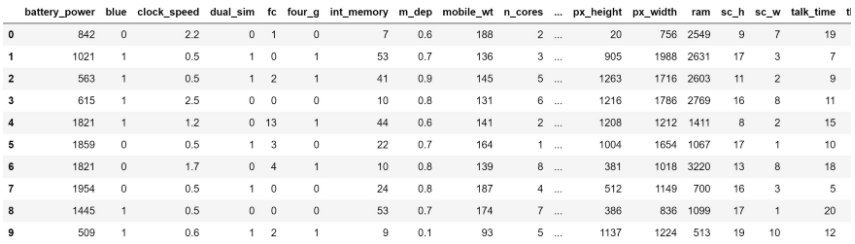
An overview of how to structure any deep learning project.

1. **Preprocess and load data-** As we have already discussed data is the key for the working of neural network and we need to process it before feeding to the neural network. In this step, we will also visualize data which will help us to gain insight into the data.
2. **Define model-** Now we need a neural network model. This means we need to specify the number of hidden layers in the neural network and their size, the input and output size.
3. **Loss and optimizer-**Now we need to define the loss function according to our task. We also need to specify the optimizer to use with learning rate and other hyperparameters of the optimizer.
4. **Fit model-**This is the training step of the neural network. Here we need to define the number of epochs for which we need to train the neural network.

After fitting model, we can test it on test data to check whether the case of overfitting. We can save the weights of the model and use it later whenever required.

import numpy as np  
import pandas as pd  
dataset = pd.read\_csv(‘train.csv’)

dataset.head()



#Changing pandas dataframe to numpy array  
X = dataset.iloc[:,:20].values  
y = dataset.iloc[:,20:21].values

#Normalizing the data  
from sklearn.preprocessing import StandardScaler  
sc = StandardScaler()  
X = sc.fit\_transform(X)

X.head()

Normalization is a technique used to change the values of an array to a common scale, without distorting differences in the ranges of values.

It is mainly required in case the dataset features vary a lot as in our case the value of battery power is in the 1000’s and clock speed is less than 3.

So if we feed unnormalized data to the neural network, the gradients will change differently for every column and thus the learning will oscillate. Study further from this [link](https://www.quora.com/Why-do-we-normalize-the-data). The X will now be changed to this form:

Normalized data:  
[-0.90259726 -0.9900495 0.83077942 -1.01918398 -0.76249466 -1.04396559  
 -1.38064353 0.34073951 1.34924881 -1.10197128 -1.3057501 -1.40894856  
 -1.14678403 0.39170341 -0.78498329 0.2831028 1.46249332 -1.78686097  
 -1.00601811 0.98609664]

**FOR TARGET, BINARY VALUES**

Next step is to one hot encode the classes, to convert integer classes into binary values.

If we have a feature 1,2,3 we need to convert to binary to be processed.

Now there is one unique binary value for the class. The new array formed will be of shape (n, number of classes), where n is the number of samples in our dataset. We can do this using simple function by sklearn:

from sklearn.preprocessing import OneHotEncoder  
ohe = OneHotEncoder()  
y = ohe.fit\_transform(y).toarray()

Our dataset has 4 classes so our new label array will look like this:

Los valores previos de y eran 0,1,2,3 y se cambiaron a:

One hot encoded array:  
[[0. 1. 0. 0.]  
 [0. 0. 1. 0.]  
 [0. 0. 1. 0.]  
 [0. 0. 1. 0.]  
 [0. 1. 0. 0.]]

Now our dataset is processed and ready to feed in the neural network.

from sklearn.model\_selection import train\_test\_split  
X\_train,X\_test,y\_train,y\_test = train\_test\_split(X,y,test\_size = 0.1)

In our dataset, the input is of 20 values and output is of 4 values. So the input and output layer is of 20 and 4 dimensions respectively.

import keras  
from keras.models import Sequential  
from keras.layers import Dense # Neural network

model = Sequential()  
model.add(Dense(16, input\_dim=20, activation=’relu’))  
model.add(Dense(12, activation=’relu’))  
model.add(Dense(4, activation=’softmax’))

In our neural network, we are using two hidden layers of 16 and 12 dimension.

**Sequential** specifies to keras that we are creating model sequentially and the output of each layer we add is input to the next layer we specify.

**model.add** is used to add a layer to our neural network. We need to specify as an argument what type of layer we want.

The **Dense** is used to specify the fully connected layer.

The optimizer is Adam. Metrics is used to specify the way we want to judge the performance of our neural network. Here we have specified it to accuracy.

Training step is simple in keras. model.fit is used to train it.

history = model.fit(X\_train, y\_train, epochs=100, batch\_size=64)

It will take around a minute to train. And after 100 epochs the neural network will be trained. The training accuracy is reached 99.5 % so our model is trained.

Now we can check the model’s performance on test data:

y\_pred = model.predict(X\_test)

#Converting predictions to label  
pred = list()  
for i in range(len(y\_pred)):  
 pred.append(np.argmax(y\_pred[i]))

#Converting one hot encoded test label to label  
test = list()  
for i in range(len(y\_test)):  
 test.append(np.argmax(y\_test[i]))

This step is inverse one hot encoding process. We will get integer labels using this step. We can predict on test data using a simple method of keras, model.predict(). It will take the test data as input and will return the prediction outputs as softmax.

from sklearn.metrics import accuracy\_score  
a = accuracy\_score(pred,test)  
print('Accuracy is:', a\*100)

We get an accuracy of 93.5%.

We can use test data as validation data and can check the accuracies after every epoch. This will give us an insight into overfitting at the time of training only and we can take steps before the completion of all epochs. We can do this by changing fit function as:

history = model.fit(X\_train, y\_train,validation\_data = (X\_test,y\_test), epochs=100, batch\_size=64)

import matplotlib.pyplot as plt  
plt.plot(history.history['acc'])  
plt.plot(history.history['val\_acc'])  
plt.title('Model accuracy')  
plt.ylabel('Accuracy')  
plt.xlabel('Epoch')  
plt.legend(['Train', 'Test'], loc='upper left')  
plt.show()

A picture containing graphical user interface

Description automatically generated

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()

Shape

Description automatically generated with medium confidence

To predict? Set a value and then normalize.