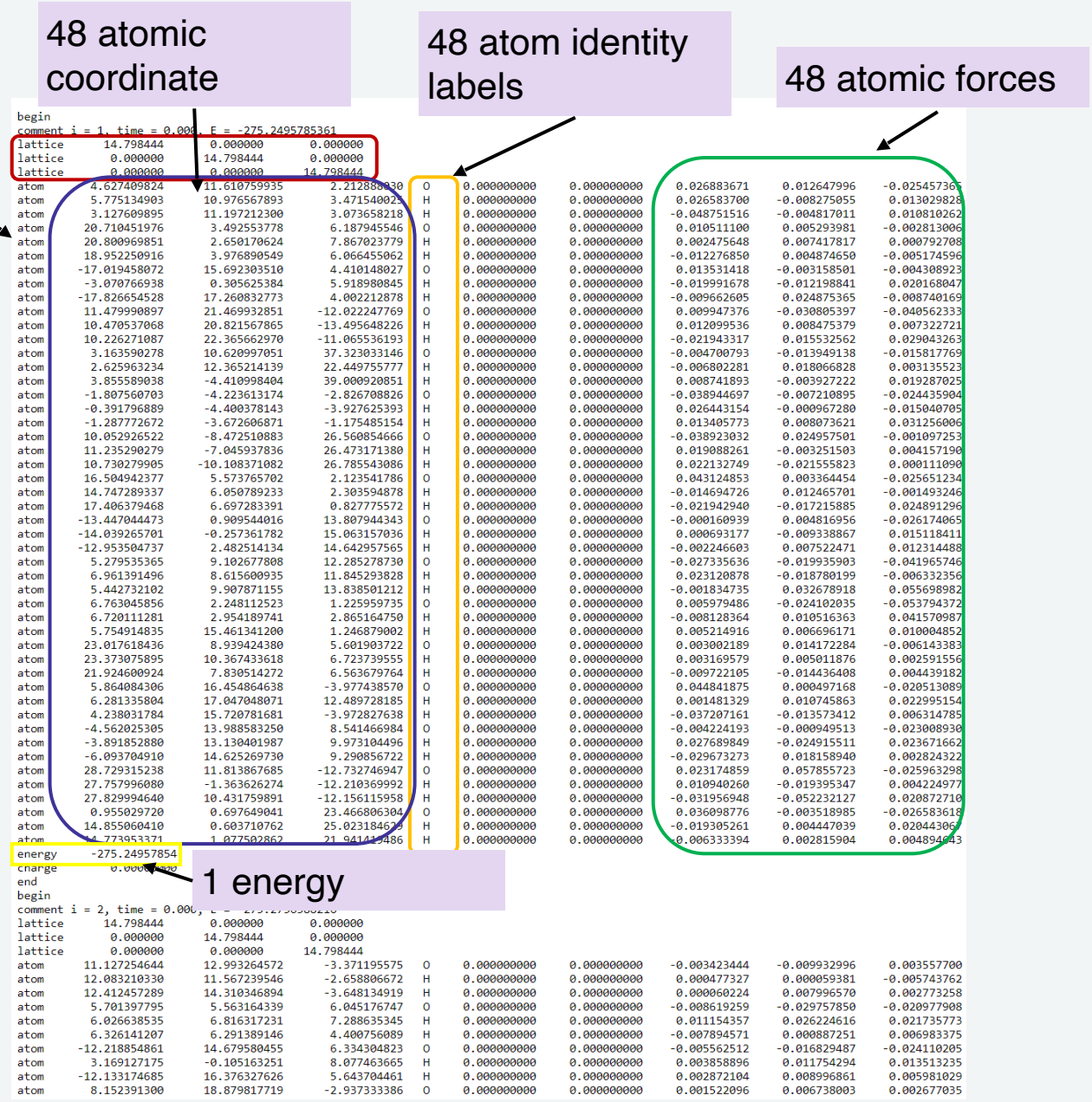
Purpose: faster electronic structure theory molecular dynamic simulations

- Ab initio molecular dynamics (AIMD) simulations remain too computationally expensive for modelling chemical systems
- Chen (Markland Group) used 8 different Behler–Parrinello neural network potentials to train system of 16 water molecules with PBC applied using a transfer learning scheme with DFT and higher order electronic structure theory methods (CCSD, CCSD(T), and AFQMC) convergence with AIMD simulations
- Schnet (Schrodiner Network) – used to train QM9 dataset of organic molecules
- Essentially, starting to reproduce Chen's learning procedure with Schnet instead of Behler-Parrinello MLPs.



The Data

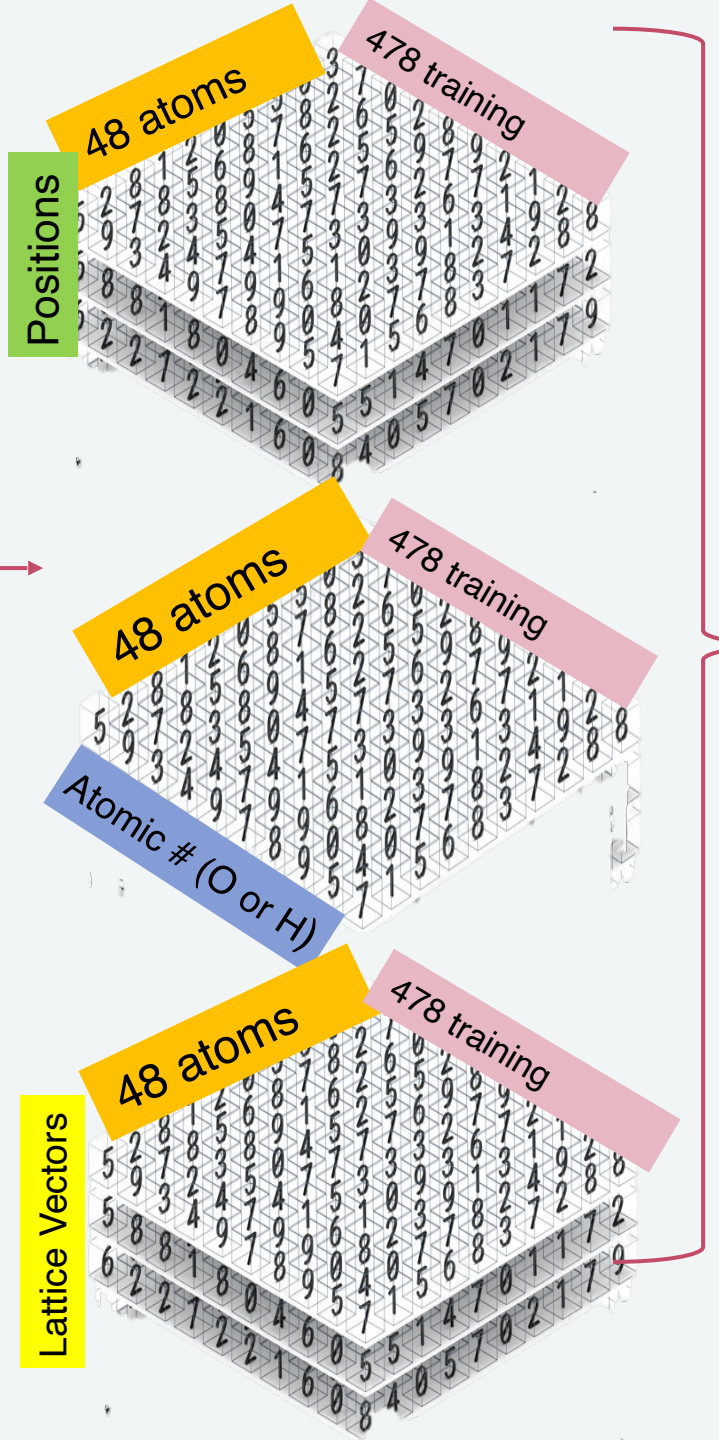
- B3LYP data files were already parsed by Clay
- 531 energy calculations, each with 48 atoms in the system specified by its coordinates, atomic number, and (computed) force vectors
- Split the data into 90% training, 10% validation
- Python data file created by Clay includes a function that creates a parsed dataset: separate tensors with atomic positions, energies, forces, numbers, and the unit cell vectors.



The Data

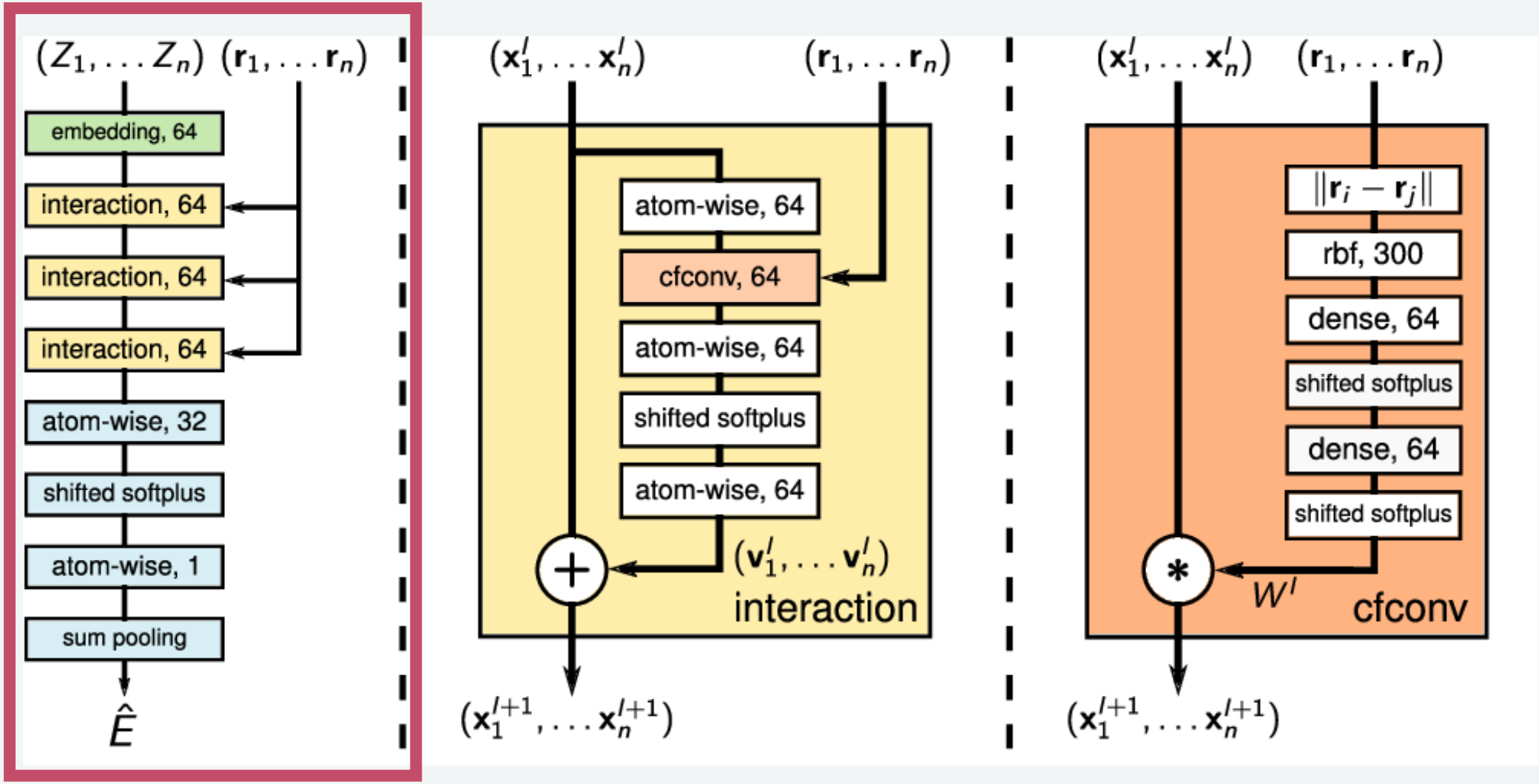
```
begin
comment i = 1, time = 0.000, E = -275.2495785361
lattice       14.798444        0.000000        0.000000
lattice        0.000000       14.798444        0.000000
lattice        0.000000        0.000000       14.798444
atom       4.627409824      11.610759935       2.212888030   O    0.000000000    0.000000000    0.026883671    0.012647996   -0.025457305
atom       5.775134903      10.976567893       3.471540021   H    0.000000000    0.000000000    0.026583700   -0.008275055    0.013029828
atom       3.127609895      11.197212300       3.073658218   H    0.000000000    0.000000000   -0.048751516   -0.004817011    0.010810262
atom      20.710451976       3.492553778       6.187945546   O    0.000000000    0.000000000    0.010511100    0.005293981   -0.002813006
atom      20.800969851       2.650170624       7.867023779   H    0.000000000    0.000000000    0.002475648    0.007417817    0.000792708
atom      18.952250916       3.976890549       6.066455062   H    0.000000000    0.000000000   -0.012276850    0.004874650   -0.005174596
atom     -17.019458072      15.692303510       4.410148027   O    0.000000000    0.000000000    0.013531418   -0.003158501   -0.004308923
atom      -3.070766938       0.305625384       5.918980845   H    0.000000000    0.000000000   -0.013991678   -0.012198841    0.020168047
atom     -17.826654528      17.260832773       4.002212878   H    0.000000000    0.000000000   -0.009662605    0.024875365   -0.008740169
atom      11.479990897      21.469932851     -12.022247769   O    0.000000000    0.000000000    0.009947376   -0.030805397   -0.040562333
atom      10.470537068      20.821567865     -13.495648226   H    0.000000000    0.000000000    0.012099536    0.008475379    0.007322721
atom      10.226271087      22.365662970     -11.065536193   H    0.000000000    0.000000000   -0.021943317    0.015532562    0.029043263
atom       3.163590278      10.620997051      37.323033146   O    0.000000000    0.000000000   -0.004700793   -0.013949138   -0.015817769
atom       2.625963234      12.365214139      22.449755777   H    0.000000000    0.000000000   -0.006802281    0.018066828    0.003135523
atom       3.855589038      -4.410998404      39.000920851   H    0.000000000    0.000000000    0.008741893   -0.003927222    0.015287025
atom      -1.807560703      -4.223613174      -2.826708826   O    0.000000000    0.000000000   -0.038944697   -0.007210895   -0.024435904
atom      -0.391796889      -4.400378143      -3.927625393   H    0.000000000    0.000000000    0.026443154   -0.000967280   -0.015040705
atom      -1.287772672      -3.672606871      -1.175485154   H    0.000000000    0.000000000    0.013405773    0.008073621    0.031256006
atom      10.052926522      -8.472510883      26.560854666   O    0.000000000    0.000000000   -0.038923032    0.024957501   -0.001097253
atom      11.235290279      -7.045937836      26.473171380   H    0.000000000    0.000000000    0.019088261   -0.003251503    0.004157196
atom      10.730279905     -10.108371082      26.785543086   H    0.000000000    0.000000000    0.022132749   -0.021555823    0.000111096
atom      16.504942377       5.573765702       2.123541786   O    0.000000000    0.000000000    0.043124853    0.003364454   -0.025651234
atom      14.747289337       6.050789233       2.303594878   H    0.000000000    0.000000000   -0.014694726    0.012465701   -0.001493246
atom      17.406379468       6.697283391       0.827775572   H    0.000000000    0.000000000   -0.021942940   -0.017215885    0.024891296
atom     -13.447044473       0.909544016      13.807944343   O    0.000000000    0.000000000   -0.000160939    0.004816956   -0.026174065
atom     -14.039265701      -0.257361782      15.063157036   H    0.000000000    0.000000000    0.000693177   -0.009338867    0.015118411
atom     -12.953504737       2.482514134      14.642957565   H    0.000000000    0.000000000   -0.002246603    0.007522471    0.012314488
atom       5.279535365       9.102677808      12.285278730   O    0.000000000    0.000000000   -0.027335636   -0.019935903   -0.041965746
atom       6.961391496       8.615600935      11.845293828   H    0.000000000    0.000000000    0.023120878   -0.018780199   -0.006332356
atom       5.442732102       9.907871155      13.838501212   H    0.000000000    0.000000000   -0.001834735    0.032678918    0.055698982
atom       6.763045856       2.248112523       1.225959735   O    0.000000000    0.000000000    0.005979486   -0.024102035   -0.053794372
atom       6.720111281       2.954189741       2.865164750   H    0.000000000    0.000000000   -0.008128364    0.010516363    0.041570987
atom       5.754914835      15.461341200       1.246879002   H    0.000000000    0.000000000    0.005214916    0.006696171    0.010004852
atom      23.017618436       8.939424380       5.601903722   O    0.000000000    0.000000000    0.003002189    0.014172284   -0.006143383
atom      23.373075895      10.367433618       6.723739555   H    0.000000000    0.000000000    0.003169579    0.005011876    0.002591556
atom      21.924600924       7.830514272       6.563679764   H    0.000000000    0.000000000   -0.009722105   -0.014436408    0.004439182
atom       5.864084306      16.454864638      -3.977438570   O    0.000000000    0.000000000    0.044841875    0.000497168   -0.020513089
atom       6.281335804      17.047048071      12.489728185   H    0.000000000    0.000000000    0.001481329    0.010745863    0.022995154
atom       4.238031784      15.720781681      -3.972827638   H    0.000000000    0.000000000   -0.037207161   -0.013573412    0.006314785
atom      -4.562025305      13.988583250       8.541466984   O    0.000000000    0.000000000   -0.004224193   -0.000949513   -0.023008930
atom      -3.891852880      13.130401987       9.973104496   H    0.000000000    0.000000000    0.027689849   -0.024915511    0.023671662
atom      -6.093704910      14.625269730       9.290856722   H    0.000000000    0.000000000   -0.029673273    0.018158940    0.002824322
atom      28.729315238      11.813867685     -12.732746947   O    0.000000000    0.000000000    0.023174859    0.057855723   -0.025963298
atom      27.757996080      -1.363626274     -12.210369992   H    0.000000000    0.000000000    0.010940260   -0.019395347    0.004224977
atom      27.829994640      10.431759891     -12.156115958   H    0.000000000    0.000000000   -0.031956948   -0.052232127    0.020872716
atom       0.955029720       0.697649041      23.466806304   O    0.000000000    0.000000000    0.036098776   -0.003518985   -0.026583618
atom      14.855060410       0.603710762      25.023184629   H    0.000000000    0.000000000   -0.019305261    0.004447039    0.020443067
atom      21.773953371       1.077503862      21.941479486   H    0.000000000    0.000000000    0.006333394    0.002815904    0.004894443
energy      -275.2495785
charge       0.0000000
end
begin
comment i = 2, time = 0.000, E = -275.2756388216
lattice       14.798444        0.000000        0.000000
lattice        0.000000       14.798444        0.000000
lattice        0.000000        0.000000       14.798444
atom      11.127254644      12.993264572      -3.371195575   O    0.000000000    0.000000000   -0.003423444   -0.009932996    0.003557700
atom      12.083210330      11.567239546      -2.658806672   H    0.000000000    0.000000000    0.000477327    0.000059381   -0.005743762
atom      12.412457289      14.310346894      -3.648134919   H    0.000000000    0.000000000    0.000060224    0.007996570    0.002773258
atom       5.701397795       5.563164339       6.045176747   O    0.000000000    0.000000000   -0.008619259   -0.029757850   -0.020977908
atom       6.026638535       6.816317231       7.288635345   H    0.000000000    0.000000000    0.011154357    0.026224616    0.021735773
atom       6.326141207       6.291389146       4.400756089   H    0.000000000    0.000000000   -0.007894571    0.000887251    0.006983375
atom     -12.218854861      14.679580455       6.334304823   O    0.000000000    0.000000000   -0.005562512   -0.016829487   -0.024110205
atom       3.169127175      -0.105163251       8.077463665   H    0.000000000    0.000000000    0.003858896    0.011754294    0.013513235
atom     -12.133174685      16.376327626       5.643704461   H    0.000000000    0.000000000    0.002872104    0.008996861    0.005981029
atom       8.152391300      18.879817719      -2.937333386   O    0.000000000    0.000000000    0.001522096    0.006738003    0.002677035
```

Clay's Parsing Python Script

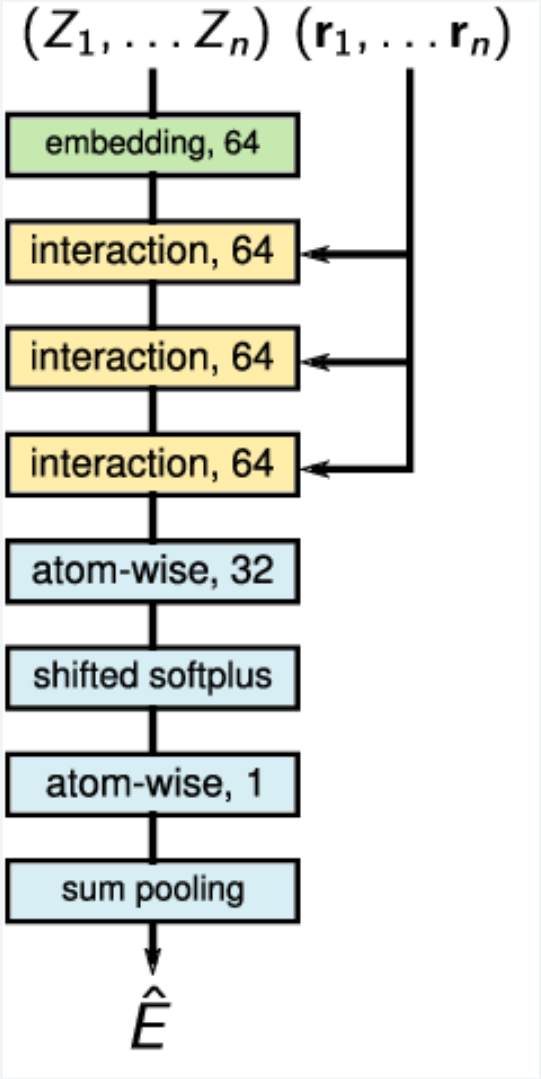


Pytorch Tensors

The Network

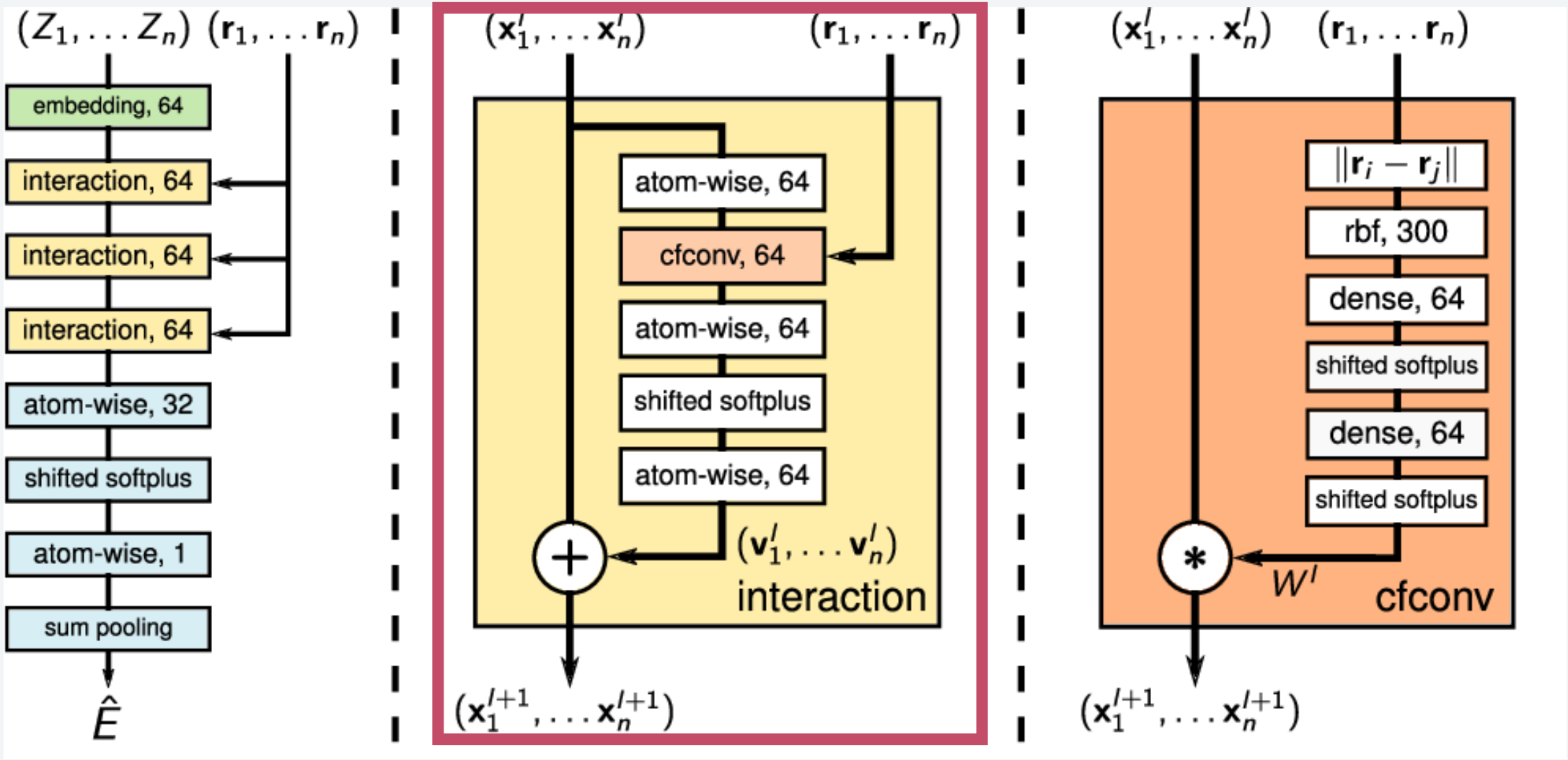

$(Z_1, \ldots Z_n)$ $(\mathbf{r}_1, \ldots \mathbf{r}_n)$
embedding, 64
interaction, 64
interaction, 64
interaction, 64
atom-wise, 32
shifted softplus
atom-wise, 1
sum pooling
$\hat{E}$
$(\mathbf{x}_1^l, \ldots \mathbf{x}_n^l)$
$(\mathbf{r}_1, \ldots \mathbf{r}_n)$
atom-wise, 64
cfconv, 64
atom-wise, 64
shifted softplus
atom-wise, 64
$(\mathbf{v}_1^l, \ldots \mathbf{v}_n^l)$
interaction
$(\mathbf{x}_1^{l+1}, \ldots \mathbf{x}_n^{l+1})$
$(\mathbf{x}_1^l, \ldots \mathbf{x}_n^l)$
$(\mathbf{r}_1, \ldots \mathbf{r}_n)$
$\|\mathbf{r}_i - \mathbf{r}_j\|$
rbf, 300
dense, 64
shifted softplus
dense, 64
shifted softplus
W^l
cfconv
$(\mathbf{x}_1^{l+1}, \ldots \mathbf{x}_n^{l+1})$

The Network

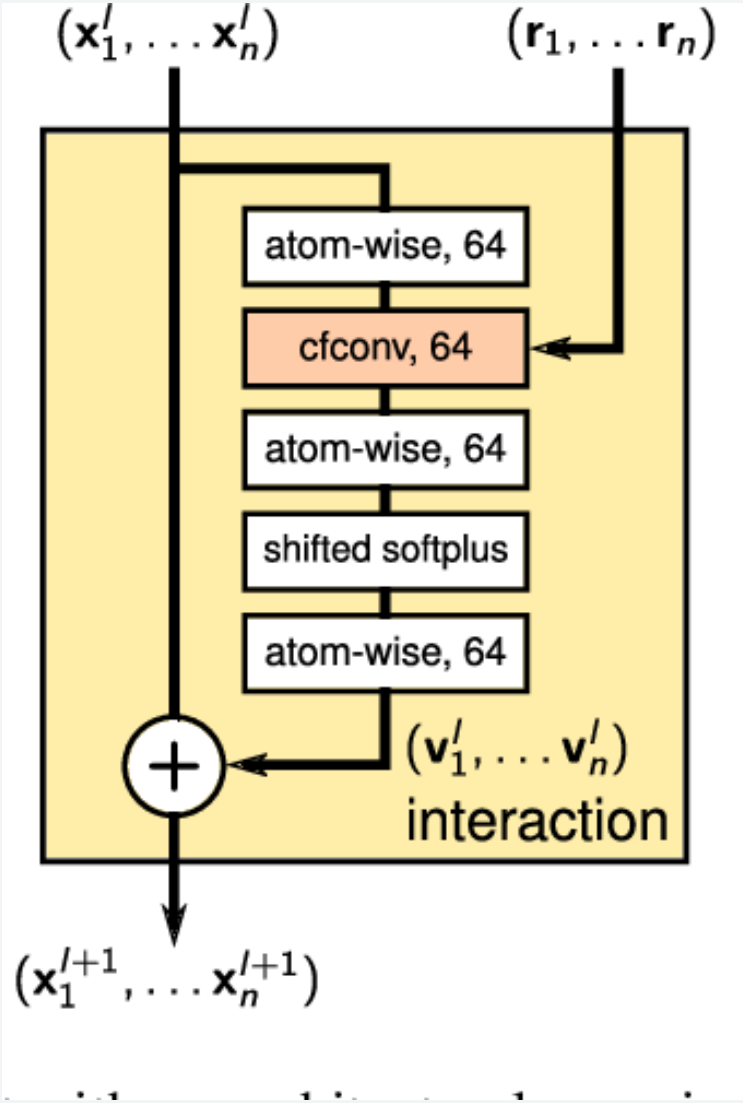



```python
class Schnet_Complete(nn.Module):
    def __init__(self):

        super().__init__()
        self.embedding = nn.Embedding(48, hidden_layers ,2)
        self.interactive_layer_1 = Schnet_I_1(hidden_layers, hidden_layers)
        self.interactive_layer_2 = Schnet_I_2(hidden_layers, hidden_layers)
        self.interactive_layer_3 = Schnet_I_3(hidden_layers, hidden_layers)
        self.linear_layers_stackedup = nn.Sequential(nn.Linear(in_features=hidden_layers, out_features=half_hidden_layers),
                                                     nn.Softplus(beta=2),
                                                     nn.Linear(in_features=half_hidden_layers, out_features=1))

    def forward(self, positions, n_configs, unit_cell_vectors, atomic_numbers, n_gaussians):
        embedding = self.embedding(atomic_numbers)
        interactive_layer_1 = self.interactive_layer_1(positions, n_configs, unit_cell_vectors, embedding, n_gaussians)
        interactive_layer_2 = self.interactive_layer_2(positions, n_configs, unit_cell_vectors, interactive_layer_1, n_gaussians)
        interactive_layer_3 = self.interactive_layer_3(positions, n_configs, unit_cell_vectors, interactive_layer_2, n_gaussians)
        final_stack = self.linear_layers_stackedup(interactive_layer_3)

        individual_atom_energies = torch.sum(final_stack, dim=2)
        # Sum along dim 1 to get a new tensor of shape [478, 1]
        pooled_energies = torch.sum(individual_atom_energies, dim=1)
        pooled_energies_reshaped = pooled_energies.unsqueeze(-1)
        pooled_energies_reshaped.requires_grad_()

        #forces computed using gradients

        forces = -torch.autograd.grad(
            pooled_energies_reshaped,
            positions,
            grad_outputs=torch.ones_like(pooled_energies_reshaped),
            create_graph=True)[0]

        return pooled_energies_reshaped, forces
```

The Network

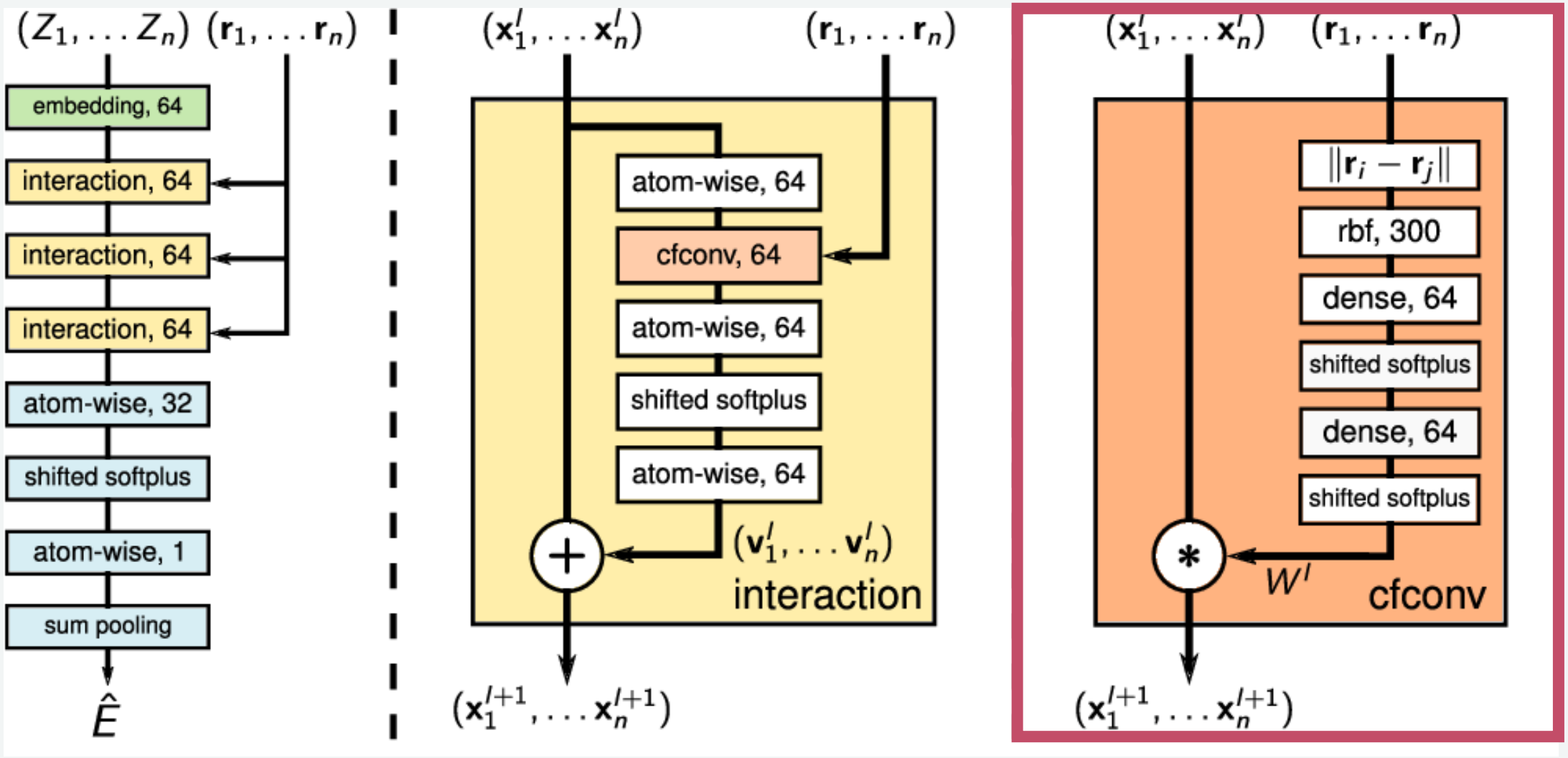

$(Z_1, \ldots Z_n)$
$(\mathbf{r}_1, \ldots \mathbf{r}_n)$
embedding, 64
interaction, 64
interaction, 64
interaction, 64
atom-wise, 32
shifted softplus
atom-wise, 1
sum pooling
$\hat{E}$
$(\mathbf{x}_1^l, \ldots \mathbf{x}_n^l)$
$(\mathbf{r}_1, \ldots \mathbf{r}_n)$
atom-wise, 64
cfconv, 64
atom-wise, 64
shifted softplus
atom-wise, 64
$(\mathbf{v}_1^l, \ldots \mathbf{v}_n^l)$
interaction
$(\mathbf{x}_1^{l+1}, \ldots \mathbf{x}_n^{l+1})$
$(\mathbf{x}_1^l, \ldots \mathbf{x}_n^l)$
$(\mathbf{r}_1, \ldots \mathbf{r}_n)$
$\|\mathbf{r}_i - \mathbf{r}_j\|$
rbf, 300
dense, 64
shifted softplus
dense, 64
shifted softplus
W^l
cfconv
$(\mathbf{x}_1^{l+1}, \ldots \mathbf{x}_n^{l+1})$

The Network –Interactive Blocks

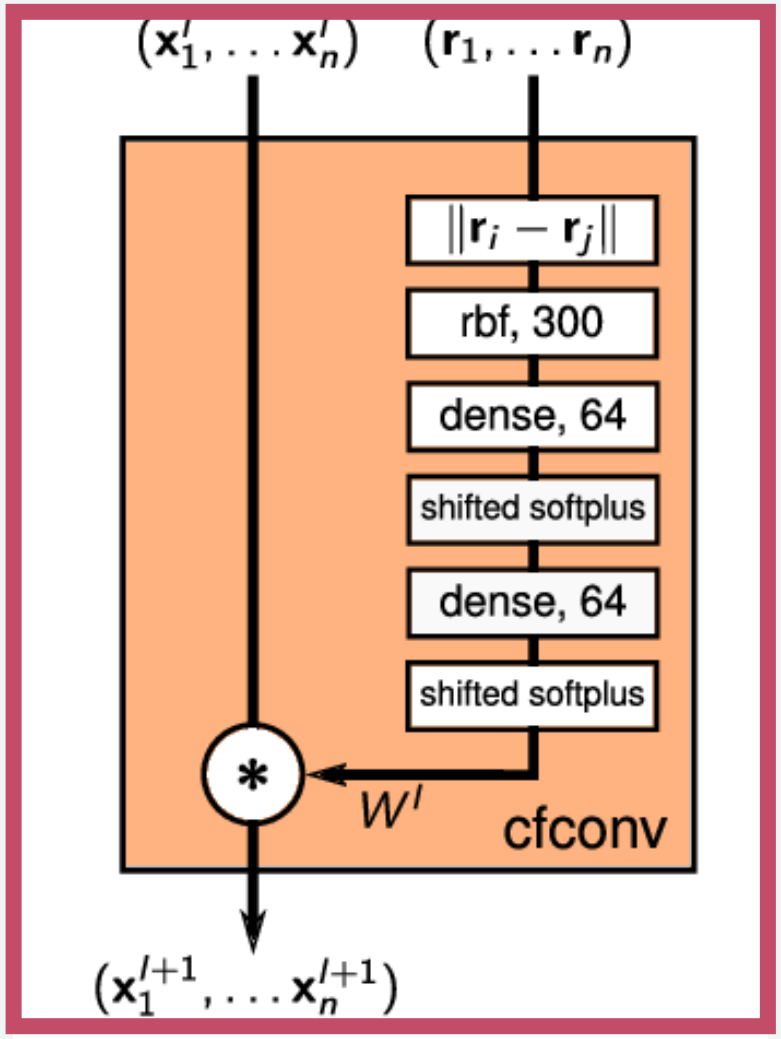



```python
class Schnet_I_1(nn.Module):
    def __init__(self, input_features, output_features, hidden_units=hidden_layers):
        '''
        Args:
            input_features (int): Number of input features to the model
            output_features (int): Number of outputs features (number of output classes)
            hidden_units (int): Number of hidden units between layers, default is hidden_layers
        '''
        super().__init__()
        self.linear_layer = nn.Linear(in_features= input_features, out_features=hidden_units)
        self.conv_layer = cconvl(n_gaussians = number_of_gaussians, output_features= hidden_layers)
        self.linear_layer_stacked = nn.Sequential(nn.Linear(in_features=hidden_units, out_features=hidden_units),
                                                  nn.Softplus(beta=2),
                                                  nn.Linear(in_features=hidden_units, out_features=output_features)
        )
    def forward(self, positions, n_configs, unit_cell_vectors, atom_wise_embedding, n_gaussians):
        atom_wise_embedding_update = self.linear_layer(atom_wise_embedding)
        cconvl_layer = self.conv_layer(positions, n_configs, unit_cell_vectors, atom_wise_embedding_update, n_gaussians)
        interactive_stack = self.linear_layer_stacked(cconvl_layer)
        return interactive_stack + atom_wise_embedding
```

The Network


$(Z_1, \ldots Z_n)$
$(\mathbf{r}_1, \ldots \mathbf{r}_n)$
embedding, 64
interaction, 64
interaction, 64
interaction, 64
atom-wise, 32
shifted softplus
atom-wise, 1
sum pooling
$\hat{E}$
$(\mathbf{x}_1^l, \ldots \mathbf{x}_n^l)$
$(\mathbf{r}_1, \ldots \mathbf{r}_n)$
atom-wise, 64
cfconv, 64
atom-wise, 64
shifted softplus
atom-wise, 64
$(\mathbf{v}_1^l, \ldots \mathbf{v}_n^l)$
interaction
$(\mathbf{x}_1^{l+1}, \ldots \mathbf{x}_n^{l+1})$
$(\mathbf{x}_1^l, \ldots \mathbf{x}_n^l)$
$(\mathbf{r}_1, \ldots \mathbf{r}_n)$
$\|\mathbf{r}_i - \mathbf{r}_j\|$
rbf, 300
dense, 64
shifted softplus
dense, 64
shifted softplus
W^l
cfconv
$(\mathbf{x}_1^{l+1}, \ldots \mathbf{x}_n^{l+1})$

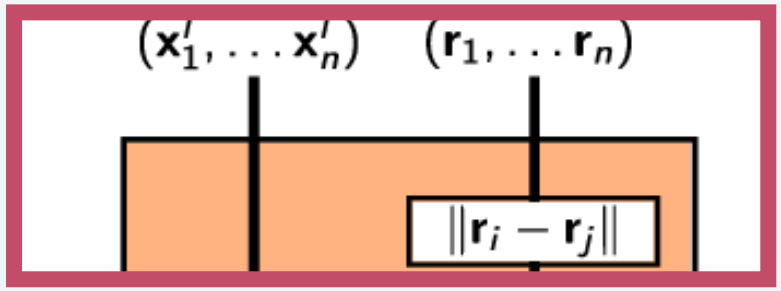
The Network – Convolutional Layer

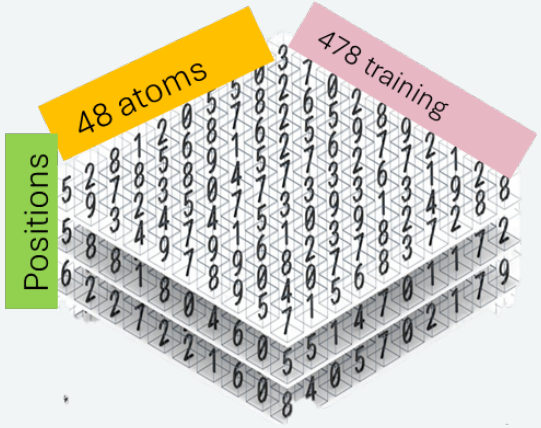



```python
class cconv1(nn.Module):
    def __init__(self, n_gaussians, output_features, hidden_units=hidden_layers):

        super().__init__()
        self.linear_layer_stack = nn.Sequential(nn.Linear(in_features= n_gaussians, out_features=hidden_units),
                                                nn.Softplus(beta=2),
                                                nn.Linear(in_features=hidden_units, out_features=hidden_units),
                                                nn.Softplus(beta=2),
                                                nn.Linear(in_features=hidden_units, out_features=output_features)
        )
    def forward(self, positions, n_configs, unit_cell_vectors, atom_wise_embedding, n_gaussians):
        displacement_vectors = positions.unsqueeze(-2) - positions.unsqueeze(-3)
        displacement_vectors = displacement_vectors.to(device)
        box_length = unit_cell_vectors.unsqueeze(-2).unsqueeze(-2)
        box_length.to(device)
        to_subtract = ((torch.abs(displacement_vectors).to(device) > 0.5 * box_length) * torch.sign(displacement_vectors).to(device)* box_length)
        to_subtract = to_subtract.to(device)
        displacement_vectors_with_PBC = displacement_vectors - to_subtract
        pairwise_dist = torch.linalg.norm(displacement_vectors_with_PBC, dim=-1) #calculates norm of the last dimension.
        pairwise_dist.to(device)
        non_zero_norm_mask = pairwise_dist != 0
        non_zero_norm_mask.to(device)
        pairwise_dist = pairwise_dist[non_zero_norm_mask]
        pairwise_dist = pairwise_dist.view(n_configs, 48, 47) #number of configs
        cutoff = 12 # centers within 12 angstroms
        centers = torch.linspace(0.0, cutoff, n_gaussians).to(device)
        gamma = 0.5 * (centers[1]- centers[0])**2
        dist_centered_squared = (pairwise_dist.unsqueeze(-1) - centers) ** 2
        expanded_pairwise_dist = torch.exp(gamma * dist_centered_squared).to(device)
        weights = self.linear_layer_stack(expanded_pairwise_dist) # shape of weight tensor = [478, 48, 47, 64]
        embedding_reshaped = atom_wise_embedding.unsqueeze(2)
        weighted_embedding = embedding_reshaped * weights
        return torch.sum(weighted_embedding, dim=2)
```

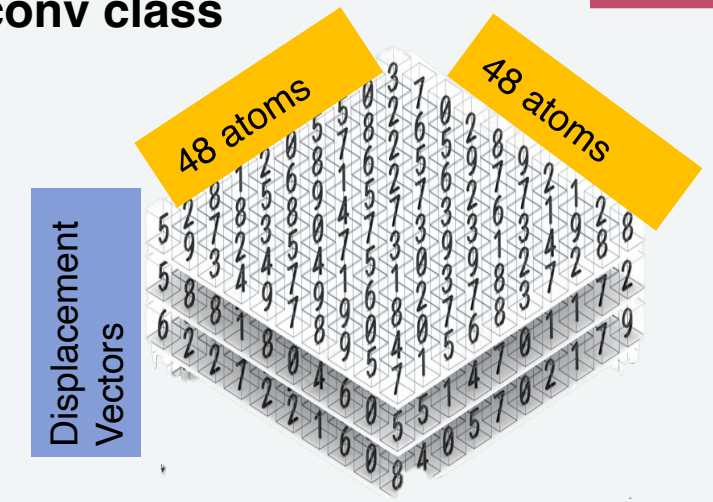
The Network – Convolutional Layer

1) Transforming Tensor of atomic coordinates into tensor that is instead rotationally invariant (replicating Chen's 16-water molecule simulation) (his system is periodic)
2) **Explain why this must be done within the conv class**





4D Tensor: Shape → [478, 48, 48, 3]



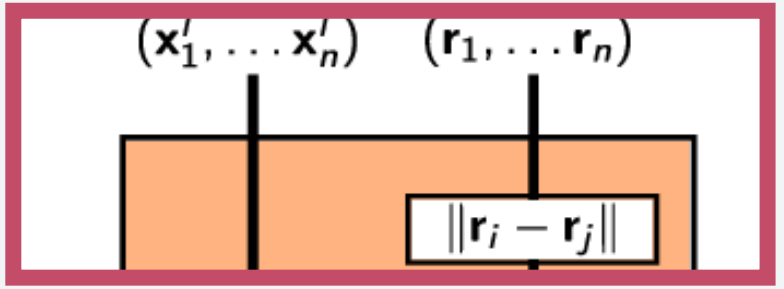
Think : 478 3D tensors of shape [48, 48, 3]

reshape the position tensor to two separate tensors: t1 =[478,48,1,3] (unsqueeze(-2)) and t2= [478,1,48,3] (unsqueeze(-3))

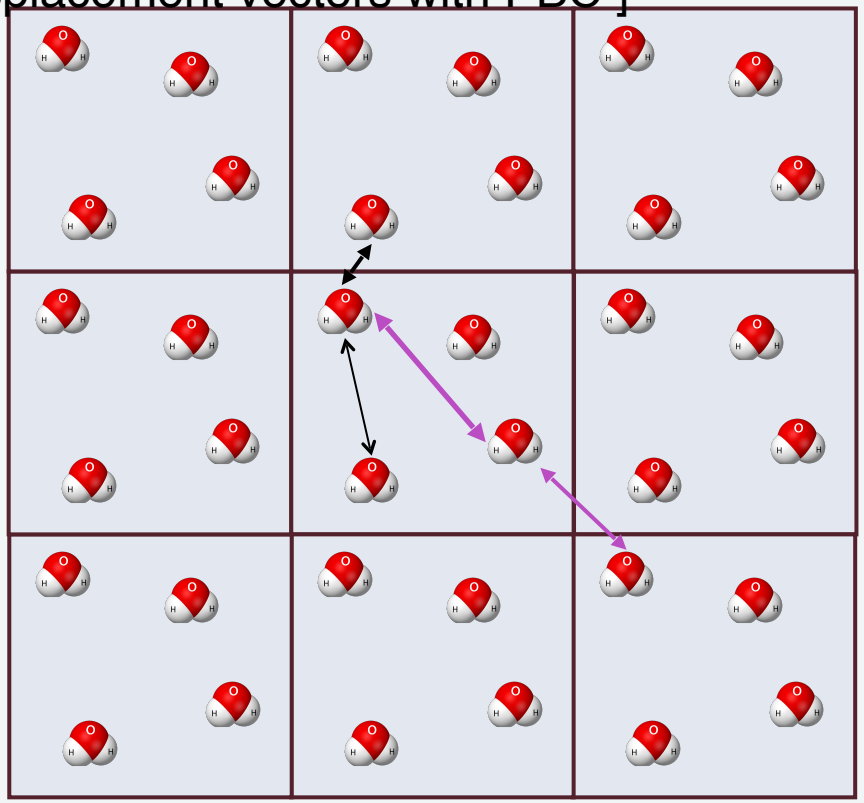
```
displacement_vectors = positions.unsqueeze(-2) -
positions.unsqueeze(-3)
displacement_vectors = displacement_vectors.to(device)
```

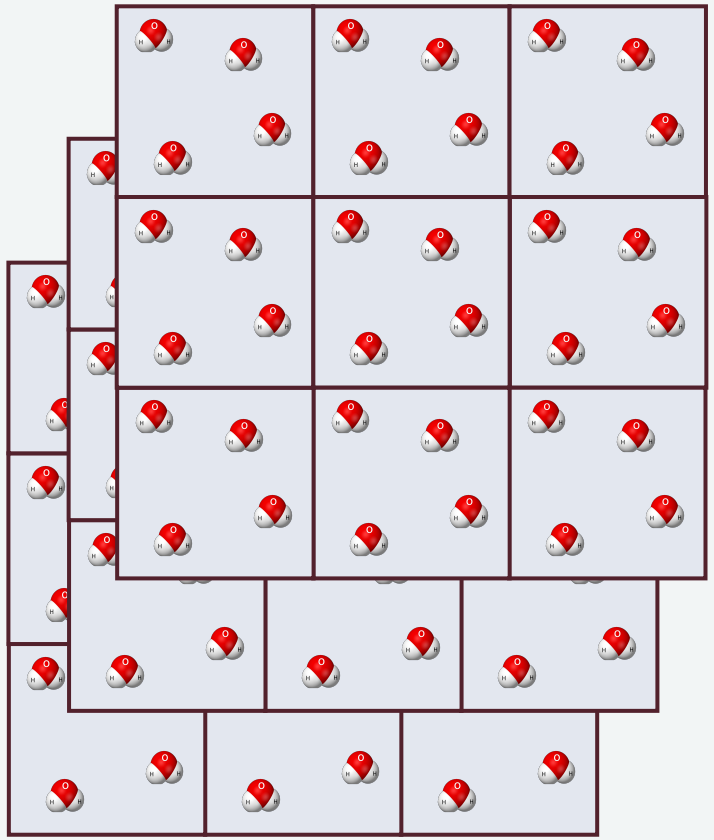
Not done! Must apply PBC!

The Network – Convolutional Layer

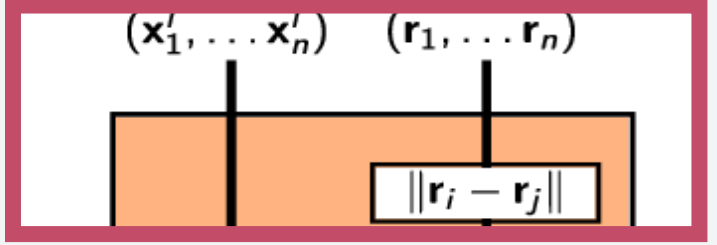


Periodic Boundary Conditions (PBC)
Tensor Transformation: [478, 48, 48, 3] → [478, 48, 48, 3]
General Tensor Transform: [#of calculations, # atoms, 3D displacement vectors no PBC]
→ [#of calculations, # atoms, 3D displacement vectors with PBC]





The Network – Convolutional Layer



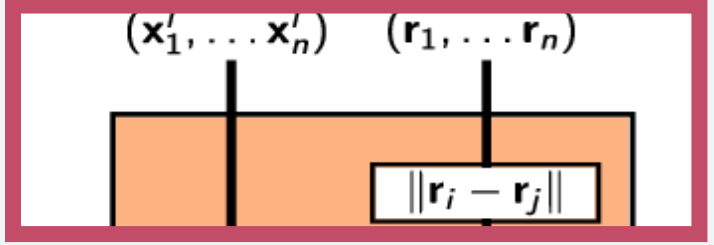
Periodic Boundary Conditions for pairwise distance calculation

Tensor Transformation: [478, 48, 48, 3] → [478, 48, 48, 3]
General Tensor Transform: [#of calculations, # atoms, 3D displacement vectors no PBC]
→ [#of calculations, # atoms, 3D

```
box_length = unit_cell_vectors.unsqueeze(-2).unsqueeze(-2)

to_subtract = ((torch.abs(displacement_vectors).to(device) > 0.5 * box_length) *
               torch.sign(displacement_vectors).to(device)* box_length)

displacement_vectors_with_PBC = displacement_vectors - to_subtract
```

- **box_length** is used to transform tensor of unit_cell_vectors [478, 3] to same size as displacement vectors [478, 1, 1, 3]
- **to_subtract** is used to transform displacement vectors to be just in octant 1 (all positive values) (torch.abs)
- Now test each dim (x,y,z) in positive displacement_vectors. If they are larger than 0.5 the box length vectors, then subtract or add the box length vectors

The Network – Convolutional Layer

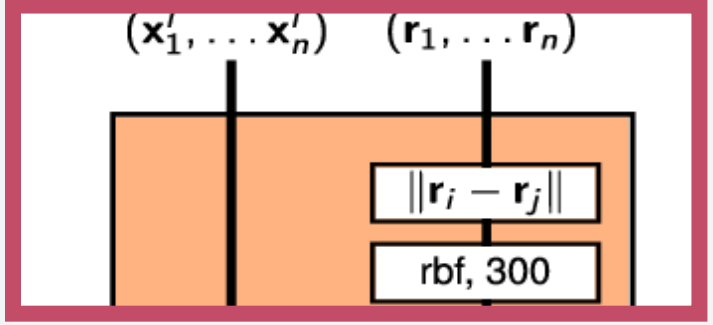


Tensor Transformation: [478, 48, 48, 3] → [478, 48, 47]
General Tensor Transform: [#of calculations, # atoms, 3D displacement vectors no PBC]
→ [#of calculations, # atoms, pairwise distances of every atom with every other atom]

- Now turn 3D vectors into distances by calculating Euclidean norm along last dimension (-1) Tensor shape is now: (478, 48, 48)
- We now want to remove norms of each atom with itself → create Boolean mask where norm does not equal zero, then apply mask to pairwise distance and reshape using "view" (applying mask turns 3D tensor into 1D tensor with every remaining element→ ex// for training tensor shape[1078369])

```
pairwise_dist = torch.linalg.norm(displacement_vectors_with_PBC, dim=-1)
non_zero_norm_mask = pairwise_dist != 0
non_zero_norm_mask.to(device)
pairwise_dist = pairwise_dist[non_zero_norm_mask]
pairwise_dist = pairwise_dist.view(n_configs, 48, 47)
```

The Network – Convolutional Layer



Tensor Transformation: [478, 48, 47] → [478, 48, 47, 300]
General Tensor Transform: [#of calculations, # atoms, pairwise distances of every atom with every other atom] → [#of calculations, # atoms, pairwise distances of every atom with every other atom, # rbfs (default = 300)]

- expand each pairwise distance into radial basis functions: $= e^{\left(\left|\left|-\gamma\left(d_{ij} - u_k\right)\right|\right|\right)^2}$
  - Introduce nonlinearity into network
- of centers = # of gaussians
- in Schnet paper, gamma is set to 10A.
- Traditionally with rbfs, gamma is set to 0.5(center distance)^2 (used traditional method)

```
cutoff = 12 # centers within 12 angstroms
centers = torch.linspace(0.0, cutoff, n_gaussians).to(device)
gamma = 0.5 * (centers[1]- centers[0])**2
dist_centered_squared = (pairwise_dist.unsqueeze(-1) - centers) ** 2
expanded_pairwise_dist = torch.exp(gamma * dist_centered_squared).to(device)
```

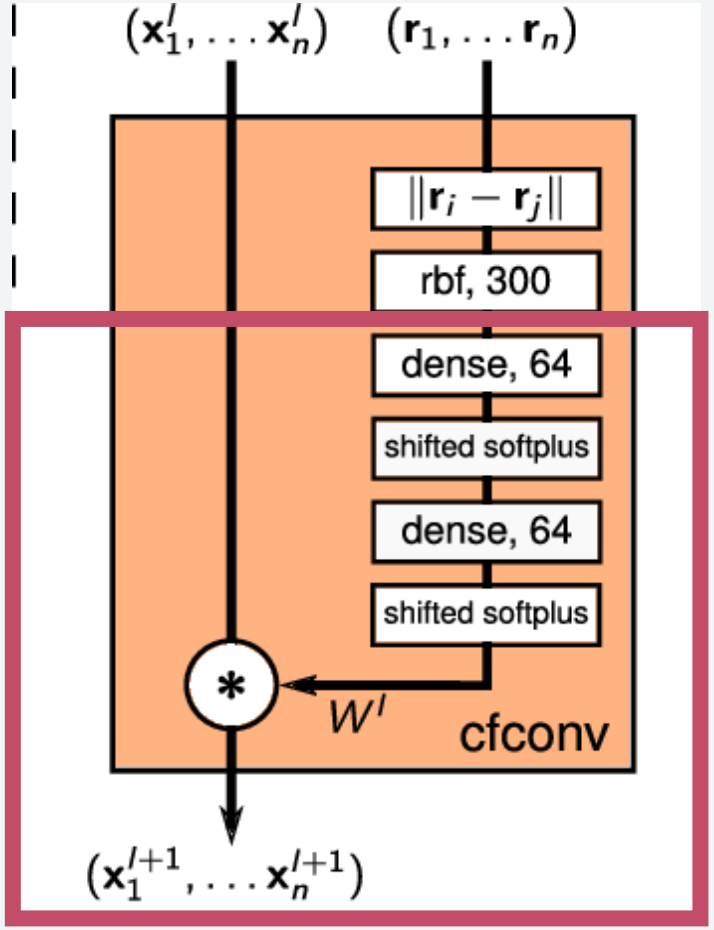
The Network – Convolutional Layer

Tensor Transformation: [478, 48, 47, 300] → [478, 48, 47, 64]
General Tensor Transform: [#of calculations, # atoms, pairwise distances of every atom with every other atom, # rbfs (default = 300)] → [#of calculations, # atoms, pairwise distances of every atom with every other atom, # of embedding features (default = 64)]

```
super().__init__()
self.linear_layer_stack = nn.Sequential(nn.Linear(in_features= n_gaussians,
out_features=hidden_units),nn.Softplus(beta=2),nn.Linear(in_features=hidden
_units,out_features=hidden_units),nn.Softplus(beta=2),nn.Linear(in_features
=hidden_units, out_features=output_features)
```

```
weights =
self.linear_layer_stack(expanded_pairwise_dist)
```

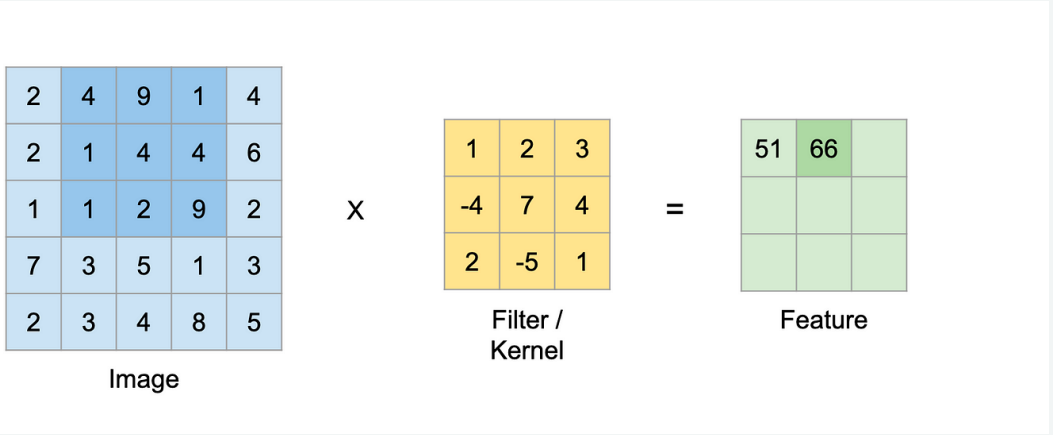
Weights are generated--- this is “continuous filter”



What does “continuous” filter mean?

Basic Discrete Filter:

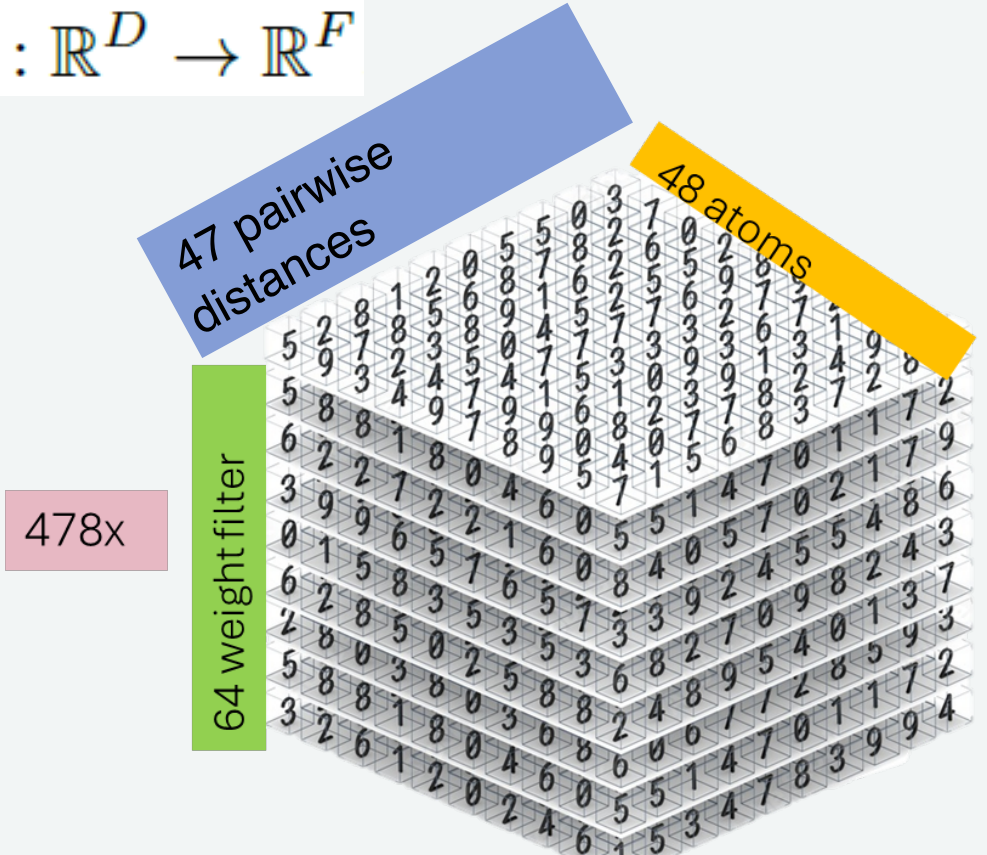
- Kernel weights are random numbers that are initialized in a matrix
- Classic Example: MNIST database



Continuous Filter:

- A separate neural network takes relative atomic positions (based on pairwise distances and PBC) to learn weights that are applied to each atom’s embedding which is related to its atomic number

$$W^l : \mathbb{R}^D \rightarrow \mathbb{R}^F$$

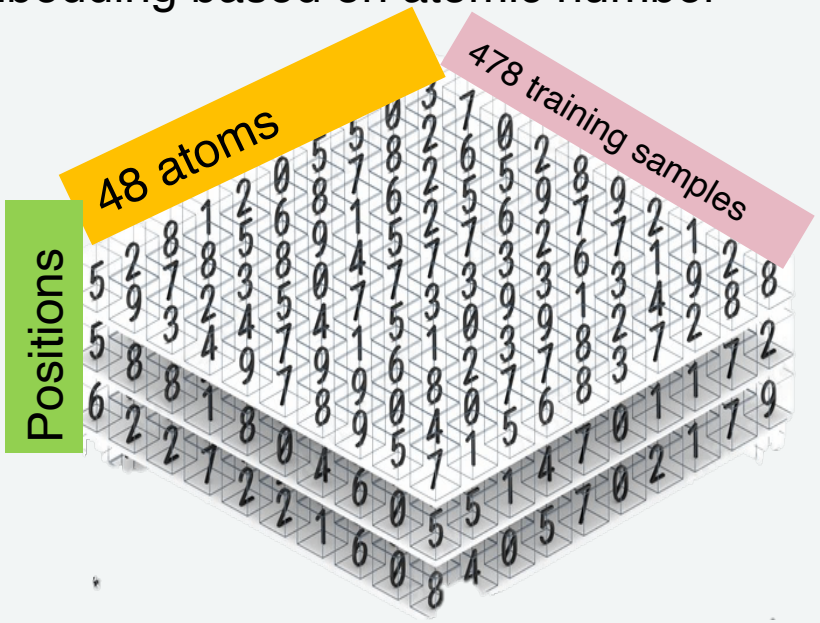


What does “continuous” filter mean?

Continuous Filter:

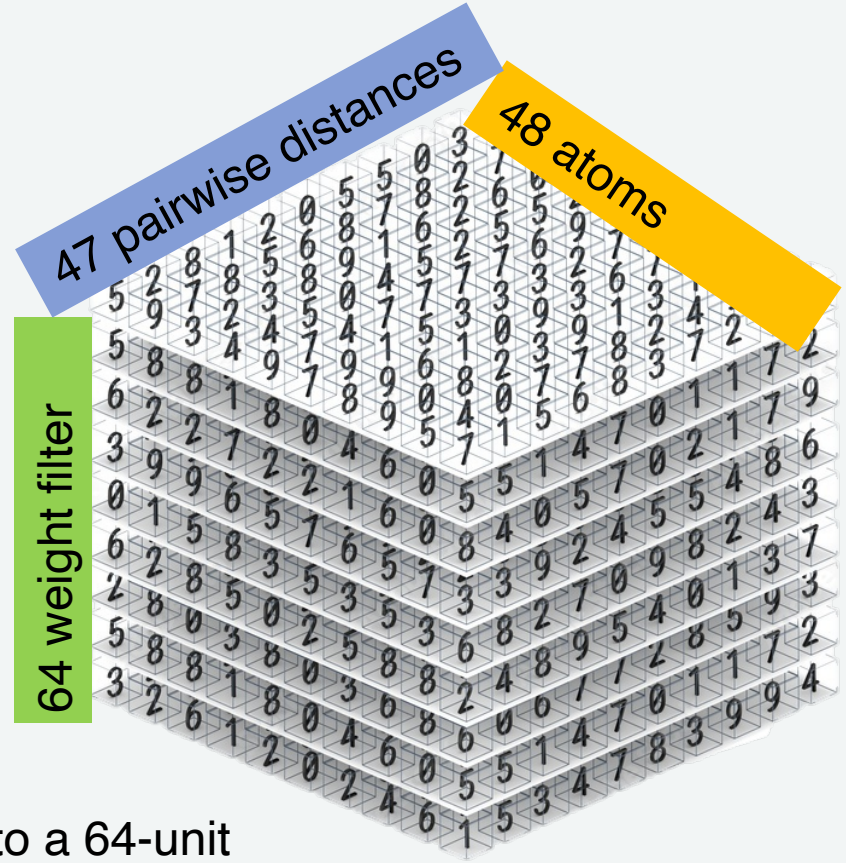
A separate neural network takes relative atomic positions (based on pairwise distances and PBC) to learn weights that are applied to each atom’s embedding based on atomic number

$$W^l : \mathbb{R}^D \rightarrow \mathbb{R}^F$$



RBF Neural Network

478x



Each position vector associated with each atom is transformed into a 64-unit vector mapped to each atom’s relative position with every other atom (translational/rotational invariance)

The Network – Convolutional Layer

Tensor Transformation: X^l : [478, 48, 64] → [478,48, 1, 64]
General Tensor Transform: Atomic # embedding needs to match the dimensions of filter [478, 48, 47, 64]

- Reshape the embedding tensor to facilitate the multiplication.. the new shape will be (478, 48, 1, 64)

```
embedding_reshaped =
atom_wise_embedding.unsqueeze(2)
```

reshaped embedding and weights

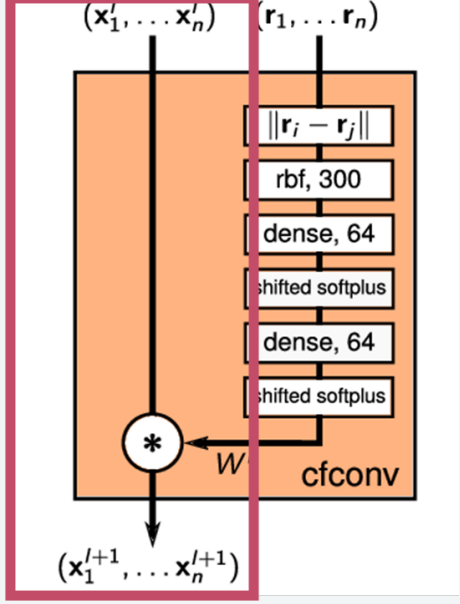
- The result will be a tensor of shape (478, 48, 47, 64)

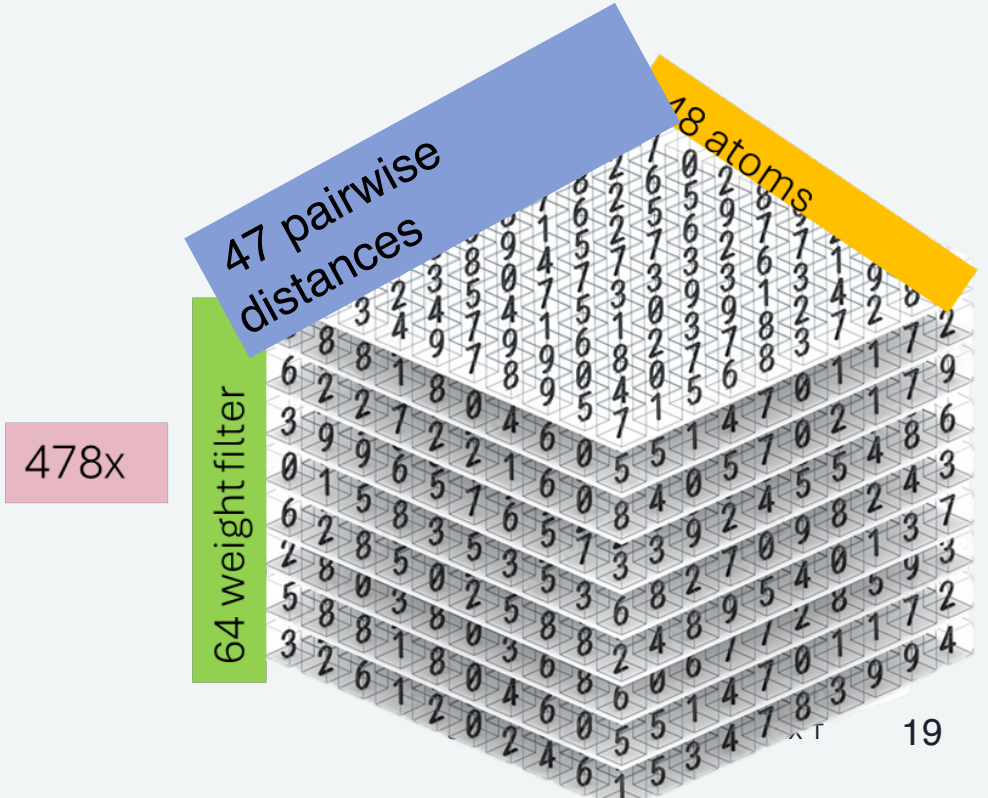
```
weighted_embedding = embedding_reshaped * weights
```

- Sum along the third dimension to get the new embedding tensor now relating each atom by its relative position and atomic identity
- The result will be a tensor of shape (478, 48, 64)

```
return torch.sum(weighted_embedding, dim=2)
```

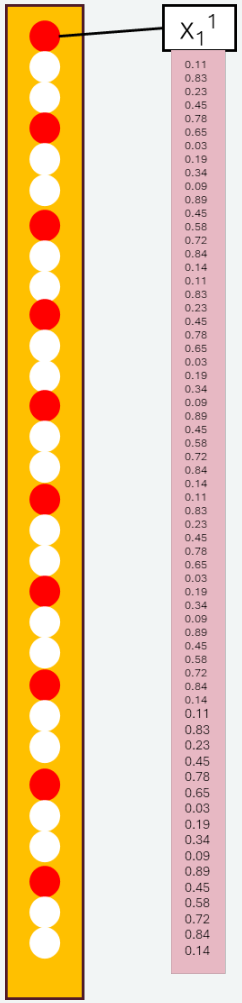
$$\mathbf{x}_i^{l+1} = (X^l * W^l)_i = \sum_j \mathbf{x}_j^l \circ W^l(\mathbf{r}_i - \mathbf{r}_j),$$

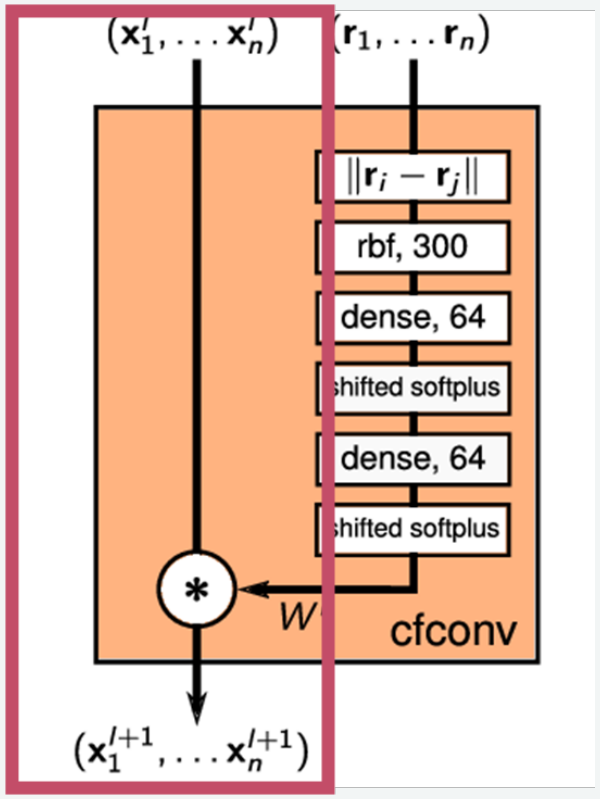


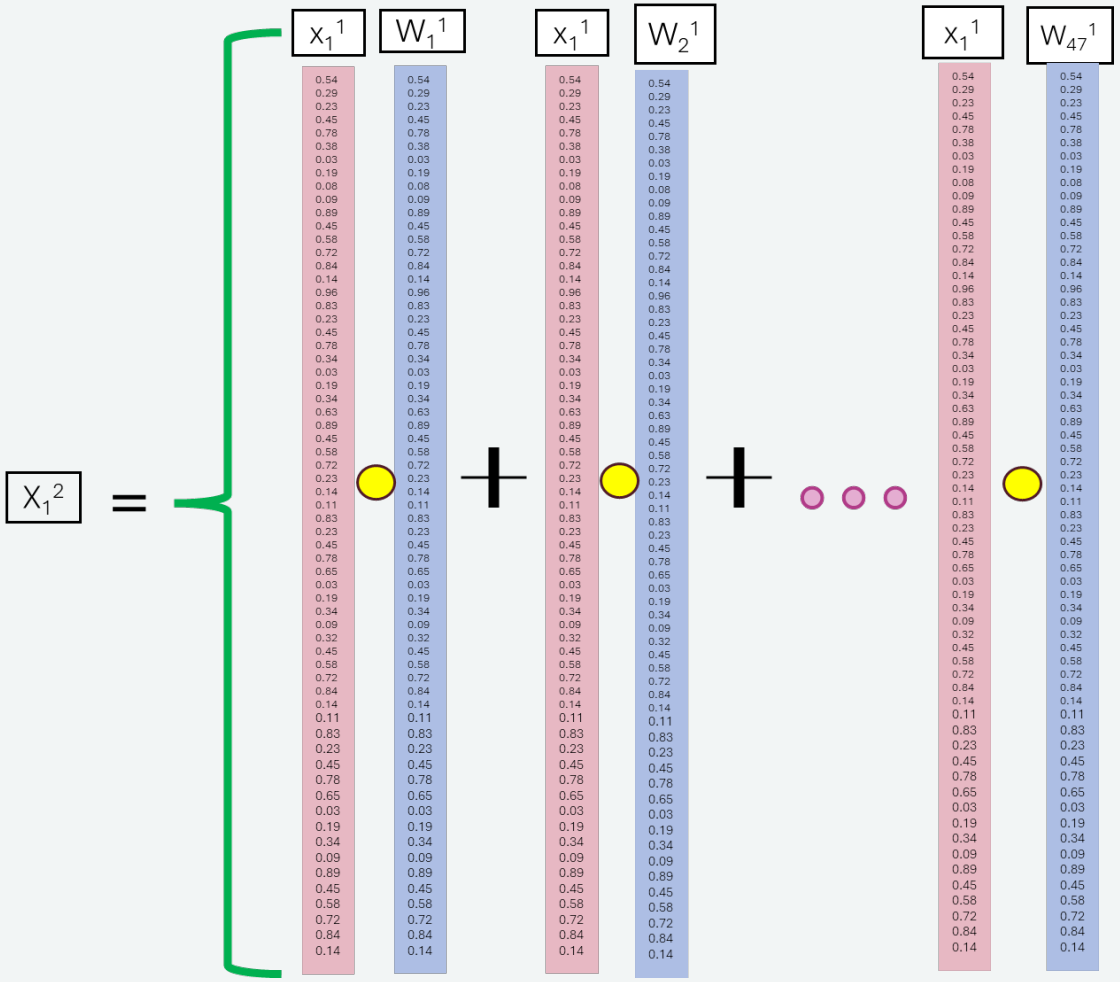


$$\mathbf{x}_i^{l+1} = (X^l * W^l)_i = \sum_j \mathbf{x}_j^l \circ W^l(\mathbf{r}_i - \mathbf{r}_j),$$

[478, 48, 1, 64] ∘ [478, 48, 47, 64]





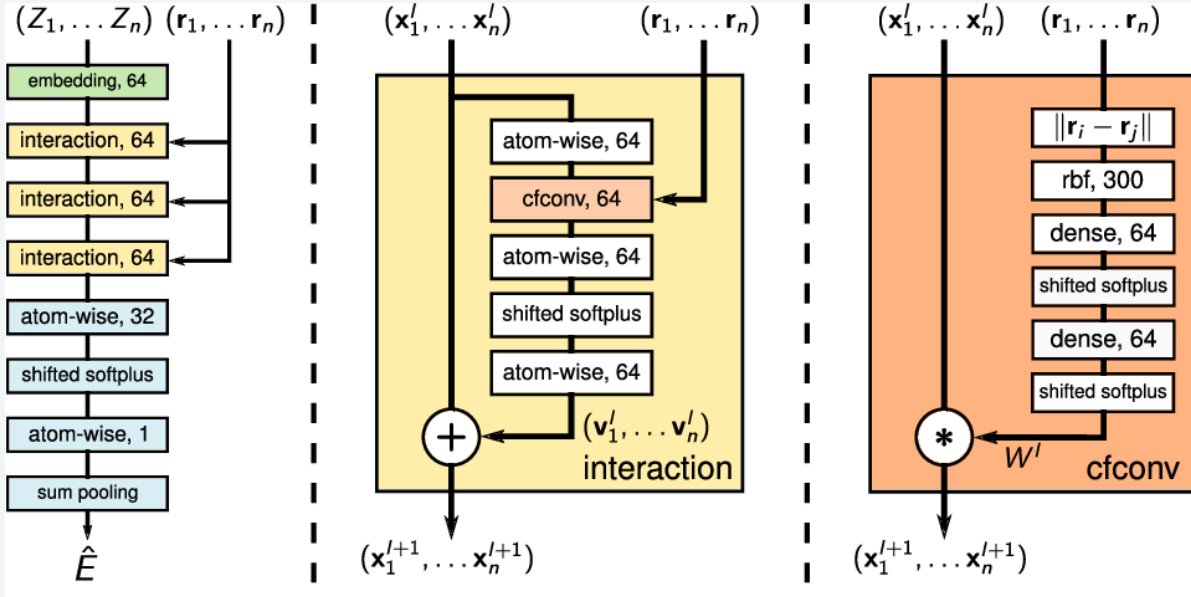


Back to the Network - Generation of Forces

- Neural Network predicts total energy of system using correlated atomic identities and positions as inputs → capable of capturing behavior predicted in higher level electronic structure calculations
- How are forces predicted? Using backpropogation algorithm, can solve for forces by gradient of Energy with respect to each atomic coordinate :

$$\hat{\mathbf{F}}_i(Z_1,\ldots,Z_n,\mathbf{r}_1,\ldots,\mathbf{r}_n) = -\frac{\partial \hat{E}}{\partial \mathbf{r}_i}(Z_1,\ldots,Z_n,\mathbf{r}_1,\ldots,\mathbf{r}_n).$$

```
class Schnet_Complete(nn.Module):
...

forces = -torch.autograd.grad(
            pooled_energies_reshaped,
            positions,

grad_outputs=torch.ones_like(pooled_energies_re
shaped),
            create_graph=True)[0]

        return pooled_energies_reshaped, forces
```



Training Parameters

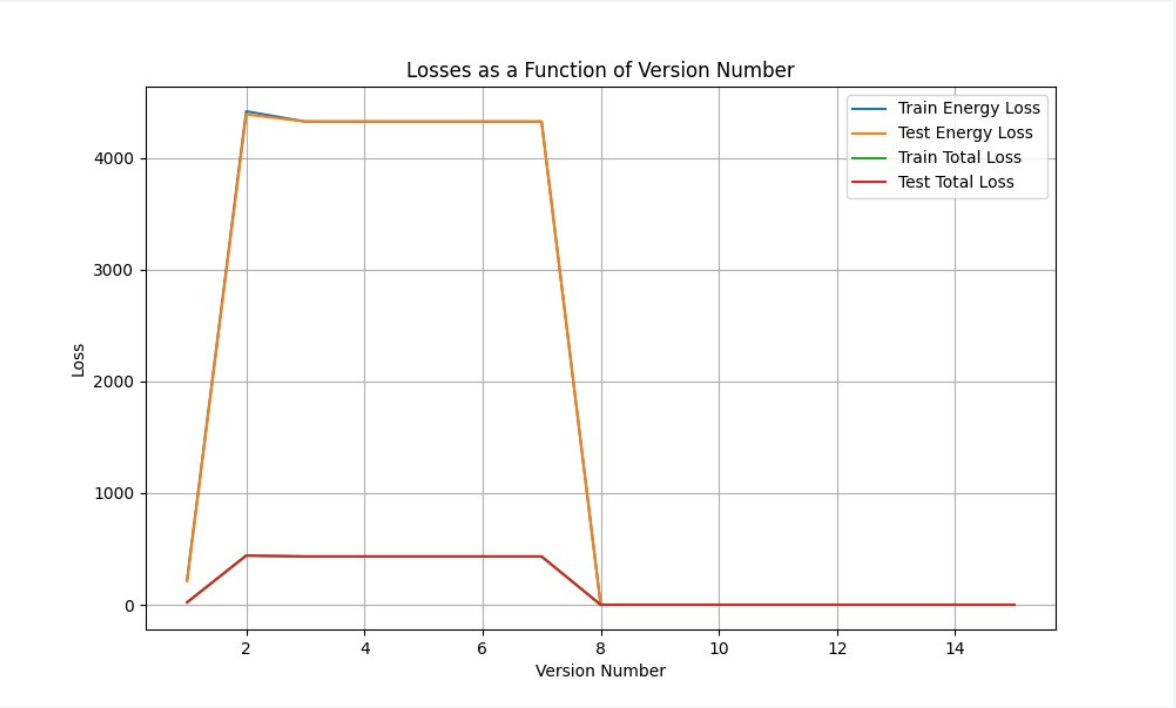
- Loss function accounts for MSE with regard to energies, as well as MSE with regard to force predictions
- MSE of force predictions is the sum of the MSE of each actual force vector – the calculated force vector using the neural network predicted energy.
- $\rho = 0.1$ (paper is 0.01)

$$\ell(\hat{E}, (E, \mathbf{F}_1, \ldots, \mathbf{F}_n)) = \rho \|E - \hat{E}\|^2 + \frac{1}{n} \sum_{i=0}^{n} \left\| \mathbf{F}_i - \left(-\frac{\partial \hat{E}}{\partial \mathbf{R}_i} \right) \right\|^2 .$$

- Started with same parameters as paper – 64 features, 300 gaussians (different gamma) , 3 interaction blocks
- Batch size = 16,
- optimizer =Adam
- lr = 0.0001
- scheduler = StepLR(optimizer, step_size=5, gamma=0.1)

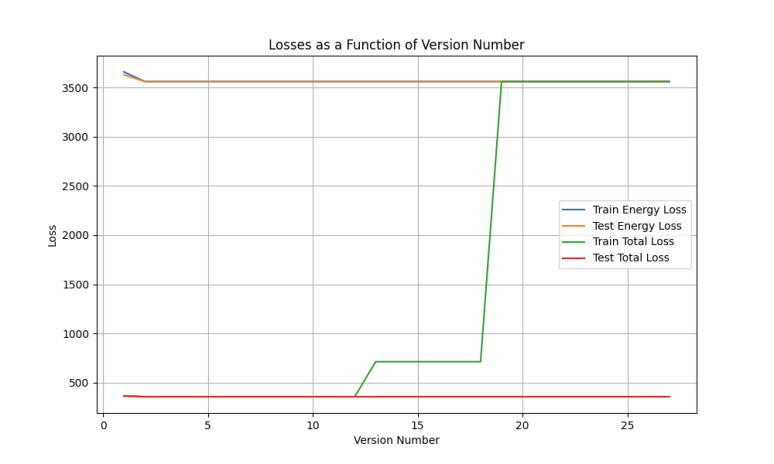
Results for Standard Training Conditions

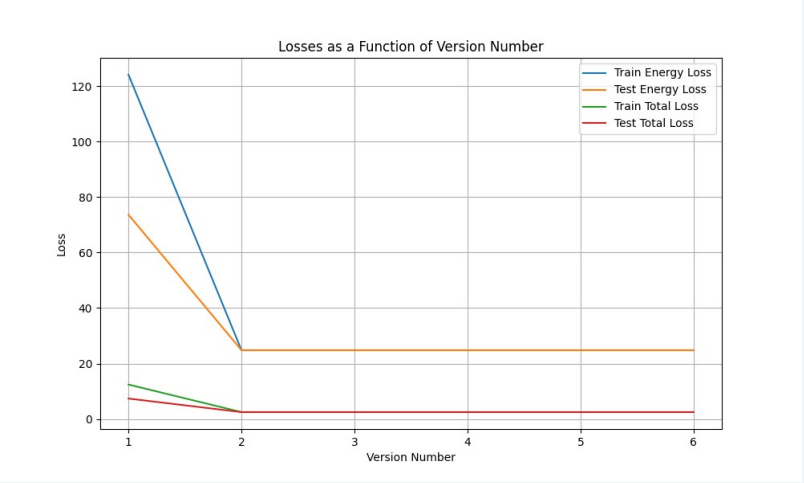
- Train Energy MSE: 0.13399571180343628 RMSE: 0.366279297
- Test Energy MSE: 0.13426832854747772 RMSE: 0.366855538
- Train Force MSE: 0.0004477987240534276 RMSE: 0.021179184
- Test Force MSE: 0.0004108284483663738 RMSE: 0.020270601
- Train Total MSE: 0.013847369700670242 RMSE: 0.117646813
- **Test Total MSE: 0.013837661594152 45 RMSE: 0.117588312**

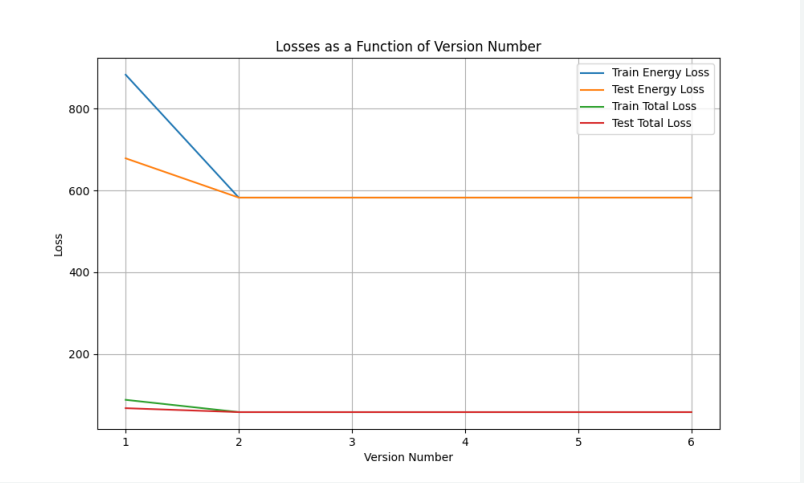


Comparison After Changing Parameters

	MSE (Test Total Energy)	RMSE (Test Total Energy)
Condition 1	**0.013837**	**0.117588312**
Condition 2 (n_gaussians = 150)	356.1743469238281	18.868876
Condition 3 (feature space = 32)	2.479239940643310 5	1.573773
Condition 4(6 interactive blocks)	58.247753143310 55	7.635289







Future Work

- Transfer Learning to higher order electronic structure theory methods
- OpenMM Simulation
- Compare to Markland Group Results

Thanks

Periodic Boundary Conditions Code

Tensor Transformation: [478, 48, 3] → [478, 48, 47]

General Tensor Transform: [#of calculations, # atoms, 3D coordinates for each atom]

→ [#of calculations, # atoms, pairwise distances of each atom]

General Intuition: For Loops

- If last dimension of tensor is 3d coordinates, and instead we want 47D vector with each distance from each 3d coordinate to every other 3d coordinate, should we iterate?

```
coordinates = dataset_train.positions
# Function to calculate Euclidean distance between two points
def euclidean_distance(point1, point2):
    return torch.sqrt(torch.sum((point1 - point2)**2))


# Initialize a tensor for pairwise distances with shape (478, 48, 47)
pairwise_distances = torch.zeros((478, 48, 47))

# Calculate pairwise distances
for i in range(48):
    for j in range(47):
        # Skip the case where i equals j
        if i != j:
            pairwise_distances[:, i, j] = euclidean_distance(coordinates[i], coordinates[j])
```