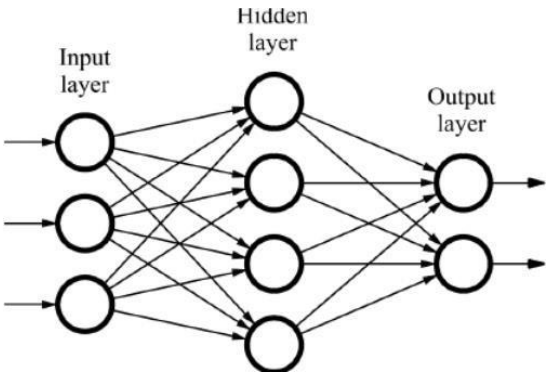
ARTIFICIAL NEURAL NETWORK
CASE STUDY

Diabetes Prediction Using Back Propagation Algorithm



Aim : - Implementation of Backpropagation algorithm in Neural Network.

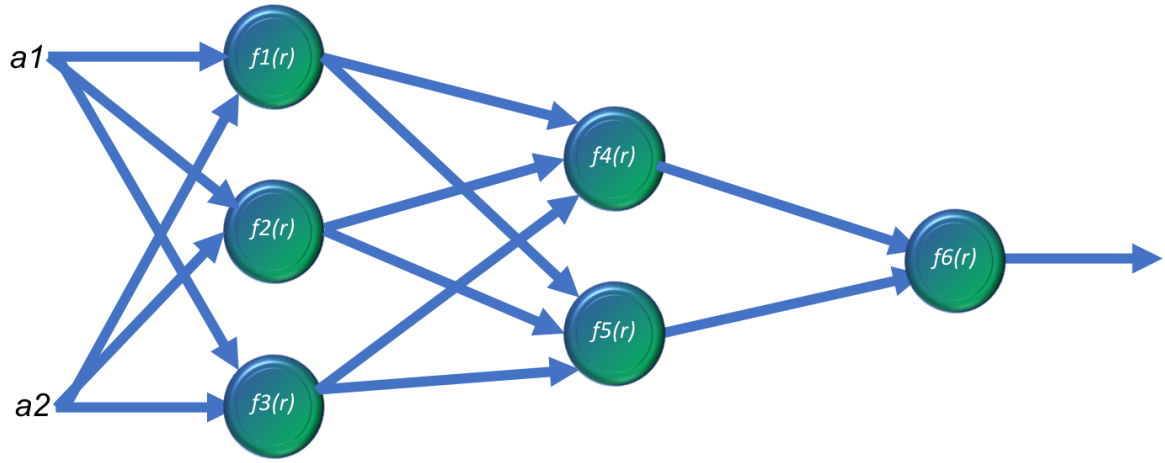
Description : - Neural Network with Backpropagation

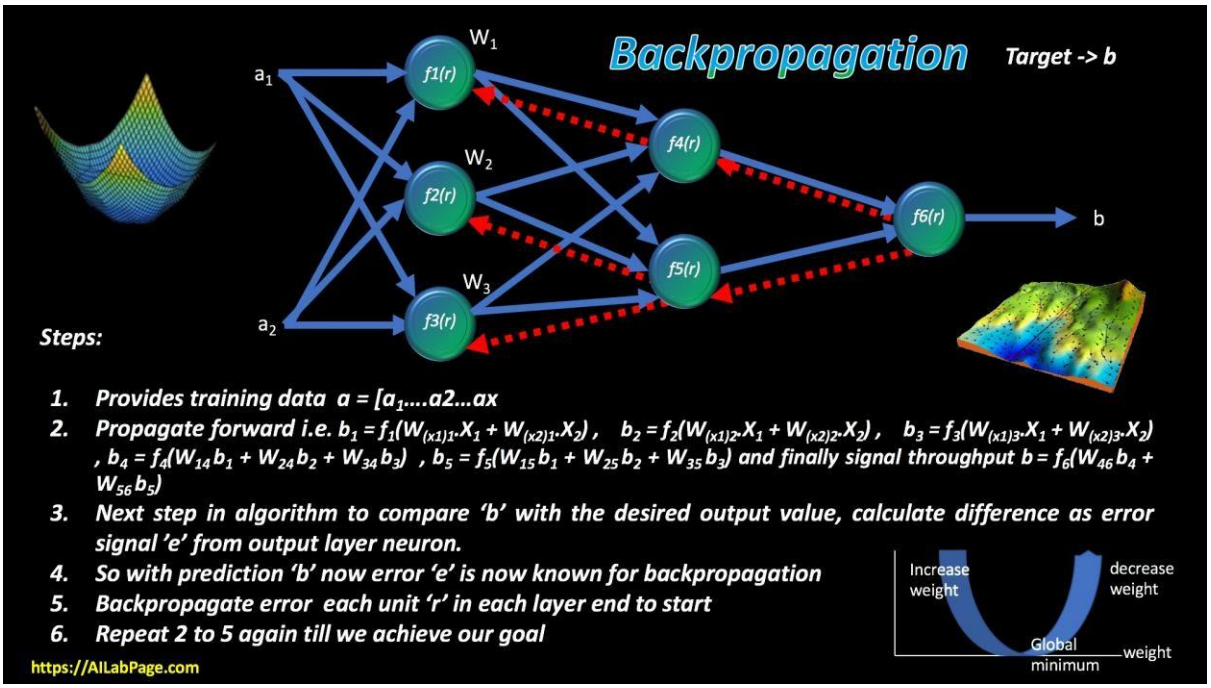
In the single-layer neural network, the training process is relatively straightforward because the error (or loss function) can be computed as a direct function of the weights, which allows easy gradient computation. In the case of multi-layer networks, the problem is that the loss is a complicated composition function of the weights in earlier layers. The gradient of a composition function is computed using the backpropagation algorithm. The backpropagation algorithm leverages the chain rule of differential calculus, which computes the error gradients in terms of summations of local-gradient products over the various paths from a node to the output. Although this summation has an exponential number of components (paths), one can compute itefficiently using dynamic programming.

The backpropagation algorithm is a direct application of dynamic programming.It contains two main phases, referred to as the forward and backward phases, respectively. The forward phase is required to compute the output values and the local derivatives at various nodes, and the backward phase is required to accumulate the products of these local values over all paths from the node to the output:

1. Forward phase: In this phase, the inputs for a training instance are fed into the neural network. This results in a forward cascade of computations across the layers, using the current set of weights. The final predicted output can be compared to that of the training instance and the derivative of the loss function with respect to the output is computed. The derivative of this loss nowneeds to be computed with respect to the weights in all layers in the backwardsphase.

2. Backward phase: The main goal of the backward phase is to learn the gradient of the loss function with respect to the different weights by using thechain rule of differential calculus. These gradients are used to update the weights. Since these gradients are learned in the backward direction, starting from the output node, this learning process is referred to as the backward phase.





Applications of Backpropagation

- Image analysis
- Source code recognition
- Face and speech recognition

Disadvantages of Backpropagation

- Backpropagation possibly be sensitive to noisy data and irregularity
- The performance of this is highly reliant on the input data
- Needs excessive time for training

```
import pandas as pd
df = pd.read_csv("/content/diabetes_data_upload.csv")
df. dropna ( axis = 0 , inplace = True )
print(df)
print(df.isnull())
```

```
     517            Yes             Yes       No     Yes  Positive
     518             No              No      Yes      No  Negative
     519             No              No       No      No  Negative

     [520 rows x 17 columns]
           Age  Gender  Polyuria  Polydipsia  sudden weight loss  weakness  \
     0    False   False     False       False               False     False
     1    False   False     False       False               False     False
     2    False   False     False       False               False     False
     3    False   False     False       False               False     False
     4    False   False     False       False               False     False
     ..     ...     ...       ...         ...                 ...       ...
     515  False   False     False       False               False     False
     516  False   False     False       False               False     False
     517  False   False     False       False               False     False
     518  False   False     False       False               False     False
     519  False   False     False       False               False     False

          Polyphagia  Genital thrush  visual blurring  Itching  Irritability  \
     0         False           False            False    False         False
     1         False           False            False    False         False
     2         False           False            False    False         False
     3         False           False            False    False         False
     4         False           False            False    False         False
     ..          ...             ...              ...      ...           ...
     515       False           False            False    False         False
     516       False           False            False    False         False
     517       False           False            False    False         False
     518       False           False            False    False         False
     519       False           False            False    False         False

          delayed healing  partial paresis  muscle stiffness  Alopecia  Obesity  \
     0              False            False             False     False    False
     1              False            False             False     False    False
     2              False            False             False     False    False
     3              False            False             False     False    False
     4              False            False             False     False    False
     ..               ...              ...               ...       ...      ...
     515            False            False             False     False    False
     516            False            False             False     False    False
     517            False            False             False     False    False
     518            False            False             False     False    False
     519            False            False             False     False    False

          class
     0    False
     1    False
     2    False
     3    False
     4    False
     ..     ...
     515  False
     516  False
     517  False
     518  False
     519  False

     [520 rows x 17 columns]
```

```
df = df.replace("Yes", 1)
```

```
df = df.replace("No",0)
```

```
df
```

	Age	Gender	Polyuria	Polydipsia	sudden weight loss	weakness	Polyphagia	Genital thrush	visual blurring	Itching	Irritabili
0	40	Male	0	1	0	1	0	0	0	1	
1	58	Male	0	0	0	1	0	0	1	0	
2	41	Male	1	0	0	1	1	0	0	1	
3	45	Male	0	0	1	1	1	1	0	1	
4	60	Male	1	1	1	1	1	0	1	1	
...	...	...	...	...	...	...	...	...	...	...	
515	39	Female	1	1	1	0	1	0	0	1	
516	48	Female	1	1	1	1	1	0	0	1	
517	58	Female	1	1	1	1	1	0	1	0	
518	32	Female	0	0	0	1	0	0	1	1	
519	42	Male	0	0	0	0	0	0	0	0	

520 rows × 17 columns

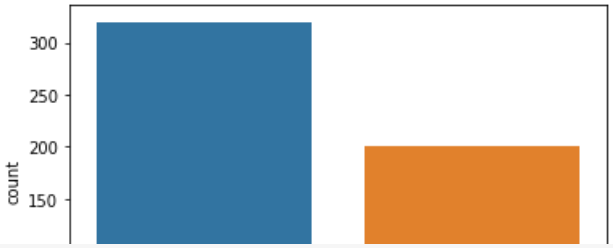
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 520 entries, 0 to 519
Data columns (total 17 columns):
 #   Column              Non-Null Count  Dtype 
---  ------              --------------  ----- 
 0   Age                 520 non-null    int64 
 1   Gender              520 non-null    object
 2   Polyuria            520 non-null    int64 
 3   Polydipsia          520 non-null    int64 
 4   sudden weight loss  520 non-null    int64 
 5   weakness            520 non-null    int64 
 6   Polyphagia          520 non-null    int64 
 7   Genital thrush      520 non-null    int64 
 8   visual blurring     520 non-null    int64 
 9   Itching             520 non-null    int64 
 10  Irritability        520 non-null    int64 
 11  delayed healing     520 non-null    int64 
 12  partial paresis     520 non-null    int64 
 13  muscle stiffness    520 non-null    int64 
 14  Alopecia            520 non-null    int64 
 15  Obesity             520 non-null    int64 
 16  class               520 non-null    object
dtypes: int64(15), object(2)
memory usage: 73.1+ KB
```

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
%matplotlib inline
import numpy as np
```

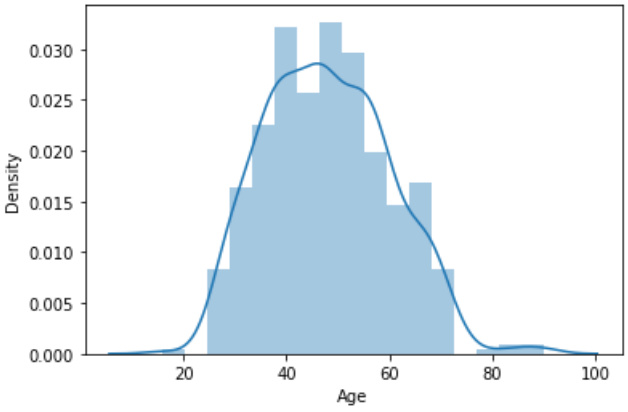
```
sns.countplot(df['class'],)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variable
  FutureWarning
<matplotlib.axes._subplots.AxesSubplot at 0x7fce529dbe50>
```




```
sns.distplot(df['Age'])
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a depreca
  warnings.warn(msg, FutureWarning)
<matplotlib.axes._subplots.AxesSubplot at 0x7fce52918c10>
```




```
df = df.drop('Gender', axis=1)
df = df.replace("Positive", 1)
df = df.replace("Negative", 0)
X = df.drop('class', axis=1)
y = df['class']
X_train,X_test,y_train,y_test = train_test_split(X,y)
```

```
X_train.head()
```

	Age	Polyuria	Polydipsia	sudden weight loss	weakness	Polyphagia	Genital thrush	visual blurring	Itching	Irritability	dela heal
150	50	1	1	1	1	1	0	0	1	1	
143	53	1	0	1	0	0	0	0	0	0	
425	62	1	1	0	1	1	0	1	0	1	
113	79	0	1	1	1	1	1	0	1	1	

```
y_train.head()
```

```
150    1
143    1
425    1
113    1
376    0
Name: class, dtype: int64
```

Keras is an open source neural network library written in Python.

There are two ways to build Keras models: sequential API, functional API

```
from keras.models import Sequential
from keras.layers import Dense, Dropout
```

The model design:

- 4 layers.
- 27 total neurons
- Relu & Sigmoid activation functions.

link text

.

```
model = Sequential()

model.add(Dense(15, input_dim=15, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(3, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss="mse", optimizer="adam", metrics=['accuracy'])
```

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_4 (Dense)             (None, 15)                240

 dense_5 (Dense)             (None, 8)                 128

 dense_6 (Dense)             (None, 3)                 27

 dense_7 (Dense)             (None, 1)                 4

=================================================================
Total params: 399
Trainable params: 399
Non-trainable params: 0
_________________________________________________________________
```

```
history = model.fit(X_train, y_train, epochs = 2500, batch_size=15, validation_data=(X_test, y_test))
```

```
Streaming output truncated to the last 5000 lines.
26/26 [==============================]  - 1s 9ms/step - loss: 0.3932 - accuracy: 0.6051 - val_loss: 0.3463
Epoch 2/2500
26/26 [==============================]  - 0s 3ms/step - loss: 0.2592 - accuracy: 0.7026 - val_loss: 0.1913
Epoch 3/2500
26/26 [==============================]  - 0s 4ms/step - loss: 0.1838 - accuracy: 0.8051 - val_loss: 0.1749
Epoch 4/2500
26/26 [==============================]  - 0s 4ms/step - loss: 0.1672 - accuracy: 0.8128 - val_loss: 0.1607
Epoch 5/2500
26/26 [==============================]  - 0s 4ms/step - loss: 0.1529 - accuracy: 0.8410 - val_loss: 0.1524
Epoch 6/2500
26/26 [==============================]  - 0s 3ms/step - loss: 0.1423 - accuracy: 0.8564 - val_loss: 0.1450
Epoch 7/2500
26/26 [==============================]  - 0s 4ms/step - loss: 0.1334 - accuracy: 0.8590 - val_loss: 0.1386
Epoch 8/2500
26/26 [==============================]  - 0s 3ms/step - loss: 0.1317 - accuracy: 0.8513 - val_loss: 0.1341
Epoch 9/2500
26/26 [==============================]  - 0s 4ms/step - loss: 0.1218 - accuracy: 0.8718 - val_loss: 0.1303
Epoch 10/2500
26/26 [==============================]  - 0s 3ms/step - loss: 0.1141 - accuracy: 0.8667 - val_loss: 0.1244
Epoch 11/2500
26/26 [==============================]  - 0s 3ms/step - loss: 0.1135 - accuracy: 0.8615 - val_loss: 0.1210
Epoch 12/2500
26/26 [==============================]  - 0s 3ms/step - loss: 0.1088 - accuracy: 0.8744 - val_loss: 0.1226
Epoch 13/2500
```

```
26/26 [==============================]  - 0s 3ms/step - loss: 0.1047 - accuracy: 0.8821 - val_loss: 0.1160
Epoch 14/2500
26/26 [==============================]  - 0s 4ms/step - loss: 0.0991 - accuracy: 0.8692 - val_loss: 0.1161
Epoch 15/2500
26/26 [==============================]  - 0s 4ms/step - loss: 0.0971 - accuracy: 0.8897 - val_loss: 0.1128
Epoch 16/2500
26/26 [==============================]  - 0s 3ms/step - loss: 0.0944 - accuracy: 0.8821 - val_loss: 0.1216
Epoch 17/2500
26/26 [==============================]  - 0s 3ms/step - loss: 0.0944 - accuracy: 0.8769 - val_loss: 0.1129
Epoch 18/2500
26/26 [==============================]  - 0s 4ms/step - loss: 0.0920 - accuracy: 0.8872 - val_loss: 0.1089
Epoch 19/2500
26/26 [==============================]  - 0s 3ms/step - loss: 0.0927 - accuracy: 0.8846 - val_loss: 0.1035
Epoch 20/2500
26/26 [==============================]  - 0s 3ms/step - loss: 0.0881 - accuracy: 0.8897 - val_loss: 0.1026
Epoch 21/2500
26/26 [==============================]  - 0s 3ms/step - loss: 0.0877 - accuracy: 0.8897 - val_loss: 0.1008
Epoch 22/2500
26/26 [==============================]  - 0s 3ms/step - loss: 0.0879 - accuracy: 0.8974 - val_loss: 0.1062
Epoch 23/2500
26/26 [==============================]  - 0s 3ms/step - loss: 0.0876 - accuracy: 0.8949 - val_loss: 0.1066
Epoch 24/2500
26/26 [==============================]  - 0s 3ms/step - loss: 0.0845 - accuracy: 0.8974 - val_loss: 0.0991
Epoch 25/2500
26/26 [==============================]  - 0s 3ms/step - loss: 0.0826 - accuracy: 0.9154 - val_loss: 0.1071
Epoch 26/2500
26/26 [==============================]  - 0s 3ms/step - loss: 0.0824 - accuracy: 0.9026 - val_loss: 0.0962
Epoch 27/2500
26/26 [==============================]  - 0s 3ms/step - loss: 0.0798 - accuracy: 0.9051 - val_loss: 0.1062
Epoch 28/2500
26/26 [==============================]  - 0s 4ms/step - loss: 0.0831 - accuracy: 0.8923 - val_loss: 0.0954
Epoch 29/2500
```

Saving the model Keras also supports a simpler interface to save both the model weights and model architecture together into a single H5 file.

Saving the model in this way includes everything we need to know about the model, including:

Model weights. Model architecture. Model compilation details (loss and metrics). Model optimizer state. This means that we can load and use the model directly, without having to re-compile it.

```
model.save('model.h5')
```

```
from keras.models import load_model
model = load_model('model.h5')
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_4 (Dense)             (None, 15)                240

 dense_5 (Dense)             (None, 8)                 128

 dense_6 (Dense)             (None, 3)                 27

 dense_7 (Dense)             (None, 1)                 4

=================================================================
Total params: 399
Trainable params: 399
Non-trainable params: 0
_________________________________________________________________
```

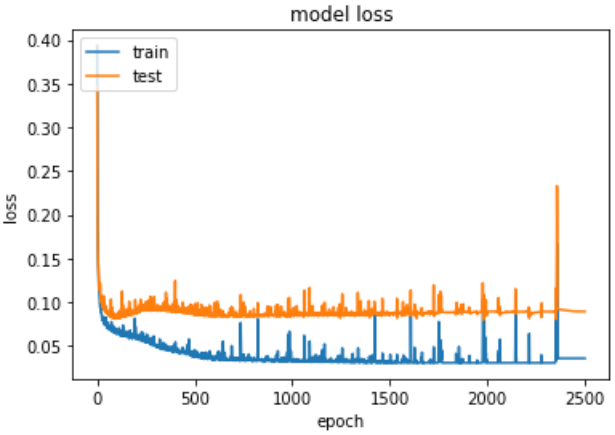
```
print(model.predict(np.array([53,0,1,0,1,0,1,0,0,0,1,0,0,1,1]).reshape((1,15))))
```

```
1/1 [==============================] - 0s 33ms/step
[[0.4941433]]
```

```
_, accuracy = model.evaluate(X_train, y_train)
print('Accuracy: %.2f' % (accuracy*100))
```

```
13/13 [==============================] - 0s 8ms/step - loss: 0.0359 - accuracy: 0.9641
Accuracy: 96.41
```

```
import matplotlib.pyplot as plt
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```




```
# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```