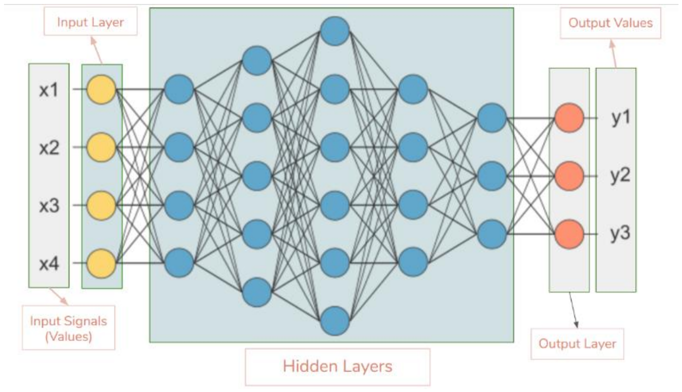
# **Assignment** 1

# Building, training and testing feed forward neural network

|  |  |
|---|---|
| **Name** | **Jayasuriya D P** |
| **Index** | 209337M |

## Question 1

**Do a literature review on how to select the number of hidden layers and the number of neurons in each layer for a given problem. List your findings in bullet form with the relevant citations.**



In a typical feed forward, neural network is comprised of three types of layers. Input layer, output layer and hidden layers. As the name stated input layer is the layer that we feed the features into the network and output layer is the layer that we take out the prediction or the resultant. Hidden layers are the layers that are in between input and output layers. These layers are can be called latent or middle layers but since latent term is too technical and middle layer doesn't make sense when there are multiple layers the preferred why the people called is the hidden layer. Additionally, the hidden layer estimates the output with respect to the input layer. But the functions or the distribution of the data is hidden in these layers. How those individual neurons and their activation functions estimate the function is also hidden inside theses layers so they are called hidden layers.

When comes to selection of layers and their sizes, the input layer can be selected according to the training data. Number of features that going to give as the input to the feed forward neural network (FFNN) is the estimation factor of the shape of the FFNN. Some rare cases there is a one additional neuron added as the bias term. Moving on the output layer, most of time there is one output layer according to the requirement. Generally according to the two main model types classification and regression models these layers is comprise of one or many neurons. If the model is a classification model it would probably have neurons with respect to number of classes. If the model is a regression model it would have number of outputs with respect to the number of outputs needs to predict. But when it comes to hidden layers there is not general method to select the number of neurons for the layer.

When deciding an optimum number of hidden layers there are many things to be considered.

- Number of input and outputs
- Number of training cases
- Noise in the dataset
- Neural network architecture
- Complexity of the problem
- Type of activation
- Training algorithm
- Regularization

These are some of the things that decide the above problem There are many researches mention couple of methods when deciding the number or layers and neurons for the hidden layers. Some of these are outdated with time. But here I list those findings

When selecting number of neurons in hidden layers

1. Hidden layer nodes should be somewhere between the input layer size and the output layer size - **Blum, 1992, p. 60**
2. The number of hidden neurons should be roughly 2/3 of number of inputs neurons plus number of output neurons. - **FAQ in a commercial neural network software company**
3. Hidden layer units no need to go beyond twice the number of hidden units as input layer - **Berry and Linoff, 1997, p. 323**

According to the above recommendation, there is some insight how to select hidden layers in a feed forward netowork.

When selecting number of hidden layers following can be considered

| **Number of hidden layers** | **Results** |
|---|---|
| None | Capable of representing linearly separable functions |
| 1 | Approximate any function that has continuous mapping from one finite space to another |
| 2 | Can represent an arbitrary decision boundary to arbitrary accuracy with rational activation functions and can approximate any smooth mapping to any accuracy. |
| > 2 | Additional layers can learn complex representations like automatic feature engineering |

For our problem We can assume going more that 2 layers is pointless.

## Question 2

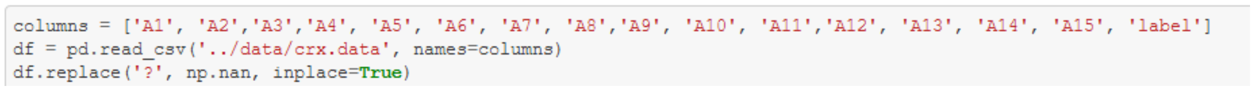
**Construct feed forward neural networks with one or more hidden layers for the above prediction problem and report their average F1 measure over 5-fold cross validation of the dataset. You can use any tool/language you prefer though Keras is recommended. You will need to try out neural networks with different sizes, different activation functions and different training loss functions (try squared error and cross entropy with logistic loss) before you arrive at the best results you can get. Report the average F1 for each neural network you tryout and highlight the best result. You can use any guidelines you may find from your literature review in Q1.**

Keras was used to implement this neural network. Full code can be found my GitHub account

Link to the code: https://github.com/DulanGit/Credit-Screening/blob/master/src/Credit%20Approval%20Version%202.0.ipynb

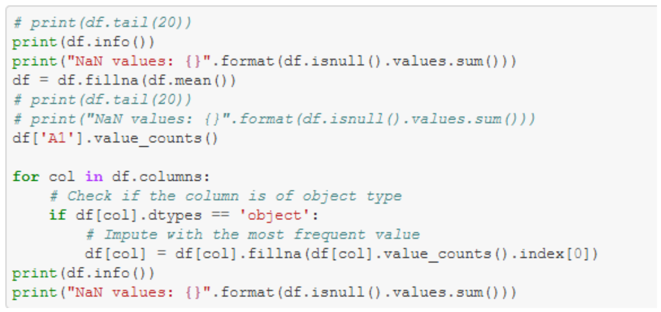
Some of the parts from the code is attached with this document.

Load data



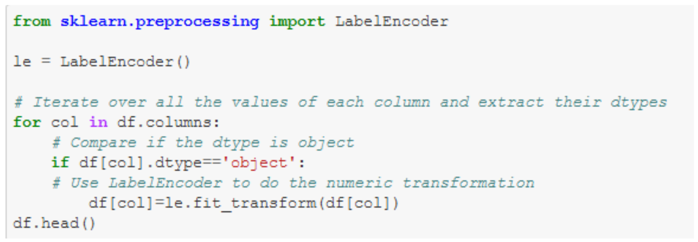
```
columns = ['A1', 'A2','A3','A4', 'A5', 'A6', 'A7', 'A8','A9', 'A10', 'A11','A12', 'A13', 'A14', 'A15', 'label']
df = pd.read_csv('../data/crx.data', names=columns)
df.replace('?', np.nan, inplace=True)
```

Remove missing values



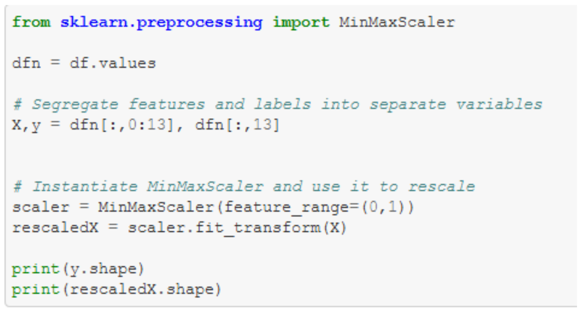
```
# print(df.tail(20))
print(df.info())
print("NaN values: {}".format(df.isnull().values.sum()))
df = df.fillna(df.mean())
# print(df.tail(20))
# print("NaN values: {}".format(df.isnull().values.sum()))
df['A1'].value_counts()

for col in df.columns:
    # Check if the column is of object type
    if df[col].dtypes == 'object':
        # Impute with the most frequent value
        df[col] = df[col].fillna(df[col].value_counts().index[0])
print(df.info())
print("NaN values: {}".format(df.isnull().values.sum()))
```

Convert none numeric data into numeric values



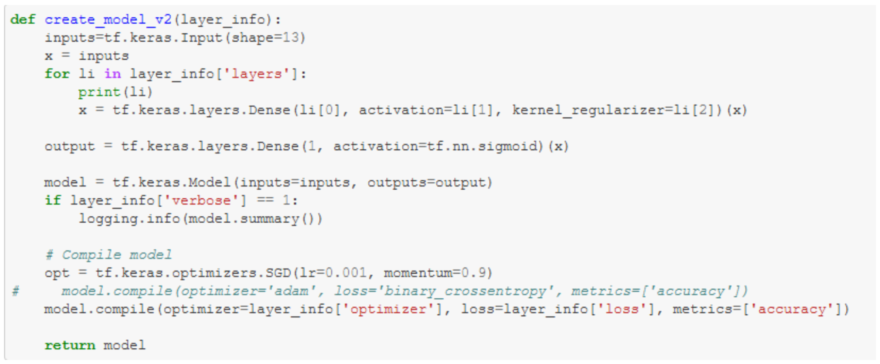
```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

# Iterate over all the values of each column and extract their dtypes
for col in df.columns:
    # Compare if the dtype is object
    if df[col].dtype=='object':
    # Use LabelEncoder to do the numeric transformation
        df[col]=le.fit_transform(df[col])
df.head()
```

Normalize data



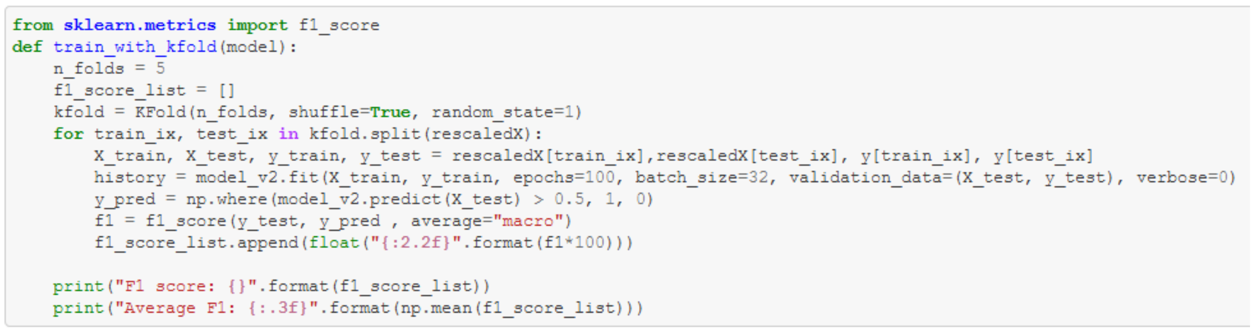
```python
from sklearn.preprocessing import MinMaxScaler

dfn = df.values

# Segregate features and labels into separate variables
X,y = dfn[:,0:13], dfn[:,13]


# Instantiate MinMaxScaler and use it to rescale
scaler = MinMaxScaler(feature_range=(0,1))
rescaledX = scaler.fit_transform(X)

print(y.shape)
print(rescaledX.shape)
```

Create model



```python
def create_model_v2(layer_info):
    inputs=tf.keras.Input(shape=13)
    x = inputs
    for li in layer_info['layers']:
        print(li)
        x = tf.keras.layers.Dense(li[0], activation=li[1], kernel_regularizer=li[2])(x)

    output = tf.keras.layers.Dense(1, activation=tf.nn.sigmoid)(x)

    model = tf.keras.Model(inputs=inputs, outputs=output)
    if layer_info['verbose'] == 1:
        logging.info(model.summary())

    # Compile model
    opt = tf.keras.optimizers.SGD(lr=0.001, momentum=0.9)
#     model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    model.compile(optimizer=layer_info['optimizer'], loss=layer_info['loss'], metrics=['accuracy'])

    return model
```

Training with KFold



```python
from sklearn.metrics import f1_score
def train_with_kfold(model):
    n_folds = 5
    f1_score_list = []
    kfold = KFold(n_folds, shuffle=True, random_state=1)
    for train_ix, test_ix in kfold.split(rescaledX):
        X_train, X_test, y_train, y_test = rescaledX[train_ix],rescaledX[test_ix], y[train_ix], y[test_ix]
        history = model_v2.fit(X_train, y_train, epochs=100, batch_size=32, validation_data=(X_test, y_test), verbose=0)
        y_pred = np.where(model_v2.predict(X_test) > 0.5, 1, 0)
        f1 = f1_score(y_test, y_pred , average="macro")
        f1_score_list.append(float("{:2.2f}".format(f1*100)))

    print("F1 score: {}".format(f1_score_list))
    print("Average F1: {:.3f}".format(np.mean(f1_score_list)))
```

## Question 3

**Explain whether your results agree or not with what you found in your literature review in Q1.**

- Different number of layers

| Number of Layers | Average F1 Score |
|---|---|
| **1** | 86.1 |
| **2** | 87.7 |
| **3** | 87.0 |
| **4** | 87.2 |
| **5** | 86.8 |

Experiment was conducted with different number of layers. The best results were coming when the layer size is 2, after that there was no big improvement. I think two layers is enough to map the data to the output variable. More that 2 layers might not help since that can result in overfitting the data to the model.

- Different number of neurons

| Number of neurons per layer | Average F1 Score |
|---|---|
| **5** | 85.3 |
| **10** | 85.8 |
| **15** | 86.4 |
| **20** | 85.5 |
| **25** | 86.1 |
| **30** | 86.0 |

In this experiment all other variables were set to constant and changed the neurons in the layer. The neuron count was increased and measured F1 score with respect to those. It was seen that with more neurons the F1 score was increased and when further increased the accuracy again was going down. Optimum number of neurons were available when the neuron size is around 15.

Below the feed forward neural network was tested with different activation functions, loss functions and optimizers. The results are as follows. For all these experiments all other conditions were fixed constant.

- Different activation functions

| Hidden layer Activation function | Average F1 Score |
|---|---|
| **Relu** | 85.85 |
| **Softmax** | 35.67 |
| **Elu** | 84.97 |
| **Relu6** | 85.54 |
| **Leaky Relu** | 85.72 |

- Different Loss functions

| Loss function | Average F1 Score |
|---|---|
| **Binary Cross entropy** | 86.13 |
| **Categorical Cross Entropy** | 30.81 |
| **Sparse Categorical Cross Entropy** | 30.76 |
| **Poisson** | 86.24 |

- Different Optimizers

| Optimizer | Average F1 Score |
|---|---|
| **SDG** | 85.54 |
| **RMSprop** | 84.98 |
| **Adam** | 86.00 |
| **Adagrad** | 85.86 |

## Question 4

**Take the best network you found in Q2 above and retrain it with the addition of regularization to the loss function. Try L1 and L2 regularization. For each training set in your 5fold cross validation, use part of it as a validation set to try out different values of λ to find the best tradeoff between the loss function and the regularizing term. Report the final average F1 score on the test sets of the 5-fold cross validation and comment on whether regularization helped or not.**

Best Network found

| Element | Value |
|---|---|
| **Hidden layers** | 2 |
| **Neurons per layer** | 15 |
| **Activation function** | Relu |
| **Loss function** | Poisson |
| **Optimizer** | Adam |

Different Regularizations and lambda values

| Lambda Value | L1 Regularization | L2 Regularization | L1 L2 Regularization |
|---|---|---|---|
| **None** | 86.42 | 85.98 | 85.85 |
| **0.1** | 35.67 | 85.20 | 35.67 |
| **0.01** | 85.27 | 85.71 | 85.29 |
| **0.001** | 84.96 | 85.84 | 85.72 |
| **0.0001** | 85.70 | 86.29 | 85.42 |
| **0.00001** | 85.55 | 85.97 | 86.12 |
| **0.000001** | 85.99 | 85.83 | 85.71 |

Looking at the above graph it's clear that regularization didn't help the network to achieve more accuracy. Regularization is one of the important pre requisites to improve the reliability, speed and accuracy of the convergence. But this is not the solution for every problem. May be regularization helps to converge the model in fever epochs but it didn't improve the accuracy.

## Conclusion

There are many ways to determine the number of hidden layers to add in the feed forward neural network. These numbers will depend on many factors. First, we can get a general idea about how many layers and neurons should be added to the architecture. Then we can do trial and test to get the optimum value. There is no direct answer how many layers are neurons to use in given feed forward neural network.

## References

http://www.faqs.org/faqs/ai-faq/neural-nets/part3/section-10.html

https://stackoverflow.com

https://www.heatonresearch.com/2017/06/01/hidden-layers.html

https://www.tutorialspoint.com/keras

https://keras.io/api

https://stats.stackexchange.com