Train a deep learning model

In this notebook you will train a deep learning model to classify the descriptions of car components as compliant or non-compliant.

Each document in the supplied training data set is a short text description of the component as documented by an authorized technician. The contents include:

- Manufacture year of the component (e.g. 1985, 2010)
- Condition of the component (poor, fair, good, new)
- Materials used in the component (plastic, carbon fiber, steel, iron)

The compliance regulations dictate: *Any component manufactured before 1995 or in fair or poor condition or made with plastic or iron is out of compliance.*

For example:

- Manufactured in 1985 made of steel in fair condition -> **Non-compliant**
- Good condition carbon fiber component manufactured in 2010 -> **Compliant**
- Steel component manufactured in 1995 in fair condition -> **Non-Compliant**

The labels present in this data are 0 for compliant, 1 for non-compliant.

The challenge with classifying text data is that deep learning models only undertand vectors (e.g., arrays of numbers) and not text. To encode the car component descriptions as vectors, we use an algorithm from Stanford called GloVe (Global Vectors for Word Representation) (https://nlp.stanford.edu/projects/glove/). GloVe provides us pre-trained vectors that we can use to convert a string of text into a vector.

Run the following cells to import the needed libraries and to download the car components data set and the GloVe vectors file.

```
import numpy as np
import pandas as pd

import keras
from keras import models
from keras import layers
from keras import optimizers

Using TensorFlow backend.

import uuid
import os

# Create a temporary folder to store locally relevant content for this notebook
rootFolderName = 'ti_labs_{0}'.format(uuid.uuid4())
tempFolderName = '/FileStore/' + rootFolderName
osLocalFolderPath = '/dbfs{0}'.format(tempFolderName)
dbutils.fs.mkdirs(tempFolderName)
print('Content files will be saved to {0}'.format(tempFolderName))

filesToDownload = ['glove.6B.100d.txt','connected-car_components.csv']

for fileToDownload in filesToDownload:
  downloadCommand = 'wget -O ''/dbfs{0}/{1}''
''https://databricksdemostore.blob.core.windows.net/data/connected-
car/{1}'''.format(tempFolderName, fileToDownload)
  print(downloadCommand)
  os.system(downloadCommand)

#List all downloaded files
dbutils.fs.ls(tempFolderName)

Content files will be saved to /FileStore/ti_labs_481687b4-0809-4c91-be8b-54ad
d58c8c20
wget -O /dbfs/FileStore/ti_labs_481687b4-0809-4c91-be8b-54add58c8c20/glove.6B.
100d.txt https://databricksdemostore.blob.core.windows.net/data/connected-car/
glove.6B.100d.txt
wget -O /dbfs/FileStore/ti_labs_481687b4-0809-4c91-be8b-54add58c8c20/connected
-car_components.csv https://databricksdemostore.blob.core.windows.net/data/con
nected-car/connected-car_components.csv
Out[2]:
[FileInfo(path='dbfs:/FileStore/ti_labs_481687b4-0809-4c91-be8b-54add58c8c20/c
onnected-car_components.csv', name='connected-car_components.csv', size=0),
 FileInfo(path='dbfs:/FileStore/ti_labs_481687b4-0809-4c91-be8b-54add58c8c20/g
love.6B.100d.txt', name='glove.6B.100d.txt', size=0)]
```

Now that you have downloaded the data, load it into a Pandas DataFrame by running the following cell.

```
# Load the car components labeled data
car_components_df = pd.read_csv(os.path.join(osLocalFolderPath +
'/','connected-car_components.csv'))
components = car_components_df["text"].tolist()
labels = car_components_df["label"].tolist()
```

```
# split data 60% for trianing, 20% for validation, 20% for test
train, validate, test = np.split(car_components_df.sample(frac=1),
[int(.6*len(car_components_df)), int(.8*len(car_components_df))])
print(train.shape)
print(test.shape)
print(validate.shape)
```

```
(60000, 2)
(20000, 2)
(20000, 2)
```

In the following cell, you use the Tokenizer from Keras to "learn" a vocabulary from the entire car components text. Then the data (both the text and the compliance labels) is split into three subsets, one that will be used for training the deep learning model, one that will be used during training batches to tune the model weights and one that will be used after the model is trained to evaluate how it performs on data the model has never seen.

Do not worry about fully understanding the code below, just get the gist.

Run the following cell.

```
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
import numpy as np

maxlen = 100
training_samples = 90000
validation_samples = 5000
max_words = 10000

tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(components)
sequences = tokenizer.texts_to_sequences(components)

word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))

data = pad_sequences(sequences, maxlen=maxlen)

labels = np.asarray(labels)
print('Shape of data tensor:', data.shape)
print('Shape of label tensor:', labels.shape)

indices = np.arange(data.shape[0])
np.random.shuffle(indices)
data = data[indices]
labels = labels[indices]

x_train = data[:training_samples]
y_train = labels[:training_samples]

x_val = data[training_samples: training_samples + validation_samples]
y_val = labels[training_samples: training_samples + validation_samples]

x_test = data[training_samples + validation_samples:]
y_test = labels[training_samples + validation_samples:]
```

```
Found 65 unique tokens.
Shape of data tensor: (100000, 100)
Shape of label tensor: (100000,)
```

Now take a look at how the text was encoded as an array in the above. Run the following cell to take a peek.

```
print("The text '{text}' is represented as the vector
'{data}'".format(text=components[indices[0]], data=x_train[0]))
```

```
The text 'carbon fiber component manufactured in 2007 in good condition' is re
presented as the vector '[ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  5  6  4  3  1
 21  1  8  2]'
```

Next, you will apply the vectors provided by GloVe to create a word embedding matrix. This matrix will be used shortly to set the model wights of the first layer of the deep neural network.

Run the following cell (don't worry too much about the details at this point).

```
glove_dir = osLocalFolderPath + '/'

embeddings_index = {}
f = open(os.path.join(glove_dir, 'glove.6B.100d.txt'))
for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()

print('Found %s word vectors.' % len(embeddings_index))

embedding_dim = 100

embedding_matrix = np.zeros((max_words, embedding_dim))
for word, i in word_index.items():
    if i < max_words:
        embedding_vector = embeddings_index.get(word)
        if embedding_vector is not None:
            embedding_matrix[i] = embedding_vector
```

```
Found 400000 word vectors.
```

In the next cell, you will use Keras to define the structure of the deep neural network. The network graph you build in this case has four layers.

Run the following cell to structure the network and view a summary description of it.

```
from keras.models import Sequential
from keras.layers import Embedding, Flatten, Dense

model = Sequential()
model.add(Embedding(max_words, embedding_dim, input_length=maxlen))
model.add(Flatten())
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 100, 100)          1000000
_________________________________________________________________
flatten_1 (Flatten)          (None, 10000)             0
_________________________________________________________________
dense_1 (Dense)              (None, 32)                320032
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 33
=================================================================
Total params: 1,320,065
Trainable params: 1,320,065
Non-trainable params: 0
_________________________________________________________________
```

Rather than train model from scratch on this car components text, you can instead use the embedding matrix derived from GloVe. In effect this boosts the model's understanding of text, because the GloVe vectors was trained against a large corpus of text in Wikipedia.

Run the following cell to fix the weights for the first layer to those provided by the embedding matrix.

```
model.layers[0].set_weights([embedding_matrix])
model.layers[0].trainable = False
```

Now you are ready to train the model.

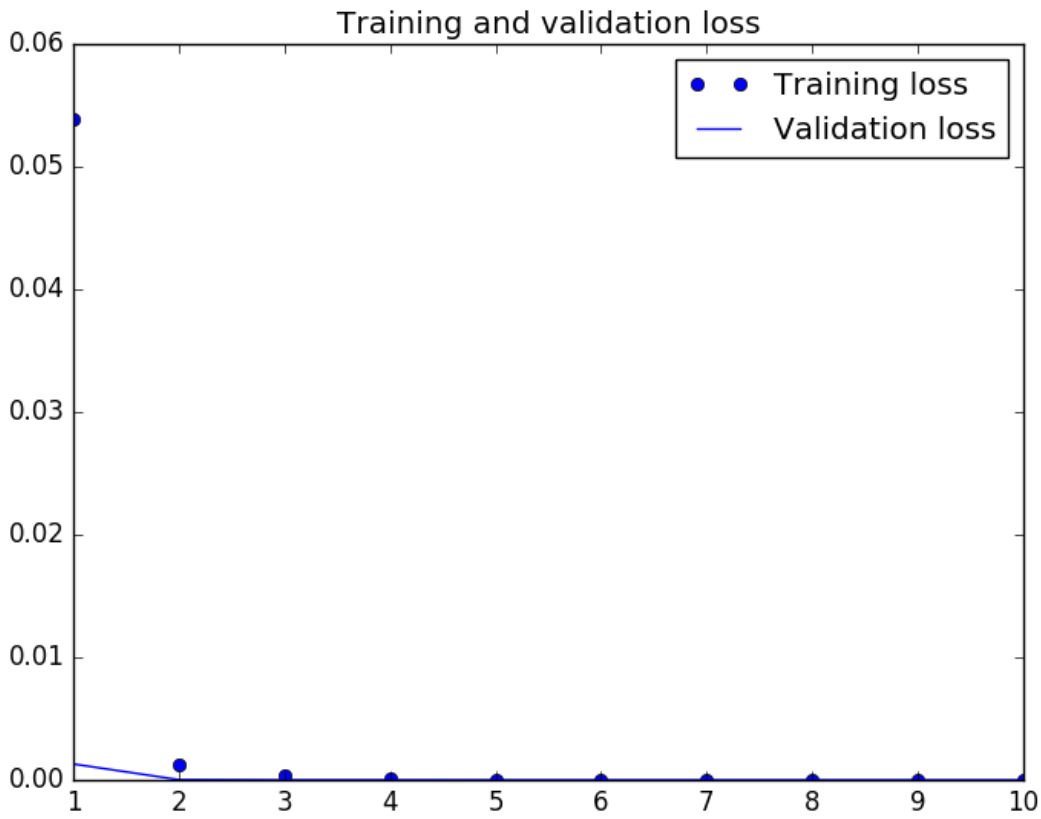
Run the following cell to train the model. This will take only **2-3 minutes** on a GPU enabled cluster.

```
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['acc'])
history = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=32,
                    validation_data=(x_val, y_val))
model.save_weights(os.path.join(glove_dir,'pre_trained_glove_model.h5'))
```

```
Train on 90000 samples, validate on 5000 samples
Epoch 1/10

   32/90000 [..............................] - ETA: 9:17 - loss: 0.6971 - acc:
0.5625
  640/90000 [..............................] - ETA: 34s - loss: 0.6941 - acc:
0.5469
 1248/90000 [..............................] - ETA: 21s - loss: 0.6579 - acc:
0.6050
 1888/90000 [..............................] - ETA: 16s - loss: 0.6241 - acc:
0.6451
 2528/90000 [..............................] - ETA: 13s - loss: 0.5888 - acc:
0.6843
 3200/90000 [>.............................] - ETA: 12s - loss: 0.5589 - acc:
0.7125
 3840/90000 [>.............................] - ETA: 11s - loss: 0.5331 - acc:
0.7365
 4480/90000 [>.............................] - ETA: 10s - loss: 0.5078 - acc:
0.7574
 5120/90000 [>.............................] - ETA: 10s - loss: 0.4841 - acc:
0.7768
```

Take a look to see how the model training went. If curves for training accuracy and validation accuracy come together, you are in good shape!

```
import matplotlib.pyplot as plt

acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(acc) + 1)

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

fig = plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

display(fig)
```



You can also evaluate how accurately the model performs against data it has not seen. Run the following cell to see the accuracy (it is the second number in the array displayed, on a scale from 0 to 1).

```
model.load_weights(os.path.join(glove_dir,'pre_trained_glove_model.h5'))
model.evaluate(x_test, y_test)
```

```
  32/5000 [..............................] - ETA: 0s
1504/5000 [========>.....................] - ETA: 0s
3008/5000 [=================>............] - ETA: 0s
4512/5000 [==========================>...] - ETA: 0s
5000/5000 [==============================] - 0s 34us/step
Out[12]: [1.0933958214991435e-07, 1.0]
```

Congratulations, you have created a deep learning model to classify the car component text descriptions as compliant or non-compliant.

Converting a Keras model to ONNX

In the steps that follow, you will convert Keras model you just trained to the ONNX format. This will enable you to use this model for classification in a very broad range of environments, outside of Azure Databricks including:

- Web services
- iOS and Android mobile apps
- Windows apps
- IoT devices

Convert the model to ONNX by running the following cell.

```
import winmltools
onnx_model_name = 'component_compliance.onnx'
converted_model = winmltools.convert_keras(model, target_opset=7)
winmltools.save_model(converted_model, os.path.join(glove_dir,
onnx_model_name))
```

The above cell created a new file called `component_compliance.onnx` that contains the ONNX version of the model.

Now try using this ONNX model to classify a component description by running the following cell. Remeber the prediction will be a value close to 0 (non-compliant) or to 1 (compliant).

```
import onnxruntime

# Load the ONNX model
onnx_session =
onnxruntime.InferenceSession(os.path.join(glove_dir,onnx_model_name))

# Grab one sample from the test data set
test_sample = x_test.astype(np.float32)[2]

# Run an ONNX session to classify the sample.
classify_output = onnx_session.run(None, {onnx_session.get_inputs()
[0].name:test_sample})[0]

print("Your model predicted the class `{pred}`, and the actual class was
`{actual}`".format(pred=classify_output[0][0], actual=y_test[0]))
```

```
Your model predicted the class `1.0`, and the actual class was `0`
```

```
import json
test_sample = x_test.astype(np.float32)[2]
test_sample_json = json.dumps(test_sample.tolist())
reloaded_sample = np.array(json.loads(test_sample_json)).astype(np.float32)

type(onnx_session.run(None, {onnx_session.get_inputs()
[0].name:reloaded_sample})[0][0][0].item())
json.dumps(onnx_session.run(None, {onnx_session.get_inputs()
[0].name:reloaded_sample})[0][0][0].item())
```

```
Out[19]: '1.0'
```

Downloading the model and test dataset

If you wanted to try this model in another environment, you can download the ONNX model from Azure Databricks with by running the following cell. The download should start within your web browser automatically.

Before you run the cell, be sure to replace with the `region` and `o` value of your Azure Databricks deployment, which can be retrieved from the URL of this notebook. For example, if the URL in the address bar for this notebook is

`https://eastus2.azuredatabricks.net/?o=6910022025589844#notebook/323983356`

then the region value is `eastus2` and the `o` value is `6910022025589844`.

```
# Replace with the region and o value of your Azure Databricks deployment (from
the URL of this notebook).
region = "eastus2"
o = "6910022025589848"

path = rootFolderName + '/' + 'component_compliance.onnx'
model_url = 'https://{region}.azuredatabricks.net/files/{path}?o=
{o}'.format(region=region,o=o,path=path)
displayHTML(model_url)
```

Show result

You can also download some test data to try your model with by running the following cell.

```
import csv
test_samples_file_name = 'test_sample.csv'
with open(glove_dir + '/' + test_samples_file_name,'w') as f:
    writer = csv.writer(f, quoting=csv.QUOTE_NONNUMERIC)
    writer.writerows(x_test.tolist())

path = rootFolderName + '/' + test_samples_file_name
samples_url = 'https://{region}.azuredatabricks.net/files/{path}?o=
{o}'.format(region=region,o=o,path=path)
displayHTML(samples_url)
```

Show result

Deploy Deep Learning ONNX format model as a web service

To demonstrate one example of using the ONNX format model in a new environment, you will deploy the ONNX model to a webservice. On the web server, the only component required by the model is the ONNX Runtime, which is used to load the

model and use it for scoring. Neither Keras nor TensorFlow are required on the web server.

In this case, you will use the Azure Machine Learning service SDK to programmatically create a Workspace, register your model, create a container image for the web service that uses it and deploy that image on to an Azure Container Instance.

Run the following cells to create some helper functions that you will use for deployment.

```
import azureml
from azureml.core import Workspace
from azureml.core.model import Model


def getOrCreateWorkspace(subscription_id, resource_group, workspace_name,
workspace_region):
    # By using the exist_ok param, if the workspace already exists we get a
reference to the existing workspace instead of an error
    ws = Workspace.create(
        name = workspace_name,
        subscription_id = subscription_id,
        resource_group = resource_group,
        location = workspace_region,
        exist_ok = True)
    return ws
```

```
def deployModelAsWebService(ws, model_folder_path="models",
model_name="component_compliance",
                scoring_script_filename="scoring-service.py",
                conda_packages=['numpy','pandas'],
                pip_packages=['azureml-sdk','onnxruntime'],
                conda_file="dependencies.yml", runtime="python",
                cpu_cores=1, memory_gb=1, tags={'name':'scoring'},
                description='Compliance classification web service.',
                service_name = "complianceservice"
               ):
    # notice for the model_path, we supply the name of the outputs folder
without a trailing slash
    # this will ensure both the model and the customestimators get uploaded.
    print("Registering and uploading model...")
    registered_model = Model.register(model_path=model_folder_path,
                                      model_name=model_name,
                                      workspace=ws)

    # create a Conda dependencies environment file
    print("Creating conda dependencies file locally...")
    from azureml.core.conda_dependencies import CondaDependencies
    mycondaenv = CondaDependencies.create(conda_packages=conda_packages,
pip_packages=pip_packages)
    with open(conda_file,"w") as f:
        f.write(mycondaenv.serialize_to_string())

    # create container image configuration
    print("Creating container image configuration...")
    from azureml.core.image import ContainerImage
    image_config = ContainerImage.image_configuration(execution_script =
scoring_script_filename,
                                                      runtime = runtime,
                                                      conda_file = conda_file)

    # create ACI configuration
    print("Creating ACI configuration...")
    from azureml.core.webservice import AciWebservice, Webservice
    aci_config = AciWebservice.deploy_configuration(
        cpu_cores = cpu_cores,
        memory_gb = memory_gb,
        tags = tags,
        description = description)

    # deploy the webservice to ACI
    print("Deploying webservice to ACI...")
    webservice = Webservice.deploy_from_model(
      workspace=ws,
```

```
      name=service_name,
      deployment_config=aci_config,
      models = [registered_model],
      image_config=image_config
    )
    webservice.wait_for_deployment(show_output=True)

    return webservice
```

In the following cell, provide the `subscription_id` of your Azure subscription into which you want to deploy. Also, provide the `resource_group` variable with a name for the resource group you want created in that subscription to hold your Workspace. Optionally, you may change the default workspace name and region.

```
#Provide the Subscription ID of your existing Azure subscription
subscription_id = "30fc406c-c745-44f0-be2d-63b1c860cde0"

#Provide values for the Resource Group and Workspace that will be created
resource_group = "aml-ws-ti"
workspace_name = "aml-workspace"
workspace_region = 'westcentralus' # eastus, westcentralus, southeastasia,
australiaeast, westeurope
```

Your web service which knows how to load the model and use it for scoring needs saved out to a file for the Azure Machine Learning service SDK to deploy it. Run the following cell to create this file.

```
# write out the file scoring-service.py
scoring_service = """
import sys
import os
import json
import numpy as np
import pandas as pd
from azureml.core.model import Model
import onnxruntime

def init():
    global model

    try:
        model_path = Model.get_model_path('component_compliance')
        model_file_path = os.path.join(model_path,'component_compliance.onnx')
        print('Loading model from:', model_file_path)

        # Load the ONNX model
        model = onnxruntime.InferenceSession(model_file_path)
    except Exception as e:
        print(e)

# note you can pass in multiple rows for scoring
def run(raw_data):
    try:
        print("Received input:", raw_data)

        input_data = np.array(json.loads(raw_data)).astype(np.float32)

        # Run an ONNX session to classify the input.
        result = model.run(None, {model.get_inputs()[0].name:input_data})[0]
        result = result[0][0].item()

        # return just the classification index (0 or 1)
        return result
    except Exception as e:
        error = str(e)
        return error
"""

with open(glove_dir + "/" + "scoring-service.py", "w") as file:
    file.write(scoring_service)
```

Next, create your Workspace (or retrieve the existing one if it already exists) and deploy the model as a web service.

Before you run the cell, modify the value of the `service_name` parameter to replace `YOUR-UNIQUE-IDENTIFIER` so that it includes your identifier.

Run the cell to perform the deployment.

```
ws =  getOrCreateWorkspace(subscription_id, resource_group,
                   workspace_name, workspace_region)
# It is important to change the current working directory so that your
generated scoring-service.py is at the root of it.
# This is required by the Azure Machine Learning SDK
os.chdir(glove_dir)
print(os.getcwd())
webservice = deployModelAsWebService(ws, model_folder_path=glove_dir,
model_name="component_compliance", service_name = "complianceservice-zst")

Warning: Falling back to use azure cli login credentials.
If you run your code in unattended mode, i.e., where you can't give a user inp
ut, then we recommend to use ServicePrincipalAuthentication or MsiAuthenticati
on.
Please refer to aka.ms/aml-notebook-auth for different authentication mechanis
ms in azureml-sdk.
Performing interactive authentication. Please follow the instructions on the t
erminal.
To sign in, use a web browser to open the page https://microsoft.com/devicelog
in and enter the code AQUJK5TJG to authenticate.
Interactive authentication successfully completed.
/dbfs/FileStore/ti_labs_481687b4-0809-4c91-be8b-54add58c8c20
Registering and uploading model...
Registering model component_compliance
Creating conda dependencies file locally...
Creating container image configuration...
Creating ACI configuration...
Deploying webservice to ACI...
Creating image
Image creation operation finished for image complianceservice-zst:1, operation
"Succeeded"
Creating service
Running.....................
SucceededACI service creation operation finished, operation "Succeeded"
```

Finally, test your deployed web service.

```
import json

# choose a sample from the test data set to send
test_sample = x_test.astype(np.float32)[2]
test_sample_json = json.dumps(test_sample.tolist())

# invoke the web service
result = webservice.run(input_data=test_sample_json)

result
```

You now have a working web service deployed that uses the ONNX version of your Keras deep learning model.