Tarek Nabih

Prof. Chinmay Hegde

ECE-UY 4563

Introduction to Machine Learning

Machine Learning Model for Quantum Experiments Simulation Prediction

Introduction

Quantum experiments have long been a cornerstone in understanding the intricacies of atomic and subatomic phenomena. Traditional methods of simulating these experiments, however, can be computationally intensive and time-consuming. Recent advancements in machine learning, particularly deep learning, offer promising avenues for simulating quantum experiments with reduced computational costs and increased accuracy. This project aims to streamline the process traditionally undertaken by scientists using the 'spinnev' program, which is known for being time-intensive and slow to yield results. By training a deep learning model to interpret and instantly respond to data inputs, the project offers a more efficient alternative. Additionally, the model is designed to furnish scientists with insightful visualizations. For example, it can illustrate the impact of variables like magnets or power on the system, leveraging tools such as Matplotlib for graphical representation. Furthermore, the model enables simultaneous comparison of two parameters against the resulting 'spin' data, employing contour plots for a more nuanced and comprehensive analysis. This report presents a novel deep learning model trained on data from the SpinEvolution platform, aiming to predict the outcomes of quantum experiments in a fraction of the traditional time. You can find more information about

the project through the video: Here is a video describing the project in details:

Tarek Nabih Machine learning final

Methods

Data Collection

Data was generated from the SpinEvolution platform, a renowned tool for simulating quantum experiments. A script, executed on NYU's Jubail supercomputer, facilitated the collection of 500,000 experiments. Each experiment is characterized by parameters including 'H1_freq', 'spinning_freq', and more, all of which affect the spin outcome. As shown in the Image below, the generated data is ready for the model to train on.

combined_2used_50000

110.73241748165100	0.43063480564611000	1137.7293504027000	90.17903050256860	-469449.52956788500	867221.891696739	78.08024009275970	2288.7899709984600
381.4065178447300	0.5857224344182690	1420.759101301180	39.71986613819010	-260705.3298699340	786575.4371431010	142.48425403566100	5199.595486894470
560.8616191502920	0.28292128803199000	1296.4856243930500	99.28275374072930	-457210.797462638	-216835.22453573900	58.39279918701670	30373.032271649000
669.9791765057880	0.9795190199203920	404.01039918328800	132.57769863861500	279953.4530409020	1058426.2144456900	26.73492804921120	23175.597558306100
923.6348850113720	0.1499976211822100	575.5601240528510	164.90200372603900	-153151.6941199570	-378715.6364349940	162.3315085235810	2121.504899416830
608.5454402974990	0.9624437226905250	1137.3241443674000	156.74728004392400	11901.9550992765	-338888.7325839540	1.873915112885830	43248.50117033970
79.66670964756450	0.6709181392984850	1338.2935808381200	161.49396296865000	303824.20419638800	-14656.744368674500	151.1653067900680	6734.615462496110
138.22081070341200	0.6409426812423400	386.84771813926800	65.52530855053450	259683.5088144810	-446072.38635835700	145.09677833310200	21281.17915298730
1248.5538205884700	0.8047869295126160	516.5760525547200	172.01125762272600	-457230.92498041200	1115302.9229107200	88.3937818141683	30529.41833740200
370.0688729144540	0.7699944505173020	821.66445456001	112.69595190538200	-478787.0608802520	540895.2382821880	0.5300421548633750	49203.2613155042

The script functions by acquiring a set of randomly generated values, sourced from an auxiliary script, and subsequently utilizes these values as input parameters for the 'spinnev' command. This process facilitates the generation of corresponding outcomes based on the provided inputs.

```
headers = ['H1_freq', 'spinning_freq' ,'power_1_1_1' ,'cs_beta_2', 'ss_iso_1_2',
'ss_ani_1_2', 'ss_beta_1_2', 'ss_ani_1_3', 'ss_beta_1_3', 'mw', 'T1e1']


ranges =
[(50,1300),(0.05,80),(0,1500),(0,180),(-600e3,600e3),(-1200e3,1200e3),(0,180),(0,50e3)
,(0,180),(-1200e3,1200e3),(0.01,50)]


# Run the spinnev command
```

```
os.system(f"spinev Tarek")

with open('input.txt', 'r') as f1:

    next(f1)  # Skip the header

    data1 = [line.strip().split() for line in f1]


# Open the second file and read in the data

with open('Tarek_re.dat', 'r') as f2:

    data2 = [line.strip().split()[1:] for line in f2]


combined_data = [d1 + d2 for d1, d2 in zip(data1, data2)]


with open('combined.csv', 'w', newline='') as f_out:

    writer = csv.writer(f_out)

    writer.writerows(combined_data)
```

Data Preprocessing

Upon collection, the dataset underwent normalization using the Min-Max scaling technique. This step ensured feature values were transformed into a consistent range, pivotal for the convergence of deep learning models.

Model Architecture

The deep learning model employed a sequential architecture. Beginning with a dense layer of 256 nodes employing the ReLU activation function, the architecture was iteratively

refined. Subsequent 8 layers, each containing 257 nodes, incorporated similar activation functions, batch normalization, and dropout mechanisms to mitigate overfitting. The final layer, a dense node, predicts the spin outcome.

Training and Hyperparameter Tuning

The Adam optimizer, with a learning rate of 0.0003, was chosen after preliminary testing. Training involved 5-fold cross-validation to ensure robustness. Early stopping was a pivotal strategy, monitoring the validation loss to prevent prolonged, redundant training phases.

The script

```
      data = np.loadtxt('data.csv', delimiter=',')

      X = data[:, :-1]
      y = data[:, -1]

      scaler = MinMaxScaler()
      X = scaler.fit_transform(X)

      joblib.dump(scaler, 'my_scaler11.pkl')

      def create_model():


         layers = [
             Dense(256, activation='relu', input_shape=(X.shape[1],),
kernel_regularizer=l2(0.5)),
             BatchNormalization(),

         ]

         layers_nodes = [257, 257, 257, 257, 257, 257, 257, 257]
```

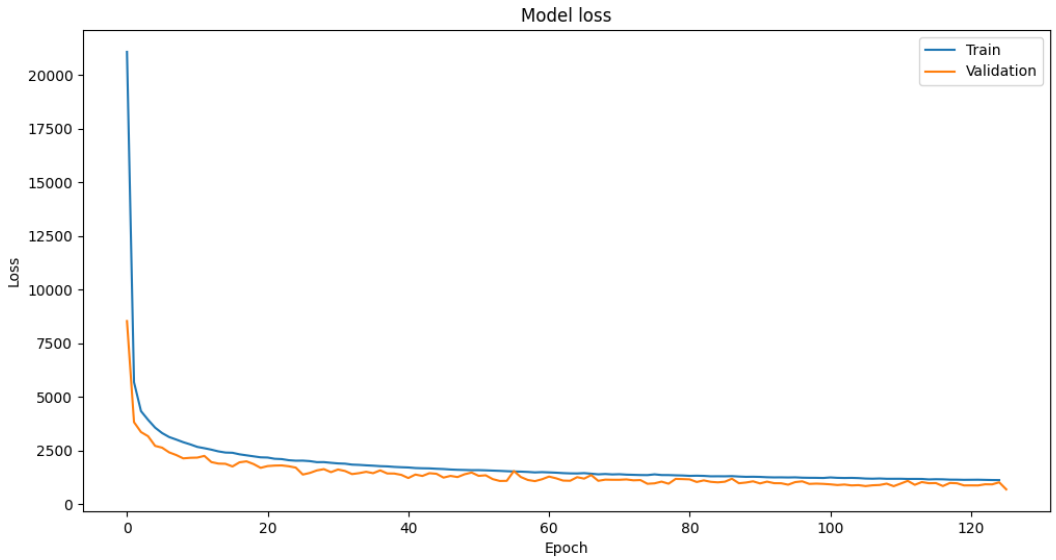
```
        for nodes in layers_nodes:
            layers.extend([
                Dense(nodes, activation='relu', kernel_regularizer=l2(0.5)),
                BatchNormalization(),
                Dropout(0.03),
            ])


        layers.append(Dense(1))

        model = Sequential(layers)
        model.compile(optimizer=Adam(learning_rate=0.0003),
loss='mean_squared_error')

        return model


    early_stopping = EarlyStopping(monitor='val_loss', patience=20)

    num_folds = 5
    kf = KFold(n_splits=num_folds, shuffle=True, random_state=42)

    model = create_model()

    fold_no = 1
    y_tests = []
    y_preds = []
    losses = []
    history = {'loss': [], 'val_loss': []}
    for train_index, test_index in kf.split(X, y):

        h = model.fit(X[train_index], y[train_index],
validation_data=(X[test_index], y[test_index]), epochs=25, batch_size=512,
callbacks=[early_stopping])

        history['loss'].extend(h.history['loss'])
        history['val_loss'].extend(h.history['val_loss'])


        y_pred = model.predict(X[test_index])
```

```
        y_preds.append(y_pred)
        y_tests.append(y[test_index])


        loss = model.evaluate(X[test_index], y[test_index], verbose=0)
        losses.append(loss)
        print(f'Loss for fold {fold_no}: {loss}')


        fold_no = fold_no + 1

    mean_loss_cv = np.mean(losses)
    print(f'Mean loss during cross-validation: {mean_loss_cv}')

    y_tests = np.concatenate(y_tests)
    y_preds = np.concatenate(y_preds)
    total_r2 = r2_score(y_tests, y_preds)
    total_mse = mean_squared_error(y_tests, y_preds)
    print(f'Total R2 Score: {total_r2}')
    print(f'Total Mean Squared Error: {total_mse}')

    model.save('my_model37.h5')

    data_test = np.loadtxt('testing.csv', delimiter=',')
    X_test = data_test[:, :-1]
    y_test = data_test[:, -1]

    X_test = scaler.transform(X_test)

    test_loss = model.evaluate(X_test, y_test, verbose=0)
    history['val_loss'].append(test_loss)

    y_test_pred = model.predict(X_test)

    r2_test = r2_score(y_test, y_test_pred)
    mse_test = mean_squared_error(y_test, y_test_pred)
    print(f'Test R2 Score: {r2_test}')
    print(f'Test Mean Squared Error: {mse_test}')



    plt.figure(figsize=(12, 6))
```

```
plt.plot(history['loss'])
plt.plot(history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper right')
plt.savefig('hello37')
```

Results

The model's inaugural performance produced an \(R^2 \) value of 0.8. Subsequent hyperparameter tuning and architectural refinements propelled this score to 0.97, highlighting the model's aptitude in predicting quantum experiment outcomes. Notably, when juxtaposed with other machine learning models, such as random forest, the deep learning model displayed superior performance and operational feasibility. The image below shows the train & test mean loss with the number of epochs.



Discussion

While the model's prowess in simulating quantum experiments is evident, it is crucial to understand its superiority. Traditional models, like the random forest, not only lagged in performance but also became computationally cumbersome as the dataset grew. Conversely, the deep learning model, even with its intricate architecture, offered a more efficient and scalable solution.

The choice of hyperparameters, particularly the learning rate and the decision to employ K-fold cross-validation, was instrumental in the model's success. An initial low learning rate hindered the model's learning capability. Adjustments to this, along with a shift to K-fold validation, substantially enhanced the model's accuracy.

Interactive User Interface for Spin Value Prediction

The latest development in our machine learning project involves the creation of a script with an interactive user interface (UI) that predicts spin values based on user inputs. This advancement integrates various Python libraries, including TensorFlow for machine learning, NumPy for numerical operations, Joblib for model loading, and Matplotlib along with Tkinter for graphical display and UI development.

implementation Details

1. Model and Scaler Loading:

- TensorFlow's `tf.keras.models.load_model` function loads the pre-trained model 'my_model34.h5'.

- Joblib loads the scaling parameters from 'my_scaler.pkl' for input normalization.

2. User Interface Setup:

- Tkinter constructs the UI, with a layout comprising input and plot frames.

- Users input parameters through entry widgets, each corresponding to a specific feature like 'H1_freq', 'spinning_freq', etc.

3. Input Validation and Preparation:

- The input values are validated against defined ranges to ensure data integrity.
- Data is reshaped and normalized using the loaded scaler to match the model's expected input format.

4. Plotting Functionality:

- The script offers two plot types: Normal and Contour.
- For 'Normal' plots, it generates a line graph for a selected parameter against the predicted output.
- The 'Contour' plot option allows users to visualize the relationship between two parameters and the predicted output, offering a more complex and detailed view.

5. Graphical Representation:

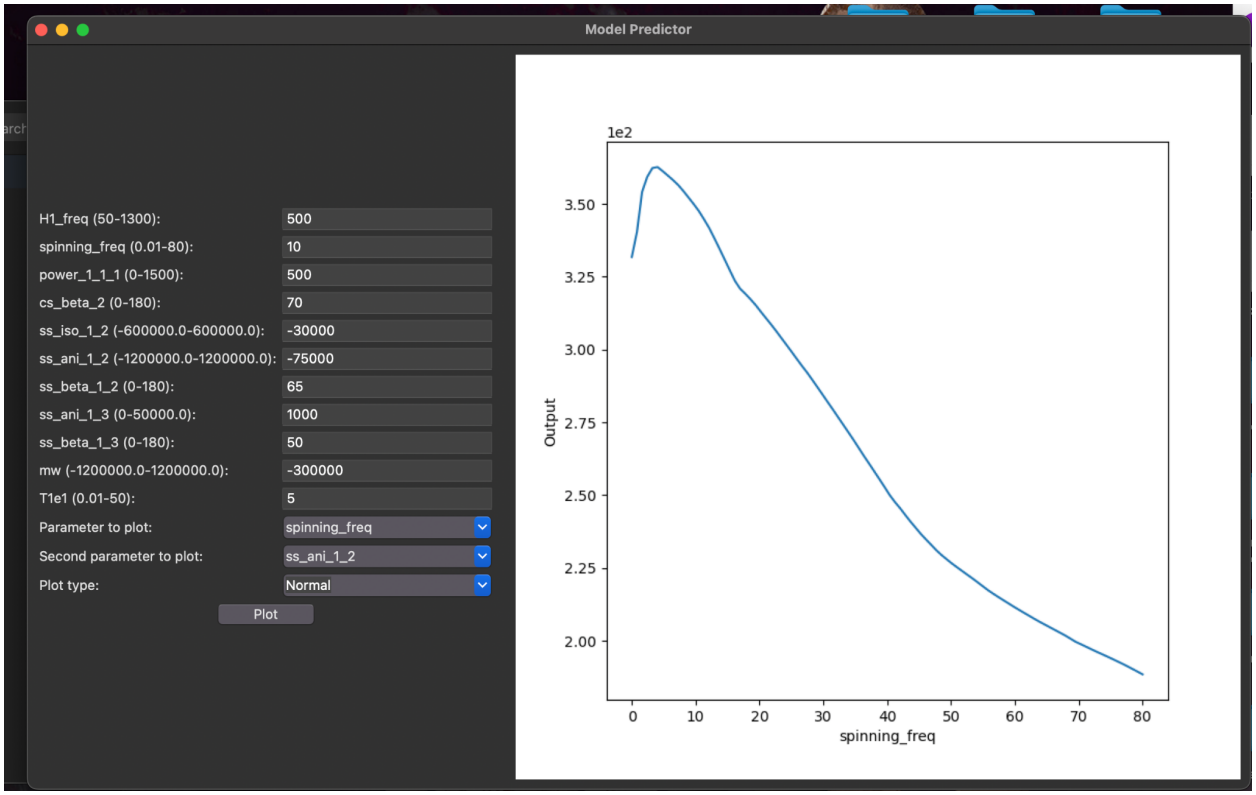
- Matplotlib creates the figures and axes for the plots.
- The 'contourf' function is used for contour plots, providing a color-coded representation of the data with 30 levels for granularity.

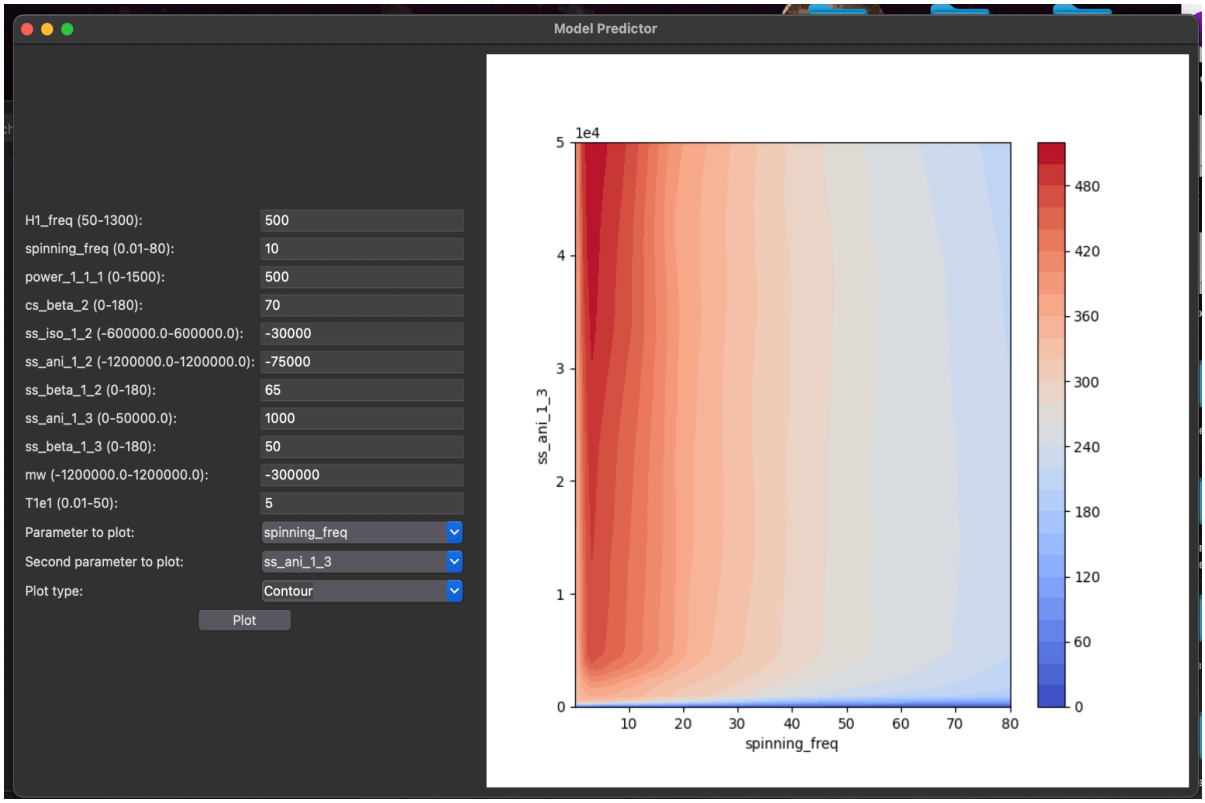
- A color bar is added for reference in contour plots.

6. Interactive Elements:

- Combobox widgets enable users to select parameters for plotting.
- A 'Plot' button triggers the prediction and plotting process based on the current inputs and selected options.

The first image illustrates the correlation between a specific parameter and the spin value, while the second image depicts the relationship between two parameters and the spinnev value using a contour plot.





Conclusion

The burgeoning field of quantum experiments stands to benefit immensely from the confluence of deep learning. The presented model, trained on extensive data from the SpinEvolution platform, showcases the potential of machine learning in revolutionizing quantum experiment simulations. Future endeavors may focus on refining the model further, exploring other architectures, and potentially integrating quantum computing for even more precise simulations.