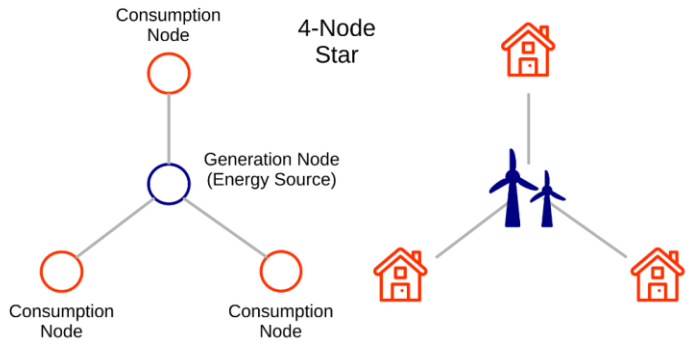
# Project: Deep Learning

## Objective:

The Main objective of the analysis is to focus on a specific type of Deep Learning and the benefits that this analysis brings to the business or stakeholders of this data is to predict the stability of grid more accurately.

## Brief description of the data set:

This project is based on the "Electrical Grid Stability Simulated Dataset", created by Vadim Arzamasov (Karlsruher Institut für Technologie, Karlsruhe, Germany) and donated to the University of California (UCI) Machine Learning Repository (link), where it is currently hosted. Learning from the project (link), I explored the utilization of GridSearchCV to perform the hyperparameter tuning with the same dataset to do more analysis in deep learning.
This dataset has a synthetic nature and contains results from simulations of grid stability for a reference 4-node star network as follows,




The original dataset contains 10,000 observations. As the reference grid is symmetric, the dataset can be augmented in 3! (3 factorial) times, or 6 times, representing a permutation of the three consumers occupying three consumer nodes. The augmented version has then 60,000 observations.

```
data.head()
```

| | tau1 | tau2 | tau3 | tau4 | p1 | p2 | p3 | p4 | g1 | g2 | g3 | g4 | stab | stabf |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 2.959060 | 3.079885 | 8.381025 | 9.780754 | 3.763085 | -0.782604 | -1.257395 | -1.723086 | 0.650456 | 0.859578 | 0.887445 | 0.958034 | 0.055347 | unstable |
| 1 | 9.304097 | 4.902524 | 3.047541 | 1.369357 | 5.067812 | -1.940058 | -1.872742 | -1.255012 | 0.413441 | 0.862414 | 0.562139 | 0.781760 | -0.005957 | stable |
| 2 | 8.971707 | 8.848428 | 3.046479 | 1.214518 | 3.405158 | -1.207456 | -1.277210 | -0.920492 | 0.163041 | 0.766689 | 0.839444 | 0.109853 | 0.003471 | unstable |
| 3 | 0.716415 | 7.669600 | 4.486641 | 2.340563 | 3.963791 | -1.027473 | -1.938944 | -0.997374 | 0.446209 | 0.976744 | 0.929381 | 0.362718 | 0.028871 | unstable |
| 4 | 3.134112 | 7.608772 | 4.943759 | 9.857573 | 3.525811 | -1.125531 | -1.845975 | -0.554305 | 0.797110 | 0.455450 | 0.656947 | 0.820923 | 0.049860 | unstable |

This dataset contains 12 primary predictive features and two dependent variables.

'tau1' to 'tau4': the reaction time of each network participant, a real value within the range 0.5 to 10 ('tau1' corresponds to the supplier node, 'tau2' to 'tau4' to the consumer nodes). 'p1' to 'p4': nominal power produced (positive) or consumed (negative) by each network participant, a real value within the

range -2.0 to -0.5 for consumers ('p2' to 'p4'). As the total power consumed equals the total power generated, p1 (supplier node) = - (p2 + p3 + p4).

'g1' to 'g4': price elasticity coefficient for each network participant, a real value within the range 0.05 to 1.00 ('g1' corresponds to the supplier node, 'g2' to 'g4' to the consumer nodes; 'g' stands for 'gamma').

'stab': the maximum real part of the characteristic differential equation root (if positive, the system is linearly unstable; if negative, linearly stable).

'stabf': a categorical (binary) label ('stable' or 'unstable').

Here is the description of the features:

```
data.describe()
```

| | tau1 | tau2 | tau3 | tau4 | p1 | p2 | p3 | p4 | g1 | g2 |
|---|---|---|---|---|---|---|---|---|---|---|
| **count** | 60000.000000 | 60000.000000 | 60000.000000 | 60000.000000 | 60000.000000 | 60000.000000 | 60000.000000 | 60000.000000 | 60000.000000 | 60000.000000 |
| **mean** | 5.250000 | 5.250001 | 5.250001 | 5.250001 | 3.750000 | -1.250000 | -1.250000 | -1.250000 | 0.525000 | 0.525000 |
| **std** | 2.742434 | 2.742437 | 2.742437 | 2.742437 | 0.752129 | 0.433017 | 0.433017 | 0.433017 | 0.274244 | 0.274243 |
| **min** | 0.500793 | 0.500141 | 0.500141 | 0.500141 | 1.582590 | -1.999945 | -1.999945 | -1.999945 | 0.050009 | 0.050028 |
| **25%** | 2.874892 | 2.875011 | 2.875011 | 2.875011 | 3.218300 | -1.624997 | -1.624997 | -1.624997 | 0.287521 | 0.287497 |
| **50%** | 5.250004 | 5.249981 | 5.249981 | 5.249981 | 3.751025 | -1.249996 | -1.249996 | -1.249996 | 0.525009 | 0.525007 |
| **75%** | 7.624690 | 7.624896 | 7.624896 | 7.624896 | 4.282420 | -0.874993 | -0.874993 | -0.874993 | 0.762435 | 0.762490 |
| **max** | 9.999469 | 9.999837 | 9.999837 | 9.999837 | 5.864418 | -0.500025 | -0.500025 | -0.500025 | 0.999937 | 0.999982 |

## Brief summary of data exploration:

At first, I needed to check all features whether there are any null values as follows.

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 60000 entries, 0 to 59999
Data columns (total 14 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   tau1    60000 non-null  float64
 1   tau2    60000 non-null  float64
 2   tau3    60000 non-null  float64
 3   tau4    60000 non-null  float64
 4   p1      60000 non-null  float64
 5   p2      60000 non-null  float64
 6   p3      60000 non-null  float64
 7   p4      60000 non-null  float64
 8   g1      60000 non-null  float64
 9   g2      60000 non-null  float64
 10  g3      60000 non-null  float64
 11  g4      60000 non-null  float64
 12  stab    60000 non-null  float64
 13  stabf   60000 non-null  object 
dtypes: float64(13), object(1)
memory usage: 6.4+ MB
```

The feature named 'stabf' is categorical (binary) label ('stable' or 'unstable'). I needed to map to 0 and 1.

```
map1 = {'unstable': 0, 'stable': 1}
data['stabf'] = data['stabf'].replace(map1)
```

```
data.head()
```

| | tau1 | tau2 | tau3 | tau4 | p1 | p2 | p3 | p4 | g1 | g2 | g3 | g4 | stab | stabf |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **51024** | 6.153085 | 4.013520 | 2.789390 | 9.355055 | 2.522928 | -0.822173 | -0.658987 | -1.041768 | 0.531863 | 0.996241 | 0.823250 | 0.578474 | 0.049802 | 0 |
| **38960** | 7.948758 | 3.485990 | 9.630111 | 3.580804 | 4.696563 | -1.953521 | -1.283568 | -1.459475 | 0.445510 | 0.457402 | 0.455086 | 0.600302 | 0.016376 | 0 |
| **21497** | 1.935822 | 0.824151 | 6.640168 | 1.853827 | 3.326405 | -1.632641 | -0.801972 | -0.891792 | 0.064151 | 0.520358 | 0.102860 | 0.966215 | -0.035227 | 1 |
| **31994** | 1.830395 | 7.286385 | 3.366041 | 8.784404 | 3.189813 | -0.988087 | -0.847310 | -1.354415 | 0.921142 | 0.831543 | 0.165711 | 0.418436 | 0.009617 | 0 |
| **22423** | 7.777096 | 1.802715 | 3.947211 | 3.925122 | 3.969584 | -1.125899 | -1.621273 | -1.222412 | 0.528602 | 0.548309 | 0.297200 | 0.847018 | 0.009683 | 0 |

It is important to find the correlation among all the features as follows,

| | tau1 | tau2 | tau3 | tau4 | p1 | p2 | p3 | p4 | g1 | g2 | g3 | g4 | stab | stabf |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **tau1** | 1.000000 | -0.002550 | -0.002550 | -0.002550 | 0.027183 | -0.015739 | -0.015739 | -0.015739 | 0.010521 | 0.006522 | 0.006522 | 0.006522 | 0.275761 | -0.234898 |
| **tau2** | -0.002550 | 1.000000 | 0.005554 | 0.005554 | 0.003004 | -0.004473 | -0.000372 | -0.000372 | -0.005832 | 0.009865 | 0.002102 | 0.002102 | 0.283417 | -0.241049 |
| **tau3** | -0.002550 | 0.005554 | 1.000000 | 0.005554 | 0.003004 | -0.000372 | -0.004473 | -0.000372 | -0.005832 | 0.002102 | 0.009865 | 0.002102 | 0.283417 | -0.241049 |
| **tau4** | -0.002550 | 0.005554 | 0.005554 | 1.000000 | 0.003004 | -0.000372 | -0.000372 | -0.004473 | -0.005832 | 0.002102 | 0.002102 | 0.009865 | 0.283417 | -0.241049 |
| **p1** | 0.027183 | 0.003004 | 0.003004 | 0.003004 | 1.000000 | -0.578983 | -0.578983 | -0.578983 | 0.000721 | 0.000341 | 0.000341 | 0.000341 | 0.010278 | -0.009938 |
| **p2** | -0.015739 | -0.004473 | -0.000372 | -0.000372 | -0.578983 | 1.000000 | 0.002833 | 0.002833 | -0.000417 | -0.002141 | 0.000774 | 0.000774 | -0.005951 | 0.005754 |
| **p3** | -0.015739 | -0.000372 | -0.004473 | -0.000372 | -0.578983 | 0.002833 | 1.000000 | 0.002833 | -0.000417 | 0.000774 | -0.002141 | 0.000774 | -0.005951 | 0.005754 |
| **p4** | -0.015739 | -0.000372 | -0.000372 | -0.004473 | -0.578983 | 0.002833 | 0.002833 | 1.000000 | -0.000417 | 0.000774 | 0.000774 | -0.002141 | -0.005951 | 0.005754 |
| **g1** | 0.010521 | -0.005832 | -0.005832 | -0.005832 | 0.000721 | -0.000417 | -0.000417 | -0.000417 | 1.000000 | 0.004718 | 0.004718 | 0.004718 | 0.282774 | -0.197664 |
| **g2** | 0.006522 | 0.009865 | 0.002102 | 0.002102 | 0.000341 | -0.002141 | 0.000774 | 0.000774 | 0.004718 | 1.000000 | -0.006939 | -0.006939 | 0.293684 | -0.218015 |

'stabf' is the primary target feature; therefore, I can drop the feature 'stab'.

Now, the dataset needs to be split. Here, 10% is test set and 90% is training set.

```
X = data.iloc[:, :12]
y = data["stabf"].values

# Split the data to Train, and Test (90%, 10%)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=42)
```

Before building the deep learning model, features are needed to be scaled with StandardScaler function.

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

## Model building with different hyperparameters:

It is difficult to choose the right layers and nodes and activation function for the deep learning model. Therefore, I used GridSearchCV to find out the right values. I created a function which takes the input layers and activation function as inputs and builds the model for all corresponding layers and activation functions.

```
Input_dimensions = X.shape[1]
def model_creation(layers,activation):
    dl_model = Sequential()
    for l,nodes in enumerate(layers):
        if l==0:
            dl_model.add(Dense(nodes,input_dim =Input_dimensions,activation = activation))
        else:
            dl_model.add(Dense(nodes,activation = activation))

    dl_model.add(Dense(1,activation='sigmoid'))

    dl_model.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
    return dl_model

dl_model = KerasClassifier(build_fn=model_creation, verbose =0)
```

Then, hyperparameter tunning is taken place with GridSearchCV to get the best parameters for this particular deep learning algorithm.

```
layers = [[24], [24,24], [24,24,12], [24,24,12,12]]
activations = ['sigmoid','relu']
param_grid = dict(layers = layers,activation=activations, batch_size = [128,256], epochs=[50])
grid = GridSearchCV(estimator=dl_model,param_grid=param_grid)
```

## Best Deep Learning model in terms of accuracy:

The best parameters of the deep learning model are as follows,

```
[results_grid.best_score_,results_grid.best_params_]
```

```
[0.9754999995231628,
 {'activation': 'relu',
  'batch_size': 128,
  'epochs': 50,
  'layers': [24, 24, 12, 12]}]
```

Here the best layers that we found are four layers, the first layer has 24 nodes, second layer has 24 nodes, third layer has 12 nodes and fourth layer has 12 nodes.

## Findings and Insights:

The main findings of this project are to utilize the hyperparameter tuning to get the right layers for deep learning algorithm and get the better accuracy of the predictions. I created confusion matrix to get the accuracy of my model, here is the result,

| | Predicted Unstable | Predicted Stable |
|---|---|---|
| **Actual Unstable** | 3827 | 31 |
| **Actual Stable** | 146 | 1996 |

```
print(f'Accuracy per the confusion matrix: {((cm.iloc[0, 0] + cm.iloc[1, 1]) / len(y_test) * 100):.2f}%')
```

```
Accuracy per the confusion matrix: 97.05%
```

## Next steps:

This exercise provides a better understanding of deep learning approach in case of predicting grid stability. I used GridSearchCV library to get the best parameters for the model such as layers and nodes and activation function. I got 97.05% accuracy of the model. From hyperparameter tuning I got four layers with ReLu activation. For further analysis with this dataset, we can increase the number of epochs and it will increase the prediction accuracy.

## Reference:


1. https://www.kaggle.com/code/pcbreviglieri/predicting-smart-grid-stability-with-deep-learning
2. https://github.com/krishnaik06/Complete-Deep-Learning-With-Materials
3. https://www.youtube.com/watch?v=Bc2dWI3vnE0
4. https://www.youtube.com/watch?v=v95p-EfWps8&t=631s