Name : Oladapo Ogunnaike
CS : 577 Number:A20435179

Training Data

Training Data are primarily used for training models ,its is used in model training, or in other words, it's the data used to fit the model

Importance:
It teaches what the expected output looks like. The neural network then analyzes the dataset repeatedly to deeply understand its characteristics and adjust itself for better performance.

Modern deep learning often involves tens or even hundreds of successive layers of representations — and they're all learned automatically from
exposure to training data

These are the data that the model will learn from, mostly used during the learning process and for fitting the parameters

Validation Data

It is used for frequent evaluation during the training phase
These are the sample of data used to provide an unbiased evaluation of a model fit on the training dataset while tuning model hyperparameters

They are important in a neural network because :

The validation data set provides an unbiased evaluation of a model fit on the training data set while tuning the model's hyper-parameters (e.g. the amount of hidden units—layers and layer widths—in a neural network

It helps protect models from overfitting and underfitting.

Validation datasets are often used for regularization by early stopping (stopping training when the error on the validation data set increases, as this is often a sign of over-fitting to the training data set)

Testing Data

It is used to evaluate the performance or accuracy of the model

They are important in a Neural network because:
They are used to get an unbiased estimate of the final model performance

testing dataset is used to evaluate how effective the training was or how accurate the model is.

These are data set used to provide an unbiased evaluation of a final model fit on the training data set

They are used to test a neural network after it has been trained on an initial training data set

2, **Data**: Data is the key for the working of neural network and we need to process it before feeding to the neural network.
Keras can also be substituted for TensorFlow, since tensorflow has the keras interface .

- You will use the NumPy library to load your dataset and two classes from the Keras library to define your model.

Below are the required imports

```
from numpy import loadtxt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

We load the file as a matrix of numbers using the NumPy function loadtxt()

will be learning a model to map rows of input variables (X) to an output variable (y), which is often summarized as y = f(X).
There would be 1 output variable (0,1) this is for the case of Binary classification.
- The first step or the Data is loading it in keras , this involves creating the training data ,train labels and the test data , test labels from the data.
- load_data splits the data into test and train data
- The training data is the data used for classification
- after loading the data set we use classes from the Keras library to define the model

np.ravel(labels) create an array from the Pandas series labels.

```
import numpy as np
from sklearn.model_selection import train_test_split
X=features
y=np.ravel(labels)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=--, random_state=--)
```

The libraries include ,
Class model: Model groups layers into an object with training and inference features.

class Sequential: Sequential groups a linear stack of layers into a tf.keras.Model.

-we can instantiate a Model by:

1 - With the "Functional API", where you start from Input, you chain layer calls to specify the model's forward pass, and finally you create your model from inputs and outputs:

```
import tensorflow as tf

class MyModel(tf.keras.Model):

  def __init__(self):
    super().__init__()
    self.dense1 = tf.keras.layers.Dense(4, activation=tf.nn.relu)
    self.dense2 = tf.keras.layers.Dense(5, activation=tf.nn.softmax)

  def call(self, inputs):
    x = self.dense1(inputs)
    return self.dense2(x)

model = MyModel()
```

```
import tensorflow as tf

class MyModel(tf.keras.Model):

  def __init__(self):
    super().__init__()
    self.dense1 = tf.keras.layers.Dense(4, activation=tf.nn.relu)
    self.dense2 = tf.keras.layers.Dense(5, activation=tf.nn.softmax)
    self.dropout = tf.keras.layers.Dropout(0.5)

  def call(self, inputs, training=False):
    x = self.dense1(inputs)
    if training:
      x = self.dropout(x, training=training)
    return self.dense2(x)

model = MyModel()
```

MODELS

1- Defining the model

Models in Keras are defined as a sequence of layers.

-We first create a Sequential model and add layers one at a time until we are satisfied with the network architecture.
-We ensure the input layer has the correct number of input features This can be specified when creating the first layer with the input_shape argument and setting it to correspond with the number of input variables.

-knowing the number of layers to use in a model. we can use the following to determine the number of layers to use in a model

Experimentation:

Use systematic experimentation to discover what works best for your specific dataset.

Using a robust test harness and controlled experiments

-Fully connected layers are defined using the Dense class. You can specify the number of neurons or nodes in the layer as the first argument and the activation function using the activation argument.

-Use the rectified linear unit activation function referred to as ReLU on the first two layers and the Sigmoid function in the output layer.

```
...
model = Sequential()
model.add(Dense((number of node), input_shape=(number of input variable), activation='relu'))
model.add(Dense((number of input variable), activation='relu'))
model.add(Dense(1, activation='sigmoid'))
...
```

Compile Keras Model /learning process

During the compilation we must specify the following :

-loss function, this is to used to evaluate a set of weights,

We use binary_crossentropy for the loss function and Stochastic Gradient Descent for the optimizer as well as different activation functions.

-Specify the optimizer used to search through different weights for the network,

-They optional metrics you want to collect and report during training.

-define the optimizer as the efficient stochastic gradient descent algorithm

- if it is a classification problem, you will collect and report the classification accuracy defined via the metrics argument.

```
from keras import optimizers
from keras import matrix

Model.compile ( optimizer='adam' ,
                loss = "categorical_crossentropy',
                Metrics=['accuracy'])
```

Training the Model/Fit Keras Model

- Training or fitting your model on your loaded data is done by calling the **fit()** function on the model.

-The training Data (images and labels),commonly know as x and y, respectively

-Training occurs over epochs, and each epoch is split into batches.

-Epoch: One pass through all of the rows in the training dataset

-Batch: One or more samples considered by the model within an epoch before weights are updated

```
...
# fit the keras model on the dataset
model.fit(X, y, epochs=150, batch_size=10)
…

Or

Model.fit(
        train_images,    #training data
        train_labels,     #training targets
        epochs =5,
        batch_size =  32,
```

Testing/Evaluate Keras Model

-Evaluate the model on the training dataset using the **evaluate()** function and pass it to the same input and output used to train the model.

-This generates a prediction for each input and output pair and collects scores, including the average loss and any metrics that have been configured, such as accuracy.

-The evaluate() function will return a list with two values: the loss of the model on the dataset, and the accuracy of the model on the dataset.

Make Predictions

Making predictions is as easy as calling the **predict()** function on the model

3, The Number of Units:

This is the most basic parameter , it uses positive integer as it value and represents the output size of the layer.
It is the unit parameter itself that plays a major role in the size of the weight matrix along with the bias vector.

It dictates the size of the weight matrix and bias vector (the bias vector will be the same size, but the weight matrix will be calculated based on the size of the input data so that the dot product will produce data that is of output size, units).

The Dense layer

Dense layer is the regular deeply connected neural network layer.

Dense Layer is used to classify images based on output from convolutional layers. Each Layer in the Neural Network contains neurons, which compute the weighted average of its input, and this weighted average is passed through a non-linear function, called an “activation function

Activation :this represents the activation function,
activation is the element-wise activation function passed as the activation argument
It is helpful when applying the element-wise activation function in a dense layer by dereliction, Linear Activation is used but we can alter and switch to any one of numerous options that Keras provides for this.

In artificial neural network, activation function is a function which transforms the incoming input signal of artificial neuron.
Activation function is useful to introduce non linearity into the neural network so that the network can learn complex relationship between input and affair data.
The popular activation functions are- step function, sigmoid function, softmax function, tanh function, ReLU, Leaky ReLU.

4, **The Optimizer** : This is the mechanism through which the model will update it self based on the training data it sees so as to improve the performance .

An optimizer updates the model in response to the output of the loss function .

It assist in minimizing the loss function.

It involves randomly initializing and manipulating the value of weights for every epoch to increase the model network’s accuracy potential
Example are : Adagrad, adam, Nadam, Adamax, RSMprop.

The Loss function : the model measures its performance on the training data an thus how it will be able to steer itself in the right direction

The purpose of loss functions is to compute the quantity that a model should seek to minimize during training.

Metrics : This is to monitor during training and testing

Difference between the Loss and the Matrix:
- the loss compute the quantity that a model should seek to minimize during training.
-The metric judges the performance of the model.
-the results from evaluating a metric are not used when training the model.

5, **Input:** this is a function in Keras, the Input() is used to instantiate a Keras tensor.
inputs: The input(s) of the model: a keras.Input object or list of keras.Input objects.
Only dicts, lists, and tuples of input tensors are supported

outputs: The output(s) of the model.The output layer is the layer in a neural network model that directly outputs a prediction.
The out put layer is the final layer of the neural network. This is where desired predictions are obtained.

Batch Size is a hyperparameter that defines the number of samples to work through before updating the internal model parameters, batch size is the number of samples that will be passed through to the network at one time, it controls the accuracy of the estimate of the error gradient when training neural networks.
Integer or None. Number of samples per gradient update. If unspecified, batch_size will default to 32.

Epoch :is a hyperparameter that defines the number times that the learning algorithm will work through the entire training dataset.
One epoch means that each sample in the training dataset has had an opportunity to update the internal model parameters

Validation Data:It's used during the training process to see how the network would perform on data it hasn't been directly trained on, it also "validate" that the neural network hasn't rather memorized the training data.

6. The techniques used to turn Text into features can be referred to as "Text Vectorization"
7.
8. **A layer** consists of a tensor-in tensor-out computation function
A Layer instance is callable, layers maintain a state, updated when the layer receives data during training, and stored in layer.weights:
the layers of a neural network are compiled and an optimizer is assigned. The optimizer is responsible to change the learning rate and weights of neurons in the neural network to reach the minimum loss function. Optimizer is very important to achieve the possible highest accuracy or minimum loss.
The **activation function** decides how to compute the input values of a layer into output values. The output values of a layer are then passed to the next layer as input values again.

10. **One-hot encoding** is meant for inputs to many models, but outputs for only a few

Step 1 - Import the library. from sklearn.preprocessing import MultiLabelBinarizer. ...

Step 2 - Setting up the Data. We have created a arrays of differnt labels with few of the labels in common. ...
Step 3 - Using MultiLabelBinarizer and Printing Output

11. the softmax activation is used to derive the final class in the classification problem.

12. **categorical_crossentropy (cce)** produces a one-hot array containing the probable match for each category, **sparse_categorical_crossentropy (scce)** produces a category index of the most likely matching category

13. **Random Classifier:**
    The equation of the classification accuracy for a random classifier
    Accuracy = 1/k (here k is the number of classes) for 5 classes
    1/5 = 20%

14. The data can be normalized by subtracting the mean (μ) of each feature and a division by the standard deviation (σ).

Normalization can help training of our neural networks as the different features are on a similar scale, which helps to stabilize the gradient descent step, allowing us to use larger learning rates or help models converge faster for a given learning rate.

15. **MAE** is the average distance between the real data and the predicted data, but fails to punish large errors in prediction.

MSE measures the average squared difference between the estimated values and the actual value

MSE is used as a default metric for calculating Loss Function despite being harder to interpret than MAE.

k-Fold Cross-Validation

Cross-validation is when the dataset is randomly split up into 'k' groups. One of the groups is used as the test set and the rest are used as the training set. The model is trained on the training set and scored on the test set.

K fold Cross - Validation is performed by :

-Spliting the dataset into training data and test data
-The parameters will undergo a Cross-Validation test to see which are the best parameters to select.
-These parameters will then be implemented into the model for retraining

Final evaluation will occur and this will depend if the cycle has to go a gain, depending on the accuracy and the level of generalization that the model performs.

K-fold Cross Validation : ensures that the score of our model does not depend on the way we select our train and test subsets.

K-Fold Cross Validation: The idea behind k-fold cross-validation is to divide all the available data items into roughly equal-sized sets. Each set is used exactly once as the test set while the remaining data is used as the training set.