How number of hidden layers effect the efficiency of a Machine Learning Model

Selection of hidden layers is very difficult task as in some cases due to the number of hidden layers a condition known as overfitting and underfitting occurs, which effect the efficiency and time complexity of the network very badly.

Overfitting condition occurs when the number of hidden layers become very large as compared to the complexity of the problem due to which a process of overtraining of the network starts and badly effects the time complexity of the network and it mostly arises when network matches the data so closely that it loses its generalization ability over the data to be tested.

Underfitting condition occurs when the number of hidden layers in the network becomes less than complexity of the problem as network barely handles such problems. This is also known as undertraining. It effects the efficiency of the network very negatively. In such case the time complexity of the network become very low and produces inefficient results.

Normally a network having large number of hidden layers shows a very large time complexity except in case of overfitting and networks having a smaller number of hidden layers show a very satisfactory time constraints except in under fitting condition.

the training process of Neural networks slows down if the large number of hidden layers are used. So, if the criteria of the problem is to get the better accuracy, then large number of hidden layers is the most suitable solution but if the time complexity is the major factor of an application then large number of hidden layers will not work in these types of application. Also, unnecessary increment in the neurons or layer will led to overfitting problem. So it is quiet essential that before designing the Neural network, training database samples must be analyzed so that approximation of number of neurons and hidden layers can be guessed properly

Hyperparameters in a neural network

A hyperparameter controls the learning process and therefore their values directly impact other parameters of the model such as weights and biases which consequently impacts how well our model performs. The accuracy of any machine learning model is most often improved by fine-tuning these hyperparameters.

1. Effect of Batch Size

The gradient descent technique is used to train neural networks. In this technique, the estimate of the error (difference between actual and predicted variables) based on a subset of the training dataset is used to update the weights in every iteration. Batch size is defined as the number of examples that are used from the training dataset for estimating the error gradient and is an important hyperparameter that influences the dynamics of the learning algorithm. In mini-batch gradient descent, the batch size is set to more than one and less than the total number of examples in the training dataset. When we increase batch size, we should also adjust the learning rate to compensate for this.

1. Effect of Activation Functions

An activation function in a neural network transforms the weighted sum of the inputs into an output from a node in a layer of the network. If an activation function is not used, the neural networks become just a linear regression model as these functions enable the non-linear transformation of the inputs making them capable to learn and perform more complex tasks. There are several commonly used non-linear activation functions like sigmoid, tanh, and ReLU.

1. Effect of Optimizers

While training a deep learning model, the weights and biases associated with each node of a layer are updated at every iteration with the objective of minimizing the loss function. This adjustment of weights is enabled by algorithms like stochastic gradient descent which are also known by the name of optimizers. AdaGrad, RMSProp, Adam, AdaMax, and all maintain and adapt learning rates for each of the weights in the model.

*Basically, anything in machine learning and deep learning that you decide their values or choose their configuration before training begins and whose values or configuration will remain the same when training ends is a hyperparameter.*

*Basically, anything in machine learning and deep learning that you decide their values or choose their configuration before training begins and whose values or configuration will remain the same when training ends is a hyperparameter.*

Here are some common examples

Train-test split ratio

Learning rate in optimization algorithms (e.g. gradient descent)

Choice of optimization algorithm (e.g., gradient descent, stochastic gradient descent, or Adam optimizer)

Choice of activation function in a neural network (nn) layer (e.g. Sigmoid, ReLU, Tanh)

The choice of cost or loss function the model will use

Number of hidden layers in a nn

Number of activation units in each layer

The drop-out rate in nn (dropout probability)

Number of iterations (epochs) in training a NN

Number of clusters in a clustering task

Kernel or filter size in convolutional layers

Pooling size

Batch size

Parameters

Parameters on the other hand are internal to the model. That is, they are learned or estimated purely from the data during training as the algorithm used tries to learn the mapping between the input features and the labels or targets.

Examples of parameters

The coefficients (or weights) of linear and logistic regression models.

Weights and biases of a NN

The cluster centroids in clustering

Recursion

def tri\_recursion(k):

  if(k > 0):

    result = k + tri\_recursion(k - 1)

    print(result)

  else:

    result = 0

  return result

print("\n\nRecursion Example Results")

tri\_recursion(6)

Factorial of a Number

n = int(input("Enter a number: "))

factorial = 1

if n < 0: #Check if the number is negative

   print("Factors do not exist for negative numbers.")

elif n == 0: #Check if the number is zero

   print("The factorial of 0 is 1.")

else:

   for i in range(1,n + 1):

       factorial = factorial\*i

   print("The factorial of",n,"is",factorial)

While Loop

n = int(input("Enter a number: "))

i = 1

while i < n:

  print(i)

  if (i == n):

    break

  i += 1

For Loop

fruits = ["apple", "banana", "cherry"]

for x in fruits:

  if x == "banana":

    break

  print(x)

Array in Python

By using an array, we can store more than one data. The Array is a process of memory allocation. It is performed as a dynamic memory allocation. We can declare an array like x[100], storing 100 data in x. It is a container that can hold a fixed number of items, and these items should be the same type. An array is popular in most programming languages like C/C++, JavaScript, etc.

import array as arr

a = arr.array('i', [2, 4, 5, 6])

print("First element is:", a[0])

print("Second element is:", a[1])

print("Third element is:", a[2])

print("Forth element is:", a[3])

print("last element is:", a[-1])

print("Second last element is:", a[-2])

print("Third last element is:", a[-3])

print("Forth last element is:", a[-4])

print(a[0], a[1], a[2], a[3], a[-1],a[-2],a[-3],a[-4])

Python code for the effect of Hidden Layers on the accuracy of the model

import pandas as pd

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from sklearn.model\_selection import train\_test\_split

from tensorflow.keras.utils import plot\_model

from google.colab import files

uploaded = files.upload()

df = pd.read\_csv("diabetes.csv")

X = df.iloc[:, 0:8]

y = df.iloc[:, 8]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=0)

model = Sequential()

model.add(Dense(64, input\_dim=8, activation='relu'))

model.add(Dense(32, activation='sigmoid'))

model.add(Dense(16, activation='sigmoid'))

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

model.compile(loss='mean\_squared\_error',optimizer='adam',metrics=['accuracy'])

model.fit(X\_train, y\_train, epochs=20)

loss, accuracy = model.evaluate(X\_test,y\_test)

Python code for Union Operation

A = {"a": 0.2, "b": 0.3, "c": 0.6, "d": 0.6}

B = {"a": 0.9, "b": 0.9, "c": 0.4, "d": 0.5}

print('The First Fuzzy Set is :', A)

print('The Second Fuzzy Set is :', B)

for A\_key, B\_key in zip(A, B):

  A\_value = A[A\_key]

  B\_value = B[B\_key]

  if A\_value > B\_value:

    Y[A\_key] = A\_value

  else:

    Y[B\_key] = B\_value

print('Fuzzy Set Union is :', Y)

Python code for Intersection Operation

# Example to Demonstrate

# Intersection of Two Fuzzy Sets

A = {"a": 0.2, "b": 0.3, "c": 0.6, "d": 0.6}

B = {"a": 0.9, "b": 0.9, "c": 0.4, "d": 0.5}

print('The First Fuzzy Set is :', A)

print('The Second Fuzzy Set is :', B)

for A\_key, B\_key in zip(A, B):

    A\_value = A[A\_key]

    B\_value = B[B\_key]

    if A\_value < B\_value:

        Y[A\_key] = A\_value

    else:

        Y[B\_key] = B\_value

print('Fuzzy Set Intersection is :', Y)

Python code for Difference Operator

# Example to Demonstrate the

# Difference Between Two Fuzzy Sets.

A = {"a": 0.2, "b": 0.3, "c": 0.6, "d": 0.6}

B = {"a": 0.8, "b": 0.8, "c": 0.4, "d": 0.5}

print('The First Fuzzy Set is :', A)

print('The Second Fuzzy Set is :', B)

for A\_key, B\_key in zip(A, B):

    A\_value = A[A\_key]

    B\_value = B[B\_key]

    B\_value = round(1 - B\_value, 2)

    if A\_value < B\_value:

        Y[A\_key] = A\_value

    else:

        Y[B\_key] = B\_value

print('Fuzzy Set Difference is :', Y)

Python code for Difference Operator

A = {"a": 0.2, "b": 0.3, "c": 0.6, "d": 0.6}

B = {"a": 0.8, "b": 0.8, "c": 0.4, "d": 0.5}

print('The First Fuzzy Set is :', A)

print('The Second Fuzzy Set is :', B)

for A\_key, B\_key in zip(A, B):

    A\_value = A[A\_key]

    B\_value = B[B\_key]

    D\_value = A\_value + B\_value - A\_value \* B\_value

    Y[B\_key] = D\_value

print('Fuzzy Set Difference is :', Y)

Python code for Max-Min Composition

import numpy as np

# Max-Min Composition given by Zadeh

def maxMin(x, y):

    z = []

    for x1 in x:

        for y1 in y.T:

            z.append(max(np.minimum(x1, y1)))

    return np.array(z).reshape((x.shape[0], y.shape[1]))

# 3 arrays for the example

r1 = np.array([[0.6, 0.5], [1, 0.1], [0, 0.7]])

r2 = np.array([[0.7, 0.3, 0.4], [0.9, 0.1, 0.6]])

print ("R1oR2 => Max-Min :\n" + str(maxMin(r1, r2)) + "\n")

#print ("R1oR2 => Max-Product :\n" + str(maxProduct(r1, r2)) + "\n\n")

Python code for Max-Product Composition

# Max-Product Composition given by Rosenfeld

def maxProduct(x, y):

    z = []

    for x1 in x:

        for y1 in y.T:

            z.append(max(np.multiply(x1, y1)))

    return np.array(z).reshape((x.shape[0], y.shape[1]))

# 3 arrays for the example

r1 = np.array([[1, 0, .7], [.3, .2, 0], [0, .5, 1]])

r2 = np.array([[.6, .6, 0], [0, .6, .1], [0, .1, 0]])

print ("R1oR2 => Max-Product :\n" + str(maxProduct(r1, r2)) + "\n\n")

Neural Network Tool Box

Deep Learning Toolbox provides functions, apps, and Simulink blocks for designing, implementing, and simulating deep neural networks. The toolbox provides a framework to create and use many types of networks, such as convolutional neural networks (CNNs) and transformers. You can visualize and interpret network predictions, verify network properties, and compress networks with quantization, projection, or pruning.

Fuzzy Logic toolbox

The toolbox lets you automatically tune membership functions and rules of a fuzzy inference system from data. You can evaluate the designed fuzzy logic systems in MATLAB and Simulink.

## Python File Handling

Python supports file handling and allows users to handle files i.e., to read and write files, along with many other file handling options, to operate on files. The concept of file handling has stretched over various other languages, but the implementation is either complicated or lengthy, like other concepts of Python, this concept here is also easy and short. [Python](https://www.geeksforgeeks.org/python-programming-language/) treats files differently as text or binary and this is important.

f = open(filename, mode)

Where the following mode is supported:

1. r: open an existing file for a read operation.
2. w: open an existing file for a write operation. If the file already contains some data, then it will be overridden but if the file is not present then it creates the file as well.
3. a:  open an existing file for append operation. It won’t override existing data.
4. r+:  To read and write data into the file. The previous data in the file will be overridden.
5. w+: To write and read data. It will override existing data.
6. a+: To append and read data from the file. It won’t override existing data.