iScience, Volume *23*

Supplemental Information

Non-linear Deep Neural Network for Rapid and Accurate Prediction of Phenotypic Responses to Kinase Inhibitors

Siddharth Vijay and Taranjit S. Gujral

Non-linear Deep Neural Network for Rapid and Accurate Prediction of Phenotypic Responses to Kinase Inhibitors

Siddharth Vijay[1], and Taranjit S. Gujral[1,2*]

1. Division of Human Biology, Fred Hutchinson Cancer Research Center, Seattle, WA. U.S.A.

2. Department of Pharmacology, University of Washington, Seattle, WA. U.S.A.

*Lead contact. Correspondence: tgujral@fredhutch.org

Running title: Artificial neural networks predict response to kinase inhibitors in cancer cells

Keywords: kinase inhibitor/ deep learning/ neural networks/cell migration/

SUPPLEMENTAL DATA

TRANSPARENT METHODS

Model development & optimization of network hyperparameters

The development and implementation of KiDNN was accomplished using the python Keras framework with a TensorFlow backend (Chollet, 2018). Leave-One-Out-Cross-Validation (LOOCV) was the primary network evaluation method used in this investigation. In LOOCV, each time [n-1] drugs are used to train the network to predict the excluded inhibitor's effect on cell migration. The process is repeated a total of [n] times leaving out and predicting the effect on migration for every inhibitor. The average mean squared error (MSE) of all [n] inhibitors was used to cross-evaluate various networks. Since large errors in predicted and observed migration are undesirable as they can cause false positives and negatives, MSE was the primary error function used to optimize the network.

For KiDNN to reach peak predictive performance for a specific cell-line, the optimal network architecture and hyperparameters need to be chosen. The complete set of hyperparameters and their definitions are shown below. Epochs; the number of iterations KiDNN is supplied the entire training dataset. Typically, the network needs to be input the entire dataset multiple times to effectively fit the network to the data(Jayachandiran). Batch Size; the number of samples (rows/individual observations of the dataset) input through KiDNN before updating weights(Chollet, 2018). Initializer; the distribution in the start of the training process from which starting weights are assigned. Activation function; function applied to the weighted sum of inputs and biases to produce output (Dongare et al., 2012). Optimizer; the algorithm that helps the network converge to the optimal set of weights. Hidden layers; the quantity of layers (consisting of several nodes) in-between the output and input layer. Multiple hidden layers can ensure that the network can capture complex, non-linear dependence between input and output (Zupan, 1994). Nodes per hidden layer; the number of individual units (nodes) per every hidden layer. Dropout rate; the percentage of individual nodes in a layer that is temporarily removed from the network along with its connections. Dropout is a common method of preventing model overfitting (increased performance in training set, but poor performance in test sets) and nodes from co-adapting too much (Srivastava et al., 2014). A complete list of all the specific values/names of the 8 hyperparameters are shown in **Table S1**. After optimization, the single top combination of the 8 hyperparameters is selected to build KiDNN. Besides these hyperparameters, the input and output layer remain set as there are 300 nodes in the input layer corresponding to the 300 kinases' inhibition measured in the activity profile and 1 output node for the predicted cell migration.

A common method of optimization used in numerous studies is Grid Search(Pontes et al., 2016), where all the various combinations of hyperparameter values are individually used to build several networks and the top combination is selected based on lowest MSE. Here, an exhaustive Grid Search optimizing all 8 hyperparameters at once wasn't a viable method because of the pure volume of required computations as more than 1,728,000 various networks would need to be evaluated. Consequently, a progressive, phase-by-phase Grid Search approach (evaluated with LOOCV MSE) was used where Grid Search was performed on 2 to 3 hyperparameters at once, rather than all 8. This significantly reduces the computation time to optimize KiDNN as only ~470 networks were evaluated. Since this method would prevent combinations of the top hyperparameter values across phases from being evaluated, a final phase 5 was performed, where the top 2 combinations of hyperparameter values were combined across multiple phases to select one final combination used to build the final KiDNN model. This multi-phase Grid Search is particularly effective because values of hyperparameters that perform poorly in terms of predictive performance are disregarded in future phases of the Grid Search, while also significantly reducing computational time to optimize KiDNN. Additionally, between phases, the top selected hyperparameters optimized in the previous phase were used in the successive phases by updating a baseline network. The initiation of the optimization process began with a completely unoptimized baseline network with default hyperparameters. The baseline neural network consisted of 300 input nodes for each kinase's activity, 2 hidden layers with 100 nodes per layer, and one output layer. The Adam optimizer, Rectified Linear Unit (ReLU) activation and Normal weight initialization distribution were used. Using the default baseline network, the network was optimized in 5 different phases to develop the final, fully-optimized KiDNN.

In phase 1, the range of overfitting was identified, where the model starts to fit the training data too much to the point where its ability to predict response of the test set was compromised. The baseline model was trained on response to 26 (~ 80%) randomly selected inhibitors in the training set and tested on the remaining 6 inhibitors (~20%) for 400 epochs. The fluctuation of MSE between predicted and observed migration of the 6 excluded inhibitors and the 26 inhibitors of the training set was measured as a function of the number of epochs. Using this data, a range of epochs is selected where MSE reaches a global minimum, before overfitting of the data starts to occur.

Using the range of epochs and batches sizes (listed in **Table S1**), Grid Search was performed to deduce the optimal combinations of epochs and batch size in phase 2. The top 5 combinations based on lowest LOOCV MSE are then re-run 5 additional times to ensure that the low MSE isn't due to chance by using average LOOCV MSE rather than single-iteration MSE. From the 5 additional runs, the 2 top combinations are identified based on average LOOCV MSE and are later used in Phase 5. In phase 3, the activation function, optimizers and weight initializers are optimized, while in phase 4, the number of hidden layers, nodes per hidden layer and Dropout rate were optimized. In both phase 3 and phase 4, the same process as executed in phase 2 was repeated.

As stated previously, the optimization process is progressive, where the top hyperparameters in one phase are used for the next. For example, the top combination of epochs and batch size selected in phase 2 were used to update the baseline network in phase 3, and the top combination of activation functions, optimizers and weight initializations in phase 3 were used to update the baseline network in phase 4. In the final phase 5, the top 2 sets of hyperparameter values across phases 2 through 4 are combined to create 8 (2^3) separate networks with various combinations of each set of hyperparameters from each phase using Grid Search. The 8 networks are run a total of 5 times and the network with the lowest average LOOCV MSE is selected as the final network architecture used to build KiDNN. The efficacy of the final KiDNN model was further evaluated using the LOOCV MSE of the predicted and observed values and the mean standard deviation between 10 iterations of predictions to ensure a low-bias and high-precision neural network.

Supervised Deep Learning Approach

KiDNN was developed with Deep Neural Networks (DNNs), a non-linear, multi-layer feed-forward network. DNNs mimic the human brain, with processing nodes analogous to neurons in our brain and collections of neurons representing complete, multi-layered neural networks (Zupan, 1994). In KiDNN, these nodes are connected by weighted links, with all nodes, except those composing the input layer, receiving weighted sums of the output from the nodes in the previous layer and transmitting their output to nodes in successive layers until the final output layer (e.g. measured phenotypes such as cell migration) is reached (Dongare et al., 2012). The output transmitted to the successive nodes from a prior node is computed using the following three steps. First, the weighted sum of the output of nodes in the previous layer is computed, then biases are added, and lastly, an activation function is applied to the output limiting the output between a finite range (-1 to 1 or 0 to 1 in this study) (Dongare et al., 2012). This computation is repeated until the final layer is reached with the final predicted migration outputted. The computation performed in an individual node is shown below:

$$output = f\left(\sum_{i=1}^{n} x_i\, w_i + b\right)$$

input: $(x_1 \; to \; x_n)$ *weights:* $(w_1 \; to \; w_n)$ *bias:* (b) *activation function* (f)

Ultimately, KiDNN is trained on activity profiles by repeatedly computing errors of millions of combinations of weights, feeding it back to the network, and adjusting its weights accordingly until the optimal set of weights and biases between individual nodes are found where the error function between predicted and observed migration is minimal. Keeping these optimal weights and biases constant, remaining untested inhibitors' activity profiles can be inputted through KiDNN and the predicted migration can be computed.

Applying KiDNN to naïve datasets (Hs578t & FOCUS)

After selecting the optimal architecture and hyperparameters to develop KiDNN for Hs578t, the network was applied to completely unseen inhibitors to predict migration. The network was trained on the 32 inhibitors' activity profile and their resulting migration to predict migration in the 178 other inhibitors. The same method of optimization used to optimize KiDNN for Hs578t was used to optimize KiDNN for a new cell line (FOCUS). After the optimal hyperparameters were found, KiDNN was trained on the 32 inhibitors' activity profile and their resulting migration to predict migration in the 178 other inhibitors for FOCUS.

Cell lines

Hs578t was obtained from American Type Culture Collection. Hepatocellular carcinoma cell line (FOCUS) was obtained from J. Wands (Brown University). Both cell lines were grown at 37°C under 5% CO2, 95% ambient atmosphere and maintained in Dulbecco's minimum essential medium (DMEM) supplemented with 10% FBS (Sigma).

Kinetic cell migration assay

To study the effect of kinase inhibitors on migration of Hs578t or FOCUS cells a wound-healing assay was employed as described previously (Gujral et al., 2014b). The details on the structure and preparation of this dataset has been disclosed previously (Gujral et al., 2014b). Briefly, cells were plated on 96-well plates (Essen Image Lock, Essen Instruments), and a wound was scratched with wound scratcher (Essen Instruments). Inhibitors at different doses were added immediately after wound scratching, wound confluence was monitored with Incucyte Live-Cell Imaging System and software (Essen Instruments).

Wound closure was observed every 2 hours for 24-72 hours by comparing the mean relative wound density of at least three biological replicates in each experiment.

Fig. S1

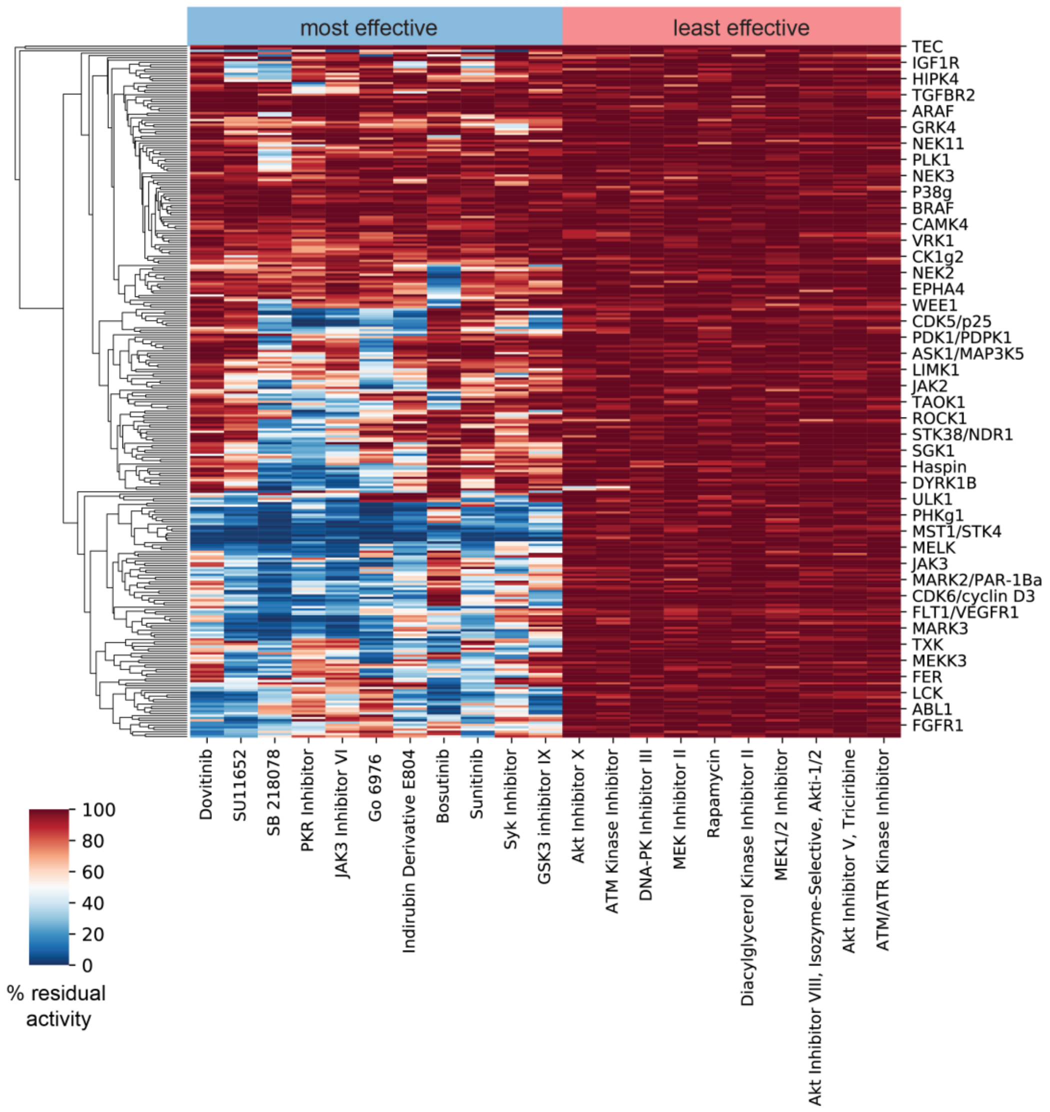

Figure S1. Kinase target profiles of the most effective KiDNN-predicted inhibitors, Related to Figure 4. A heatmap of showing kinase targets profiles of the most effective and the least effective KiDNN predicted inhibitors.

Table S1. List of Parameters Optimized, Related to Figure 2. Epochs were selected from the training and validation loss plot (Fig. 2B) by choosing 3 values above and 3 below the 125 epochs at which overfitting was observed. Batch size is an integer. Various types of kernel initializers, optimizer and activation functions were evaluated.

Epochs	*Batch Size*	*Weight Initializer*	*Optimizer*	*Activation*	*Hidden Layers (HL)*	*Nodes per HL*	*Dropout Rate*
50	1	Uniform	RMSprop	Sigmoid	1	10	0
75	2	Truncated Normal	Adagrad	TanH	2	25	0.05
100	4	Normal	Adamax	ReLU	3	50	0.1
125	8	Lecun Uniform	Adadelta	ELU		100	0.2
150	16	Glorot Normal	Adam	SELU		150	0.5
175	32	He Normal	Nadam			200	
200		Glorot Uniform				250	
		Variance Scaling				300	
		He Uniform					
		Orthogonal					

ReLU; rectified linear unit, ELU; exponential linear unit, SELU; scaled exponential linear unit, TanH; hyperbolic tangent
RMSprop; root mean square propagation, Adagrad; adaptive gradient, Adam; Adaptive moment estimation, Nadam; Nesterov-accelerated adaptive moment estimation

Table S2. Network Evaluation of Top Batch Size and Epoch Combinations, Related to Figure 2. The top 2 combinations are shaded. The one that was used for subsequent optimization is indicated in bold.

Batch Size	*Epoch*	*Average MSE*
8	50	106.41
8	125	111.92
16	50	101.47
32	**75**	**100.48**
32	125	103.00

Table S3. Network Evaluation of Top Weight Initializers, Optimizers and Activation Function Combinations, Related to Figure 2. The top 2 combinations are shaded. The one that was used for subsequent optimization is indicated in bold.

Weight Initializer	*Optimizer*	*Activation*	*Mean MSE*
Uniform	Adagrad	ReLU	99.4
Truncated Normal	Adagrad	ReLU	94.6
Truncated Normal	**Adagrad**	**ELU**	**92.9**
Lecun Uniform	Adamax	ELU	107.3
Variance Scaling	Nadam	ReLU	101.7

ReLU; rectified linear unit, ELU; exponential linear unit, Adagrad; adaptive gradient, Adam; Adaptive moment estimation, Nadam; Nesterov-accelerated adaptive moment estimation

Table S4. Network Evaluation of Top Hidden Layer Quantity, Nodes and Dropout Rate Combinations, Related to Figure 2. The top 2 combinations are shaded. The one that was used for subsequent optimization is indicated in bold.

Hidden Layers (HL)	*Nodes per HL*	*Dropout Rate*	*Average MSE*
2	200	0	89.9
2	**300**	**0**	**85.7**
2	200	0.05	91.4
3	250	0	94.0
3	50	0.05	92.8

Table S5. Predicted migration of Hs578t cells in response to top 20 most effective kinase inhibitors selected by KiDNN, Related to Figure 4

Rank	*Kinase Inhibitor*	*Predicted Migration*
1	Staurosporine	12.3
2	K-252a	28.1
3	SB 218078	35.9
4	Cdk1/2 Inhibitor III	37.5
5	PKR Inhibitor	46.8
6	JAK3 Inhibitor VI	48.1
7	SU11652	48.6
8	Go 6976	49.9
9	Indirubin Derivative E804	50.2
10	Staurosporine, N-benzoyl-	51.4
11	Bosutinib	52.7
12	Sunitinib	54.2
13	Syk Inhibitor	56.3
14	GSK3 inhibitor IX	57.0
15	JAK Inhibitor I	57.5
16	Dasatinib	57.5
17	AMPK Inhibitor, Compound C	57.8
18	Dovitinib	58.2
19	Aurora Kinase/Cdk Inhibitor	61.8
20	Indirubin-3’-monoxime	61.8

Table S6. Predicted migration of FOCUS cells in response to top 20 most effective kinase inhibitors selected by KiDNN-FOCUS, Related to Figure 5

Rank	*Kinase Inhibitor*	*Predicted Migration*
1	Staurosporine	5.0
2	Dasatinib	14.2
3	Dovitinib	33.8
4	Bosutinib	34.5
5	Staurosporine, N-benzoyl-	38.8
6	LCK inhibitor	38.8
7	GSK3 inhibitor IX	41.6
8	SB 218078	43.1
9	SU11652	48.5
10	Indirubin Derivative E804	48.6
11	PDGFR RTK inhibitor	52.0
12	K-252a	53.4
13	PDK1/Akt/Flt Dual Pathway Inhibitor	55.5
14	GSK-3 Inhibitor X	55.9
15	Syk Inhibitor	56.4
16	Sunitinib	57.6
17	Flt-3 Inhibitor II	58.0
18	TWS119	59.2
19	Tozasertib	61.3
20	GSK-3 Inhibitor XIII	61.4

Table S7. Predicted and measured migration of Hs578t cells in response to all 40 kinase inhibitors, Related to Figure 4.

Kinase Inhibitor	***Measured Migration***	***KiDNN Predictions***	***KiR Predictions***
Bosutinib	39.9	52.7	42.5
Casein Kinase I Inhibitor D44	70	70.0	70.0
Cdk1/2 Inhibitor III	36.2	37.5	35.5
Dasatinib	21.2	57.5	28.7
EGFR ErbB-2 Erbb-4	70	70.0	68.7
Erlotinib	70	70.0	70.0
Go 6983	70	67.8	67.2
Gefitinib	70	70.0	66.5
GSK3 inhibitor IX	49	57.0	51.7
GSK-3b Inhibitor I	70	70.0	70.0
H-89	70	70.0	68.3
Imatinib	70	70.0	70.0
JNK Inhibitor II	70	68.1	66.9
K-252a	27.5	28.1	27.6
Lapatinib	70	70.0	70.0
LCK inhibitor	70	68.5	68.3
LY294002	70	70.0	70.0
Masitinib	70	70.0	68.4
Met Kinase Inhibitor	68.6	65.3	67.5
Nilotinib	70	68.2	67.8
PDGFR RTK inhibitor	69	67.7	67.6
Rapamycin	70	70.0	68.5

ROCK Inhibitor	70	70.0	68.9
SB 218078	29.4	35.9	30.0
SB220025	56.9	70.0	57.6
Sorafenib	65.4	70.0	68.4
Src Kinase Inhibitor I	70	70.0	69.0
Staurosporine	0.98	12.3	6.0
Sunitinib	54.3	54.2	52.9
Tofacitinib	70	70.0	70.0
TWS119	68.3	62.2	61.2
Vandetanib	66.6	65.3	62.4
Aminopurvalanol A	70	68.6	61.8
Staurosporine, N-benzoyl	56.3	51.4	45.5
AMPK Inhibitor Compound C	65.8	57.8	56.6
PDK1/Akt/Flt Dual Pathway Inhibitor	70	63.6	65.0
SU11652	44.4	48.6	50.8
JAK Inhibitor I	70	57.5	46.8
PD 98059	70	70.0	70.0
Dovitinib	55.7	58.2	54.9
Mean Squared Error		***38.91***	***109.39***
Mean Absolute Error		***4.99***	***7.95***

Table S8. Predicted and measured migration of FOCUS cells in response to all 39 kinase inhibitors, Related to Figure 5.

Kinase Inhibitor	***Measured Migration***	***KiDNN Prediction***	***KiR Prediction***
Staurosporine	4.5	5.0	20.1
Dasatinib	11.6	14.2	25.2
Bosutinib	33.4	34.5	39.7
LCK inhibitor	36.5	38.8	42.0
GSK3 inhibitor IX	40.9	41.6	49.5
SB 218078	42.0	43.1	45.0
PDGFR RTK inhibitor	47.7	52.0	49.7
K-252a	52.4	53.4	43.2
Sunitinib	56.2	57.6	57.0
Sorafenib	58.1	64.7	65.6
TWS119	58.3	59.2	47.3
Vandetanib	64.2	65.9	58.6
EGFR ErbB-2 Erbb-4	67.3	67.1	70.0
Nilotinib	67.5	65.7	60.5
Go 6983	68.0	68.9	68.5
Src Kinase Inhibitor I	68.5	66.7	60.8
Gefitinib	68.9	70.0	67.2
SB220025	69.7	69.9	69.6
Casein Kinase I Inhibitor D44	70.0	70.0	70.0
Cdk1/2 Inhibitor III	70.0	70.0	61.9
Erlotinib	70.0	70.0	70.0
GSK-3b Inhibitor I	70.0	70.0	70.0

H-89	70.0	68.8	70.0
Imatinib	70.0	70.0	68.9
JNK Inhibitor II	70.0	70.0	69.3
Lapatinib	70.0	70.0	70.0
LY294002	70.0	70.0	70.0
Masitinib	70.0	65.6	62.9
Met Kinase Inhibitor	70.0	70.0	65.4
Rapamycin	70.0	70.0	68.5
ROCK Inhibitor	70.0	70.0	70.0
Tofacitinib	70.0	70.0	70.0
Aminopurvalanol A	66.78	64.18	60.34
AMPK Compound C	64.82	57.12	54.46
Cdk2 Inhibitor IV, NU6140	70	66.31	52.71
Dovitinib	32.34	51.26	33.6
GSK-3 Inhibitor XIII	59.5	59.93	53.69
Staurosporine, N-benzoyl-	37.94	55.32	30.87
SU11652	47.04	49.41	20.72
Mean Squared Error		**106.48**	**175.12**

Data S1. KiDNN code, Related to Figures 1, 2 and 3.

Predicting Effect of Untested Kinase Inhibitors on Hs578t Cell Migration

```
# Importing the libraries
import numpy as np
import pandas as pd

#Importing migration and activity profile data
response_data = pd.read_csv('hs578t_migration.csv')
drug_list = response_data.iloc[:, 0].values
alldrugs = pd.read_csv('allDrugs_migration.csv', encoding='latin1')
alldrugs = alldrugs.set_index('compound')
dataset = alldrugs.loc[drug_list]
response = response_data['Hs578t'].values
dataset["migration"] = response

# Slicing the dataset
X = dataset.iloc[:, 0:300].values
y = dataset.iloc[:, 300].values


# Importing the Keras libraries and packages
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout

# Initializing the ANN
classifier = Sequential()

# Adding the input layer and the first hidden layer
classifier.add(Dense(units = 200, kernel_initializer = 'TruncatedNormal', activation = 'elu',
input_dim = 300))

# Adding the second hidden layer
classifier.add(Dense(units = 200, kernel_initializer = 'TruncatedNormal', activation = 'elu'))

# Adding the output layer
classifier.add(Dense(units = 1, kernel_initializer = 'TruncatedNormal' ))

# Compiling the ANN
classifier.compile(loss = 'mean_squared_error', optimizer='adagrad')

# Fitting the ANN to the Training set
classifier.fit(X, y, batch_size = 32, epochs = 75)
```

```
# Predicting effect on cell migration for all 178 kinase inhibitors
X_predict = alldrugs
prediction_index = X_predict.index.tolist()
X_predict = X_predict.iloc[:, 0:300].values
y_pred = classifier.predict(X_predict)
```

Training & Validation Loss – Hs578t

```
# Importing the libraries
import numpy as np
import pandas as pd
import keras
from keras.models import Sequential
from keras.layers import Dense
import matplotlib.pyplot as plt

# Importing migration and activity profile data
response_data2 = pd.read_csv('hs578t_migration.csv')
drug_list2 = response_data2.iloc[:, 0].values
alldrugs2 = pd.read_csv('allDrugs_migration.csv', encoding='latin1')
alldrugs2 = alldrugs2.set_index('compound')
dataset2 = alldrugs2.loc[drug_list2]
response2 = response_data2['Hs578t'].values
dataset2["migration"] = response2

# Importing the dataset
X2 = dataset2.iloc[:, 0:300].values
y2 = dataset2.iloc[:, 300].values

#building model function
def build_classifier():
    classifier = Sequential()
    classifier.add(Dense(units = 100, kernel_initializer = 'normal', activation = 'relu', input_dim =
300))
    classifier.add(Dense(units = 100, kernel_initializer = 'normal', activation = 'relu'))
    classifier.add(Dense(units = 1, kernel_initializer = 'normal'))
    classifier.compile(loss = 'mean_squared_error', optimizer= 'adam', metrics=[ 'mse'])
    return classifier
classifier = build_classifier()
history = classifier.fit(X2, y2, validation_split= .2, epochs=500, batch_size=1, verbose=0)
hist = pd.DataFrame(history.history)
print(history.history.keys())

#plotting training and validation MSE as a function of epochs
```

```
def plot_history(history):
  hist = pd.DataFrame(history.history)
  hist['epoch'] = history.epoch

  plt.figure()
  plt.xlabel('Epoch')
  plt.ylabel('Mean Squared Error')
  plt.plot(hist['epoch'], hist['mean_squared_error'],
           label='Train Error')
  plt.plot(hist['epoch'], hist['val_mean_squared_error'],
           label = 'Validation Error')
  z = np.polyfit(hist['epoch'].values, hist['val_mean_squared_error'].values, 5)
  f = np.poly1d(z)
  x_new = np.linspace(hist['epoch'].values[0], hist['epoch'].values[-1], 50)
  y_new = f(x_new)
  plt.plot(x_new, y_new, label = 'Polynomial Fit')
  plt.title("Training & Validation MSE")
  plt.ylim([0,200])
  plt.xlim([0,450])
  plt.legend()

  plt.savefig("Training & Validation MSE.svg")
  plt.show()


plot_history(history)

# exporting data
best_fit_df = pd.DataFrame({"xnew": x_new, "ynew": y_new})
best_fit_df.to_excel("best_fit_line.xlsx", sheet_name='1')
hist.to_excel("Train & Val Loss.xlsx", sheet_name='1')
```

Epoch & Batch Size Optimization – Hs578t

```
# Importing the libraries
import numpy as np
import pandas as pd
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras import initializers
from keras.layers import Dropout
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from scipy.stats.stats import pearsonr
```

```
#Importing migration and activity profile data
response_data2 = pd.read_csv('hs578t_migration.csv')
drug_list2 = response_data2.iloc[:, 0].values
alldrugs2 = pd.read_csv('alldrugs_migration.csv', encoding='latin1')
alldrugs2 = alldrugs2.set_index('compound')
dataset2 = alldrugs2.loc[drug_list2]
response2 = response_data2['Hs578t'].values
dataset2["migration"] = response2

# Slicing the dataset
X2 = dataset2.iloc[:, 0:300].values
y2 = dataset2.iloc[:, 300].values

def cross_val( hl_units, init, opt, batch_size, epochs, act, dropout):
    y_pred_all = []
    y_test_all = []
    for num in range(len(X2)):
        y_test = y2[num]
        X_test = X2[num, :]
        X_test = np.array([X_test])
        X_test.T
        X_train = np.delete(X2, (num), axis=0)
        y_train = np.delete(y2, (num), axis=0)
        classifier = Sequential()
        classifier.add(Dense(units = hl_units, kernel_initializer = init, input_dim = 300, activation =
act))
        classifier.add(Dense(units = hl_units, kernel_initializer = init, activation = act))
        classifier.add(Dense(units = 1, kernel_initializer = init))
        classifier.compile(loss = 'mean_squared_error', optimizer = opt)
        classifier.fit(X_train, y_train, batch_size = batch_size, epochs = epochs)
        y_pred = classifier.predict(X_test)
        y_pred_all.append(y_pred[0][0])
        y_test_all.append(y_test)

    return (mean_squared_error(y_pred_all, y_test_all), mean_absolute_error(y_pred_all,
y_test_all), pearsonr(y_pred_all, y_test_all)[0])



ep = [50,75,100,125,150,175,200]
bs = [1,2,4,8,16,32]

scores = []

#Performing Grid Search
for epoch in ep:
    for batch_size in bs:
        scores.append(cross_val(100, 'normal', 'adam', batch_size, epoch, 'relu', 0.0))
```

Weight Initialization, Optimizer, Activation Function Optimization – Hs578t

```
# Importing the libraries
import numpy as np
import pandas as pd
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras import initializers
from keras.layers import Dropout
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from scipy.stats.stats import pearsonr

#Importing migration and activity profile data
response_data2 = pd.read_csv('hs578t_migration.csv')
drug_list2 = response_data2.iloc[:, 0].values
alldrugs2 = pd.read_csv('alldrugs_migration.csv', encoding='latin1')
alldrugs2 = alldrugs2.set_index('compound')
dataset2 = alldrugs2.loc[drug_list2]
response2 = response_data2['Hs578t'].values
dataset2["migration"] = response2

# Slicing the dataset
X2 = dataset2.iloc[:, 0:300].values
y2 = dataset2.iloc[:, 300].values

def cross_val( hl_units, init, opt, batch_size, epochs, act, dropout):
    y_pred_all = []
    y_test_all = []
    for num in range(len(X2)):
        y_test = y2[num]
        X_test = X2[num, :]
        X_test = np.array([X_test])
        X_test.T
        X_train = np.delete(X2, (num), axis=0)
        y_train = np.delete(y2, (num), axis=0)
        classifier = Sequential()
        classifier.add(Dense(units = hl_units, kernel_initializer = init, input_dim = 300, activation =
act))
        classifier.add(Dense(units = hl_units, kernel_initializer = init, activation = act))
        classifier.add(Dense(units = 1, kernel_initializer = init))
        classifier.compile(loss = 'mean_squared_error', optimizer = opt)
        classifier.fit(X_train, y_train, batch_size = batch_size, epochs = epochs)
        y_pred = classifier.predict(X_test)
        y_pred_all.append(y_pred[0][0])
        y_test_all.append(y_test)

    return (mean_squared_error(y_pred_all, y_test_all), mean_absolute_error(y_pred_all,
y_test_all), pearsonr(y_pred_all, y_test_all)[0])


initializer = ['uniform', 'TruncatedNormal', 'normal', 'lecun_uniform', 'glorot_normal',
'he_normal', 'glorot_uniform', 'VarianceScaling', 'orthogonal', 'he_uniform']
optimizer = ['rmsprop','adagrad', 'adamax', 'adadelta', 'adam', 'nadam']
activation = ['sigmoid', 'tanh', 'relu', 'elu', 'selu']

scores = []

#Performing Grid Search
for init in initializer:
    for opt in optimizer:
        for act in activation:
            scores.append(cross_val(100, init, opt, 32, 75, act, 0.0))
```

Hidden layer, Nodes per Hidden Layer and Dropout Rate Optimization – Hs578t

```
# Importing the libraries
import numpy as np
import pandas as pd
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras import initializers
from keras.layers import Dropout
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from scipy.stats.stats import pearsonr

#Importing migration and activity profile data
response_data2 = pd.read_csv('hs578t_migration.csv')
drug_list2 = response_data2.iloc[:, 0].values
alldrugs2 = pd.read_csv('alldrugs_migration.csv', encoding='latin1')
alldrugs2 = alldrugs2.set_index('compound')
dataset2 = alldrugs2.loc[drug_list2]
response2 = response_data2['Hs578t'].values
```

```
dataset2["migration"] = response2

# Slicing the dataset
X2 = dataset2.iloc[:, 0:300].values
y2 = dataset2.iloc[:, 300].values

#LOOCV function for 1 hidden layer
def cross_val1( hl_units, init, opt, batch_size, epochs, act, dropout):
    y_pred_all = []
    y_test_all = []
    for num in range(len(X2)):
        y_test = y2[num]
        X_test = X2[num, :]
        X_test = np.array([X_test])
        X_test.T
        X_train = np.delete(X2, (num), axis=0)
        y_train = np.delete(y2, (num), axis=0)
        classifier = Sequential()
        classifier.add(Dense(units = hl_units, kernel_initializer = init, input_dim = 300, activation = 
act))
        classifier.add(Dropout(dropout))
        classifier.add(Dense(units = 1, kernel_initializer = init))
        classifier.compile(loss = 'mean_squared_error', optimizer = opt)
        classifier.fit(X_train, y_train, batch_size = batch_size, epochs = epochs)
        y_pred = classifier.predict(X_test)
        y_pred_all.append(y_pred[0][0])
        y_test_all.append(y_test)

    return (mean_squared_error(y_pred_all, y_test_all), mean_absolute_error(y_pred_all, 
y_test_all), pearsonr(y_pred_all, y_test_all)[0])

#LOOCV function for 2 hidden layers
def cross_val2( hl_units, init, opt, batch_size, epochs, act, dropout):
    y_pred_all = []
    y_test_all = []
    for num in range(len(X2)):
        y_test = y2[num]
        X_test = X2[num, :]
        X_test = np.array([X_test])
        X_test.T
        X_train = np.delete(X2, (num), axis=0)
        y_train = np.delete(y2, (num), axis=0)
        classifier = Sequential()
        classifier.add(Dense(units = hl_units, kernel_initializer = init, input_dim = 300, activation = 
act))
        classifier.add(Dropout(dropout))
        classifier.add(Dense(units = hl_units, kernel_initializer = init, input_dim = 300, activation =
act))
        classifier.add(Dropout(dropout))
        classifier.add(Dense(units = 1, kernel_initializer = init))
        classifier.compile(loss = 'mean_squared_error', optimizer = opt)
        classifier.fit(X_train, y_train, batch_size = batch_size, epochs = epochs)
        y_pred = classifier.predict(X_test)
        y_pred_all.append(y_pred[0][0])
        y_test_all.append(y_test)

    return (mean_squared_error(y_pred_all, y_test_all), mean_absolute_error(y_pred_all,
y_test_all), pearsonr(y_pred_all, y_test_all)[0])

#LOOCV function for 3 hidden layers
def cross_val3( hl_units, init, opt, batch_size, epochs, act, dropout):
    y_pred_all = []
    y_test_all = []
    for num in range(len(X2)):
        y_test = y2[num]
        X_test = X2[num, :]
        X_test = np.array([X_test])
        X_test.T
        X_train = np.delete(X2, (num), axis=0)
        y_train = np.delete(y2, (num), axis=0)
        classifier = Sequential()
        classifier.add(Dense(units = hl_units, kernel_initializer = init, input_dim = 300, activation =
act))
        classifier.add(Dropout(dropout))
        classifier.add(Dense(units = hl_units, kernel_initializer = init, input_dim = 300, activation =
act))
        classifier.add(Dropout(dropout))
        classifier.add(Dense(units = hl_units, kernel_initializer = init, input_dim = 300, activation =
act))
        classifier.add(Dropout(dropout))
        classifier.add(Dense(units = 1, kernel_initializer = init))
        classifier.compile(loss = 'mean_squared_error', optimizer = opt)
        classifier.fit(X_train, y_train, batch_size = batch_size, epochs = epochs)
        y_pred = classifier.predict(X_test)
        y_pred_all.append(y_pred[0][0])
        y_test_all.append(y_test)

    return (mean_squared_error(y_pred_all, y_test_all), mean_absolute_error(y_pred_all,
y_test_all), pearsonr(y_pred_all, y_test_all)[0])


#List of possible values
```

```
nodes_per_hidden_layer = [10,25,50,100,150,200,250,300]
dropout_rate = [0,0.05,0.1,0.2,0.5]

scores = []

#Performing Grid Search
for nodes in nodes_per_hidden_layer:
    for dr in dropout_rate:
            scores.append(cross_val1(nodes, ‘TruncatedNormal’, ‘adagrad’, 32, 75, ‘elu’, dr))

for nodes in nodes_per_hidden_layer:
    for dr in dropout_rate:
            scores.append(cross_val2(nodes, ‘TruncatedNormal’, ‘adagrad’, 32, 75, ‘elu’, dr))

for nodes in nodes_per_hidden_layer:
    for dr in dropout_rate:
            scores.append(cross_val3(nodes, ‘TruncatedNormal’, ‘adagrad’, 32, 75, ‘elu’, dr))
```

Predicting Effect of Untested Kinase Inhibitors on FOCUS Cell Migration

```
# Importing the libraries
import numpy as np
import pandas as pd

#Importing migration and activity profile data
response_data = pd.read_csv('focus_migration.csv')
drug_list = response_data.iloc[:, 0].values
alldrugs = pd.read_csv('allDrugs_migration.csv', encoding='latin1')
alldrugs = alldrugs.set_index('compound')
dataset = alldrugs.loc[drug_list]
response = response_data['FOCUS'].values
dataset["migration"] = response

# Slicing the dataset
X = dataset.iloc[:, 0:300].values
y = dataset.iloc[:, 300].values


# Importing the Keras libraries and packages
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
```

```
# Initializing the ANN
classifier = Sequential()

# Adding the input layer and the first hidden layer
classifier.add(Dense(units = 200, kernel_initializer = 'TruncatedNormal', activation = 'elu',
input_dim = 300))

# Adding the second hidden layer
classifier.add(Dense(units = 200, kernel_initializer = 'TruncatedNormal', activation = 'elu'))

# Adding the output layer
classifier.add(Dense(units = 1, kernel_initializer = 'TruncatedNormal' ))

# Compiling the ANN
classifier.compile(loss = 'mean_squared_error', optimizer='adagrad')

# Fitting the ANN to the Training set
classifier.fit(X, y, batch_size = 32, epochs = 75)

# Predicting effect on cell migration for all 178 kinase inhibitors
X_predict = alldrugs
prediction_index = X_predict.index.tolist()
X_predict = X_predict.iloc[:, 0:300].values
y_pred = classifier.predict(X_predict)
```

Training & Validation Loss – FOCUS

```
# Importing the libraries
import numpy as np
import pandas as pd
import keras
from keras.models import Sequential
from keras.layers import Dense
import matplotlib.pyplot as plt

# Importing migration and activity profile data
response_data2 = pd.read_csv('focus_migration.csv')
drug_list2 = response_data2.iloc[:, 0].values
alldrugs2 = pd.read_csv('allDrugs_migration.csv', encoding='latin1')
alldrugs2 = alldrugs2.set_index('compound')
dataset2 = alldrugs2.loc[drug_list2]
response2 = response_data2['FOCUS'].values
dataset2["migration"] = response2
```

```
# Importing the dataset
X2 = dataset2.iloc[:, 0:300].values
y2 = dataset2.iloc[:, 300].values

#building model function
def build_classifier():
    classifier = Sequential()
    classifier.add(Dense(units = 100, kernel_initializer = 'normal', activation = 'relu', input_dim = 
300))
    classifier.add(Dense(units = 100, kernel_initializer = 'normal', activation = 'relu'))
    classifier.add(Dense(units = 1, kernel_initializer = 'normal'))
    classifier.compile(loss = 'mean_squared_error', optimizer= 'adam', metrics=[ 'mse'])
    return classifier
classifier = build_classifier()
history = classifier.fit(X2, y2, validation_split= .2, epochs=500, batch_size=1, verbose=0)
hist = pd.DataFrame(history.history)
print(history.history.keys())

#plotting training and validation MSE as a function of epochs
def plot_history(history):
  hist = pd.DataFrame(history.history)
  hist['epoch'] = history.epoch

  plt.figure()
  plt.xlabel('Epoch')
  plt.ylabel('Mean Squared Error')
  plt.plot(hist['epoch'], hist['mean_squared_error'],
           label='Train Error')
  plt.plot(hist['epoch'], hist['val_mean_squared_error'],
           label = 'Validation Error')
  z = np.polyfit(hist['epoch'].values, hist['val_mean_squared_error'].values, 5)
  f = np.poly1d(z)
  x_new = np.linspace(hist['epoch'].values[0], hist['epoch'].values[-1], 50)
  y_new = f(x_new)
  plt.plot(x_new, y_new, label = 'Polynomial Fit')
  plt.title("Training & Validation MSE")
  plt.ylim([0,200])
  plt.xlim([0,450])
  plt.legend()

  plt.savefig("Training & Validation MSE.svg")
  plt.show()


plot_history(history)
```

```
# exporting data
best_fit_df = pd.DataFrame({"xnew": x_new, "ynew": y_new})
best_fit_df.to_excel("best_fit_line.xlsx", sheet_name='1')
hist.to_excel("Train & Val Loss.xlsx", sheet_name='1')
```

Epoch & Batch Size Optimization – FOCUS

```
# Importing the libraries
import numpy as np
import pandas as pd
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras import initializers
from keras.layers import Dropout
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from scipy.stats.stats import pearsonr

#Importing migration and activity profile data
response_data2 = pd.read_csv('focus_migration.csv')
drug_list2 = response_data2.iloc[:, 0].values
alldrugs2 = pd.read_csv('alldrugs_migration.csv', encoding='latin1')
alldrugs2 = alldrugs2.set_index('compound')
dataset2 = alldrugs2.loc[drug_list2]
response2 = response_data2['FOCUS'].values
dataset2["migration"] = response2

# Slicing the dataset
X2 = dataset2.iloc[:, 0:300].values
y2 = dataset2.iloc[:, 300].values

def cross_val( hl_units, init, opt, batch_size, epochs, act, dropout):
    y_pred_all = []
    y_test_all = []
    for num in range(len(X2)):
        y_test = y2[num]
        X_test = X2[num, :]
        X_test = np.array([X_test])
        X_test.T
        X_train = np.delete(X2, (num), axis=0)
        y_train = np.delete(y2, (num), axis=0)
        classifier = Sequential()
        classifier.add(Dense(units = hl_units, kernel_initializer = init, input_dim = 300, activation =
act))
```

```
        classifier.add(Dense(units = hl_units, kernel_initializer = init, activation = act))
        classifier.add(Dense(units = 1, kernel_initializer = init))
        classifier.compile(loss = 'mean_squared_error', optimizer = opt)
        classifier.fit(X_train, y_train, batch_size = batch_size, epochs = epochs)
        y_pred = classifier.predict(X_test)
        y_pred_all.append(y_pred[0][0])
        y_test_all.append(y_test)

    return (mean_squared_error(y_pred_all, y_test_all), mean_absolute_error(y_pred_all, 
y_test_all), pearsonr(y_pred_all, y_test_all)[0])


ep = [50,75,100,125,150,175,200]
bs = [1,2,4,8,16,32]

scores = []

#Performing Grid Search
for epoch in ep:
    for batch_size in bs:
        scores.append(cross_val(100, 'normal', 'adam', batch_size, epoch, 'relu', 0.0))
```

Weight Initialization, Optimizer, Activation Function Optimization – FOCUS

```
# Importing the libraries
import numpy as np
import pandas as pd
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras import initializers
from keras.layers import Dropout
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from scipy.stats.stats import pearsonr

#Importing migration and activity profile data
response_data2 = pd.read_csv('focus_migration.csv')
drug_list2 = response_data2.iloc[:, 0].values
alldrugs2 = pd.read_csv('alldrugs_migration.csv', encoding='latin1')
alldrugs2 = alldrugs2.set_index('compound')
dataset2 = alldrugs2.loc[drug_list2]
response2 = response_data2['FOCUS'].values
dataset2["migration"] = response2
```

```
# Slicing the dataset
X2 = dataset2.iloc[:, 0:300].values
y2 = dataset2.iloc[:, 300].values

def cross_val( hl_units, init, opt, batch_size, epochs, act, dropout):
    y_pred_all = []
    y_test_all = []
    for num in range(len(X2)):
        y_test = y2[num]
        X_test = X2[num, :]
        X_test = np.array([X_test])
        X_test.T
        X_train = np.delete(X2, (num), axis=0)
        y_train = np.delete(y2, (num), axis=0)
        classifier = Sequential()
        classifier.add(Dense(units = hl_units, kernel_initializer = init, input_dim = 300, activation = 
act))
        classifier.add(Dense(units = hl_units, kernel_initializer = init, activation = act))
        classifier.add(Dense(units = 1, kernel_initializer = init))
        classifier.compile(loss = 'mean_squared_error', optimizer = opt)
        classifier.fit(X_train, y_train, batch_size = batch_size, epochs = epochs)
        y_pred = classifier.predict(X_test)
        y_pred_all.append(y_pred[0][0])
        y_test_all.append(y_test)

    return (mean_squared_error(y_pred_all, y_test_all), mean_absolute_error(y_pred_all, 
y_test_all), pearsonr(y_pred_all, y_test_all)[0])


initializer = ['uniform', 'TruncatedNormal', 'normal', 'lecun_uniform', 'glorot_normal', 
'he_normal', 'glorot_uniform', 'VarianceScaling', 'orthogonal', 'he_uniform']
optimizer = ['rmsprop','adagrad', 'adamax', 'adadelta', 'adam', 'nadam']
activation = ['sigmoid', 'tanh', 'relu', 'elu', 'selu']

scores = []

#Performing Grid Search
for init in initializer:
    for opt in optimizer:
        for act in activation:
            scores.append(cross_val(100, init, opt, 2, 120, act, 0.0))
```

Hidden layer, Nodes per Hidden Layer and Dropout Rate Optimization – FOCUS

```
# Importing the libraries
import numpy as np
import pandas as pd
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras import initializers
from keras.layers import Dropout
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from scipy.stats.stats import pearsonr

#Importing migration and activity profile data
response_data2 = pd.read_csv('focus_migration.csv')
drug_list2 = response_data2.iloc[:, 0].values
alldrugs2 = pd.read_csv('alldrugs_migration.csv', encoding='latin1')
alldrugs2 = alldrugs2.set_index('compound')
dataset2 = alldrugs2.loc[drug_list2]
response2 = response_data2['FOCUS'].values
dataset2["migration"] = response2

# Slicing the dataset
X2 = dataset2.iloc[:, 0:300].values
y2 = dataset2.iloc[:, 300].values

#LOOCV function for 1 hidden layer
def cross_val1( hl_units, init, opt, batch_size, epochs, act, dropout):
    y_pred_all = []
    y_test_all = []
    for num in range(len(X2)):
        y_test = y2[num]
        X_test = X2[num, :]
        X_test = np.array([X_test])
        X_test.T
        X_train = np.delete(X2, (num), axis=0)
        y_train = np.delete(y2, (num), axis=0)
        classifier = Sequential()
        classifier.add(Dense(units = hl_units, kernel_initializer = init, input_dim = 300, activation =
act))
        classifier.add(Dropout(dropout))
        classifier.add(Dense(units = 1, kernel_initializer = init))
        classifier.compile(loss = 'mean_squared_error', optimizer = opt)
        classifier.fit(X_train, y_train, batch_size = batch_size, epochs = epochs)
        y_pred = classifier.predict(X_test)
        y_pred_all.append(y_pred[0][0])
        y_test_all.append(y_test)
    return (mean_squared_error(y_pred_all, y_test_all), mean_absolute_error(y_pred_all,
y_test_all), pearsonr(y_pred_all, y_test_all)[0])

#LOOCV function for 2 hidden layers
def cross_val2( hl_units, init, opt, batch_size, epochs, act, dropout):
    y_pred_all = []
    y_test_all = []
    for num in range(len(X2)):
        y_test = y2[num]
        X_test = X2[num, :]
        X_test = np.array([X_test])
        X_test.T
        X_train = np.delete(X2, (num), axis=0)
        y_train = np.delete(y2, (num), axis=0)
        classifier = Sequential()
        classifier.add(Dense(units = hl_units, kernel_initializer = init, input_dim = 300, activation =
act))
        classifier.add(Dropout(dropout))
        classifier.add(Dense(units = hl_units, kernel_initializer = init, input_dim = 300, activation =
act))
        classifier.add(Dropout(dropout))
        classifier.add(Dense(units = 1, kernel_initializer = init))
        classifier.compile(loss = 'mean_squared_error', optimizer = opt)
        classifier.fit(X_train, y_train, batch_size = batch_size, epochs = epochs)
        y_pred = classifier.predict(X_test)
        y_pred_all.append(y_pred[0][0])
        y_test_all.append(y_test)

    return (mean_squared_error(y_pred_all, y_test_all), mean_absolute_error(y_pred_all,
y_test_all), pearsonr(y_pred_all, y_test_all)[0])

#LOOCV function for 3 hidden layers
def cross_val3( hl_units, init, opt, batch_size, epochs, act, dropout):
    y_pred_all = []
    y_test_all = []
    for num in range(len(X2)):
        y_test = y2[num]
        X_test = X2[num, :]
        X_test = np.array([X_test])
        X_test.T
        X_train = np.delete(X2, (num), axis=0)
        y_train = np.delete(y2, (num), axis=0)
        classifier = Sequential()
        classifier.add(Dense(units = hl_units, kernel_initializer = init, input_dim = 300, activation =
act))
```

```
        classifier.add(Dropout(dropout))
        classifier.add(Dense(units = hl_units, kernel_initializer = init, input_dim = 300, activation = 
act))
        classifier.add(Dropout(dropout))
        classifier.add(Dense(units = hl_units, kernel_initializer = init, input_dim = 300, activation = 
act))
        classifier.add(Dropout(dropout))
        classifier.add(Dense(units = 1, kernel_initializer = init))
        classifier.compile(loss = 'mean_squared_error', optimizer = opt)
        classifier.fit(X_train, y_train, batch_size = batch_size, epochs = epochs)
        y_pred = classifier.predict(X_test)
        y_pred_all.append(y_pred[0][0])
        y_test_all.append(y_test)

    return (mean_squared_error(y_pred_all, y_test_all), mean_absolute_error(y_pred_all, 
y_test_all), pearsonr(y_pred_all, y_test_all)[0])


#List of possible values
nodes_per_hidden_layer = [10,25,50,100,150,200,250,300]
dropout_rate = [0,0.05,0.1,0.2,0.5]

scores = []

#Performing Grid Search
for nodes in nodes_per_hidden_layer:
    for dr in dropout_rate:
            scores.append(cross_val1(nodes, ‘uniform’, ‘adagrad’, 2, 120, ‘selu’, dr))

for nodes in nodes_per_hidden_layer:
    for dr in dropout_rate:
            scores.append(cross_val2(nodes, ‘uniform’, ‘adagrad’, 2, 120, ‘selu’, dr))

for nodes in nodes_per_hidden_layer:
    for dr in dropout_rate:
            scores.append(cross_val3(nodes, ‘uniform’, ‘adagrad’, 2, 120, ‘selu’, dr))
```