ML For Molecular Simulation: Exercise 2

1 Linear Model

1.1 Data generation

1.1.1 What does the `numpy.random.randn` function do?

The function `numpy.random.randn` samples numbers from a normal (Gaussian) distribution with a mean of 0 and a variance of 1.

If you call the function without passing an argument. The function returns a single floating point number. But you can also pass a list of integers $i_1, i_2, ..., i_n$ to get an n-dimensional array of shape $(i_1, i_2, ..., i_n)$, which contains random floating point numbers.

1.1.2 How is the function used in the workflow?

The function is used to add some noise to the data that the model is trained on. It does this my computing the an array of y-values for the array of x-values according to the linear function and then adding some random gaussian noise to the y-values. The gaussian noise is scaled by the variable `dmax` to control the amount of noise added to the data.

Adding noise to the data is a common practice in machine learning to make the model more robust to real data, which usually always contains some noise, and to prevent overfitting.

1.2 Datasets and DataLoaders

1.2.1 Concept of Dataset

The `Dataset` class is an utility class from `torch` for storing the samples and the corresponding labels to the samples. These samples are stored as torch tensors. For our case we can store our array of x values with the corresponding y values. The `Dataset` class is especially useful

when you have some preprocessing that you want to do to your data before letting the model learn on the data.

1.2.2 Concept of DataLoader

The `DataLoader` is responsible for loading our data from the `Dataset`. Must fundamentally it is an special iterator meaning that we can use it very simply in a for loop syntax such that we can load the data from the `Dataset` one at the time. The `DataLoader` has some special features compared to a normal iterator which make it more useful for machine learning. For example it can load the data from the `Dataset` in mini-batches, which means that it can load the data in small chunks instead of one at a time, for example we set `batch_size=16` which means that during each iteration the `DataLoader` will load 16 samples from the `Dataset`. This is very important to machine learning since it can speed up the training of the model but also prevent overfitting of the model. The `DataLoader` can also shuffle the data, which means that it can load the data in a random order, which is also important to prevent overfitting.

1.3 Building and Training a model

The linear model that is used in the regression has a single input and a single output, therefore we can simplify the general vectorial form to a scalar equation, and it also contains a bias term. The mathematical representation of the model is:

$$y(x; w, b) = wx + b,$$

where w is the weight, x is the input, and b is the bias term. The learnable parameters of the model are the weight w and the bias b.

The parameters that correspond to the weights in the model are retrieved in the `LinearRegression.forward` method. The linear nueral network layer is is assigned to the `linear` class variable, and in the forward methos, the input `x` is passed through the layer and the output is returned.

To make predictions with the fitted model, you simply call the model with the input data. Under the hood this calls the `forward` method of the model. We can best explain this using an example. First you create an instance of the model with:

```
model = LinearRegression()
```

After fitting the model you can make predictions using the model by calling,

```
model(x_input)
```

which will return the predictions of the model.

1.4 Model evaluation

The mathematical equation to compute the coefficient of determination R^2 is given by:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2},$$

where y_i is the true value, $\hat{y}_i$ is the predicted value, and $\bar{y}$ is the mean of the true values. The nominator of the fraction is colloquially known as the sum of residual squares (RSS), and the denominator is known as the total sum of squares (TSS).

A R^2 value of 1 indicates that the model perfectly predicts the data, while a value of 0 indicates that the model does not predict the data at all.

2 Experiments

2.1 Different datasets

We generate the datasets with different values of d_max to see how the noise affects the model. We generate the datasets with d_max = {1, 10, 20}.

After generating the datasets we first plot the data to see that the data generation process was correct. A plot of the data is shown below.

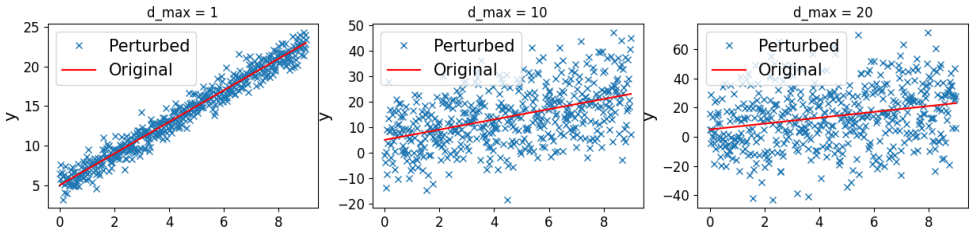
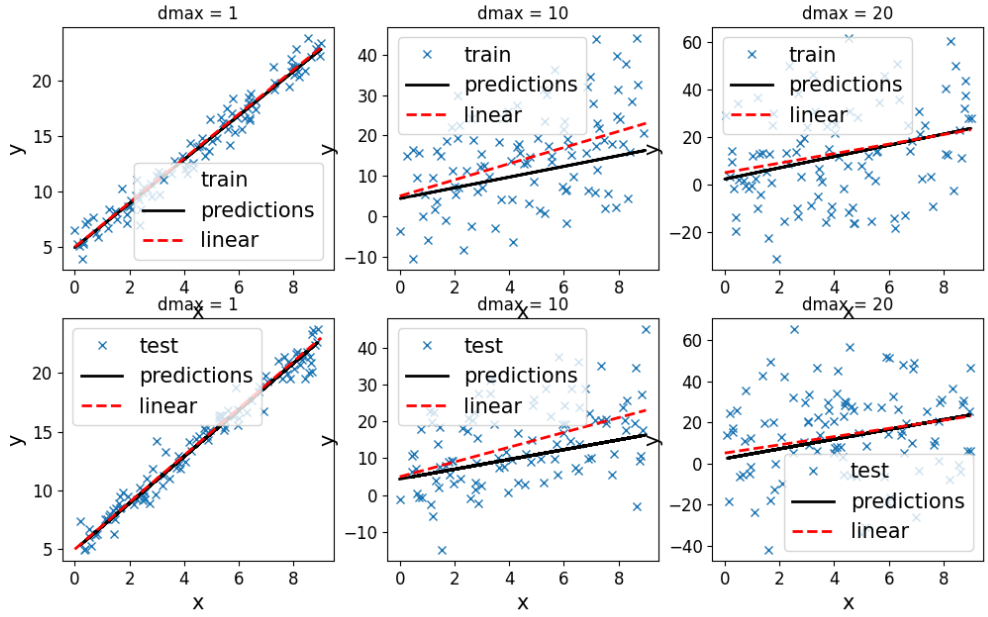


Figure 1: Generated datasets for different values of d_max = {1, 10, 20}

We can clearly observe that which the value of d_max increases the noise in the data increases.

Next we train the model on each of the datasets and plot the predictions of the model on the training data as well as the testing data. The plots are shown below.



The top row shows the predictions on the training data for different values of d_max, and the bottom row shows the predictions on the testing data for different values of d_max.

We can then look at the values of the trained parameters and see if the match with the actual values, i.e $m = 2$ and $b = 5$. The ouput is shown below

```
d_max = 1
Learned m: 1.9541736841201782
Learned b: 5.058658123016357

d_max = 10
Learned m: 1.4539545774459839
Learned b: 6.422358989715576

d_max = 20
Learned m: 2.035374641418457
Learned b: 3.4162681102752686
```

We observe that the model is able to learn the parameters of the linear function when the noise is low, i.e. d_max = 1. However, as the noise increases the model is not able to learn the parameters of the linear function correctly.

To provide a more quantitative measure of the performance of the models we compute the R^2, MAE, and MSE values for each of the models on the training and testing data. The output is shown below.

```
Dataset (dmax=1)
R2 score =  0.97191
MSE train =   1.0142    MAE train =   0.8114
MSE test  =   0.8314    MAE test  =   0.7315

Dataset (dmax=10)
R2 score =  0.19578
MSE train = 100.2779    MAE train =   8.1824
MSE test  =  91.3155    MAE test  =   7.2007

Dataset (dmax=20)
R2 score =  0.06730
MSE train = 420.1023    MAE train =  16.1607
MSE test  = 482.8817    MAE test  =  17.9652
```

This clearly shows that as the we increase the amount of noise in our data, that the R^2 decrease from almost 1 to 0. This illustrates that with low noise the model almost perfectly predicts the data, while with high noise the model does not predict the data at all. This is also reflected in the MSE and MAE values, where the values are very low for low noise and very high for high noise.

2.2 Scaled Data

When training the model over a larger interval with `high_val = 100` we observe the the loss per epoch show up as `nan`, which means not a number. This usally happens when there is some numerical error. When investigating this in detail by printing the loss for each minibatch when training, we notice that the losses per iteration become very large. This means that the gradients calculated on backpropagation will also become very large and until they become `inf` and finally `nan`. Once it is `nan` it will remain not a number and the model is unable to train.

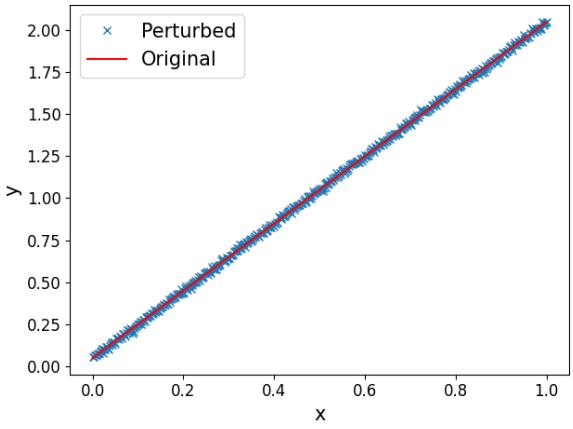
If we try to train the model with a smaller learning rate, the model is able to train. This is because the learning rate scales the large gradients down to a number that is representable by the computer, such that we do not get the numerical errors that we had before.

```
Dataset (Scaled Data Low Learning Rate)
R2 score =  0.99812
```

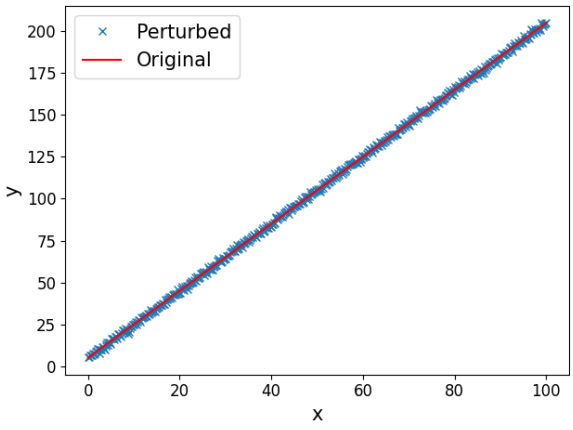
```
MSE train =   6.2022    MAE train =   2.0814
MSE test  =   6.0931    MAE test  =   1.9914
```

We observe from the evaluations metrics that the model is able to predict the data with high accuracy. However, adjusting the learning rate based on the values of the data is not manageable in practice. A better approach for this is to normalize the data such that the values are between 0 and 1, which can be done with the `PTMinMaxScaler` class.

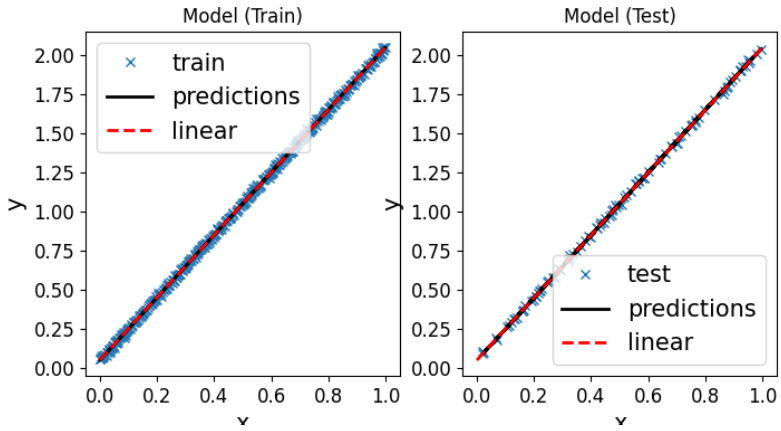
After scaling the x and y values. We first plot the data to see whether the scaling was done correctly. The plot is shown below.



We can see that the x-values are now scaled between 0 and 1, and the y-values are also scaled by the same factor. Next we test whether we can retrieve the original with inverse scaling. The plot is shown below.



Now that we have tested and verified that the scaling and inverse scaling works correctly, we train the model on the data. The evaluation metrics are shown below.



The plot shows that the model is able to predict the data well for the train and the test set. This is even more clearly illustrated by the evaluation metrics below.

```
Dataset (Scaled Data With Scaler)
R2 score =  0.99966
MSE train =   0.9583    MAE train =   0.7719
MSE test  =   1.0424    MAE test  =   0.8229
```

The R^2 value is almost 1, which means that the model almost perfectly predicts the data. This is also reflected in the MSE and MAE values, which are very low. This shows that scaling the data is a good way to ensure that the model is able to learn the data correctly.