COGS 181 Final Project - TensorFlow & Keras

June 13, 2020

1 COGS 181 - TensorFlow and Keras

Lilyanne Kurth-Yontz

[1]:
```
import string
import random

import tensorflow as tf
import numpy as np
import os
import time

import matplotlib.pyplot as plt
import pylab

import datetime
```

[2]:
```
tf.enable_eager_execution()
```

1.1 Import Text

[3]:
```
path_to_file = tf.keras.utils.get_file('war-and-peace.txt',
                'https://raw.githubusercontent.com/mmcky/nyu-econ-370/master/
 ↪notebooks/data/book-war-and-peace.txt')

text          = open(path_to_file, 'rb').read().decode(encoding = 'utf-8')
vocab         = sorted(set(text))
vocab_size    = len(vocab)

print("EXCERPT FROM " + str(path_to_file) + ":\n----------------\n" +␣
 ↪str(text[10000:10500]))
print("----------------\nFILE LENGTH: " + str(len(text)) + "\n----------------")
print("UNIQUE CHARACTERS: " + str(vocab_size))
```

```
EXCERPT FROM /home/lkurthyo/.keras/datasets/war-and-peace.txt:
----------------
```

```
able to check the sad
current of his thoughts, "that Anatole is costing me forty thousand
rubles a year? And," he went on after a pause, "what will it be in five
years, if he goes on like this?" Presently he added: "That's what we
fathers have to put up with… Is this princess of yours rich?"

"Her father is very rich and stingy. He lives in the country. He is the
well-known Prince Bolkonski who had to retire from the army under the
late Emperor, and was nicknamed 'the King of Prussia.' He is
----------------
FILE LENGTH: 3202303
----------------
UNIQUE CHARACTERS: 82
```

1.2 Data Processing

[4]:
```
char2int = {c:i for i, c in enumerate(vocab)}
int2char = np.array(vocab)

text_as_int = np.array([char2int[ch] for ch in text], dtype = np.int32)

# Split training and validation ~ 50/50.
tr_text  = text_as_int[:1600000]
val_text = text_as_int[1600000:]
```

1.3 Parameters

[5]:
```
batch_size    = 64
buffer_size   = 10000
embedding_dim = 256
epochs        = 10
seq_length    = 100
rnn_units     = 1024
```

1.4 More Data Processing

[6]:
```
tr_char_dataset  = tf.data.Dataset.from_tensor_slices(tr_text)
val_char_dataset = tf.data.Dataset.from_tensor_slices(val_text)

tr_sequences  = tr_char_dataset.batch(seq_length + 1, drop_remainder = True)
val_sequences = val_char_dataset.batch(seq_length + 1, drop_remainder = True)

def split_input_target(chunk):
    '''
```

```
    Creates input and target from text.

    Parameters:
    chunk = chunk of text.

    Returns:
    input_text = input text generated from the chunk.
    target_text = target text generated from the chunk.
    '''
    input_text = chunk[:-1]
    target_text = chunk[1:]
    return input_text, target_text

tr_dataset  = tr_sequences.map(split_input_target).shuffle(buffer_size).
 ↪batch(batch_size, drop_remainder = True)
val_dataset = val_sequences.map(split_input_target).shuffle(buffer_size).
 ↪batch(batch_size, drop_remainder = True)
```

1.5 Model Building

[7]:
```
def build_model(vocab_size, embedding_dim, rnn_units, batch_size):
    '''
    Builds RNN model.

    Parameters:
    vocab_size = number of unique characters.
    embedding_dim = embedding dimension.
    rnn_units = number of RNN units.
    batch_size = size of the batch.

    Returns:
    model = RNN model.
    '''
    model = tf.keras.Sequential([
        tf.keras.layers.Embedding(vocab_size,
                                  embedding_dim,
                                  batch_input_shape = [batch_size, None]),
        tf.keras.layers.GRU(rnn_units,
                            return_sequences = True,
                            stateful = True,
                            recurrent_initializer = 'glorot_uniform'),
        tf.keras.layers.Dense(vocab_size)])
    return model

model = build_model(vocab_size = len(vocab),
        embedding_dim = embedding_dim,
```

```
        rnn_units = rnn_units,
        batch_size = batch_size)
```

[8]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (64, None, 256)           20992
_________________________________________________________________
gru (GRU)                    (64, None, 1024)          3935232
_________________________________________________________________
dense (Dense)                (64, None, 82)            84050
=================================================================
Total params: 4,040,274
Trainable params: 4,040,274
Non-trainable params: 0
_________________________________________________________________
```

1.6 Loss

[9]:
```
def loss(labels, logits):
    '''
    Generates loss from each epoch.

    Parameters:
    labels = labels of the batch.
    logits = predictors of the log-likelihood of the next character.
    '''
    return tf.keras.losses.sparse_categorical_crossentropy(labels, logits,␣
 ↪from_logits = True)
```

1.7 Setup

[10]:
```
optimizer = tf.keras.optimizers.Adam(learning_rate = 0.005)    # Create␣
 ↪optimizer.
model.compile(optimizer = optimizer, loss = loss)              # Compile model.

# Define early stopping.
early_stop = tf.keras.callbacks.EarlyStopping(monitor = 'val_loss')

# Create checkpoints.
checkpoint_dir      = './checkpoints' + datetime.datetime.now().strftime("_%Y.
 ↪%m.%d-%H:%M:%S")
```

```
checkpoint_prefix   = os.path.join(checkpoint_dir, "ckpt_{epoch}")
checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(
                        filepath = checkpoint_prefix,
                        save_weights_only = True)

# Fit the model.
history = model.fit(tr_dataset,
                    epochs = epochs,
                    callbacks = [checkpoint_callback, early_stop],
                    validation_data=val_dataset)
```

```
Epoch 1/10
WARNING:tensorflow:From /opt/conda/lib/python3.7/site-
packages/tensorflow/python/ops/math_grad.py:1250:
add_dispatch_support.<locals>.wrapper (from tensorflow.python.ops.array_ops) is
deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
247/247 [==============================] - 168s 680ms/step - loss: 2.3361 -
val_loss: 1.7906
Epoch 2/10
247/247 [==============================] - 161s 651ms/step - loss: 1.5741 -
val_loss: 1.5146
Epoch 3/10
247/247 [==============================] - 161s 652ms/step - loss: 1.3860 -
val_loss: 1.4311
Epoch 4/10
247/247 [==============================] - 160s 649ms/step - loss: 1.2995 -
val_loss: 1.4084
Epoch 5/10
247/247 [==============================] - 158s 642ms/step - loss: 1.2452 -
val_loss: 1.4069
Epoch 6/10
247/247 [==============================] - 153s 619ms/step - loss: 1.2122 -
val_loss: 1.4179
```

1.8 The Process

[11]:
```
model = build_model(vocab_size, embedding_dim, rnn_units, batch_size=1)
model.load_weights(tf.train.latest_checkpoint(checkpoint_dir))
model.build(tf.TensorShape([1, None]))

def generate_text(model, start_string):
    '''
    Predicts and generates text based on a starting string.
```

```
    Parameters:
    model = the building model (in this case, RNN).
    start_string = starting string to generate predictions from.

    Returns:
    The starting string plus the predictions generated from it.
    '''
    print('Generating with seed: "' + start_string + '"\n----------------')

    num_generate = 1000   # Length of string to be printed.

    input_eval = [char2int[s] for s in start_string]
    input_eval = tf.expand_dims(input_eval, 0)

    text_generated = []
    temperature    = 1.0

    model.reset_states()

    for i in range(num_generate):
        predictions = model(input_eval)
        predictions = tf.squeeze(predictions, 0)
        predictions = predictions / temperature
        predicted_id = tf.random.categorical(predictions, num_samples =␣
↪1)[-1,0].numpy()

        input_eval = tf.expand_dims([predicted_id], 0)
        text_generated.append(int2char[predicted_id])

    return (start_string + ''.join(text_generated))
```

1.9 The Result

[12]:
```
print(generate_text(model, start_string = "and "))
```

```
Generating with seed: "and "
----------------
and another there there
tried its musurery morthver to TeloCas so capea there not be and good--princed
in considering at the room all
without expression, lit attentively could not khow no one squeezed befusing).

"Well, my dear,"

"Pierre rece the only reception vous saved them all."
```

```
She parted-black our officers, looks, and the first man verses, generally about
moral
with court yet met Denisov's laugh. I help for you and your up?"

And little cannon, saw there sometimes finished, twics, and won't; but I
coune we'll go."

Whenchantagon's self-respendente.

On that colonel on his friend wold look at the
carriage.

Old men to the old prince, talked and French.

Pierre smiling, watches
were longed there. He was ofender this and how her
arms you never might him."
```

```
Plack it sounded from at earlon the book of cilvisity. The social place of
an before answerlitzenced, not write today it would be able to success.l…" she
was which terrified vanify of
Anna
Pavlough.

The hilds of what she was a
```

1.10 Tracking the Loss

[20]:
```
plt.title("Red = Training Loss  |  Green = Validation Loss")
pylab.xlim([0, 5])   # 0-5 because only 6 epochs were completed.
pylab.ylim([1, 3])

# Plot training loss until 6/10 epochs.
plt.plot(range(0, 6),
         [2.3361, 1.5741, 1.3860, 1.2995, 1.2452, 1.2122],
         'r')

# Plot validation loss until 6/10 epochs.
plt.plot(range(0, 6),
         [1.7906, 1.5146, 1.4311, 1.4084, 1.4069, 1.4179],
         'g')
plt.show()
```


Red = Training Loss | Green = Validation Loss
3.00
2.75
2.50
2.25
2.00
1.75
1.50
1.25
1.00
0
1
2
3
4
5

[ ]: