Objective

This document is a comprehensive study of the DMI problem using DMInet. It tries to solve the following tasks:

1. inputs: 3 (bead image, membrane type, coordinates) vs 2 (bead image, membrane type)?
2. graph construction (adjacency matrix vs DeltaG value);
3. one-hot encoding the membrane type vs simple number representation;
4. hyperparameter tuning.

In [1]:
```
from utils import *
import matplotlib.pyplot as plt
%matplotlib inline
import pandas as pd
import h5py
from sklearn.model_selection import train_test_split
from scipy.stats import gaussian_kde
from sklearn.metrics import r2_score, mean_absolute_error
from tensorflow.keras.models import Sequential, Model, load_model
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras.models import model_from_json
from tensorflow.keras.layers import Input, Dense, Conv2D, MaxPooling2D, Flatten, concaten
```

In [2]:
```
# load data
beadList = np.load("beadList.npy")
beadMat = np.load("beadMat.npy")
membraneTypes = np.load("membraneTypes.npy")
coords = np.load("coords.npy")
DeltaG = np.load("PMF.npy")
print(beadList.shape)
print(beadMat.shape)
print(membraneTypes.shape)
print(coords.shape)
print(DeltaG.shape)
```

```
(630,)
(630, 14, 14)
(630,)
(630, 10)
(630, 10)
```

Without the coords as inputs

In [3]:
```
# get train, test dataset
X_train_beadlist, X_test_beadlist, X_train_bead, X_test_bead, X_train_membrane, X_test_meml
                                                train_test_split( \
                                                beadList, beadMat, membraneTypes, coords, l
                                                shuffle=True, test_size=0.2, random_state=
```

In [4]:
```python
# save test data
np.save("beadList_test.npy", X_test_beadlist)
np.save("beadMat_test.npy", X_test_bead)
np.save("membrane_test.npy", X_test_membrane)
np.save("coords_test.npy", coords_test)
np.save("PMF_test.npy", y_test)
```

In [5]:
```python
# # convert datatype
X_train_bead = X_train_bead.astype('float32')
X_train_membrane = X_train_membrane.astype('float32')
X_test_bead = X_test_bead.astype('float32')
X_test_membrane = X_test_membrane.astype('float32')
```

In [6]:
```python
# reshape data
X_train_bead = X_train_bead.reshape((X_train_bead.shape[0], Num_beads, Num_beads, 1))
X_test_bead = X_test_bead.reshape((X_test_bead.shape[0], Num_beads, Num_beads, 1))
print(X_train_bead.shape, X_test_bead.shape)
```

```
(504, 14, 14, 1) (126, 14, 14, 1)
```

In [7]:
```
# define the input: only 2 considered
input1 = Input(shape=(Num_beads, Num_beads, 1)) # the bead matrix
input2 = Input(shape=(1,))                       # the membrane type
# input3 = Input(shape=(Num_labels,))            # the coordinates

# part I: the feature of the beads
input_model1 = Conv2D(5, (3,3), padding="same", activation='relu', kernel_initializer='he_
input_model1 = BatchNormalization()(input_model1)
input_model1 = MaxPooling2D((2,2))(input_model1)
input_model1 = Conv2D(3, (3,3), padding="same", activation='relu', kernel_initializer='he_
input_model1 = BatchNormalization()(input_model1)
input_model1 = MaxPooling2D((2,2))(input_model1)
input_model1 = Flatten()(input_model1)
input_model1 = Model(inputs=input1, outputs=input_model1)

# part II: the membrane type
input_model2 = Dense(1)(input2)
input_model2 = Model(inputs=input2, outputs=input2)

# # part III: the coords along the membrane
# input_model3 = Dense(Num_labels)(input3)
# input_model3 = Model(inputs=input3, outputs=input3)

# concatenate together
Merged = concatenate([input_model1.output, input_model2.output])
# Merged = concatenate([input_model1.output, input_model2.output, input_model3.output])
out_model = Dense(30, activation='relu', kernel_initializer='he_uniform')(Merged)
out_model = BatchNormalization()(out_model)
out_model = Dense(20, activation='relu', kernel_initializer='he_uniform')(out_model)
out_model = BatchNormalization()(out_model)
out_model = Dense(Num_labels)(out_model)

# GCN_model = Model(inputs=[input_model1.input, input_model2.input, input_model3.input], ou
GCN_model = Model(inputs=[input_model1.input, input_model2.input], outputs=out_model)
```

In [8]:
```
# compile model
GCN_model.compile(optimizer='adam',
            loss='mean_squared_error',
            metrics=['mean_absolute_error'])
```

In [9]:
```
# add callbacks
es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=20)
mc = ModelCheckpoint('./res_2inputs/best_model.h5', monitor='val_loss', mode='min', verbos

# train model
history = GCN_model.fit(x=[X_train_bead, X_train_membrane], y=y_train,\
                        validation_split=0.1, epochs=500, batch_size=32, callbacks=[es, mc
```

```
Epoch 1/500
15/15 [==============================] - ETA: 0s - loss: 8.6980 - mean_absolute_erro
r: 1.9535
Epoch 00001: val_loss improved from inf to 8.13835, saving model to ./res_2inputs/bes
t_model.h5
15/15 [==============================] - 0s 27ms/step - loss: 8.6980 - mean_absolute_
error: 1.9535 - val_loss: 8.1383 - val_mean_absolute_error: 1.7380
Epoch 2/500
10/15 [===================>..........] - ETA: 0s - loss: 7.7111 - mean_absolute_erro
r: 1.8158
Epoch 00002: val_loss improved from 8.13835 to 7.72950, saving model to ./res_2input
s/best_model.h5
15/15 [==============================] - 0s 11ms/step - loss: 7.8309 - mean_absolute_
error: 1.8247 - val_loss: 7.7295 - val_mean_absolute_error: 1.6339
Epoch 3/500
10/15 [===================>..........] - ETA: 0s - loss: 7.5067 - mean_absolute_erro
r: 1.7683
Epoch 00003: val_loss improved from 7.72950 to 7.39563, saving model to ./res_2input
s/best_model.h5
```

In [10]:
```
# load the best model
best_model = load_model("./res_2inputs/best_model.h5")
y_pred = best_model.predict([X_test_bead, X_test_membrane])
# save prediction
np.save("PMF_pred.npy", y_pred)
```

In [11]:
```
print(best_model.summary())
```

```
Model: "model_2"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 14, 14, 1)]  0
__________________________________________________________________________________________________
conv2d (Conv2D)                 (None, 14, 14, 5)    50          input_1[0][0]
__________________________________________________________________________________________________
batch_normalization (BatchNorma (None, 14, 14, 5)    20          conv2d[0][0]
__________________________________________________________________________________________________
max_pooling2d (MaxPooling2D)    (None, 7, 7, 5)      0           batch_normalization[0]
[0]
__________________________________________________________________________________________________
conv2d_1 (Conv2D)               (None, 7, 7, 3)      138         max_pooling2d[0][0]
__________________________________________________________________________________________________
batch_normalization_1 (BatchNor (None, 7, 7, 3)      12          conv2d_1[0][0]
__________________________________________________________________________________________________
max_pooling2d_1 (MaxPooling2D)  (None, 3, 3, 3)      0           batch_normalization_1
[0][0]
__________________________________________________________________________________________________
flatten (Flatten)               (None, 27)           0           max_pooling2d_1[0][0]
__________________________________________________________________________________________________
input_2 (InputLayer)            [(None, 1)]          0
__________________________________________________________________________________________________
concatenate (Concatenate)       (None, 28)           0           flatten[0][0]
                                                                 input_2[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 30)           870         concatenate[0][0]
__________________________________________________________________________________________________
batch_normalization_2 (BatchNor (None, 30)           120         dense_1[0][0]
__________________________________________________________________________________________________
dense_2 (Dense)                 (None, 20)           620         batch_normalization_2
[0][0]
__________________________________________________________________________________________________
batch_normalization_3 (BatchNor (None, 20)           80          dense_2[0][0]
__________________________________________________________________________________________________
dense_3 (Dense)                 (None, 10)           210         batch_normalization_3
[0][0]
```

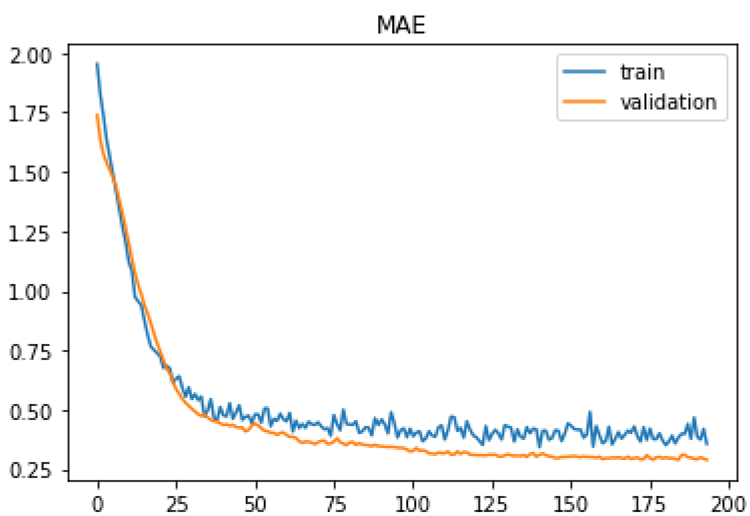
```
==========================================================================================
===========
Total params: 2,120
Trainable params: 2,004
Non-trainable params: 116
__________________________________________________________________________________________
___________
None
```

In [12]:
```
history.history.keys()
```

Out[12]:
```
dict_keys(['loss', 'mean_absolute_error', 'val_loss', 'val_mean_absolute_error'])
```

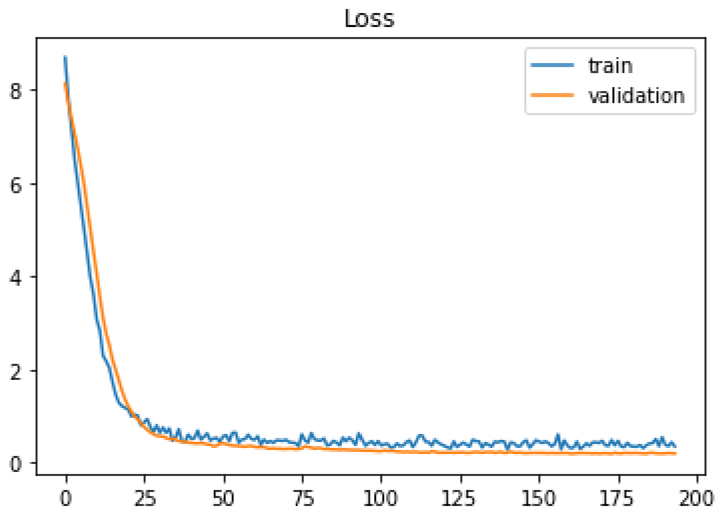
In [13]:
```
plt.plot(history.history["mean_absolute_error"]) #['acc', 'mean_absolute_error'])
plt.plot(history.history["val_mean_absolute_error"])
plt.legend(['train', 'validation'])
plt.title("MAE")
```

Out[13]:
```
Text(0.5, 1.0, 'MAE')
```



In [14]:
```
plt.plot(history.history["loss"]) #['acc', 'mean_absolute_error'])
plt.plot(history.history["val_loss"])
plt.legend(['train', 'validation'])
plt.title("Loss")
```

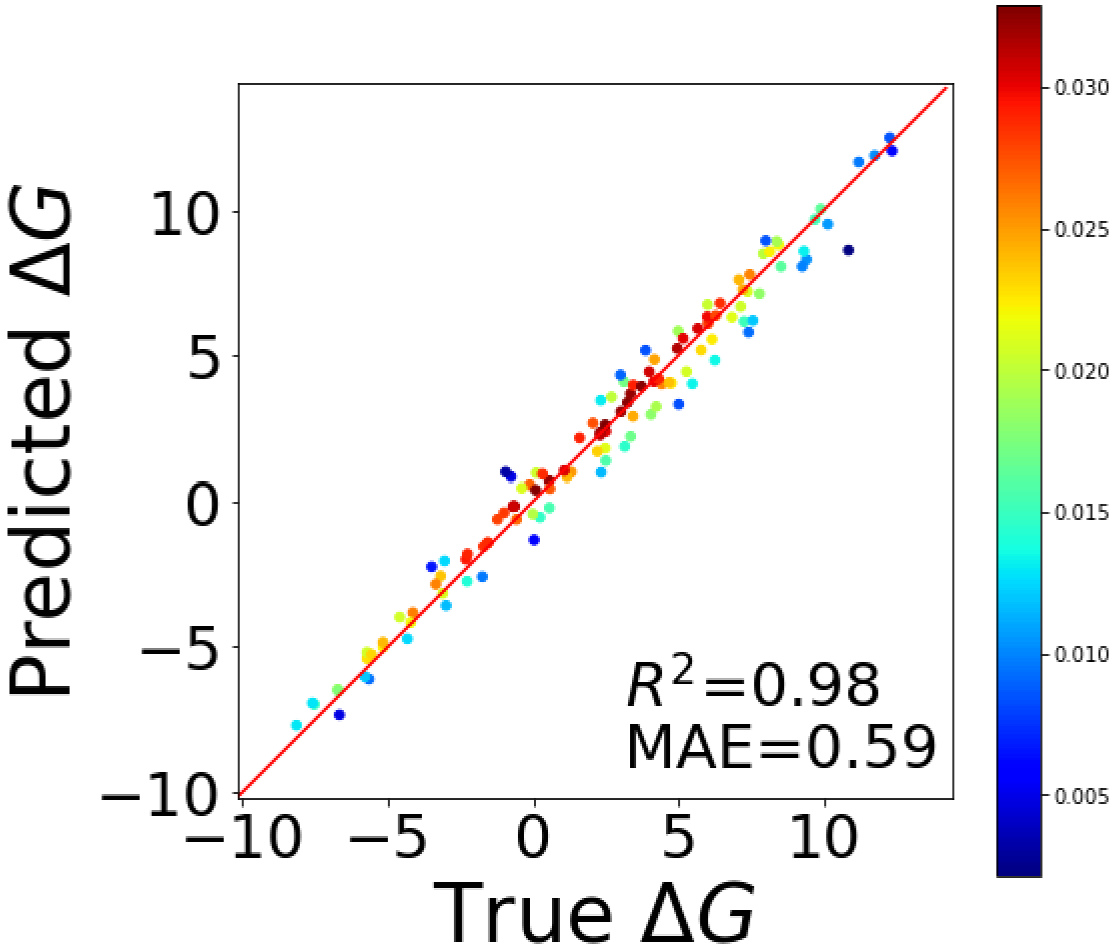
Out[14]:
```
Text(0.5, 1.0, 'Loss')
```



In [29]:

```
# make plots for the comparison: predict the Delta_G
Delta_G_pred = GetDeltaG(y_pred)
Delta_G_test = GetDeltaG(y_test)

# get the metrics
R2_score = r2_score(Delta_G_test, Delta_G_pred)
mae_score = mean_absolute_error(Delta_G_test, Delta_G_pred)

# Calculate the point density
xy = np.vstack([Delta_G_test, Delta_G_pred])
z = gaussian_kde(xy)(xy)

fig, ax = plt.subplots()
cax = ax.scatter(Delta_G_test, Delta_G_pred, c=z, s = 20, cmap='jet') #  edgecolor='',

# Sort the points by density, so that the densest points are plotted last
idx = z.argsort()
Delta_G_test, Delta_G_pred, z = Delta_G_test[idx], Delta_G_pred[idx], z[idx]
plt.colorbar(cax)

plt.xlabel("True $\Delta G$", fontsize=40)
plt.ylabel("Predicted $\Delta G$", fontsize=40)
plt.xticks(fontsize=30)
plt.yticks(fontsize=30)

x0, x1 = min(Delta_G_test), max(Delta_G_test)
length = x1 - x0
x_start, x_end = x0-0.1*length, x1+0.1*length
plt.xlim([x_start, x_end])
plt.ylim([x_start, x_end])
plt.gca().set_aspect("equal", adjustable="box")
plt.plot(np.arange(x_start, x_end, 0.01*length), np.arange(x_start, x_end, 0.01*length), '
plt.text(x_end - 0.55*length, x_start + 0.15 *length, '$R^2$='+"{:.2f}".format(R2_score),
plt.text(x_end - 0.55*length, x_start + 0.05 *length, 'MAE='+"{:.2f}".format(mae_score), f

fig=plt.gcf()
fig.set_size_inches(8,8)

plt.savefig('./res_2inputs/r2dimer.eps', format='eps', dpi=1000)
plt.savefig("./res_2inputs/r2dimer.png", dpi=1000, bbox_tight=True)
```

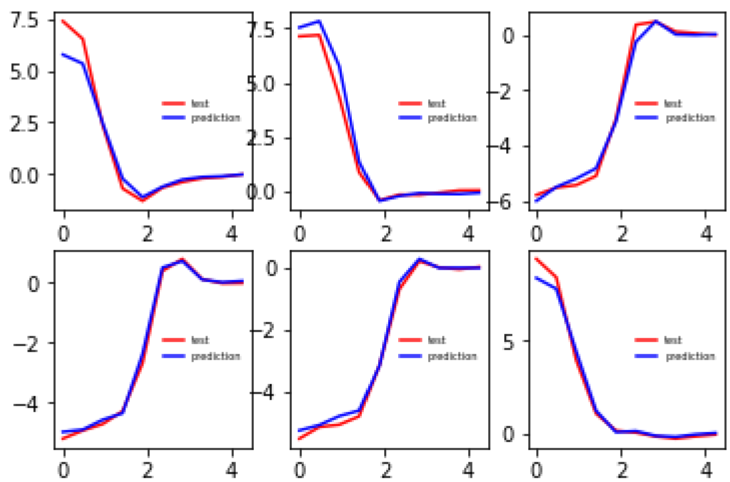
```
<ipython-input-29-2224cbeda42f>:40: MatplotlibDeprecationWarning: savefig() got unexpec
ted keyword argument "bbox_tight" which is no longer supported as of 3.3 and will becom
e an error two minor releases later
  plt.savefig("./res_2inputs/r2dimer.png", dpi=1000, bbox_tight=True)
```



In [16]:
```
pos = np.linspace(0.0, 4.25, 10)
```

In [24]:
```
# save the pandas of prediction
data_df.to_csv("r2_pred.csv", index=False)
```

In [25]:
```
pos = np.linspace(0.0, 4.25, 10)
random_idx = np.random.randint(0, y_test.shape[0], size=(6))
# get plots
for i in range(0, 6):
    plt.subplot(2, 3, i+1)
    idx = random_idx[i]
    plt.plot(pos,y_test[idx,:], color="red",label='test')
    plt.plot(pos,y_pred[idx,:], color="blue", label='prediction')
    plt.legend(loc='center right', fontsize=5, frameon=False)

# plt.savefig("res_2inputs/comparison.png", dpi=600, bbox_tight=True)
```



In [26]:
```
random_idx
```

Out[26]:
```
array([117,  85,  39,   2, 107,  51])
```

In [ ]: