CAP 6619 - Deep Learning

Project 5

Sequential data (timeseries + text)

Richard Acs (Z23536011) and Matthew Acs (Z23536012)

Part 1: Temperature forecasting using RNNs

Following closely along Chapter 10 of our textbook, Part 1 uses a temperature-forecasting task as a example of using DL to process and make predictions on sequential data.

Dataset: recorded at the weather station at the Max Planck Institute for Biogeochemistry in Jena, Germany, it consists of 14 different quantities (such as temperature, pressure, humidity, wind direction, and so on) recorded every 10 minutes over several years. The original data goes back to 2003, but the subset of the data we'll download is limited to 2009–2016.

Useful sources and references for Part 1:

https://colab.research.google.com/github/fchollet/deep-learning-with-python-notebooks/blob/master/chapter10_dl-for-timeseries.ipynb

Acquiring and inspecting the data

```
!wget https://s3.amazonaws.com/keras-datasets/jena_climate_2009_2016.csv.zip
!unzip jena_climate_2009_2016.csv.zip
```

```
--2022-07-28 16:31:41--  https://s3.amazonaws.com/keras-datasets/jena_climate_20
Resolving s3.amazonaws.com (s3.amazonaws.com)... 52.217.41.158
Connecting to s3.amazonaws.com (s3.amazonaws.com)|52.217.41.158|:443... connecte
HTTP request sent, awaiting response... 200 OK
Length: 13565642 (13M) [application/zip]
Saving to: ‘jena_climate_2009_2016.csv.zip’

jena_climate_2009_2 100%[===================>]  12.94M  16.6MB/s    in 0.8s

2022-07-28 16:31:42 (16.6 MB/s) - ‘jena_climate_2009_2016.csv.zip’ saved [135656

Archive:  jena_climate_2009_2016.csv.zip
```

```
  inflating: jena_climate_2009_2016.csv
  inflating: __MACOSX/._jena_climate_2009_2016.csv
```

Inspecting the data

```
import os
fname = os.path.join("jena_climate_2009_2016.csv")

with open(fname) as f:
    data = f.read()

lines = data.split("\n")
header = lines[0].split(",")
lines = lines[1:]
print(header)
print(len(lines))
```

```
['"Date Time"', '"p (mbar)"', '"T (degC)"', '"Tpot (K)"', '"Tdew (degC)"', '"rh
420451
```

Parsing the data

```
import numpy as np
temperature = np.zeros((len(lines),))
raw_data = np.zeros((len(lines), len(header) - 1))
for i, line in enumerate(lines):
    values = [float(x) for x in line.split(",")[1:]]
    temperature[i] = values[1]
    raw_data[i, :] = values[:]
```

Plotting the temperature timeseries

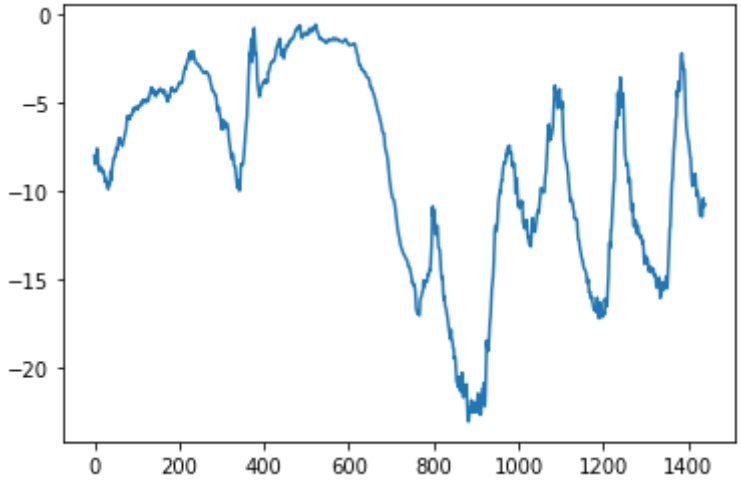
```
from matplotlib import pyplot as plt
plt.plot(range(len(temperature)), temperature)
```

```
[<matplotlib.lines.Line2D at 0x7fbca32f5590>]
```

Plotting the first 10 days of the temperature timeseries

```
plt.plot(range(1440), temperature[:1440])
```

```
[<matplotlib.lines.Line2D at 0x7fbca2dcd950>]
```



Computing the number of samples we'll use for each data split

```
num_train_samples = int(0.5 * len(raw_data))
num_val_samples = int(0.25 * len(raw_data))
num_test_samples = len(raw_data) - num_train_samples - num_val_samples
print("num_train_samples:", num_train_samples)
print("num_val_samples:", num_val_samples)
print("num_test_samples:", num_test_samples)
```

```
num_train_samples: 210225
num_val_samples: 105112
num_test_samples: 105114
```

Preparing the data

Normalizing the data

```
mean = raw_data[:num_train_samples].mean(axis=0)
raw_data -= mean
std = raw_data[:num_train_samples].std(axis=0)
raw_data /= std
```

```
import numpy as np
```

```
from tensorflow import keras
int_sequence = np.arange(10)
dummy_dataset = keras.utils.timeseries_dataset_from_array(
    data=int_sequence[:-3],
    targets=int_sequence[3:],
    sequence_length=3,
    batch_size=2,
)

for inputs, targets in dummy_dataset:
    for i in range(inputs.shape[0]):
        print([int(x) for x in inputs[i]], int(targets[i]))
```

```
[0, 1, 2] 3
[1, 2, 3] 4
[2, 3, 4] 5
[3, 4, 5] 6
[4, 5, 6] 7
```

Instantiating datasets for training, validation, and testing

```
sampling_rate = 6
sequence_length = 120
delay = sampling_rate * (sequence_length + 24 - 1)
batch_size = 256

train_dataset = keras.utils.timeseries_dataset_from_array(
    raw_data[:-delay],
    targets=temperature[delay:],
    sampling_rate=sampling_rate,
    sequence_length=sequence_length,
    shuffle=True,
    batch_size=batch_size,
    start_index=0,
    end_index=num_train_samples)

val_dataset = keras.utils.timeseries_dataset_from_array(
    raw_data[:-delay],
    targets=temperature[delay:],
    sampling_rate=sampling_rate,
    sequence_length=sequence_length,
    shuffle=True,
    batch_size=batch_size,
    start_index=num_train_samples,
    end_index=num_train_samples + num_val_samples)

test_dataset = keras.utils.timeseries_dataset_from_array(
    raw_data[:-delay],
    targets=temperature[delay:],
    sampling_rate=sampling_rate,
```

```
    sequence_length=sequence_length,
    shuffle=True,
    batch_size=batch_size,
    start_index=num_train_samples + num_val_samples)
```

Inspecting the output of one of our datasets

```
for samples, targets in train_dataset:
    print("samples shape:", samples.shape)
    print("targets shape:", targets.shape)
    break
```

```
samples shape: (256, 120, 14)
targets shape: (256,)
```

Building a baseline "model"

In this case we will try to predict the temperature by simply assuming that the temperature 24 hours from now will be equal to the temperature right now.

We shall use the mean absolute error (MAE) as a metric of performance and consider this (rather silly) "model" as our baseline.

Computing the common-sense baseline MAE

```
def evaluate_naive_method(dataset):
    total_abs_err = 0.
    samples_seen = 0
    for samples, targets in dataset:
        preds = samples[:, -1, 1] * std[1] + mean[1]
        total_abs_err += np.sum(np.abs(preds - targets))
        samples_seen += samples.shape[0]
    return total_abs_err / samples_seen

print(f"Validation MAE: {evaluate_naive_method(val_dataset):.2f}")
print(f"Test MAE: {evaluate_naive_method(test_dataset):.2f}")
```

```
Validation MAE: 2.44
Test MAE: 2.62
```

Building our *real* first model

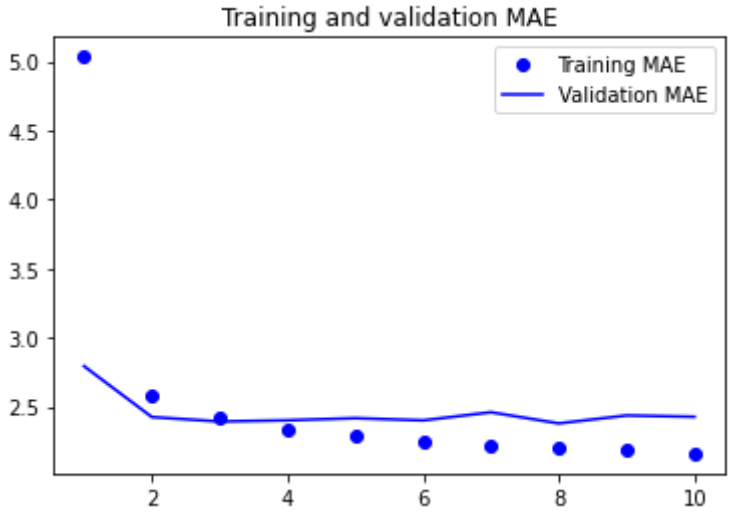
This is essentially the "simple LSTM-based model" from Listing 10.12 in the textbook.

```
from tensorflow import keras
from tensorflow.keras import layers

inputs = keras.Input(shape=(sequence_length, raw_data.shape[-1]))
x = layers.LSTM(16)(inputs)
outputs = layers.Dense(1)(x)
model = keras.Model(inputs, outputs)

callbacks = [
    keras.callbacks.ModelCheckpoint("jena_lstm.keras",
                                    save_best_only=True)
]
model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
history = model.fit(train_dataset,
                    epochs=10,
                    validation_data=val_dataset,
                    callbacks=callbacks)

model = keras.models.load_model("jena_lstm.keras")
print(f"Test MAE: {model.evaluate(test_dataset)[1]:.2f}")
```

```
Epoch 1/10
819/819 [==============================] - 53s 56ms/step - loss: 47.7903 - mae:
Epoch 2/10
819/819 [==============================] - 46s 56ms/step - loss: 11.1244 - mae:
Epoch 3/10
819/819 [==============================] - 46s 56ms/step - loss: 9.6039 - mae: 2
Epoch 4/10
819/819 [==============================] - 46s 56ms/step - loss: 8.9704 - mae: 2
Epoch 5/10
819/819 [==============================] - 46s 56ms/step - loss: 8.5662 - mae: 2
Epoch 6/10
819/819 [==============================] - 46s 56ms/step - loss: 8.2745 - mae: 2
Epoch 7/10
819/819 [==============================] - 46s 56ms/step - loss: 8.0717 - mae: 2
Epoch 8/10
819/819 [==============================] - 46s 56ms/step - loss: 7.9281 - mae: 2
Epoch 9/10
819/819 [==============================] - 46s 56ms/step - loss: 7.8116 - mae: 2
Epoch 10/10
819/819 [==============================] - 46s 56ms/step - loss: 7.6233 - mae: 2
405/405 [==============================] - 15s 35ms/step - loss: 11.1342 - mae:
Test MAE: 2.60
```

```
import matplotlib.pyplot as plt
loss = history.history["mae"]
val_loss = history.history["val_mae"]
epochs = range(1, len(loss) + 1)
plt.figure()
plt.plot(epochs, loss, "bo", label="Training MAE")
plt.plot(epochs, val_loss, "b", label="Validation MAE")
plt.title("Training and validation MAE")
```

```
plt.legend()
plt.show()
```



TODO 1: Improve the solution for temperature forecast

Write code to produce another solution to the temperature forecasting problem that outperforms the one above, i.e., has a lower Test MAE.

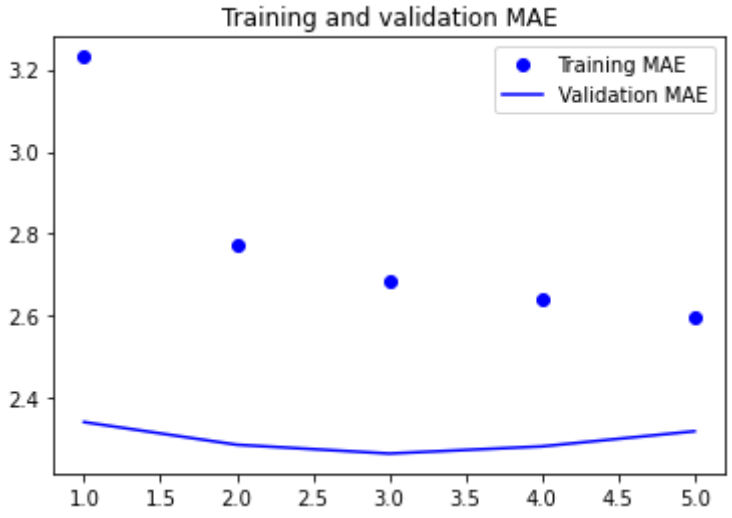
You can use a (combination of) different architecture (e.g., bidirectional RNN, see Listing 10.24 in the textbook), dropout and/or other regularization strategies, hyperparameter optimizations, or any other acceptable "trick" in the deep learning world.

```
from tensorflow import keras
from tensorflow.keras import layers

inputs = keras.Input(shape=(sequence_length, raw_data.shape[-1]))
x = layers.GRU(32, recurrent_dropout=0.5, return_sequences=True)(inputs)
x = layers.GRU(32, recurrent_dropout=0.5)(x)
x = layers.Dropout(0.5)(x)
x = layers.Dense(16)(x)
outputs = layers.Dense(1)(x)
model = keras.Model(inputs, outputs)

callbacks = [
    keras.callbacks.ModelCheckpoint("jena_stacked_gru_dropout.keras",
                                    save_best_only=True)
]
model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
history = model.fit(train_dataset,
                    epochs=5,
                    validation_data=val_dataset,
                    callbacks=callbacks)
```

```
model = keras.models.load_model("jena_stacked_gru_dropout.keras")
print(f"Test MAE: {model.evaluate(test_dataset)[1]:.2f}")
```

```
WARNING:tensorflow:Layer gru will not use cuDNN kernels since it doesn't meet th
WARNING:tensorflow:Layer gru_1 will not use cuDNN kernels since it doesn't meet
Epoch 1/5
819/819 [==============================] - 600s 728ms/step - loss: 18.4665 - mae
Epoch 2/5
819/819 [==============================] - 593s 724ms/step - loss: 12.7604 - mae
Epoch 3/5
819/819 [==============================] - 587s 716ms/step - loss: 11.9294 - mae
Epoch 4/5
819/819 [==============================] - 590s 720ms/step - loss: 11.4949 - mae
Epoch 5/5
819/819 [==============================] - 584s 712ms/step - loss: 11.1067 - mae
WARNING:tensorflow:Layer gru will not use cuDNN kernels since it doesn't meet th
WARNING:tensorflow:Layer gru_1 will not use cuDNN kernels since it doesn't meet
405/405 [==============================] - 37s 89ms/step - loss: 9.6089 - mae: 2
Test MAE: 2.41
```

```
loss = history.history["mae"]
val_loss = history.history["val_mae"]
epochs = range(1, len(loss) + 1)
plt.figure()
plt.plot(epochs, loss, "bo", label="Training MAE")
plt.plot(epochs, val_loss, "b", label="Validation MAE")
plt.title("Training and validation MAE")
plt.legend()
plt.show()
```



Summary table

Method	Test MAE	Remarks
Baseline	2.62	Common sense baseline
Real first model (LSTM)	2.60	Not much better than the baseline

Method	Test MAE	Remarks

Part 2: Sentiment analysis using LSTMs

In Part 2 we will revisit the IMDB movie review classification task from an earlier assignment, this time using more sophisticated approaches and architectures.

Please refer to Chapter 11 of our textbook for background information on NLP and approaches for text representation in deep learning architectures.

The code (and much of the text) below is essentially from https://www.tensorflow.org/text/tutorials/text_classification_rnn

Useful sources and references for Part 2:

- https://colab.research.google.com/github/fchollet/deep-learning-with-python-notebooks/blob/master/chapter11_part01_introduction.ipynb
- https://colab.research.google.com/github/fchollet/deep-learning-with-python-notebooks/blob/master/chapter11_part02_sequence-models.ipynb

Imports + auxiliary function

```
import numpy as np

import tensorflow_datasets as tfds
import tensorflow as tf

tfds.disable_progress_bar()
```

Import `matplotlib` and create a helper function to plot graphs:

```
import matplotlib.pyplot as plt


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

Setup input pipeline

The IMDB large movie review dataset is a *binary classification* dataset—all the reviews have either a *positive* or *negative* sentiment.

Download the dataset using TFDS. See the loading text tutorial for details on how to load this sort of data manually.

```
dataset, info = tfds.load('imdb_reviews', with_info=True,
                          as_supervised=True)
train_dataset, test_dataset = dataset['train'], dataset['test']

train_dataset.element_spec
```

```
Downloading and preparing dataset imdb_reviews/plain_text/1.0.0 (download: 80.23
Shuffling and writing examples to /root/tensorflow_datasets/imdb_reviews/plain_t
Shuffling and writing examples to /root/tensorflow_datasets/imdb_reviews/plain_t
WARNING:absl:Dataset is using deprecated text encoder API which will be removed
Shuffling and writing examples to /root/tensorflow_datasets/imdb_reviews/plain_t
Dataset imdb_reviews downloaded and prepared to /root/tensorflow_datasets/imdb_r
(TensorSpec(shape=(), dtype=tf.string, name=None),
 TensorSpec(shape=(), dtype=tf.int64, name=None))
```

Initially this returns a dataset of (text, label pairs):

```
for example, label in train_dataset.take(1):
  print('text: ', example.numpy())
  print('label: ', label.numpy())
```

```
text:  b"This was an absolutely terrible movie. Don't be lured in by Christopher
label:  0
```

Next shuffle the data for training and create batches of these `(text, label)` pairs:

```
BUFFER_SIZE = 10000
BATCH_SIZE = 64
```

```
train_dataset = train_dataset.shuffle(BUFFER_SIZE).batch(BATCH_SIZE).prefetch(tf.data
test_dataset = test_dataset.batch(BATCH_SIZE).prefetch(tf.data.AUTOTUNE)
```

```
for example, label in train_dataset.take(1):
  print('texts: ', example.numpy()[:3])
  print()
  print('labels: ', label.numpy()[:3])
```

```
texts:  [b"A convict serving time comes forward to give the Cold Case unit infor
 b'This movie was very, very strange and very, very funny. All of the actors are
 b'There is a reason why Jay Leno himself will not acknowledge this film. It con
```

```
labels:  [1 1 0]
```

Create the text encoder

The raw text loaded by `tfds` needs to be processed before it can be used in a model. The simplest way to process text for training is using the `TextVectorization` layer. This layer has many capabilities, but this tutorial sticks to the default behavior.

Create the layer, and pass the dataset's text to the layer's `.adapt` method:

```
VOCAB_SIZE = 1000
encoder = tf.keras.layers.TextVectorization(
    max_tokens=VOCAB_SIZE)
encoder.adapt(train_dataset.map(lambda text, label: text))
```

The `.adapt` method sets the layer's vocabulary. Here are the first 20 tokens. After the padding and unknown tokens they're sorted by frequency:

```
vocab = np.array(encoder.get_vocabulary())
vocab[:20]
```

```
array(['', '[UNK]', 'the', 'and', 'a', 'of', 'to', 'is', 'in', 'it', 'i',
       'this', 'that', 'br', 'was', 'as', 'for', 'with', 'movie', 'but'],
      dtype='<U14')
```

Once the vocabulary is set, the layer can encode text into indices. The tensors of indices are 0-padded to the longest sequence in the batch (unless you set a fixed `output_sequence_length`):

```
encoded_example = encoder(example)[:3].numpy()
encoded_example
```

```
array([[ 4,  1,  1, ...,  0,  0,  0],
       [11, 18, 14, ...,  0,  0,  0],
       [48,  7,  4, ...,  0,  0,  0]])
```

With the default settings, the process is not completely reversible. There are three main reasons for that:

1. The default value for `preprocessing.TextVectorization`'s `standardize` argument is `"lower_and_strip_punctuation"`.

2. The limited vocabulary size and lack of character-based fallback results in some unknown tokens.

```
for n in range(3):
  print("Original: ", example[n].numpy())
  print("Round-trip: ", " ".join(vocab[encoded_example[n]]))
  print()
```

```
Original:  b"A convict serving time comes forward to give the Cold Case unit inf
Round-trip:  a [UNK] [UNK] time comes forward to give the [UNK] case [UNK] [UNK]

Original:  b'This movie was very, very strange and very, very funny. All of the
Round-trip:  this movie was very very strange and very very funny all of the act

Original:  b'There is a reason why Jay Leno himself will not acknowledge this fi
Round-trip:  there is a reason why [UNK] [UNK] himself will not [UNK] this film
```

Create the first model

Please refer to https://www.tensorflow.org/text/tutorials/text_classification_rnn for detailed explanation + diagram.

```
model = tf.keras.Sequential([
    encoder,
    tf.keras.layers.Embedding(
        input_dim=len(encoder.get_vocabulary()),
        output_dim=64,
        # Use masking to handle the variable sequence lengths
        mask_zero=True),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1)
])
```

Please note that Keras sequential model is used here since all the layers in the model only have single input and produce single output. In case you want to use stateful RNN layer, you might want to build your model with Keras functional API or model subclassing so that you can retrieve and reuse the RNN layer states. Please check Keras RNN guide for more details.

The embedding layer uses masking to handle the varying sequence-lengths. All the layers after the `Embedding` support masking:

```
print([layer.supports_masking for layer in model.layers])
```

```
[False, True, True, True, True]
```

To confirm that this works as expected, evaluate a sentence twice. First, alone so there's no padding to mask:

```
# predict on a sample text without padding.

sample_text = ('The movie was cool. The animation and the graphics '
               'were out of this world. I would recommend this movie.')
predictions = model.predict(np.array([sample_text]))
print(predictions[0])
```

```
[-0.01116458]
```

Now, evaluate it again in a batch with a longer sentence. The result should be identical:

```
# predict on a sample text with padding

padding = "the " * 2000
predictions = model.predict(np.array([sample_text, padding]))
print(predictions[0])
```

```
[-0.01116458]
```

Compile the Keras model to configure the training process:

```
model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              optimizer=tf.keras.optimizers.Adam(1e-4),
              metrics=['accuracy'])
```

Train the model

```
history = model.fit(train_dataset, epochs=10,
                    validation_data=test_dataset,
                    validation_steps=30)
```

```
====================] - 40s 83ms/step - loss: 0.6512 - accuracy: 0.5658 - val_los

====================] - 31s 78ms/step - loss: 0.4438 - accuracy: 0.7968 - val_los

====================] - 31s 79ms/step - loss: 0.3795 - accuracy: 0.8262 - val_los

====================] - 31s 78ms/step - loss: 0.3325 - accuracy: 0.8548 - val_los
```

```
====================] - 31s 78ms/step - loss: 0.3236 - accuracy: 0.8581 - val_los

====================] - 32s 80ms/step - loss: 0.3127 - accuracy: 0.8646 - val_los

====================] - 31s 79ms/step - loss: 0.3100 - accuracy: 0.8667 - val_los

====================] - 31s 79ms/step - loss: 0.3021 - accuracy: 0.8717 - val_los

====================] - 31s 79ms/step - loss: 0.3021 - accuracy: 0.8699 - val_los

====================] - 31s 79ms/step - loss: 0.2979 - accuracy: 0.8716 - val_los
```

```
test_loss, test_acc = model.evaluate(test_dataset)

print('Test Loss:', test_loss)
print('Test Accuracy:', test_acc)
```

```
391/391 [==============================] - 18s 45ms/step - loss: 0.3265 - accura
Test Loss: 0.3264799416065216
Test Accuracy: 0.8632000088691711
```

```
plt.figure(figsize=(16, 8))
plt.subplot(1, 2, 1)
plot_graphs(history, 'accuracy')
plt.ylim(None, 1)
plt.subplot(1, 2, 2)
plot_graphs(history, 'loss')
plt.ylim(0, None)
```

```
(0.0, 0.6688877314329147)
```

Run a prediction on a new sentence:

If the prediction is >= 0.0, it is positive else it is negative.

```
sample_text = ('The movie was cool. The animation and the graphics '
               'were out of this world. I would recommend this movie.')
predictions = model.predict(np.array([sample_text]))
```

Second model: stacking two LSTM layers

Please refer to https://www.tensorflow.org/text/tutorials/text_classification_rnn for additional explanation + diagram.

The interesting thing about using an `RNN` with `return_sequences=True` is that the output still has 3-axes, like the input, so it can be passed to another RNN layer, like this:

```
model = tf.keras.Sequential([
    encoder,
    tf.keras.layers.Embedding(len(encoder.get_vocabulary()), 64, mask_zero=True),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64,  return_sequences=True)),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(1)
])
```

```
model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              optimizer=tf.keras.optimizers.Adam(1e-4),
              metrics=['accuracy'])
```

```
history = model.fit(train_dataset, epochs=10,
                    validation_data=test_dataset,
                    validation_steps=30)
```

```
Epoch 1/10
391/391 [==============================] - 68s 142ms/step - loss: 0.6366 - accur
Epoch 2/10
391/391 [==============================] - 52s 132ms/step - loss: 0.3990 - accur
Epoch 3/10
391/391 [==============================] - 53s 135ms/step - loss: 0.3414 - accur
```

```
Epoch 4/10
391/391 [==============================] - 52s 133ms/step - loss: 0.3227 - accur
Epoch 5/10
391/391 [==============================] - 52s 132ms/step - loss: 0.3156 - accur
Epoch 6/10
391/391 [==============================] - 52s 132ms/step - loss: 0.3083 - accur
Epoch 7/10
391/391 [==============================] - 53s 134ms/step - loss: 0.3070 - accur
Epoch 8/10
391/391 [==============================] - 53s 134ms/step - loss: 0.3015 - accur
Epoch 9/10
391/391 [==============================] - 53s 133ms/step - loss: 0.2976 - accur
Epoch 10/10
391/391 [==============================] - 53s 133ms/step - loss: 0.2957 - accur
```

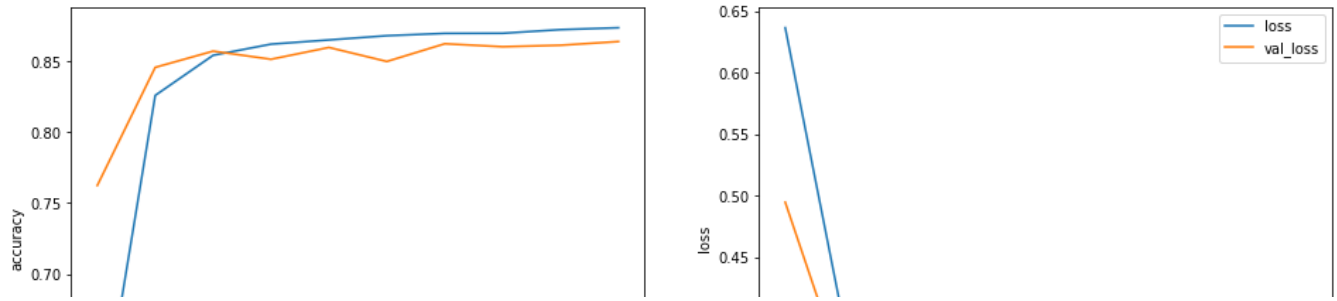
```
test_loss, test_acc = model.evaluate(test_dataset)

print('Test Loss:', test_loss)
print('Test Accuracy:', test_acc)
```

```
391/391 [==============================] - 28s 71ms/step - loss: 0.3222 - accura
Test Loss: 0.3222462236881256
Test Accuracy: 0.8645600080490112
```

```
# predict on a sample text without padding.

sample_text = ('The movie was not good. The animation and the graphics '
               'were terrible. I would not recommend this movie.')
predictions = model.predict(np.array([sample_text]))
print(predictions)
```

```
[[-2.0999575]]
```

```
plt.figure(figsize=(16, 6))
plt.subplot(1, 2, 1)
plot_graphs(history, 'accuracy')
plt.subplot(1, 2, 2)
plot_graphs(history, 'loss')
```



TODO 2: Improve the solution for IMDB sentiment analysis

Write code to produce another solution to the movie review problem that outperforms the two solutions provided above, i.e, has a higher test accuracy.

You can use a (combination of) different architecture, dropout and/or other regularization strategies, hyperparameter optimizations, masking, pretrained embeddings, or any other acceptable "trick" in the deep learning world.

```
VOCAB_SIZE = 10000
encoder1 = tf.keras.layers.TextVectorization(
    ngrams=2,
    max_tokens=VOCAB_SIZE,
    output_mode="multi_hot",
    )
encoder1.adapt(train_dataset.map(lambda text, label: text))
```

```
model = tf.keras.Sequential([
    encoder1,
    tf.keras.layers.Dense(16, activation="relu"),
    tf.keras.layers.Dense(16, activation="relu"),
    tf.keras.layers.Dense(1, activation="sigmoid"),
])
```

```
model.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
```

```
history = model.fit(train_dataset, epochs=3,
                    validation_data=test_dataset,
                    validation_steps=30)
```

```
Epoch 1/3
391/391 [==============================] - 6s 13ms/step - loss: 0.3324 - accurac
Epoch 2/3
391/391 [==============================] - 5s 13ms/step - loss: 0.1943 - accurac
Epoch 3/3
391/391 [==============================] - 5s 12ms/step - loss: 0.1516 - accurac
```

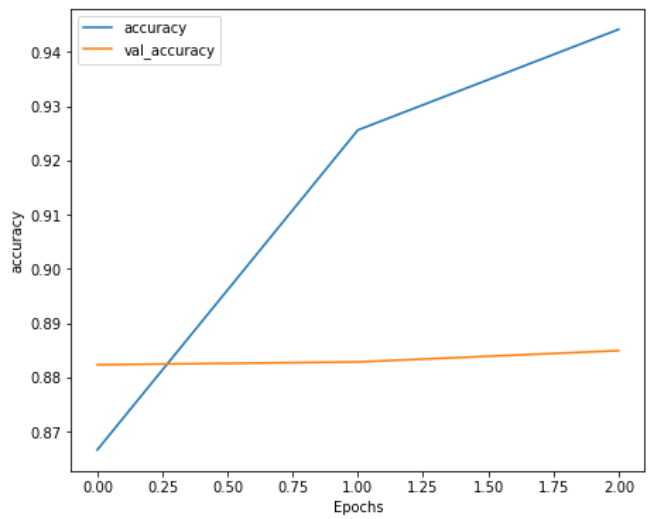
```
test_loss, test_acc = model.evaluate(test_dataset)

print('Test Loss:', test_loss)
print('Test Accuracy:', test_acc)
```

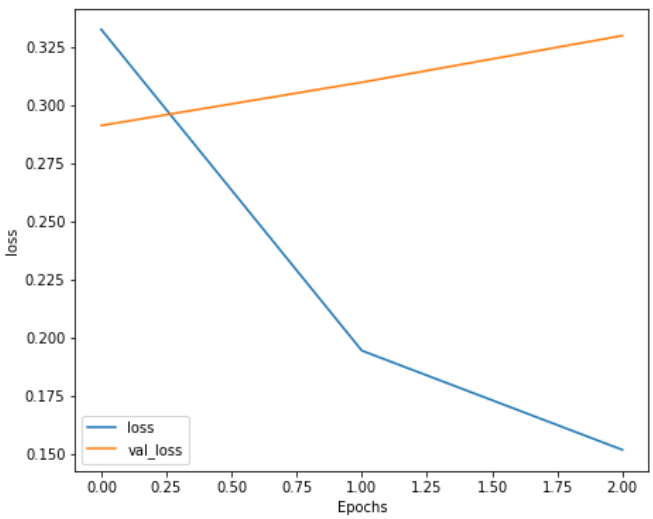
```
391/391 [==============================] - 5s 11ms/step - loss: 0.3049 - accurac
Test Loss: 0.30489251017570496
Test Accuracy: 0.8885200023651123
```

```
# predict on a sample text without padding. 0 is negative and 1 is positive

sample_text = ('The movie was not good. The animation and the graphics '
               'were terrible. I would not recommend this movie.')
predictions = model.predict(np.array([sample_text]))
print(int(predictions))
```

```
0
```

```
plt.figure(figsize=(16, 6))
plt.subplot(1, 2, 1)
plot_graphs(history, 'accuracy')
plt.subplot(1, 2, 2)
plot_graphs(history, 'loss')
```





Summary table

Method	Test Accuracy	Remarks
First model (LSTM)	0.863	Baseline

Method	Test Accuracy	Remarks
Second model (2 stacked LSTM layers)	0.865	Very similar to the baseline
Bigram bag-of-words model	0.889	Much better than the sequence models

Part 3: NLP using Transformers

In Part 3 we will look at the Transformer architecture and how it can be used in a specific NLP task, machine translation (from English to Spanish).

Please refer to Chapter 11 of our textbook for additional information.

Useful sources and references for Part 3:

- https://colab.research.google.com/github/fchollet/deep-learning-with-python-notebooks/blob/master/chapter11_part03_transformer.ipynb
- https://colab.research.google.com/github/fchollet/deep-learning-with-python-notebooks/blob/master/chapter11_part04_sequence-to-sequence-learning.ipynb

Setup

```
!wget http://storage.googleapis.com/download.tensorflow.org/data/spa-eng.zip
!unzip -q spa-eng.zip
```

```
--2022-07-28 17:54:02--  http://storage.googleapis.com/download.tensorflow.org/d
Resolving storage.googleapis.com (storage.googleapis.com)... 74.125.197.128, 74.
Connecting to storage.googleapis.com (storage.googleapis.com)|74.125.197.128|:80
HTTP request sent, awaiting response... 200 OK
Length: 2638744 (2.5M) [application/zip]
Saving to: 'spa-eng.zip'

spa-eng.zip         100%[===================>]   2.52M  --.-KB/s    in 0.006s

2022-07-28 17:54:02 (450 MB/s) - 'spa-eng.zip' saved [2638744/2638744]
```

```
text_file = "spa-eng/spa.txt"
with open(text_file) as f:
    lines = f.read().split("\n")[:-1]
text_pairs = []
for line in lines:
    english, spanish = line.split("\t")
    spanish = "[start] " + spanish + " [end]"
    text_pairs.append((english, spanish))
```

```
import random
print(random.choice(text_pairs))
```

```
('I feel great about this.', '[start] Me siento muy bien con esto. [end]')
```

```
import random
random.shuffle(text_pairs)
num_val_samples = int(0.15 * len(text_pairs))
num_train_samples = len(text_pairs) - 2 * num_val_samples
train_pairs = text_pairs[:num_train_samples]
val_pairs = text_pairs[num_train_samples:num_train_samples + num_val_samples]
test_pairs = text_pairs[num_train_samples + num_val_samples:]
```

Vectorizing the English and Spanish text pairs

```
import tensorflow as tf
import string
import re

from tensorflow import keras
from tensorflow.keras import layers

strip_chars = string.punctuation + "¿"
strip_chars = strip_chars.replace("[", "")
strip_chars = strip_chars.replace("]", "")

def custom_standardization(input_string):
    lowercase = tf.strings.lower(input_string)
    return tf.strings.regex_replace(
        lowercase, f"[{re.escape(strip_chars)}]", "")

vocab_size = 15000
sequence_length = 20

source_vectorization = layers.TextVectorization(
    max_tokens=vocab_size,
    output_mode="int",
    output_sequence_length=sequence_length,
)
target_vectorization = layers.TextVectorization(
    max_tokens=vocab_size,
    output_mode="int",
    output_sequence_length=sequence_length + 1,
    standardize=custom_standardization,
)
train_english_texts = [pair[0] for pair in train_pairs]
train_spanish_texts = [pair[1] for pair in train_pairs]
source_vectorization.adapt(train_english_texts)
target_vectorization.adapt(train_spanish_texts)
```

Preparing datasets for the translation task

```
batch_size = 64

def format_dataset(eng, spa):
    eng = source_vectorization(eng)
    spa = target_vectorization(spa)
    return ({
        "english": eng,
        "spanish": spa[:, :-1],
    }, spa[:, 1:])

def make_dataset(pairs):
    eng_texts, spa_texts = zip(*pairs)
    eng_texts = list(eng_texts)
    spa_texts = list(spa_texts)
    dataset = tf.data.Dataset.from_tensor_slices((eng_texts, spa_texts))
    dataset = dataset.batch(batch_size)
    dataset = dataset.map(format_dataset, num_parallel_calls=4)
    return dataset.shuffle(2048).prefetch(16).cache()

train_ds = make_dataset(train_pairs)
val_ds = make_dataset(val_pairs)
```

```
for inputs, targets in train_ds.take(1):
    print(f"inputs['english'].shape: {inputs['english'].shape}")
    print(f"inputs['spanish'].shape: {inputs['spanish'].shape}")
    print(f"targets.shape: {targets.shape}")
```

```
inputs['english'].shape: (64, 20)
inputs['spanish'].shape: (64, 20)
targets.shape: (64, 20)
```

The Transformer encoder

[ ] ↳ *1 cell hidden*

The Transformer decoder

```
class TransformerDecoder(layers.Layer):
    def __init__(self, embed_dim, dense_dim, num_heads, **kwargs):
        super().__init__(**kwargs)
        self.embed_dim = embed_dim
        self.dense_dim = dense_dim
```

```
        self.num_heads = num_heads
        self.attention_1 = layers.MultiHeadAttention(
            num_heads=num_heads, key_dim=embed_dim)
        self.attention_2 = layers.MultiHeadAttention(
            num_heads=num_heads, key_dim=embed_dim)
        self.dense_proj = keras.Sequential(
            [layers.Dense(dense_dim, activation="relu"),
             layers.Dense(embed_dim),]
        )
        self.layernorm_1 = layers.LayerNormalization()
        self.layernorm_2 = layers.LayerNormalization()
        self.layernorm_3 = layers.LayerNormalization()
        self.supports_masking = True

    def get_config(self):
        config = super().get_config()
        config.update({
            "embed_dim": self.embed_dim,
            "num_heads": self.num_heads,
            "dense_dim": self.dense_dim,
        })
        return config

    def get_causal_attention_mask(self, inputs):
        input_shape = tf.shape(inputs)
        batch_size, sequence_length = input_shape[0], input_shape[1]
        i = tf.range(sequence_length)[:, tf.newaxis]
        j = tf.range(sequence_length)
        mask = tf.cast(i >= j, dtype="int32")
        mask = tf.reshape(mask, (1, input_shape[1], input_shape[1]))
        mult = tf.concat(
            [tf.expand_dims(batch_size, -1),
             tf.constant([1, 1], dtype=tf.int32)], axis=0)
        return tf.tile(mask, mult)

    def call(self, inputs, encoder_outputs, mask=None):
        causal_mask = self.get_causal_attention_mask(inputs)
        if mask is not None:
            padding_mask = tf.cast(
                mask[:, tf.newaxis, :], dtype="int32")
            padding_mask = tf.minimum(padding_mask, causal_mask)
        attention_output_1 = self.attention_1(
            query=inputs,
            value=inputs,
            key=inputs,
            attention_mask=causal_mask)
        attention_output_1 = self.layernorm_1(inputs + attention_output_1)
        attention_output_2 = self.attention_2(
            query=attention_output_1,
            value=encoder_outputs,
            key=encoder_outputs,
```

```
            attention_mask=padding_mask,
        )
        attention_output_2 = self.layernorm_2(
            attention_output_1 + attention_output_2)
        proj_output = self.dense_proj(attention_output_2)
        return self.layernorm_3(attention_output_2 + proj_output)
```

Putting it all together: A Transformer for machine translation

PositionalEmbedding layer

```
class PositionalEmbedding(layers.Layer):
    def __init__(self, sequence_length, input_dim, output_dim, **kwargs):
        super().__init__(**kwargs)
        self.token_embeddings = layers.Embedding(
            input_dim=input_dim, output_dim=output_dim)
        self.position_embeddings = layers.Embedding(
            input_dim=sequence_length, output_dim=output_dim)
        self.sequence_length = sequence_length
        self.input_dim = input_dim
        self.output_dim = output_dim

    def call(self, inputs):
        length = tf.shape(inputs)[-1]
        positions = tf.range(start=0, limit=length, delta=1)
        embedded_tokens = self.token_embeddings(inputs)
        embedded_positions = self.position_embeddings(positions)
        return embedded_tokens + embedded_positions

    def compute_mask(self, inputs, mask=None):
        return tf.math.not_equal(inputs, 0)

    def get_config(self):
        config = super(PositionalEmbedding, self).get_config()
        config.update({
            "output_dim": self.output_dim,
            "sequence_length": self.sequence_length,
            "input_dim": self.input_dim,
        })
        return config
```

End-to-end Transformer

```
embed_dim = 256
dense_dim = 2048
num_heads = 8
```

```
encoder_inputs = keras.Input(shape=(None,), dtype="int64", name="english")
x = PositionalEmbedding(sequence_length, vocab_size, embed_dim)(encoder_inputs)
encoder_outputs = TransformerEncoder(embed_dim, dense_dim, num_heads)(x)

decoder_inputs = keras.Input(shape=(None,), dtype="int64", name="spanish")
x = PositionalEmbedding(sequence_length, vocab_size, embed_dim)(decoder_inputs)
x = TransformerDecoder(embed_dim, dense_dim, num_heads)(x, encoder_outputs)
x = layers.Dropout(0.5)(x)
decoder_outputs = layers.Dense(vocab_size, activation="softmax")(x)
transformer = keras.Model([encoder_inputs, decoder_inputs], decoder_outputs)
```

Training the sequence-to-sequence Transformer

```
transformer.compile(
    optimizer="rmsprop",
    loss="sparse_categorical_crossentropy",
    metrics=["accuracy"])
transformer.fit(train_ds, epochs=10, validation_data=val_ds)
```

```
Epoch 1/10
1302/1302 [==============================] - 98s 71ms/step - loss: 1.6393 - accu
Epoch 2/10
1302/1302 [==============================] - 92s 70ms/step - loss: 1.3112 - accu
Epoch 3/10
1302/1302 [==============================] - 92s 70ms/step - loss: 1.1650 - accu
Epoch 4/10
1302/1302 [==============================] - 91s 70ms/step - loss: 1.0820 - accu
Epoch 5/10
1302/1302 [==============================] - 92s 70ms/step - loss: 1.0369 - accu
Epoch 6/10
1302/1302 [==============================] - 91s 70ms/step - loss: 1.0078 - accu
Epoch 7/10
1302/1302 [==============================] - 91s 70ms/step - loss: 0.9873 - accu
Epoch 8/10
1302/1302 [==============================] - 91s 70ms/step - loss: 0.9699 - accu
Epoch 9/10
1302/1302 [==============================] - 91s 70ms/step - loss: 0.9531 - accu
Epoch 10/10
1302/1302 [==============================] - 91s 70ms/step - loss: 0.9382 - accu
<keras.callbacks.History at 0x7fbca45d42d0>
```

Translating new sentences with our Transformer model

```
import numpy as np
spa_vocab = target_vectorization.get_vocabulary()
spa_index_lookup = dict(zip(range(len(spa_vocab)), spa_vocab))
max_decoded_sentence_length = 20

def decode_sequence(input_sentence):
```

```
    tokenized_input_sentence = source_vectorization([input_sentence])
    decoded_sentence = "[start]"
    for i in range(max_decoded_sentence_length):
        tokenized_target_sentence = target_vectorization(
            [decoded_sentence])[:, :-1]
        predictions = transformer(
            [tokenized_input_sentence, tokenized_target_sentence])
        sampled_token_index = np.argmax(predictions[0, i, :])
        sampled_token = spa_index_lookup[sampled_token_index]
        decoded_sentence += " " + sampled_token
        if sampled_token == "[end]":
            break
    return decoded_sentence

test_eng_texts = [pair[0] for pair in test_pairs]
for _ in range(20):
    input_sentence = random.choice(test_eng_texts)
    print("-")
    print(input_sentence)
    print(decode_sequence(input_sentence))
```

```
 [start] no es tan bueno como parece [end]
 -
 He's been a patron of this store for many years.
 [start] ha sido una de esta tienda de [UNK] por muchos años [end]
 -
 I wish I didn't have to do all those things I don't want to do.
 [start] ojalá no tenía que hacer todas las cosas que quiero hacer [end]
 -
 Please tell me how to get to Boston.
 [start] por favor [UNK] a boston [end]
 -
 Law and politics are two different things.
 [start] la vista y la lista son [UNK] las cosas son [UNK] [end]
 -
 Do you want to watch this movie again?
 [start] quieres ver esta película a esta película [end]
 -
 I don't care who kisses Tom.
 [start] no me creo quién hizo tom [end]
 -
 Tom gave Mary some food.
 [start] tom le dio a mary algo de comida [end]
 -
 I want everyone to hear what I have to say.
 [start] quiero oír a todo el mundo debe decir lo que tengo que decir [end]
 -
 Tom could see Mary was about ready to cry.
 [start] tom podía ver que mary estaba listo para de que estaba esperando [end]
 -
 Tom depends on Mary too much.
 [start] tom habla mucho inglés [end]

 -
 What a team!
 [start] ¡qué equipo [end]
```

```
[start] ¡que equipo [end]
-
You worry too much about your weight.
[start] te estás muy bien de tu peso [end]
-
I love to teach.
[start] me encanta escribir [end]
-
Tom told Mary not to wait for John.
[start] tom le dijo a mary que no se lo que para no se lo que le [UNK] a [end]
-
She turned away in anger.
[start] se quedó fuera de la [UNK] [end]
-
I'm ready for anything.
[start] estoy listo para todo [end]
-
The thief ran away in the direction of the station.
[start] el aeropuerto se quedó sin la dirección del niño [end]
-
Nobody cares what you think.
[start] no le importa lo que crees [end]
-
Wait until I'm done eating.
[start] espera a que me he terminado de comer [end]
```

Conclusions

This assignment challenged us to dig into our machine learning toolbox to improve the given solutions. In particular, this assignment was the most patience-testing of all the assignments. This is because the models had long training times on the google collab servers and they often disconnected from the runtime in the middle of training. This means that the model had to be trained again from the beginning. We tried many experiments to beat the base models by implementing many different ideas that were aimed at increasing the power and generalization of the models. This was very time-consuming because each iteration took around half an hour to train, so just a few iterations took hours to complete. Despite this challenge, we were able to beat both models by implementing strategies that increased the capacity of the model while also curbing overfitting.

For the first part, we increased the capacity of the model by stacking two GRU layers. This helped the model have the necessary capacity to identify the relevant patterns in the dataset. To mitigate overfitting, we also added a dropout layer and a recurrent dropout layer. These techniques helped achieve a test mae that is about 0.10 less than the previous best model in the assignment.

For the second part, we completely changed the model architecture. The previous model for part 2 used an LSTM, which is a sequence model. However, the better model would be a bag-of-word model, which disregards the order of the words in the model. This is because each movie review is relatively long compared to the number of reviews. This scenario has been shown to be better

modeled with a bag-of-words model rather than a sequence model such as an LSTM. Besides changing the model to a bag-of-words model, we also used bigrams to introduce some local ordering and we increased the vocabulary size. This resulted in a test accuracy that was noticeable better than the LSTM model. Furthermore, the bag-of-words model was significantly less computationally expensive to train, which made it easier to fine-tune the parameters.

Overall, we enjoyed the challenge of having to “beat” the given solutions. Since training took a very long time, we wrote a list of improvements in order of feasibility and experimented with each improvement in the list until something worked or the list ran out. For part 1 we only had one list and we achieved an improved accuracy after relatively few experiments. On the other hand, for part 2 we had multiple lists and hours of training time until we finally found an improved solution. It was very satisfying to finally achieve an improved solution, but the process was extremely frustrating at times (especially in part 2). To improve the models, we had to draw upon techniques learned throughout this entire semester, which allowed us to really review and understand the material from a more holistic and applied viewpoint.

✓ 5s completed at 2:13 PM