Neural Networks

Practical – W8

Task 1: Training a simple Neural Network on Iris dataset

Neural networks, inspired by the human brain, offer powerful tools for classification tasks. In today's lab sessions, we'll provide a concise overview of the essential steps in training a neural network on the Iris dataset—from data preparation and model construction to training and evaluation. You can train a NN using the following steps.

1. **Data Loading:**
   Load the Iris dataset containing measurements of three iris species.
2. **Data Preprocessing:**
   Split the dataset into features (X) and labels (y).
   Standardize or normalize the feature values.
3. **Model Construction:**
   Initialize a sequential model using Keras.
   Add layers to the model, specifying the input dimensions and activation functions.
   Typically, include one or more hidden layers with ReLU activation and an output layer with softmax activation for multiclass classification.
4. **Model Compilation:**
   Compile the model by specifying the optimizer, loss function, and evaluation metric.
   Common choices include 'adam' optimizer and 'categorical_crossentropy' loss for classification.
5. **Model Training:**
   Fit the model to the training data using the fit function.
   Specify the number of epochs and batch size for training.
6. **Model Evaluation:**
   Evaluate the trained model on the test dataset.
   Use metrics such as accuracy to assess the model's performance.

```
from keras.models import Sequential
from keras.layers import Dense
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import numpy as np
import tensorflow as tf
```

The code block imports the required libraries

```
# Load Iris dataset
iris = load_iris()
X, y = iris.data, iris.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Standardize features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Load Iris Dataset:

- Use the load_iris function from the sklearn.datasets module to load the Iris dataset.
- X contains the feature values (sepal length, sepal width, petal length, petal width), and y contains the corresponding target labels (species of iris).

Split the Dataset:

- Employ the train_test_split function from sklearn.model_selection to split the dataset into training and testing sets.
- X_train and y_train store the feature and target values for the training set, respectively.
- X_test and y_test hold the feature and target values for the testing set.
- The test_size parameter specifies the proportion of the dataset for the test split (20% in this case), and random_state ensures reproducibility.

Standardize Features:

- Utilize the StandardScaler from sklearn.preprocessing to standardize the feature values.
- The fit_transform method computes the mean and standard deviation of each feature on the training set and standardizes the values.
- The transform method applies the same transformation to the test set, ensuring consistent feature scaling between training and testing data.

```
# Encoding y
print("Before label conversion to categorical: ", y_train[0:5]) # The
format of the labels before conversion
y_train  = tf.keras.utils.to_categorical(y_train, num_classes=3)
print("After label conversion to categorical: ", y_train[0:5]) # The
format of the labels after conversion
y_test = tf.keras.utils.to_categorical(y_test, num_classes=3)


print("Shape of y_train",y_train.shape)
print("Shape of y_test",y_test.shape)
```

The above code segment handles the encoding of categorical labels using one-hot encoding. Here's a breakdown:

Encoding y with One-Hot Encoding:

- Utilizes the to_categorical function from tf.keras.utils to convert categorical class labels (y_train and y_test) into one-hot encoded format.
- One-hot encoding represents each categorical label as a binary vector, where each dimension corresponds to a class, marked by 1, and others are 0.
- num_classes=3 indicates there are three classes in the dataset, corresponding to the three species of iris flowers.

Displaying Labels Before and After Conversion:

- Prints the format of labels before and after one-hot encoding, offering insight into the transformation that occurred in the label representation.

Displaying the Shape of y_train and y_test:

- Prints the shapes of the one-hot encoded label arrays for the training and testing sets.
- Shape information is essential for confirming the expected format after the encoding process.

```
#Building Model
model = Sequential()

# Input layer and first hidden layer
model.add(Dense(8, activation='relu', input_dim=4))

# Second hidden layer
model.add(Dense(8, activation='relu'))

# Output layer
model.add(Dense(3, activation='softmax'))
```

The above code constructs a neural network model using the Keras Sequential API:

Sequential Model Initialization:

- Initializes a sequential model, representing a linear stack of layers.

Input Layer and First Hidden Layer:

- Adds the first dense layer with 8 neurons and ReLU activation.
- input_dim=4 specifies the input layer with four features (dimensions).

Second Hidden Layer:

- Adds a second dense layer with 8 neurons and ReLU activation.
- The model automatically infers the input shape from the previous layer.

Output Layer:

- Adds the output layer with 3 neurons and softmax activation.
- Softmax activation is suitable for multi-class classification, providing probabilities for each class.

```
# compiling model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

The above code is responsible for compiling the previously defined neural network model. Here's a breakdown without code blocks:

Compilation:

- The compile method is called on the model to configure its learning process.

Optimizer:

- 'adam' is chosen as the optimizer.
- Adam is a popular optimization algorithm known for its efficiency in training neural networks.

Loss Function:

- 'categorical_crossentropy' is selected as the loss function.
- Categorical crossentropy is commonly used for multi-class classification problems with one-hot encoded labels.

Metrics:

- 'accuracy' is specified as the metric to be monitored during training.
- Accuracy is a common metric for classification tasks, representing the proportion of correctly classified instances.

```
# training model
model.fit(X_train, y_train, epochs=50, batch_size=32,
validation_data=(X_test, y_test))
```

Training the Model:

- The fit method is called on the model to train it on the specified training data (X_train and y_train).
- The number of training epochs is set to 50, indicating the number of times the entire training dataset will be processed by the neural network.
- The batch_size is set to 32, specifying the number of samples used in each iteration. It is a common practice to use mini-batches for efficiency.
- The validation_data parameter is provided with the testing data (X_test and y_test), allowing the model's performance to be evaluated on the validation set after each epoch.

```
# Evluating model
prediction=model.predict(X_test)
length=len(prediction)
y_label=np.argmax(y_test,axis=1)
predict_label=np.argmax(prediction,axis=1)

accuracy=np.sum(y_label==predict_label)/length * 100
print("Accuracy of the dataset",accuracy )
```

The above segment evaluates the trained neural network model on the test dataset:

Making Predictions:

- Uses the predict method on the model to generate predictions for the test input data (X_test).
- The resulting prediction variable holds the model's output for each test sample.

Calculating Accuracy:

- The length of the prediction array is determined, representing the number of test samples.
- Extracts the true labels (y_label) and predicted labels (predict_label) by finding the indices of the maximum values along the second axis of the one-hot encoded arrays.
- Calculates the accuracy by comparing the true labels with the predicted labels and computing the percentage of correct predictions.
- The accuracy is then printed to the console, providing an assessment of the model's performance on the test dataset.

Task 2: Improve accuracy!!!

The code provided in task 1 acquires an accuracy of 70-83%. Your task is to improve accuracy to 100%. You can do so by increasing the neural networks learning capabilities. It could be done in two ways.

1. Increase the number of hidden layers.
2. Increase the number of neurons in the hidden layers.
3. Remember!! You cannot change number of neurons in input or output layers as they depend upon the data.