DeepLearning.AI TensorFlow Developer

Contents

1 Introduction to TensorFlow for AI, ML, and DL **1**
- 1.0.1 Callbacks . . . 1
- 1.0.2 Upload Custom Images . . . 1
- 1.0.3 ImageDataGenerator . . . 2

2 CNN's in TensorFlow **3**
- 2.0.1 Using OS and Splitting Data . . . 3
- 2.0.2 Building a CNN with IDG . . . 4
- 2.0.3 Transfer Learning: Built-in Models . . . 4
- 2.0.4 Reading Images from CSVs . . . 5
- 2.0.5 Multi-Class Classification . . . 6

3 Natural Language Processing **7**
- 3.0.1 Setup & Reading CSVs . . . 7
- 3.0.2 Tokenizer . . . 7
- 3.0.3 Modeling . . . 8
- 3.0.4 Visualizing Clustering . . . 8
- 3.0.5 Sequence Model with Pretrained Weights . . . 9
- 3.0.6 Predicting Words . . . 10

4 Sequences: Time Series and Predictions **12**
- 4.0.1 12

1 Introduction to TensorFlow for AI, ML, and DL

1.0.1 Callbacks

We can use **callbacks** in order to stop training when we reach a certain accuracy we desire. This is to stop the loss from beginning to increase again if we start to overfit the model. Click here to see the TensorFlow Callbacks documentation.

```python
import tensorflow as tf
print(tf.__version__)

class myCallback(tf.keras.callbacks.Callback):
  def on_epoch_end(self, epoch, logs={}):
    if(logs.get('accuracy')>0.6): # might need to use 'acc' instead
      print("\nReached 60% accuracy so cancelling training!")
      self.model.stop_training = True

callbacks = myCallback()

mnist = tf.keras.datasets.fashion_mnist
(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(512, activation=tf.nn.relu),
  tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
model.fit(x_images, y_labels, epochs=10, callbacks=[callbacks])
```

1.0.2 Upload Custom Images

We can use the below code to upload a custom image and use it on a trained model.

```python
import numpy as np
from google.colab import files
from keras.preprocessing import image

uploaded = files.upload()

for fn in uploaded.keys():
  # predicting images
  path = '/content/' + fn
  img = image.load_img(path, target_size=(300, 300))
  x = image.img_to_array(img)
  x = np.expand_dims(x, axis=0)

  images = np.vstack([x])
  classes = model.predict(images, batch_size=10)
  print(classes[0])
  if classes[0]>0.5:
    print(fn + " is a human")
  else:
    print(fn + " is a horse")
```

1.0.3 ImageDataGenerator

```
import tensorflow as tf
import os
import zipfile
from os import path, getcwd, chdir
from tensorflow.keras.optimizers import RMSprop
  from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Import and extract zip file containing images
path = f"{getcwd()}/../tmp2/happy-or-sad.zip"
zip_ref = zipfile.ZipFile(path, 'r')
zip_ref.extractall("/tmp/h-or-s")
zip_ref.close()

def train_happy_sad_model():
  DESIRED_ACCURACY = 0.999

  class myCallback(tf.keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs={}):
      if(logs.get('acc')>DESIRED_ACCURACY):
        print('\nReached 100% accuracy so stopping training.')
        self.model.stop_training = True

  callbacks = myCallback()

  # Define and Compile the Model.
  model = tf.keras.models.Sequential([
      tf.keras.layers.Conv2D(64, (3,3), activation='relu', input_shape=(150,150,3)),
      tf.keras.layers.MaxPooling2D(2,2),
      tf.keras.layers.Conv2D(32, (3,3), activation='relu'),
      tf.keras.layers.MaxPooling2D(2,2),
      tf.keras.layers.Conv2D(16, (3,3), activation='relu'),
      tf.keras.layers.MaxPooling2D(2,2),
      tf.keras.layers.Flatten(),
      tf.keras.layers.Dense(512),
      tf.keras.layers.Dense(1, activation='sigmoid')
  ])

  model.compile(optimizer=RMSprop(lr=0.001),
                loss='binary_crossentropy',
                metrics=['accuracy'])

  # Create an instance of an ImageDataGenerator
  train_datagen = ImageDataGenerator(rescale=1./255)

  train_generator = train_datagen.flow_from_directory(
      '/tmp/h-or-s', # directory containing the images
      target_size=(150,150),
      class_mode='binary'
  )

  history = model.fit(
      train_generator, # data generator object
      epochs=50,
      callbacks=[callbacks],
      verbose=1
  )

  return history.history['acc'][-1]
```

2 CNN's in TensorFlow

2.0.1 Using OS and Splitting Data

```python
path_cats_and_dogs = f"{getcwd()}/../tmp2/cats-and-dogs.zip"
shutil.rmtree('/tmp')

local_zip = path_cats_and_dogs
zip_ref = zipfile.ZipFile(local_zip, 'r')
zip_ref.extractall('/tmp')
zip_ref.close()

try: # Make directories for training and testing, along with class subdirectories
  os.mkdir('/tmp/cats-v-dogs/')
  os.mkdir('/tmp/cats-v-dogs/training/')
  os.mkdir('/tmp/cats-v-dogs/testing/')
  os.mkdir('/tmp/cats-v-dogs/training/cats/')
  os.mkdir('/tmp/cats-v-dogs/testing/cats/')
  os.mkdir('/tmp/cats-v-dogs/training/dogs/')
  os.mkdir('/tmp/cats-v-dogs/testing/dogs/')
except OSError:
  pass

def split_data(SOURCE, TRAINING, TESTING, SPLIT_SIZE):
  """
  a SOURCE directory containing the files
  a TRAINING directory that a portion of the files will be copied to
  a TESTING directory that a portion of the files will be copie to
  a SPLIT SIZE to determine the portion
  """
  source_list = os.listdir(SOURCE) # get list of files
  random.sample(source_list, len(source_list)) # shuffle the list
  train_images = source_list[:int(len(source_list)*SPLIT_SIZE)]
  testing_images = source_list[int(len(source_list)*SPLIT_SIZE):]

  for img in train_images:
    if os.path.getsize(SOURCE+img) != 0: # make sure not empty file
      copyfile(SOURCE+img, TRAINING+img)

  for img in testing_images:
    if os.path.getsize(SOURCE+img) != 0: # make sure not empty file
      copyfile(SOURCE+img, TESTING+img)

  return None

CAT_SOURCE_DIR = "/tmp/PetImages/Cat/"
TRAINING_CATS_DIR = "/tmp/cats-v-dogs/training/cats/"
TESTING_CATS_DIR = "/tmp/cats-v-dogs/testing/cats/"
DOG_SOURCE_DIR = "/tmp/PetImages/Dog/"
TRAINING_DOGS_DIR = "/tmp/cats-v-dogs/training/dogs/"
TESTING_DOGS_DIR = "/tmp/cats-v-dogs/testing/dogs/"

split_size = .9
split_data(CAT_SOURCE_DIR, TRAINING_CATS_DIR, TESTING_CATS_DIR, split_size)
split_data(DOG_SOURCE_DIR, TRAINING_DOGS_DIR, TESTING_DOGS_DIR, split_size)
```

2.0.2 Building a CNN with IDG

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(16, (3,3), activation='relu', input_shape=(150,150,3)),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(32, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

model.compile(optimizer=RMSprop(lr=0.001), loss='binary_crossentropy',
              metrics=['acc'])

# Feed training data into our IDG object
TRAINING_DIR = '/tmp/cats-v-dogs/training/'
train_datagen = ImageDataGenerator(rescale=1./255,
                                   rotation_range=45,
                                   horizontal_flip=True,
                                   vertical_flip=True,
                                   shear_range=0.2,
                                   zoom_range=0.2,
                                   fill_mode='nearest')

train_generator = train_datagen.flow_from_directory(TRAINING_DIR,
                                                    target_size=(150,150),
                                                    batch_size=10,
                                                    class_mode='binary')
# Feed testing data into our IDG object
VALIDATION_DIR = '/tmp/cats-v-dogs/testing/'
validation_datagen = ImageDataGenerator(rescale=1./255)

validation_generator = validation_datagen.flow_from_directory(VALIDATION_DIR,
                                                              target_size=(150,150),
                                                              batch_size=10,
                                                              class_mode='binary')
history = model.fit_generator(train_generator,
                              epochs=2,
                              verbose=1,
                              validation_data=validation_generator)
```

2.0.3 Transfer Learning: Built-in Models

```
path_inception = f"{getcwd()}/../tmp2/
  inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5"

# Import the inception model
from tensorflow.keras.applications.inception_v3 import InceptionV3

# Create an instance of the inception model from the local pre-trained weights
local_weights_file = path_inception

pre_trained_model = InceptionV3(input_shape=(150,150,3), include_top=False,
                                weights=None)

pre_trained_model.load_weights(local_weights_file)
```

```python
# Make all the layers in the pre-trained model non-trainable
for layer in pre_trained_model.layers:
  layer.trainable = False

last_layer = pre_trained_model.get_layer('mixed7')
print('last layer output shape: ', last_layer.output_shape) # (None, 7, 7, 768)
last_output = last_layer.output

# Flatten the output layer to 1 dimension (with input from last pretrained layers)
x = layers.Flatten()(last_output)

x = layers.Dense(1024, activation='relu')(x)
x = layers.Dropout(0.2)(x)
x = layers.Dense(1, activation='sigmoid')(x)

model = Model(pre_trained_model.input, x)

model.compile(optimizer = RMSprop(lr=0.0001),
              loss = 'binary_crossentropy',
              metrics = ['acc'])
```

2.0.4 Reading Images from CSVs

```python
def get_data(filename):
  """
  Read the file passed into the function. The first line contains
  the header (so skip it). Each line contains 785 values, with
  the first being the label and remaining being the pixel values.
  You will need to reshape the images into 28x28.
  """
  with open(filename) as training_file:
    labels = []
    images = []
    reader = csv.reader(training_file, delimiter = ',')
    next(reader, None) # skip first line

    for row in reader:
      labels.append(row[0])
      images.append(np.array(row[1:]).reshape(28,28))

  labels = np.array(labels).astype(float)
  images = np.array(images).astype(float)
  return images, labels

path_sign_mnist_train = f"{getcwd()}/../tmp2/sign_mnist_train.csv"
path_sign_mnist_test = f"{getcwd()}/../tmp2/sign_mnist_test.csv"
training_images, training_labels = get_data(path_sign_mnist_train)
testing_images, testing_labels = get_data(path_sign_mnist_test)

print(training_images.shape) # (27455, 28, 28)
print(training_labels.shape) # (27455)
print(testing_images.shape) # (7172, 28, 28)
print(testing_labels.shape) # (7172)

# Expand dimensions (add 1 to the end)
training_images = np.expand_dims(training_images, axis=3) # (27455, 28, 28, 1)
testing_images = np.expand_dims(testing_images, axis=3) # (7172, 28, 28, 1)
```

2.0.5 Multi-Class Classification

```
# Create an ImageDataGenerator objects
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=45,
    horizontal_flip=True,
    vertical_flip=True,
    zoom_range=0.2 )

validation_datagen = ImageDataGenerator(rescale=1./255)

# Create generators (images already loaded, not from directory)
train_generator = train_datagen.flow(training_images,
                                     training_labels,
                                     batch_size=32)

valid_generator = validation_datagen.flow(testing_images,
                                          testing_labels,
                                          batch_size=32)

# Define the model
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(64, (3,3), activation='relu', input_shape=(28,28,1)),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(26, activation='softmax') # 26 classes
])

# Compile Model.
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['acc'])

# Train the Model
history = model.fit_generator(train_generator,
                              validation_data=valid_generator,
                              epochs=2,
                              verbose=1)

# Access the metrics for plotting
acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']
```

3 Natural Language Processing

3.0.1 Setup & Reading CSVs

```
vocab_size = 1000
embedding_dim = 16
max_length = 120
trunc_type = 'post'
pad_type = 'post'
oov_tok = '<OOV>'
training_portion = .8
sentences = []
labels = []
# stopwords is an array of 153 words to be removed

with open("/tmp/bbc-text.csv", 'r') as csvfile:
  reader = csv.reader(csvfile, delimiter=',')
  next(reader)
  for row in reader:
    labels.append(row[0])
    sentence = row[1]
    for word in stopwords:
      token = ' ' + word + ' '
      sentence = sentence.replace(token, ' ')
      sentence = sentence.replace('  ', ' ')
    sentences.append(sentence)
```

3.0.2 Tokenizer

```
train_size = int(len(sentences)*training_portion)

train_sentences = sentences[:train_size] # 1780
train_labels = labels[:train_size] # 1780

validation_sentences = sentences[train_size:] # 445
val_labels = labels[train_size:] # 445

tokenizer = Tokenizer(num_words=vocab_size, oov_token=oov_tok)
tokenizer.fit_on_texts(train_sentences)
word_index = tokenizer.word_index

train_sequences = tokenizer.texts_to_sequences(train_sentences)
train_padded = pad_sequences(train_sequences, padding=pad_type, maxlen=max_length)

validation_sequences = tokenizer.texts_to_sequences(validation_sentences)
validation_padded = pad_sequences(validation_sequences, padding=pad_type,
                                  maxlen=max_length)

label_tokenizer = Tokenizer()
label_tokenizer.fit_on_texts(labels)

training_label_seq = np.array(label_tokenizer.texts_to_sequences(train_labels))
validation_label_seq = np.array(label_tokenizer.texts_to_sequences(val_labels))

```

3.0.3 Modeling

```
model = tf.keras.Sequential([
  tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=max_length),
  tf.keras.layers.GlobalAveragePooling1D(),
  tf.keras.layers.Dense(24, activation='relu'),
  tf.keras.layers.Dense(6, activation='softmax')
])
model.compile(loss='sparse_categorical_crossentropy',optimizer='adam',
              metrics=['accuracy'])

num_epochs = 30
history = model.fit(train_padded, training_label_seq,
                    epochs=num_epochs,
                    validation_data=(validation_padded, validation_label_seq),
                    verbose=1)

def plot_graphs(history, string):
  plt.plot(history.history[string])
  plt.plot(history.history['val_'+string])
  plt.xlabel("Epochs")
  plt.ylabel(string)
  plt.legend([string, 'val_'+string])
  plt.show()

plot_graphs(history, "accuracy")
plot_graphs(history, "loss")
```

3.0.4 Visualizing Clustering

```
reverse_word_index = dict([(value, key) for (key, value) in word_index.items()])
e = model.layers[0]
weights = e.get_weights()[0]

out_v = io.open('vecs.tsv', 'w', encoding='utf-8')
out_m = io.open('meta.tsv', 'w', encoding='utf-8')
for word_num in range(1, vocab_size):
  word = reverse_word_index[word_num]
  embeddings = weights[word_num]
out_m.write(word + "\n")
out_v.write('\t'.join([str(x) for x in embeddings]) + "\n")
out_v.close()
out_m.close()

try:
  from google.colab import files
except ImportError:
  pass
else:
  files.download('vecs.tsv')
  files.download('meta.tsv')
```

Go to TensorFlow Projector and upload both the *vecs.tsv* and *meta.tsv* files.
Click *sphereize data*.

3.0.5 Sequence Model with Pretrained Weights

Note that the word vector can be downloaded here.

```
# TOKENIZE THE TEXTS
sentences=[]
labels=[]
random.shuffle(corpus) # corpus contains our sentences and labels
for x in range(training_size):
  sentences.append(corpus[x][0])
  labels.append(corpus[x][1])

tokenizer = Tokenizer()
tokenizer.fit_on_texts(sentences)
word_index = tokenizer.word_index
vocab_size=len(word_index)

sequences = tokenizer.texts_to_sequences(sentences)
padded = pad_sequences(sequences, maxlen=max_length, padding=padding_type,
                       truncating=trunc_type)

split = int(test_portion * training_size)

test_sequences = padded[0:split]
training_sequences = padded[split:training_size]
test_labels = labels[0:split]
training_labels = labels[split:training_size]

# LOAD THE PRETRAINED VECTORS (138532)
embeddings_index = {};
with open('/tmp/glove.6B.100d.txt') as f:
  for line in f:
    values = line.split();
    word = values[0];
    coefs = np.asarray(values[1:], dtype='float32');
    embeddings_index[word] = coefs;

embeddings_matrix = np.zeros((vocab_size+1, embedding_dim));
for word, i in word_index.items():
  embedding_vector = embeddings_index.get(word);
  if embedding_vector is not None:
    embeddings_matrix[i] = embedding_vector;

# CREATE THE MODEL
model = tf.keras.Sequential([
  tf.keras.layers.Embedding(vocab_size+1, embedding_dim, input_length=max_length,
                            weights=[embeddings_matrix], trainable=False),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Conv1D(64, 5, activation='relu'),
  tf.keras.layers.MaxPooling1D(pool_size=4),
  tf.keras.layers.LSTM(64),
  tf.keras.layers.Dense(1, activation='sigmoid')
])
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])

training_padded = np.array(training_sequences)
training_labels = np.array(training_labels)
testing_padded = np.array(test_sequences)
testing_labels = np.array(test_labels)

history = model.fit(training_padded, training_labels, epochs=num_epochs,
                    validation_data=(testing_padded, testing_labels), verbose=1)
```

3.0.6 Predicting Words

Click here to see how to generate text using characters instead of words.

```
tokenizer = Tokenizer()
!wget --no-check-certificate \
  https://storage.googleapis.com/laurencemoroney-blog.appspot.com/sonnets.txt \
  -O /tmp/sonnets.txt
data = open('/tmp/sonnets.txt').read()

corpus = data.lower().split("\n")

tokenizer.fit_on_texts(corpus)
total_words = len(tokenizer.word_index) + 1

input_sequences = []
for line in corpus:
  token_list = tokenizer.texts_to_sequences([line])[0] # [0] converts to 1D array
  for i in range(1, len(token_list)): # loop over length of each sequence
    n_gram_sequence = token_list[:i+1]
    input_sequences.append(n_gram_sequence)

max_sequence_len = max([len(x) for x in input_sequences])
input_sequences = np.array(pad_sequences(input_sequences, maxlen=max_sequence_len,
                                         padding='pre'))

# Use last word in sequence as label (to predict)
predictors, label = input_sequences[:,:-1],input_sequences[:,-1]

# One-Hot Encode our labels (each label has size 3211)
label = ku.to_categorical(label, num_classes=total_words)

print(input_sequences[:5])
# [[   0,    0,    0,    0,    0,    0,    0,    0,    0,   34,  417],
#  [   0,    0,    0,    0,    0,    0,    0,    0,   34,  417,  877],
#  [   0,    0,    0,    0,    0,    0,    0,   34,  417,  877,  166],
#  [   0,    0,    0,    0,    0,    0,   34,  417,  877,  166,  213],
#  [   0,    0,    0,    0,    0,   34,  417,  877,  166,  213,  517]]
```

Note that the *input_sequences* that is built from the for loop is just adding the next word of each sentence into the next line. We then use the last element of each row as the label to predict. This helps us choose which words come after certain inputs. So for row 1, [34, 417] is input and [877] is the label.

```
model = Sequential()
model.add(Embedding(total_words, 100, input_length=max_sequence_len-1))
model.add(Bidirectional(LSTM(120, return_sequences=True)))
model.add(Dropout(0.2))
model.add((LSTM(100, return_sequences=False)))
model.add(Dense(total_words/2, activation='relu',
                kernel_regularizer=regularizers.l2(0.01)))
model.add(Dense(total_words, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam',
              metrics=['accuracy'])

history = model.fit(predictors, label, epochs=100, verbose=1)
```

Note that for our *input_length*, we had to subtract 1. This is because we used the last value as our label, so out longest sequence length was reduced by 1. One thing to note is that our accuracy will start very low (near 2%) but will climb steadily as we train for a long period of time (high 90%'s).

```python
seed_text = "Help me Obi Wan Kenobi, you're my only hope"
next_words = 100

# Predict next 100 words in the sequence
for _ in range(next_words):
  token_list = tokenizer.texts_to_sequences([seed_text])[0]
  token_list = pad_sequences([token_list], maxlen=max_sequence_len-1, padding='pre')
  predicted = np.argmax(model.predict(token_list), axis=1)
  output_word = ""
  for word, index in tokenizer.word_index.items():
    if index == predicted:
      output_word = word
      break
  seed_text += " " + output_word
print(seed_text)
```

4 Sequences: Time Series and Predictions

4.0.1 ...

Time series are an ordered sequence of values that are usually equally spaced over time. Time series data can be used for forecasting, imputing (previous or filling in missing data), anomalies, sound waves, etc.

Trend: time series is moving in a specific direction.
Seasonality: patterns repeat at predicted intervals.
Combination: directional trend with repeated patterns.
Neither: random values that create white noise.
Fixed Partitioning: split the time series data in only train and validation sets.
- Note that we need to include full seasons (don't split data in mid season).
- Skip creating a test set, and just retrain model on train & validation sets.

Common metrics for **evaluation** include: errors (forecast-actual), mse, rmse, mae, mape.
- We can use the keras.metrics library.

Differencing: looking at difference between data at time t and time t-a.
- Forecast = trailing moving average of differencing + centered moving average of t-a.