Homework 4

Deepak Akhare

December 12, 2022

Problem 1A:

A function to compute the spectral filters $\{\hat{g}_{j=1}^{J}\}$ is constructed using the half-cosine kernel window. The code to implement is as follows.

```
class kernel:
    def __init__(self, K, R, d, J, lamb_max):
        # -- filter properties
        self.R = float(R)
        self.J = J
        self.K = K
        self.d = d
        self.lamb_max = torch.tensor(lamb_max)

        # -- Half-Cosine kernel
        self.a = R*torch.log(self.lamb_max) / (J-R+1)
        self.g_hat = lambda lamb: sum([self.d[k]*torch.cos(2*torch.pi*k*(lamb/self.a+
                                                    0.5))*(-lamb>=0 and -lamb<self.a)
                                                    for k in range(K+1)])

    def wavelet(self, lamb, j):
        """
            constructs wavelets ($j\in [2, J]$).
        :param lamb: eigenvalue (analogue of frequency).
        :param j: filter index in the filter bank.
        :return: filter response to input eigenvalues.
        """
        assert(j>=2 and j<=self.J)
        return self.g_hat(torch.log(torch.tensor(lamb)) - self.a*(j-1)/self.R)

    def scaling(self, lamb):
        """
            constructs scaling function (j=1).
        :param lamb: eigenvalue (analogue of frequency).
        :return: filter response to input eigenvalues.
        """
        g2 = self.R/2*sum([di**2 for di in self.d])
        g3 = sum([self.wavelet(lamb,i)**2 for i in range(2,self.J+1)])
        gj = torch.sqrt(self.R*self.d[0]**2 + g2 - g3)
        return gj
```

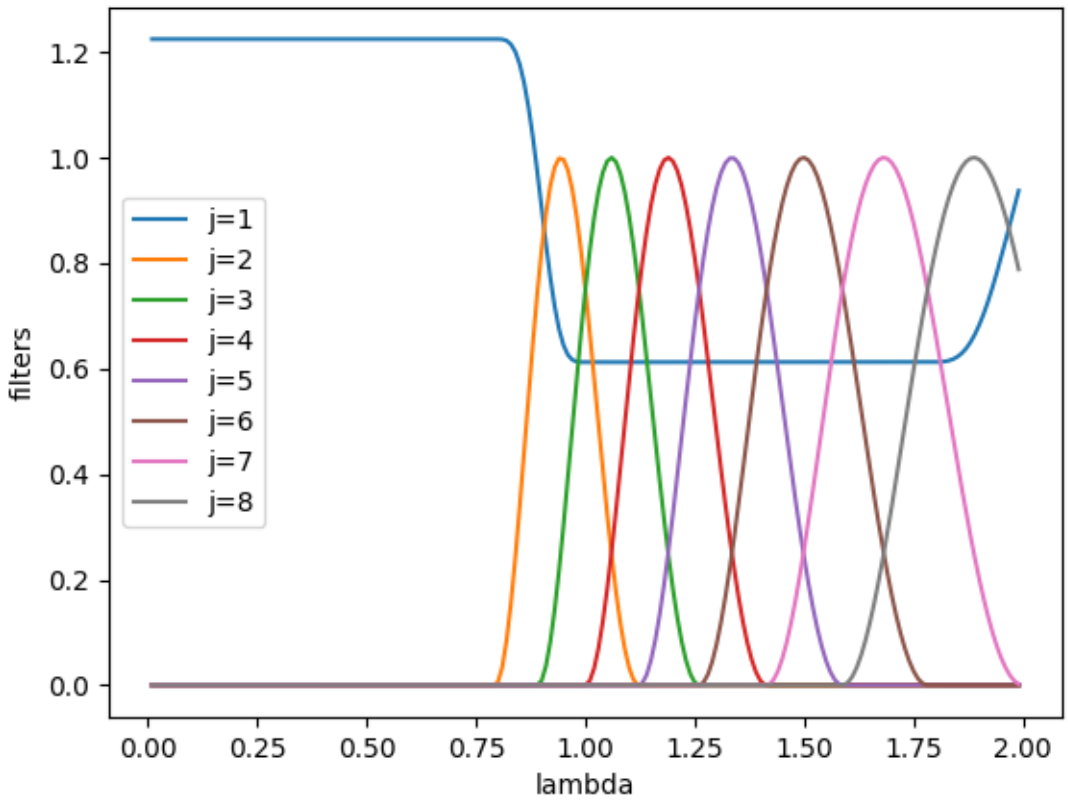

Figure 1: spectral filters $\{\hat{g}_{j=1}^{J}\}$ as a function of λ

Problem 1B:

spectral filters $\{\hat{g}_{j=1}^{J}\}$ as a function of λ is shown bellow.

Problem 2A:

A code is written to construct scattering feature maps z_G from molecules in the QM9 dataset, using J = 8 and L.

```
class scattering(nn.Module):
    def __init__(self, J, L, V, d_f, K, d, R, lamb_max):
        super(scattering, self).__init__()

        # -- graph parameters
        self.n_node = V
        self.n_atom_features = d_f

        # -- filter parameters
        self.K = K
        self.d = d
        self.J = J
        self.R = R
        self.lamb_max = lamb_max
        self.filters = kernel(K=1, R=3, d=[0.5, 0.5], J=8, lamb_max=2)

        # -- scattering parameters
        self.L = L

    def compute_spectrum(self, W):
        """
            Computes eigenvalues of normalized graph Laplacian.
        :param W: tensor of graph adjacency matrices.
        :return: eigenvalues of normalized graph Laplacian
        """

        # -- computing Laplacian
        # W = W + torch.eye(len(W))
        D = torch.diag_embed(W.sum(1))
        L = D - W

        # -- normalize Laplacian
        diag  = W.sum(1)
        dhalf = torch.diag_embed(1. / torch.sqrt(torch.max(torch.ones(diag.size()),
                                                           diag)))
        # print(torch.isnan(dhalf))
        L     = dhalf @ L @ dhalf

        # -- eig decomposition
        E, V = torch.symeig(L, eigenvectors=True)

        return abs(E), V

    def filtering_matrices(self, W):
        """
            Compute filtering matrices (frames) for spectral filters
        :return: a collection of filtering matrices of each wavelet kernel and the
                                                    scaling function in the filter-
```

```
                                                    bank.
        """

        filter_matrices = []
        E, V = self.compute_spectrum(W)
        # print(E)

        # -- scaling frame
        LM = torch.diag_embed(torch.tensor([self.filters.scaling(e) for e in E]))
        filter_matrices.append(V @ LM @ V.T)

        # -- wavelet frame
        for j in range(2, self.J+1):
            LM = torch.diag_embed(torch.tensor([self.filters.wavelet(e, j=j) for e in
                                                E])).type(torch.float64)
            filter_matrices.append(V @ LM @ V.T)
            # filter_matrices.append(V @ ... @ V.T)

        return torch.stack(filter_matrices)

    def forward(self, W, f):
        """
            Perform wavelet scattering transform
        :param W: tensor of graph adjacency matrices.
        :param f: tensor of graph signal vectors.
        :return: wavelet scattering coefficients
        """

        # -- filtering matrices
        g = self.filtering_matrices(W)

        # --
        U_ = [f]

        # -- zero-th layer
        # S = ...  # S_(0,1)
        S = f.mean(0)

        for l in range(self.L):
            U = U_.copy()
            U_ = []

            for f_ in U:
                for g_j in g:

                    U_.append(abs(g_j@f_))
                    S_li = torch.mean(abs(g_j@f_),0)

                    # -- append scattering feature S_(l,i)
                    S = torch.cat((S,S_li), dim=0)

        return S

# -- initialize scattering function
```

```
scat = scattering(L=2, V=9, d_f=5, K=1, R=3, d=[0.5, 0.5], J=8, lamb_max=2)

# -- load data
training_points = 5000
testing_points  = 1000
data = MolecularDataset(N=training_points + testing_points)

# -- Compute scattering feature maps

S = torch.cat([scat.forward(data.adjs[b], data.sigs[b]).unsqueeze(0) for b in range(
                                            len(data.adjs))])
```

Problem 2B:

A 2D projection feature maps of z_G is created using PCA. The code and plot are as follows.

```
# -- PCA projection
pca    = PCA(n_components=2)
latent = pca.fit_transform(S)
```

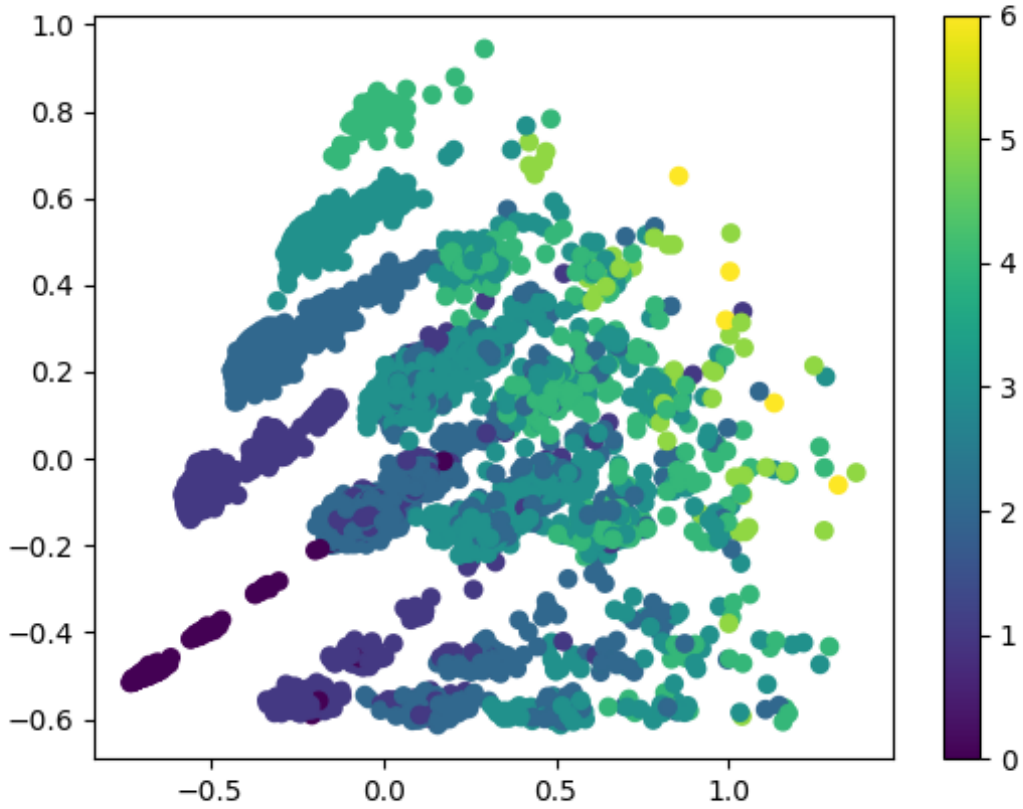

Figure 2: Scatter plot to visualize the 2D map of the scattering latent space with the number of hydrogen-bond acceptor atoms corresponding to the molecular graph

Problem 2C:

A code is developed to implement a 2-layer neural network to predict the Highest Occupied Molecular Orbital (HOMO) energy of the molecule from scattering representations. The model is trained using 5000 data points. The loss plot is as follows.

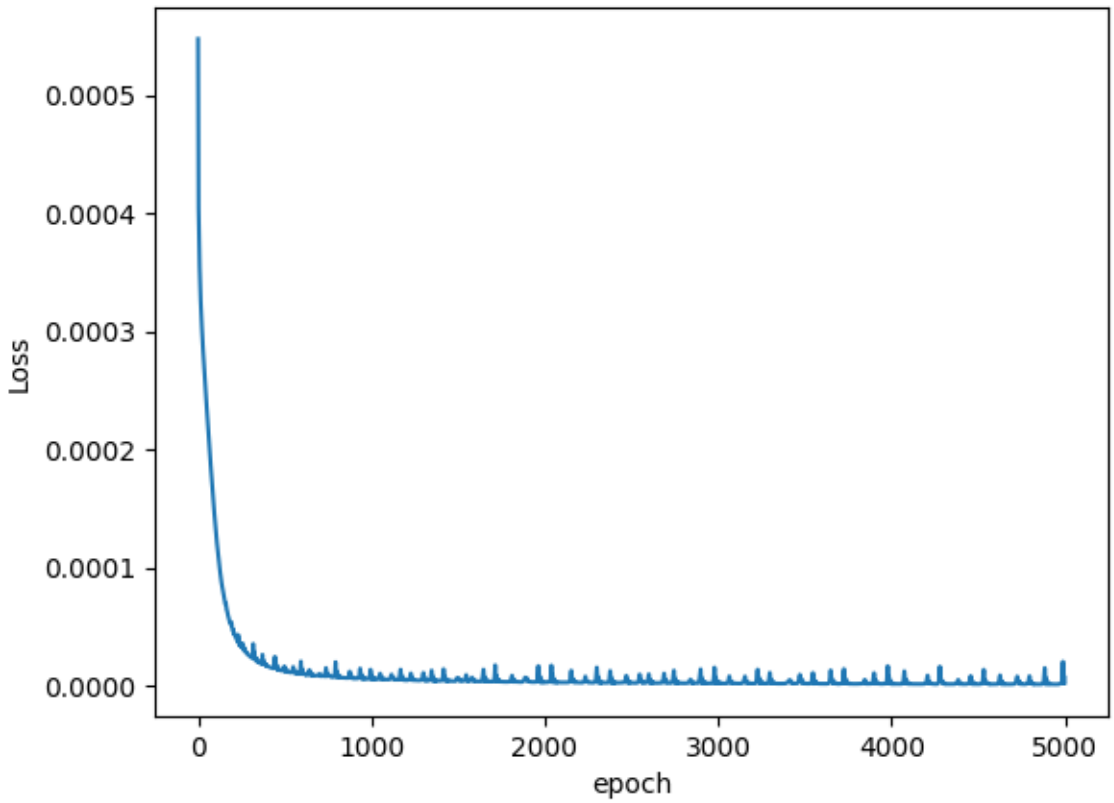

Figure 3: Loss plot of a 2-layer neural network to predict the Highest Occupied Molecular Orbital (HOMO) energy of the molecule

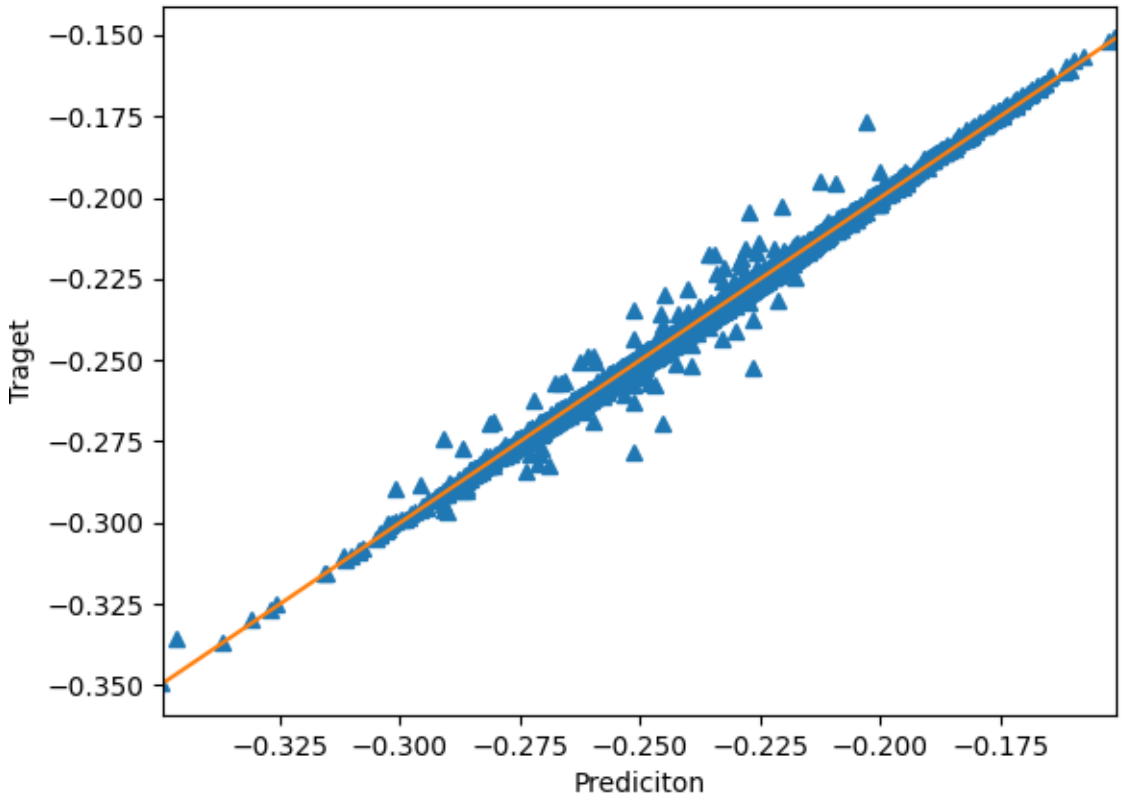

Figure 4: comparison in target and prediction of the Highest Occupied Molecular Orbital (HOMO) energy of the molecule on 5000 training data points

Problem 2D:

A 1000 test data points are used to evaluate the model. The comparison in target and prediction is shown below figure.

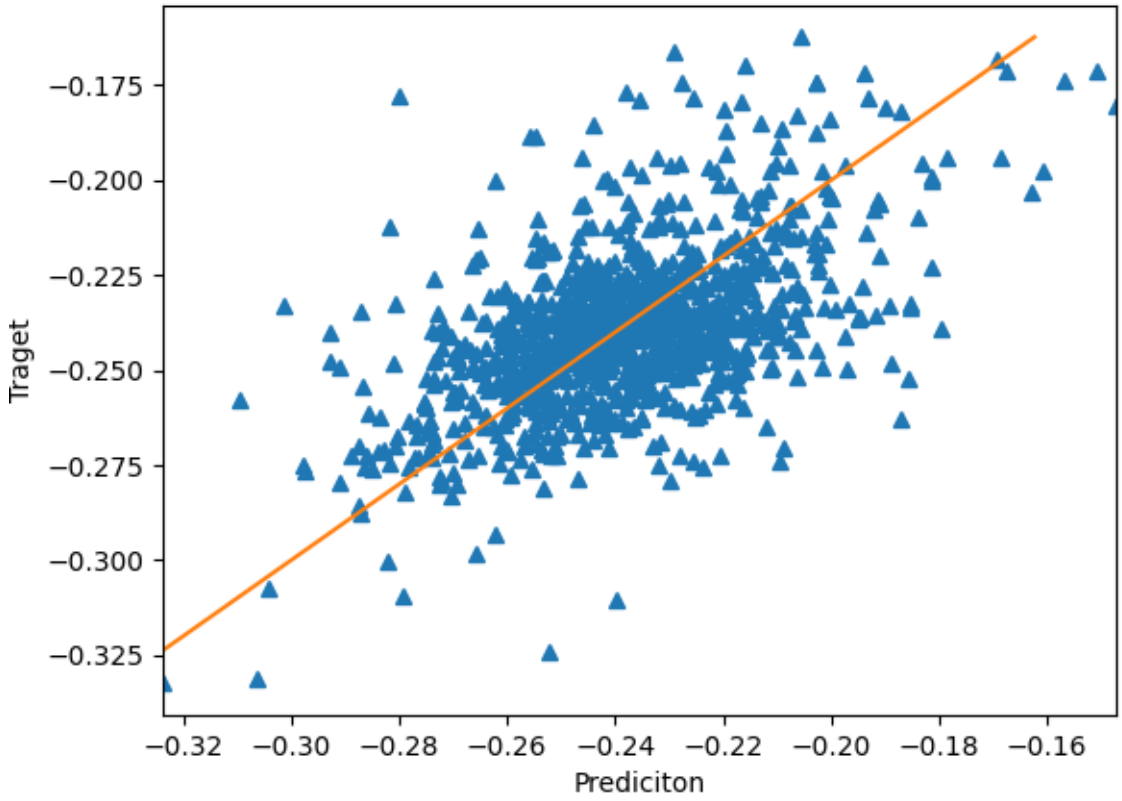

Figure 5: comparison in target and prediction of the Highest Occupied Molecular Orbital (HOMO) energy of the molecule on 5000 testing data points