Application of Machine Learning in Predicting Molecular Properties

By Edward Sun

Author Bio

Edward Sun is currently a high school junior attending Union County Magnet High School in Scotch Plains, New Jersey. He is, and has been, a National Chemistry Olympiad Finalist in 2022 and 2021, and has participated in the North Jersey Regional Science Fair, hosted by Nokia Bell Labs. Ever since he was a child, he was fascinated by chemistry, and in middle school, took an interest in computer science as well, and sought to find a way to unite the two subjects. In this paper, he utilizes machine learning algorithms to predict the melting points of organic compounds from their structures, and hopes that this research will build a foundation for many revolutionary steps to come in utilizing machine learning in diverse scientific subjects, ranging from medicine to astronomy.

Abstract

As developing new compounds and determining their properties is expensive and possibly dangerous, it becomes necessary to develop a model to predict the molecular properties without having to synthesize and test it experimentally. Two systematic ways to represent a compound are through the schematic diagram of the molecular structure and the simplified molecular-input line-entry system (SMILES). In this study, these representations were used to train two neural network models, a convolutional neural network (CNN) and a recurrent neural network (RNN), respectively, in order to predict the compounds' melting points. By representing the compounds as an image of the structure, the CNN was unsuccessful at fitting the given data, appearing to remain constant at around the average melting point of the given data. However, by representing the compounds as a systematically generated string of text, the RNN was successful at fitting the data, with an overall trend similar to the actual trend and a lower mean absolute error. However, the SMILES data used for the RNN does not contain orientation information, unlike the structure diagram data. For future studies, it may be possible to combine the two representations to reach a more accurate prediction model.

Keywords: Chemistry, computational, machine learning, molecular properties, neural network, convolutional, recurrent

Background

Chemistry is the science that deals with the properties, composition, and structures of substances, and how they transform into other substances. Therefore, it is one of the most useful fields for medicine by synthesizing new compounds as novel drugs. To effectively synthesize stable and reliable chemicals for medical treatment, it is important that the physicochemical properties of specific organic compounds can be determined in order to infer its use as a drug, such as solubility, membrane permeability, or stability (Mi et al., 2021). However, determining these properties can be both dangerous and costly. Specifically, it is inefficient and expensive to measure all the necessary properties of every organic compound in order to quantify its use as a drug. For example, determining properties like the precise melting point of a compound is difficult to do with a huge number of these compounds. In addition, some of these experiments can be hazardous or dangerous to the experimentalist. Thus, it becomes necessary to develop a model to predict such properties with relative accuracy without using experimental methods.

With the rise of machine learning, neural networks are becoming an increasingly popular model to predict various properties, including chemical properties (Salahinejad et al., 2013). While it is easy for a human to determine, for instance, around what range a compound's melting point is based only on its structure, it is far more difficult to get an accurate number for the melting point from the structure only. It is possible that machine learning could achieve similar or superior predictions on the melting point compared to human prediction. By analyzing the structure in a complex mathematical form, a neural network can take in the structure of a compound and output a number for the melting point. Despite the challenges to understand how the neural network calculates the melting point explicitly, the mechanism behind the learning process is nevertheless generally understood.

Previous research (Mi et al., 2021) has attempted to predict melting points using the simplified molecular-input line-entry system (SMILES) with various machine learning models. The SMILES system has the advantage over the actual structure schematic of being a single line of text; however, it does not contain complete information on the orientation, which could also affect the melting point. However, no studies have been performed using orientation information to predict melting point. Therefore, the objective of this study is to develop a machine learning model to predict melting point of chemicals and to investigate whether orientation information affects the melting point.

Method

In this study, Google Collab was used to run all the programs, using the built-in neural network foundation TensorFlow, interfacing with the Python 3 programming language with the Keras library.

First, the dataset of compound names, their SMILES data, and their melting points were obtained and extracted into a list (Bradley et al., 2019). Then, each entry of the list was formatted such that all SMILES text was converted into an array of numbers, with each number representing a unique letter of the text. Also, images of the structures were extracted from PubChem using an automated scraper and compiled separately Then, the list of formatted data was divided into training and validation inputs and outputs, with 20% of the original data being used for validation.

Afterward, the neural network architecture was constructed using Keras. Two models were used for this project: a bidirectional recurrent neural network and a convolutional neural network. For the first RNN, the first layer was an embedding layer, converting the indices of the inputs into dense vectors. The second and third layers were both bidirectional LSTM layers, and the final layer was a dense layer with only one node as the output. For the second CNN, an input layer of a 128 x 128 image was used, with the input images being converted into grayscale and rescaled to be a two dimensional tensor with values between 0 and 1. Then, two convolution and pooling layers were used, and then a flattening layer leading into three dense layers, with the final dense layer having only one node as output. In both, mean squared error was used as the loss function.

After all the data was collected, each network was trained for ten epochs, training on 90% of the original dataset and validating using the remaining 10%. After the models were fit, the loss values over time were plotted. Then, using a sample of 500 random inputs from the validation set, a scatterplot of melting point predictions were plotted, with the x-axis being the actual melting point and the y-axis being the predicted melting point. The scatterplot was compared to a straight line corresponding to the actual and predicted numbers being equal. To evaluate the accuracy of the models, the mean absolute error of the validation set in the final epoch of training was used, as well as the correlation coefficients of the scatterplots.

Results

The CNN model was not successful in predicting the melting point of chemicals. Specifically, it predicted almost no variance in the melting point compared to the actual data. The model does not seem to be sensitive to various predictors (Figure 1). The predicted melting point was consistently at 76.4 ^{o}C.

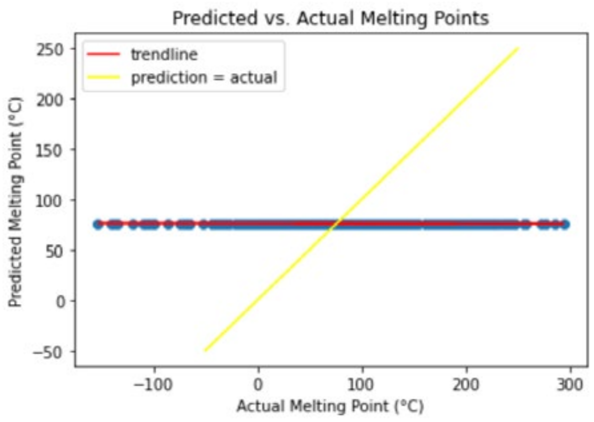


Figure 1. Predicted melting point against actual melting point from CNN. Each dot represents a prediction of one chemical. The yellow line represents when the prediction is equal to the actual value.

However, when zooming into the CNN graph (Figure 2), an opposite trend to the actual trend was observed. As the actual melting point increases, the predicted melting point slightly decreases instead, resulting in a negative correlation coefficient (Table 1).

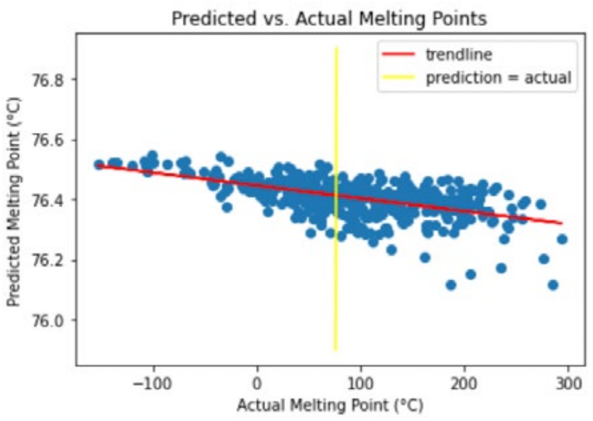


Figure 2. Zoomed in graph from Fig. 1 concentrated on predictions to show the slight downward trend of CNN predictions.

Compared to the CNN, predicted melting points by the RNN showed a higher sensitivity to various predictors and a higher variance. The overall trend is close to the actual trend, albeit with a less steep slope (Figure 3). The melting point predictions for the RNN appear to have a minimum of -50oC and a maximum of 204oC. Overall, the RNN showed a lower mean absolute error and a higher correlation coefficient than CNN (Table 1).

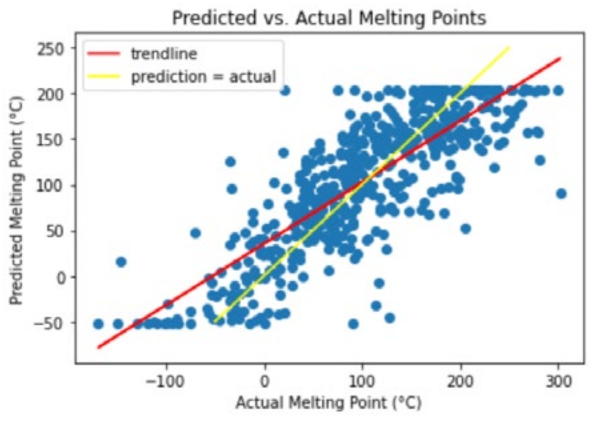


Figure 3. Predicted melting point against actual melting point from RNN. Each dot represents a prediction of one chemical.

Table 1. Correlation coefficient and mean absolute error of the two machine learning models.

Neural Network Type	Correlation Coefficient	Mean Absolute Error
Convolutional	-0.56775847	61.9855
Recurrent	0.81269509	38.5271

Discussion and Conclusion

Of the two neural network models, the recurrent neural network is better at prediction than the convolutional neural network. A possible explanation for the extremely poor performance of the CNN could be incorrect model design, such as inappropriate activation functions or loss functions. However, the dataset used may also be a contributing factor. The CNN used images of the compounds' structures as input, while the RNN used the SMILES data as input. The images used have a lot of empty space, which wastes a lot of computational power on the network on processing empty areas. Also, the scale of the molecule changes, since all images were originally in a 300 x 300 resolution, which could negatively affect the network's ability to recognise bonding structures.

The RNN is most likely useful for predicting melting points because of its ability to recognize sequences. Since organic compounds are typically made of functional groups attached to a long carbon chain, identifying these groups is extremely useful in determining the melting point. Another advantage of using the RNN is that it can recognize branches efficiently. Under the SMILES system, branches are enclosed in parentheses, which the RNN can interpret as a new branch. Since the RNN is primarily built for text processing, the SMILES system translates into regular text well, with parentheses acting like new "words" in the sentence.

Possibly a major factor contributing to the error in predictions for the RNN is the orientation information of the compound. SMILES cannot describe how a molecule is oriented, which makes it impossible to differentiate between stereoisomers such as diastereomers or enantiomers, even though this could have a large impact on the melting point of the compound. Another possibility is the limited data set; there was not enough data to form more accurate predictions. It is also unclear how the data from the original source was obtained, meaning there may be some sampling bias in the original dataset.

Future studies may use a more complex model for the predictions, as the current model does not contain that many neurons in order to save time and resources; a more complex model may be able to predict more accurately. Also, there are other molecular properties that can be predicted, such as boiling point, solubility, or heat of formation. With sufficient complexity, deep learning models may even be able to predict properties like toxicity or medical abilities by potentially finding previously unknown patterns in the structures of chemical compounds correlating to its effects on biological organisms.
It may even be possible to reverse engineer the mechanism of a trained neural network by adjusting inputs and observing the changes in the output values, such as how effective a compound is in treating cancer.
In conclusion, the most successful model of deep learning to predict molecular properties from the compound's structure, as was shown in this study, is a recurrent neural network trained using the SMILES format of text input.

References

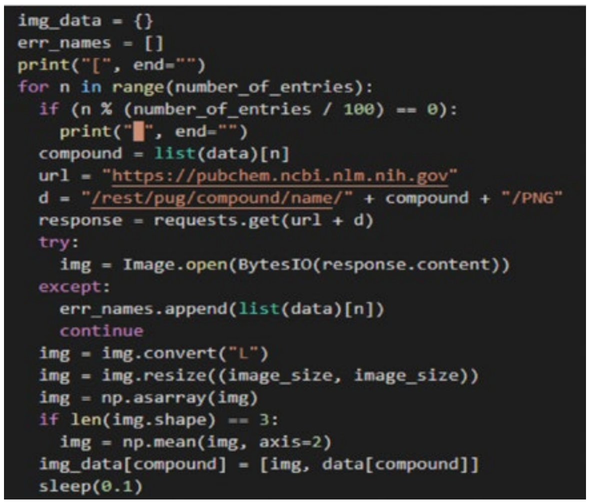
1. Bradley, J.-C., Williams, A., & Lang, A. (2019, October 4). *Jean-Claude Bradley open melting point dataset*. figshare. Retrieved January 4, 2022, from https://figshare.com/articles/dataset/Jean_Claude_Bradley_Open_Melting_Point_Datset/1031637

2. Mi, W., Chen, H., Zhu, D. A., Zhang, T., & Qian, F. (2021). Melting point prediction of organic molecules by deciphering the chemical structure into a natural language. Chemical Communications, 57(21), 2633–2636.https://doi.org/10.1039/d0cc07384a

3. Salahinejad, M., Le, T. C., & Winkler, D. A. (2013). Capturing the crystal: Prediction of enthalpy of sublimation, crystal lattice energy, and melting points of organic compounds. *Journal of Chemical Information and Modeling*, 53(1), 223–229. https://doi.org/10.1021/ci3005012

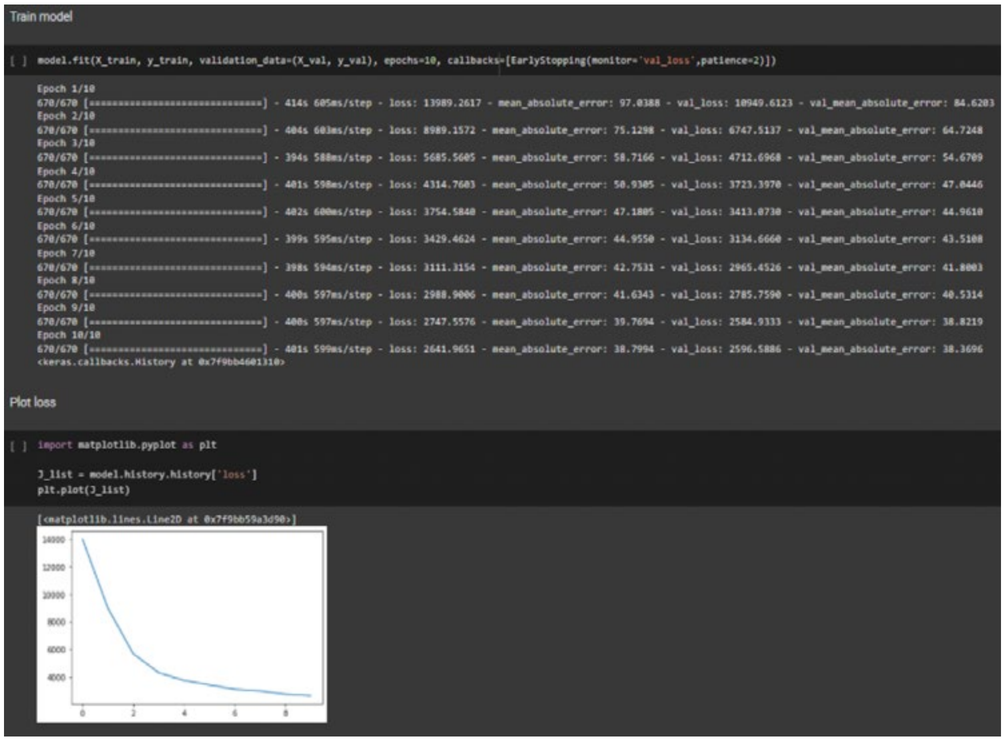
Appendix

Below is some of the code that I used for the project



```
img_data = {}
err_names = []
print("[", end="")
for n in range(number_of_entries):
  if (n % (number_of_entries / 100) == 0):
    print("█", end="")
  compound = list(data)[n]
  url = "https://pubchem.ncbi.nlm.nih.gov"
  d = "/rest/pug/compound/name/" + compound + "/PNG"
  response = requests.get(url + d)
  try:
    img = Image.open(BytesIO(response.content))
  except:
    err_names.append(list(data)[n])
    continue
  img = img.convert("L")
  img = img.resize((image_size, image_size))
  img = np.asarray(img)
  if len(img.shape) == 3:
    img = np.mean(img, axis=2)
  img_data[compound] = [img, data[compound]]
  sleep(0.1)
```

Code that iteratively obtains and processes images from PubChem for each compound in the dataset using the compound's conventional IUPAC name in the URL for PubChem's PUG-REST interface, for use in the CNN. First it assembles the URL, then after obtaining the image of the structure (printing errored requests if they occur), converts it into a Numpy array and inserts it into the global img_data dictionary that will serve as the input set for the network.

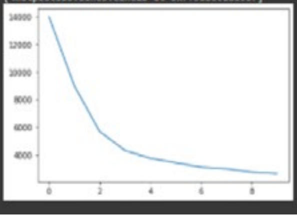


Train model

```
[ ]  model.fit(X_train, y_train, validation_data=(X_val, y_val), epochs=10, callbacks=[EarlyStopping(monitor='val_loss',patience=2)])

     Epoch 1/10
     670/670 [==============================] - 414s 605ms/step - loss: 13989.2617 - mean_absolute_error: 97.0388 - val_loss: 10949.6123 - val_mean_absolute_error: 84.6203
     Epoch 2/10
     670/670 [==============================] - 404s 603ms/step - loss: 8989.1572 - mean_absolute_error: 75.1298 - val_loss: 6747.5137 - val_mean_absolute_error: 64.7248
     Epoch 3/10
     670/670 [==============================] - 394s 588ms/step - loss: 5685.5605 - mean_absolute_error: 58.7166 - val_loss: 4712.6968 - val_mean_absolute_error: 54.6709
     Epoch 4/10
     670/670 [==============================] - 401s 598ms/step - loss: 4314.7603 - mean_absolute_error: 50.9305 - val_loss: 3723.3970 - val_mean_absolute_error: 47.0446
     Epoch 5/10
     670/670 [==============================] - 402s 600ms/step - loss: 3754.5840 - mean_absolute_error: 47.1805 - val_loss: 3413.0730 - val_mean_absolute_error: 44.9610
     Epoch 6/10
     670/670 [==============================] - 399s 595ms/step - loss: 3429.4624 - mean_absolute_error: 44.9550 - val_loss: 3134.6660 - val_mean_absolute_error: 43.5108
     Epoch 7/10
     670/670 [==============================] - 398s 594ms/step - loss: 3111.3154 - mean_absolute_error: 42.7531 - val_loss: 2965.4526 - val_mean_absolute_error: 41.8003
     Epoch 8/10
     670/670 [==============================] - 400s 597ms/step - loss: 2988.9006 - mean_absolute_error: 41.6343 - val_loss: 2785.7590 - val_mean_absolute_error: 40.5314
     Epoch 9/10
     670/670 [==============================] - 400s 597ms/step - loss: 2747.5576 - mean_absolute_error: 39.7694 - val_loss: 2584.9333 - val_mean_absolute_error: 38.8219
     Epoch 10/10
     670/670 [==============================] - 401s 599ms/step - loss: 2641.9651 - mean_absolute_error: 38.7994 - val_loss: 2596.5886 - val_mean_absolute_error: 38.3696
     <keras.callbacks.History at 0x7f9bb4601310>
```

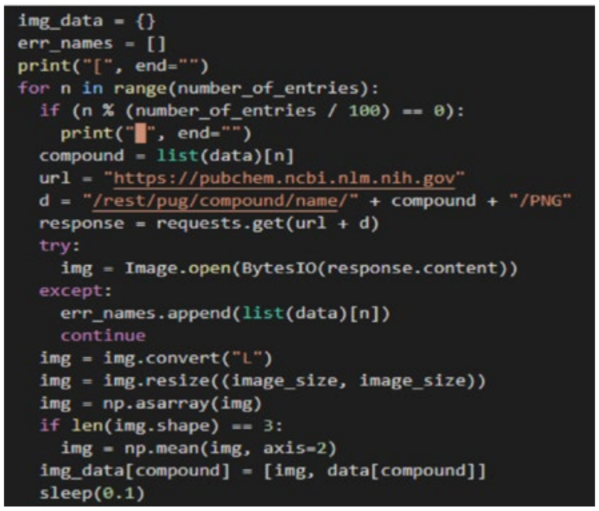
Plot loss

```
[ ]  import matplotlib.pyplot as plt

     J_list = model.history.history['loss']
     plt.plot(J_list)

     [<matplotlib.lines.Line2D at 0x7f9bb59a3d90>]
```



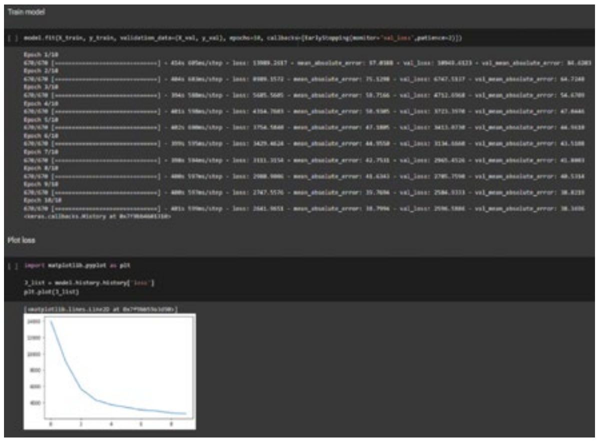
Appendix

Below is some of the code that I used for the project



```
img_data = {}
err_names = []
print("[", end="")
for n in range(number_of_entries):
  if (n % (number_of_entries / 100) == 0):
    print("█", end="")
  compound = list(data)[n]
  url = "https://pubchem.ncbi.nlm.nih.gov"
  d = "/rest/pug/compound/name/" + compound + "/PNG"
  response = requests.get(url + d)
  try:
    img = Image.open(BytesIO(response.content))
  except:
    err_names.append(list(data)[n])
    continue
  img = img.convert("L")
  img = img.resize((image_size, image_size))
  img = np.asarray(img)
  if len(img.shape) == 3:
    img = np.mean(img, axis=2)
  img_data[compound] = [img, data[compound]]
  sleep(0.1)
```

Code that iteratively obtains and processes images from PubChem for each compound in the dataset using the compound's conventional IUPAC name in the URL for PubChem's PUG-REST interface, for use in the CNN. First it assembles the URL, then after obtaining the image of the structure (printing errored requests if they occur), converts it into a Numpy array and inserts it into the global img_data dictionary that will serve as the input set for the network.



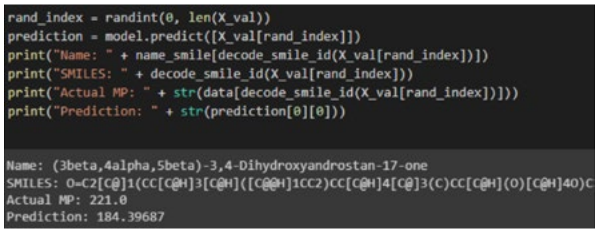
Train model

```
[ ] model.fit(X_train, y_train, validation_data=(X_val, y_val), epochs=10, callbacks=[EarlyStopping(monitor='val_loss',patience=2)])

    Epoch 1/10
    670/670 [==============================] - 454s 685ms/step - loss: 13989.2037 - mean_absolute_error: 87.0388 - val_loss: 10949.6323 - val_mean_absolute_error: 84.6383
    Epoch 2/10
    670/670 [==============================] - 404s 603ms/step - loss: 8989.1572 - mean_absolute_error: 75.5298 - val_loss: 6747.5537 - val_mean_absolute_error: 64.7248
    Epoch 3/10
    670/670 [==============================] - 394s 588ms/step - loss: 5685.5605 - mean_absolute_error: 58.7166 - val_loss: 4752.6968 - val_mean_absolute_error: 54.6709
    Epoch 4/10
    670/670 [==============================] - 401s 598ms/step - loss: 4354.7683 - mean_absolute_error: 50.9305 - val_loss: 3723.3970 - val_mean_absolute_error: 47.0446
    Epoch 5/10
    670/670 [==============================] - 402s 600ms/step - loss: 3754.5848 - mean_absolute_error: 47.1805 - val_loss: 3413.0738 - val_mean_absolute_error: 44.5618
    Epoch 6/10
    670/670 [==============================] - 399s 595ms/step - loss: 3429.4624 - mean_absolute_error: 44.9550 - val_loss: 3134.6668 - val_mean_absolute_error: 43.5188
    Epoch 7/10
    670/670 [==============================] - 398s 594ms/step - loss: 3131.3154 - mean_absolute_error: 42.7531 - val_loss: 2965.4526 - val_mean_absolute_error: 41.8083
    Epoch 8/10
    670/670 [==============================] - 400s 597ms/step - loss: 2988.9006 - mean_absolute_error: 41.6343 - val_loss: 2705.7598 - val_mean_absolute_error: 40.5314
    Epoch 9/10
    670/670 [==============================] - 400s 597ms/step - loss: 2747.5576 - mean_absolute_error: 39.7604 - val_loss: 2584.3333 - val_mean_absolute_error: 38.8239
    Epoch 10/10
    670/670 [==============================] - 401s 599ms/step - loss: 2641.9653 - mean_absolute_error: 38.7994 - val_loss: 2596.5866 - val_mean_absolute_error: 38.1696
    <keras.callbacks.History at 0x7f9b64481310>
```

Plot loss

```
[ ] import matplotlib.pyplot as plt

    l_list = model.history.history['loss']
    plt.plot(l_list)

    [<matplotlib.lines.Line2D at 0x7f9b6b53a1d50>]
```

Top: Training output of RNN; for each epoch of training, the loss and mean absolute error are printed as each data entry is tested; after the training set for the epoch is completed, the validation subset is used on the network and printed to the right.
Bottom: plotted graph of loss value over time (mean squared error of melting point prediction) using Matplotlib. Loss decreases exponentially as the number of epochs increases.

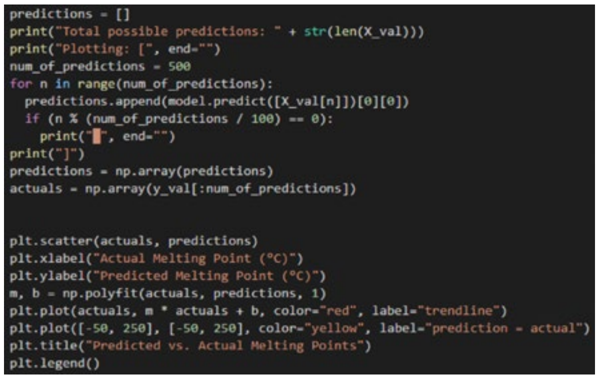


```
rand_index = randint(0, len(X_val))
prediction = model.predict([X_val[rand_index]])
print("Name: " + name_smile[decode_smile_id(X_val[rand_index])])
print("SMILES: " + decode_smile_id(X_val[rand_index]))
print("Actual MP: " + str(data[decode_smile_id(X_val[rand_index])]))
print("Prediction: " + str(prediction[0][0]))
```

```
Name: (3beta,4alpha,5beta)-3,4-Dihydroxyandrostan-17-one
SMILES: O=C2[C@]1(CC[C@H]3[C@H]([C@@H]1CC2)CC[C@H]4[C@]3(C)CC[C@H](O)[C@H]4O)C
Actual MP: 221.0
Prediction: 184.39687
```

Random prediction from RNN, chooses random entry in the original dataset and feeds it into the network, then prints output.

Here, the selected compound is (3β,4α,5β)-3,4-dihydroxyandrostan-17-one, a steroid with a melting point of about 221.0ºC. The melting point predicted by the RNN was about 184.4ºC. The error in this instance is 36.6ºC, close to the mean absolute error of the RNN.



```
predictions = []
print("Total possible predictions: " + str(len(X_val)))
print("Plotting: [", end="")
num_of_predictions = 500
for n in range(num_of_predictions):
  predictions.append(model.predict([X_val[n]])[0][0])
  if (n % (num_of_predictions / 100) == 0):
    print("█", end="")
print("]")
predictions = np.array(predictions)
actuals = np.array(y_val[:num_of_predictions])


plt.scatter(actuals, predictions)
plt.xlabel("Actual Melting Point (°C)")
plt.ylabel("Predicted Melting Point (°C)")
m, b = np.polyfit(actuals, predictions, 1)
plt.plot(actuals, m * actuals + b, color="red", label="trendline")
plt.plot([-50, 250], [-50, 250], color="yellow", label="prediction = actual")
plt.title("Predicted vs. Actual Melting Points")
plt.legend()
```

Code to generate a scatterplot of 500 random predictions of both the CNN and RNN using Matplotlib. First it generates all its predictions, then it plots all the points using the predicted melting points as the y-axis and the actual melting points as the x-axis. Then, it generates a linear curve of best fit for the data, and plots this line, along with another line that represents y = x, i.e. when the prediction is equal to the actual value.