Building Models In Keras

Building models

In this lecture, we will delve into the implementation of various models using the Keras library, which is renowned for its simplicity and flexibility in building deep learning applications. We will begin with an exploration of a linear regression model, a fundamental statistical method that helps us understand the relationship between input features and a continuous target variable.
Following this, we will dive into the- creation of a multi-layer perceptron (MLP) using the Iris data set, a classic machine learning dataset that contains information about different species of irises.

Building models

Here are the 7 steps to building models in TensorFlow:

1. Load the data set that we will be using for the model.
2. Divide the data set into two parts: a training set for model training and a testing set.
3. Normalize the data if needed. This step can improve convergence during training.
4. Build the Keras model by adding layers and specifying the activation functions.
5. Compile the model. This means to configure the learning process by specifying the loss function, optimizer, and any additional metrics to be monitored during training.
6. Train the model by feeding it the training data.
7. Evaluate the model's performance using the test data.

Linear Regression model

Now, let us examine an example of a linear regression model built using Keras. In this example, we will assume that we have already completed steps 1 through 3, which include loading the dataset, splitting the data, and normalizing it if necessary.
We will now build our linear regression model using Keras. For this task, we will utilize a Sequential model, which allows us to stack layers in a linear fashion. Then, we start adding the dense layers and setting the activation functions.

```
#4. Build the model

model = Sequential()
model.add(Dense(16, input_shape = (4,)))
model.add(Activation('sigmoid'))
#3 classes in the output layer, corresponding to the 3 species
model.add(Dense(3))
model.add(Activation('softmax'))
```

We will next compile the model and configure the learning process.

```
#5. Compile the model

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

Linear Regression model

Now, we will begin to train our model using the training data. Note that we set the number of epochs to 100. This means the training loop continues for 100 iterations.

```
#6. Train the model by feeding it the training data

model.fit(x_train, y_train, verbose = 1, batch_size = 1, nb_epoch = 100)
```

Next, we will evaluate the model using the test data.

```
#7. Test the model

loss, accuracy = mode.evaluate(x_test, y_test, verbose = 0)
print('\nAccuracy is using keras prediction {:.2f}'.format(accuracy))
```

Linear Regression model

Now, let us take a closer look at the output of our linear regression model built using Keras. After training the model on the dataset, we can evaluate its performance and interpret the results.

```
Epoch 1/100
75/75 [==============================] - 0s - loss: 1.2947 - acc: 0.4533
Epoch 2/100
75/75 [==============================] - 0s - loss: 1.0353 - acc: 0.6400
Epoch 3/100
75/75 [==============================] - 0s - loss: 0.8930 - acc: 0.6533
...                                   ...                                   ...
...                                   ...                                   ...
...                                   ...                                   ...
...                                   ...                                   ...
Epoch 99/100
75/75 [==============================] - 0s - loss: 0.1186 - acc: 0.9733
Epoch 100/100
75/75 [==============================] - 0s - loss: 0.1167 - acc: 0.9867
```

MLP on the Iris data set

Now, let us examine an example of a Multi-Layer Perceptron (MLP) built using Keras. In this example, we will use the Iris dataset, which is a dataset in machine learning that contains measurements of iris flowers from three different species.
First, we will import the necessary libraries.

```
import pandas as pd
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.utils import np_utils
```

We will proceed to load the data set and modify it.

```
data_train = pd.read_csv('./Datasets/iris/iris_train.csv')

#convert string value to numeric value
data_train.set_value(data_train['species'] == 'Iris-setosa', ['species'], 0)
data_train.set_value(data_train['species'] == 'Iris-versicolor', ['species'], 1)
data_train.set_value(data_train['species'] == 'Iris-virginica', ['species'], 2)
data_train = data_train.apply(pd.to_numeric)

#change dataframe to array
data_train_array = data_train.as_matrix()
```

Linear Regression model

Now, we will split the training data into inputs and outputs.

```
x_train = data_train_array[:,:4]
y_train = data_train_array[:,4]
```

Next, we will change the target format using NumPy. This transformation is essential for ensuring that our data is in the appropriate format for analysis and modeling.

```
y_train = np_utils.tocategorical(y_train)
```

MLP on the Iris data set

Now, let us dive into building our MLP model with one hidden layer. We have three main layers:

- Input layer: The input layer contains 4 neurons, representing the features of an iris: sepal length, sepal width, petal length, and petal width.
- Hidden layer: The hidden layer contains 10 neurons, and uses ReLU as its activation function.
- Output layer: The output layer contains 3 neurons, representing-

the iris SoftMax layer, Setosa, Versicolor, and Virginica. We will use the SoftMax activation function in this layer to obtain probability distributions for each class.

Linear Regression model

Here is the code for building the model with the explained characteristics.

```
#Build the model

model = Sequential()
model.add(Dense(output_dim = 10, input_dim = 4))
model.add(Activation('relu'))
model.add(Dense(output_dim = 3))
model.add(Activation('softmax'))
```

Next, we will compile the model and configure the learning process.

```
#Compile the model

mode.compile(loss = 'categorical_crossentropy', optimizer = 'sgd', metrics = ['accuracy'])
```

Linear Regression model

Now, it is time to start training our model.

```
#Train the model using the training data

model.fit(x_train, y_train, nb_epoch = 100, batch_size = 120)
```

We will now load and prepare the test data that we want to use to evaluate out model.

```
#Load the testing data

data_test = pd.read_csv('./Datasets/iris/iris_test.csv')

#convert string value to numeric value
data_test.set_value(data_test['species'] == 'Iris-setosa', ['species'], 0)
data_test.set_value(data_test['species'] == 'Iris-versicolor', ['species'], 1)
data_test.set_value(data_test['species'] == 'Iris-virginica', ['species'], 2)
data_test = data_test.apply(pd.to_numeric)

#change dataframe to array
data_test_array = data_test.as_matrix()
```

Linear Regression model

Now, we will split the test data into inputs and outputs.

```
x_test = data_test_array[:,:4]
y_test = data_test_array[:,4]
```

Next, we will make a prediction using our model

```
#Get prediction by the model

classes = model.predict_classes(x_test, batch_size = 120)
```

We can also calculate the accuracy of the model.

```
#Calculate accuracy

accuracy = np.sum(classes == y_test) / 30.0 * 100
```

Linear Regression model

After running these codes, we will have the following results:

```
Epoch 1/100
120/120 [==============================] - 0s - loss: 2.7240 -
acc: 0.3667
Epoch 2/100
120/120 [==============================] - 0s - loss: 2.4166 -
acc: 0.3667
Epoch 3/100
120/120 [==============================] - 0s - loss: 2.1622 -
acc: 0.4083
Epoch 4/100
120/120 [==============================] - 0s - loss: 1.9456 -
acc: 0.6583
```

```
Epoch 98/100
120/120 [==============================] - 0s - loss: 0.5571 -
acc: 0.9250
Epoch 99/100
120/120 [==============================] - 0s - loss: 0.5554 -
acc: 0.9250
Epoch 100/100
120/120 [==============================] - 0s - loss: 0.5537 -
acc: 0.9250
```

Conclusion

In this presentation, we explored the implementation of two fundamental machine learning models using Keras, the Multi-Layer Perceptron (MLP) and Linear Regression. Both MLPs and Linear Regression have wide-ranging applications across various domains, from predicting numerical values to classifying data points, showcasing their versatility in solving real-world problems.

You are encouraged further- exploration of other neural network architectures and machine learning models to broaden your understanding and capabilities within the field.