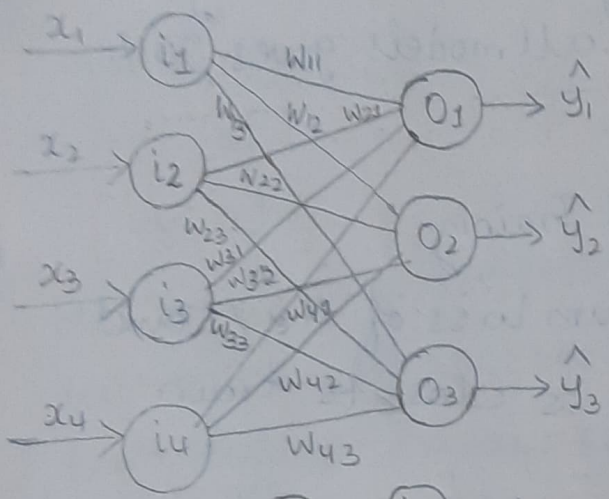
8. Multiclass Classification of Iris Dataset Using Keras

Aim: To perform multiclass classification using Keras, on Iris dataset

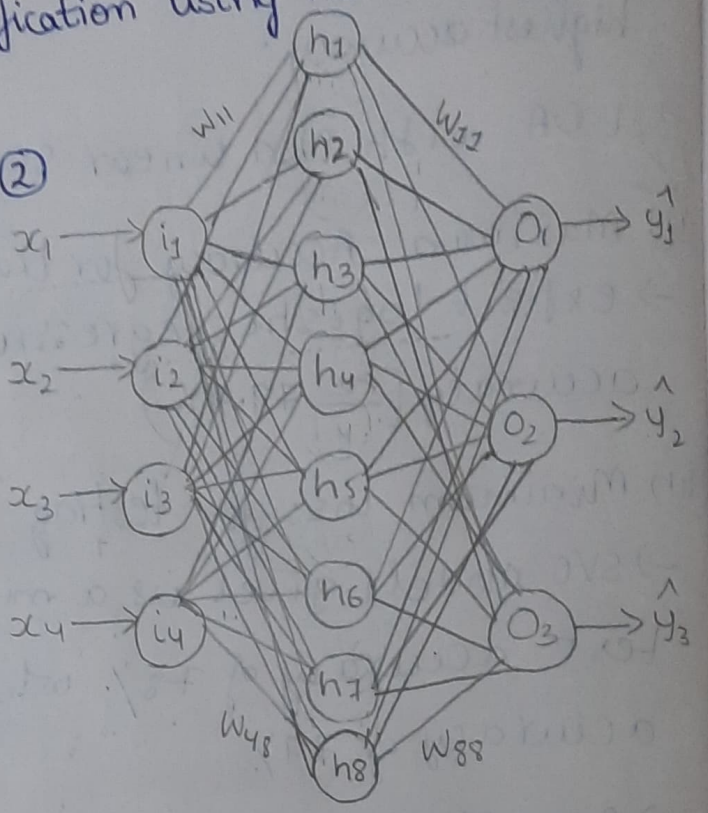
Description:

Architectures:

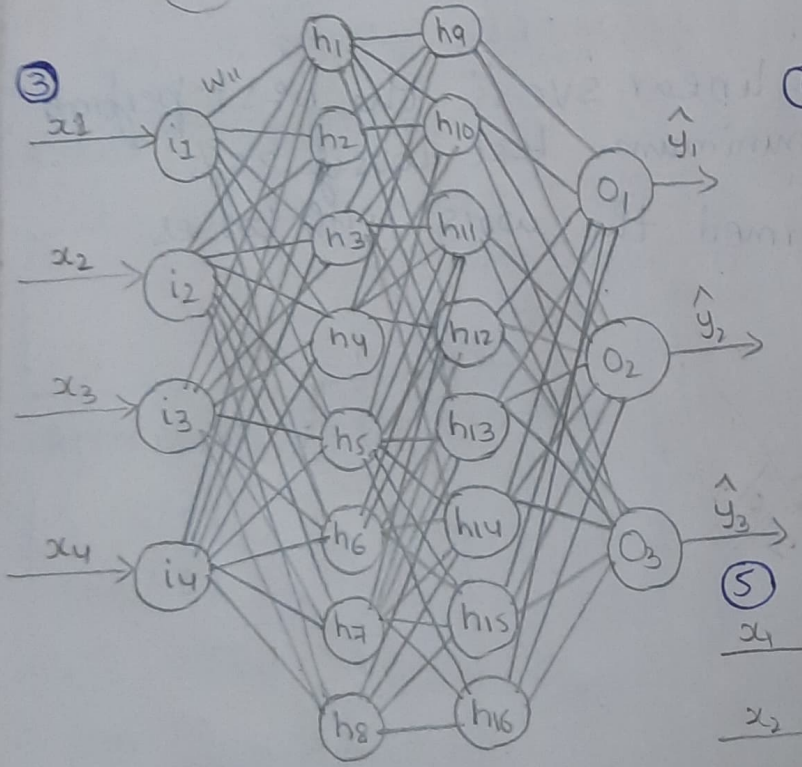
①



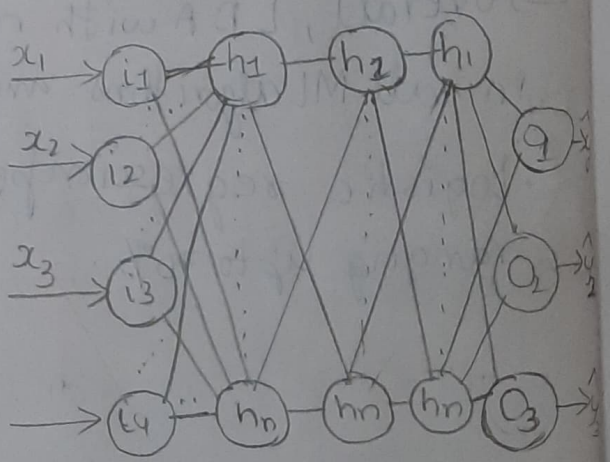
②



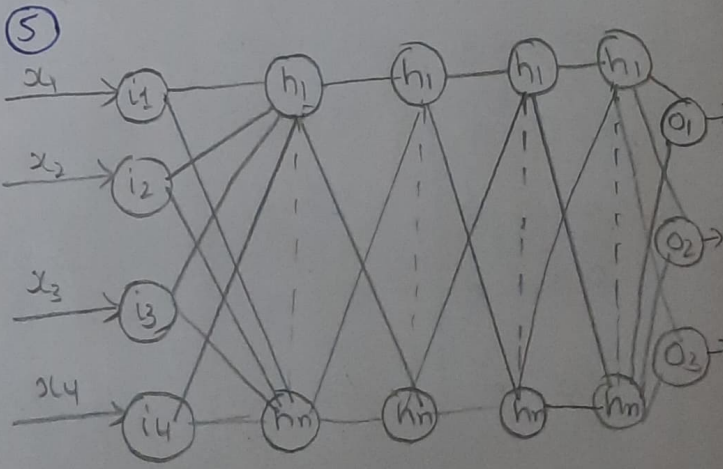
③



④



⑤



→ inputs are x_1, x_2, x_3 and x_4 which is sepal length, sepal width, petal length and petal width. Output neuron O_1, O_2 and O_3 represent the predicted class (Iris setosa, Iris versicolor or Iris virginica)

Activation function: (i) Relu (ii) (Sigmoid or) Softmax

(i) Relu – used in hidden layers of the model

formula: $\boxed{\phi(x_1) = \max\{x_1, 0\}}, \phi$

(ii) Sigmoid/max used in Output layer of the model

formula: $\boxed{\phi(x_i) = \frac{e^{x_j}}{\sum_{i=1}^{3} e^{x_i}}, \phi(x_2) = \frac{e^{x_2}}{\sum_{i=1}^{3} e^{x_i}}, \phi(x_3) = \frac{e^{x_3}}{\sum_{i=1}^{3} e^{x_i}}}$

such that, $\phi(x_1) + \phi(x_2) + \phi(x_3) = 1$

Loss function (for Multinomial logistic Regression)

(i) Categorial crossentropy - It is a multiclass extension/version of binary cross entropy.

formula: $\boxed{-\log(\hat{y}_{c(i)})}$ which is the loss of true class c(i

One hot encoding:

→ It is a machine learning technique which converts given categorical information into a format such that all other inputs are zeros

Ex: for iris setosa $x_1 = [1\,0\,0]$ and iris virginica $= [0\,0$
iris versicolor, $x_2 = [0\,1\,0]$

Code:
```
import sklearn
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
df = pd.read_csv('Iris.csv')
x = df.loc[:, :5], Value
y = df.loc[:, -1], Value
```

Code:

```
# no hidden layer
model = Sequential()
model.add(normalizer)
model.add(layers.Dense(units=3, activation='softmax'))
model.compile(loss="categorical-crossentropy", optimizer="adam",
              metrics=["accuracy"])

model.summary()
history = model.fit(x_train, y_train, epochs=10, batch_size=2, Validation
          _data=(x_test, y_test))
loss, accuracy = model.evaluate(x_test, y_test)
Test_loss.append(loss)
Test_accuracy.append(accuracy)
loss, accuracy = model.evaluate(x_train, y_train)
Train_loss.append(loss)
Train_accuracy.append(accuracy)
```

Output: Model: "sequential"

Layer (type)	Output Shape	Param #
normalization (Normalization)	(None, 6)	11
dense	(None, 3)	18

Total params: 29

Trainable params: 18

Non-trainable params: 11

Code: # with 1 hidden layer

```
model = sequential( )
model.add(normalizer)
model.add(layers.Dense(8, activation = "relu", input_dim=5))
model.add(layer.Dense(Units = 3, activation= 'softmax'))
model.compile(loss= "categorical_crossentropy", optimizer = "adam
              metrics =["accuracy"])
model.summary( )
history = model.fit(x_train, y_train, epochs=10, batch_size = 2,
                    validation_data = (x_test, y_test).
loss, accuracy = model.evaluate(x_test, y_text)
Test_loss.append (loss)
Test_accuracy: append(accuracy)
Test, accuracy = model.evaluate(x_train, y_train)
Train_loss.append(loss)
Train_accuracy.append(accuracy)
```

Output: Model: "sequential 1"

Layer(type)	Output Shape	Param #
normalization (Normalization)	(None, 5)	11
dense	(None, 8)	48
dense_1	(None, 3)	27

Total params: 86
Trainable Params: 75
non-trainable Params: 11

Code: # 2 hidden layers

```
model = sequential( )
model.add(normalizer)
model.add(layers.Dense( 8, activation = "relu", input_dim=5))
model.add(Dense(units=8, Kernel_initializer='uniform', activation=
model.add (layers.Dense(Units=3, activation = 'softmax'))
model.compile (loss="Categorical_crossentropy", optimizer = "adam",
                metrics = ["_accuracy"]
model.summary ( )
history = model.fit(x_train, y_train, epoch = 10, batch_size = 2, Validation
                _data = (x.test, y_test))
loss, accuracy = model.evaluate (x_test, y_test)
Test_loss.append(loss)
Test_accuracy.append(accuracy)
Loss, accuracy = model.evaluate(x_train, y_train)
Train_loss.append(loss)
Train_accuracy.append(accuracy)
```

Code: # With 3 hidden layers

```
model_sequential( )
model.add (normalizer)
model.add(layers.Dense(8, activation = "relu", input_dim=5))
model.add(Dense(Units=8, Kernel_initializer = 'uniform', activation
                = 'relu')
model.add(Dense(Units=8, Kernel_initializer = 'uniform', activation
                = 'relu')
```

```
model.add(layers.Dense(units=3, activation="softmax"))
model.add(Dense(units=8, ker
model.compile(loss="categorical_crossentropy", optimizer="adam", metrics
    =["accuracy"])
model.summary()
history = model.fit(x_train, y_train, epoch=10, batch_size=2, validation
    _data=(x_test, y_test))
loss, accuracy = model.evaluate(x_test, y_test)
loss, accuracy = model.evaluate(x_train, y_train)
```

Outputs: (2 hidden layers)

model: "sequential_2"

Layer (type)	Output shape	Param #
normalization	(None, 5)	11
dense (Dense)	(None, 8)	48
dense_1 (Dense)	(None, 8)	72
dense_2 (Dense)	(None, 3)	27

Total params: 158
Trainable Params: 147
Non-trainable params: 11

(3 hidden layer)

model: "sequential_3"

Layer (type)	Output Shape	Param #
normalization	(None, 5)	11
dense (Dense)	(None, 8)	48
dense_1 (Dense)	(None, 8)	72
dense_2 (Dense)	(None, 8)	72
dense_3 (Dense)	(None, 3)	27

Total param: 230
Trainable Params: 219
Non-trainable params: 11

Code: # 4 hidden layers

```
model = sequential()
model.add(normalizer)
model.add(layers.Dense(8, activation="relu", input_dim=5))
model.add(Dense(units=8, kernel_initializer='uniform',
    activation='relu'))
```

```
model.add(Dense(Units=8, kernel_initializer='uniform' activation='relu'))
model.add(Dense(Units=8 kernel_initializer='uniform'
model.add(Layers.Dense(Units=3, activation='softmax'))
model.Compile(loss="categorical_crossentropy", optimizer="adam",
              metrices=["accuracy"],)
model.summary()
history = model.fit(x_train, y_train, epoch=10, batch_size=2,
          validation_data=(x_test, y_test).
loss, accuracy = model.evaluate(x_tst, y_test)
loss, accuracy = model.evaluate(x_train, y_train)
```

Output:

Model: "Sequential 4"

Layer(type)	Output Shape	Param#
normalization	(None, 5)	11
dense (dense)	(None, 8)	48
dense_1 (dense)	(None, 8)	72
dense_2 (dense)	(None, 8)	72
dense_3 (dense)	(None, 8)	72
dense_4 (dense)	(None, 3)	27

Total Params: 302
Trainable Params: 291
Non-trainable Params: 11

Result/Observation:-

Test loss	Test accuracy	Train Loss	Train accuracy	Model Number	
0.44	0.80	0.38	0.85	0 hidden layer	
0.43	0.77	0.29	0.93	1 hidden layer	
0.16	0.97	0.15	0.98	2 hidden layer	→ best model
0.22	0.97	0.18	0.98	3 hidden layer	
0.42	0.60	0.35	0.74	4 hidden layer	→ worst model

Best Model:

→ Model with 2 hidden layers had the best test and train accuracy along with least loss in terms of testing and training, that is, only 16% and 15%

→ 3 hidden layers is the next best performer with test loss of only 20% and train loss of 18% along with train accuracy of 98%

Worst Model:

→ Upon adding 4 hidden layers, the model's accuracy is observered to be the lowest, that is, test accuracy is only 60% and train accuracy is 74%.

Conclusion:

∴ Models with about 2-3 hidden layers perform well.