Topics : translator (encoder-decoder recurrent neural network)

In [1]:
```
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

In [2]:
```
import os
import tensorflow as tf
import numpy as np

# Set the seed for random operations.
# This let our experiments to be reproducible.
SEED = 1234
tf.random.set_seed(SEED)
np.random.seed(SEED)

# Get current working directory
cwd = os.getcwd()
```

In [3]:
```
from google.colab import drive
drive.mount('/content/drive/')
```

```
Mounted at /content/drive/
```

In [4]:
```
!cat /content/drive/My\ Drive/ita.txt
```

```
Output streaming troncato alle ultime 5000 righe.
Please smile.	Sorridi, per favore.	CC-BY 2.0 (France) Attribution: tatoeba.org #1852296 (CK) & #4140430 (Guybrush8
8)
Please smile.	Sorridi, per piacere.	CC-BY 2.0 (France) Attribution: tatoeba.org #1852296 (CK) & #4140431 (Guybrush8
8)
Please smile.	Sorrida, per favore.	CC-BY 2.0 (France) Attribution: tatoeba.org #1852296 (CK) & #4140432 (Guybrush8
8)
Please smile.	Sorrida, per piacere.	CC-BY 2.0 (France) Attribution: tatoeba.org #1852296 (CK) & #4140433 (Guybrush8
8)
Please smile.	Sorridete, per favore.	CC-BY 2.0 (France) Attribution: tatoeba.org #1852296 (CK) & #4140434 (Guybrush8
8)
Please smile.	Sorridete, per piacere.	CC-BY 2.0 (France) Attribution: tatoeba.org #1852296 (CK) & #4140435 (Guybrush8
8)
Put it there.	Mettilo lì.	CC-BY 2.0 (France) Attribution: tatoeba.org #1553401 (CK) & #5209591 (Guybrush88)
Put it there.	Mettila lì.	CC-BY 2.0 (France) Attribution: tatoeba.org #1553401 (CK) & #5209592 (Guybrush88)
Put it there.	Lo metta lì.	CC-BY 2.0 (France) Attribution: tatoeba.org #1553401 (CK) & #5209593 (Guybrush88)
Put it there.	La metta lì.	CC-BY 2.0 (France) Attribution: tatoeba.org #1553401 (CK) & #5209594 (Guybrush88)
Put it there.	Mettetelo lì.	CC-BY 2.0 (France) Attribution: tatoeba.org #1553401 (CK) & #5209596 (Guybrush88)
Put it there.	Mettetela lì.	CC-BY 2.0 (France) Attribution: tatoeba.org #1553401 (CK) & #5209597 (Guybrush88)
```

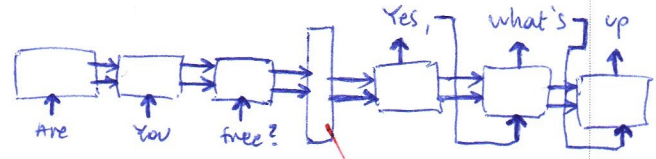
Neural Machine Translation

sentence as input and sentence as output (a Seq2Seq can work)

Italian/English

Dataset

└ the architecture is an encoder-decoder one



encoder (implemented through a recurrent architecture)

ENCODING of the input sentence

decoder (e.g. LSTM)
it produces an output after each recurrent step based on the input

the encoder produces a final output vector which represents all the input sentences

difference between encoder and decoder

We have to encode the words, we can't feed the model with base words. We could simply do one-hot-encoding but with that every word would be 1 to all the others, while we would want to preserve some closeness in case of words-correlation.

In [5]:
```
# Prepare dataset
# ---------------

MAX_NUM_SENTENCES = 40000
MAX_NUM_WORDS = 20000

ita_sentences = []
eng_sentences = []
eng_sentences_train = []

# Simplify the dataset
MAX_LEN = 3 # words

# Read all lines in translation dataset
count = 0
for line in open(os.path.join('/content/drive/My Drive', 'ita.txt'), encoding='utf-8'):

    if count > MAX_NUM_SENTENCES:
        break

    if '\t' not in line:
        continue

    eng_sentence_, ita_sentence, _ = line.rstrip().split('\t')

    if (len(eng_sentence_.split(' ')) > MAX_LEN or
        len(ita_sentence.split(' ')) > MAX_LEN):
      continue

    eng_sentence = eng_sentence_ + ' <eos>'
    eng_sentence_train = '<sos> ' + eng_sentence_

    ita_sentences.append(ita_sentence)
    eng_sentences.append(eng_sentence)
    eng_sentences_train.append(eng_sentence_train)

    count += 1

print('Number of sentences:', len(ita_sentences))
```

limitations that we can put

encoder input

target sentences

input of the decoder during training

└ we take sentences with max 3 words

we read line per line and we split eng-ita by the "TAB"

there are some lines without the TAB, if we encounter them we skip them with "continue"

(= go to the next one)

encoding for the END OF SENTENCE

encoding for the START OF SENTENCE

```
Number of sentences: 32499
```

In [6]:
```
max(len(sentence.split(' ')) for sentence in eng_sentences)
```

Out[6]: 4

Tokenization

The tokenizer allows to obtain automatically the vocabulary of the words given a text

Converts words to integers

In [7]:
```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

# Create Tokenizer to convert words to integers
ita_tokenizer = Tokenizer(num_words= MAX_NUM_WORDS)
ita_tokenizer.fit_on_texts(ita_sentences)
ita_tokenized = ita_tokenizer.texts_to_sequences(ita_sentences)

ita_wtoi = ita_tokenizer.word_index
print('Total italian words:', len(ita_wtoi))

max_ita_length = max(len(sentence) for sentence in ita_tokenized)
print('Max italian sentence length:', max_ita_length)

eng_tokenizer = Tokenizer(num_words=MAX_NUM_WORDS, filters='?!,."')
eng_tokenizer.fit_on_texts(eng_sentences+eng_sentences_train)
eng_tokenized = eng_tokenizer.texts_to_sequences(eng_sentences)
eng_tokenized_train = eng_tokenizer.texts_to_sequences(eng_sentences_train)

eng_wtoi = eng_tokenizer.word_index
print('Total english words:', len(eng_wtoi))

max_eng_length = max(len(sentence) for sentence in eng_tokenized)
print('Max english sentence length:', max_eng_length)

num_eng_words = len(eng_wtoi) + 1
```

"texts to sequences" converts words to numbers

We do 2 different tokenizers: one for the italian lang. and one for the english

the tokenizer returns the word index organizing the input words of the sentences (it automatically recognizes the words). The indeces are based on the frequence: the word with index 1 is the most frequent word [etc.]

This gives us the dictionary

```
Total italian words: 8524
Max italian sentence length: 4
Total english words: 3858
Max english sentence length: 4
```

Padding sequences

we have to padd the sequences in order to have in a batch sentences which are of the same length (required for LSTM)

In [8]:
```
# Pad to max italian sentence length
ita_encoder_inputs = pad_sequences(ita_tokenized, maxlen=max_ita_length)

print("Italian encoder inputs shape:", ita_encoder_inputs.shape)

# Pad to max italian sentence length
eng_decoder_inputs = pad_sequences(eng_tokenized_train, maxlen=max_eng_length, padding='post')

print("English decoder inputs shape:", eng_decoder_inputs.shape)
```

(we add some zeros at the beginning to obtain the length we want: length that we want = max_ita_length)

(for the outputs we add zeros at the end: padding = 'post')

```
Italian encoder inputs shape: (32499, 4)
English decoder inputs shape: (32499, 4)
```

In [9]:
```
# Pad to max english sentence length
eng_outputs = pad_sequences(eng_tokenized, maxlen=max_eng_length, padding='post')
```

In [10]:
```
ita_encoder_inputs
```

Out[10]:
```
array([[   0,    0,    0, 1211],
       [   0,    0,    0, 2893],
       [   0,    0,    0, 1603],
       ...,
       [   0,    3, 2261, 1561],
       [   0,    1,  861,  158],
       [   0,    1,  500, 1210]], dtype=int32)
```

Model

Hyperparameter that we have to tune

In [11]:
```
# Build Encoder-Decoder Model
# ---------------------------
EMBEDDING_SIZE = 32

# ENCODER
# -------
encoder_input = tf.keras.Input(shape=[max_ita_length])
encoder_embedding_layer = tf.keras.layers.Embedding(len(ita_wtoi)+1, EMBEDDING_SIZE, input_length=max_ita_length, mask_zero=True)
encoder_embedding_out = encoder_embedding_layer(encoder_input)
encoder = tf.keras.layers.LSTM(units=128, return_state=True)
encoder_output, h, c = encoder(encoder_embedding_out)
encoder_states = [h, c]

# DECODER
# -------
decoder_input = tf.keras.Input(shape=[max_eng_length])
decoder_embedding_layer = tf.keras.layers.Embedding(len(eng_wtoi)+1, EMBEDDING_SIZE)
decoder_embedding_out = decoder_embedding_layer(decoder_input)
decoder_lstm = tf.keras.layers.LSTM(units=128, return_sequences=True, return_state=True)

# Initialize decoder state with final encoder state (initial_state=encoder_states)
decoder_lstm_out, _, _ = decoder_lstm(decoder_embedding_out, initial_state=encoder_states)

decoder_dense = tf.keras.layers.Dense(len(eng_wtoi)+1, activation='softmax')
decoder = decoder_dense(decoder_lstm_out)

# MODEL
model = tf.keras.Model([encoder_input, decoder_input], decoder)
```

each word is embedded with 32 values

length of italian vocabulary (from ↑)

because we have the padding

the layers following the embedding one will not output a value for the padding values (= LSTM is not run on the padding (zeros) values)

Alternative to LSTM: GRU (Gated Recurrent Unit) (the choice of which one is an hyperparam that we have to tune)

↳ length of the output vector

IF True, the output will be the output and the state: state = [h, c]

these will initialize the decoder state

If it's "True" then we have the output after each time step

In [13]:
```
model.summary()
#model.weights
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   Connected to
```

```
==================================================================================================
input_1 (InputLayer)            [(None, 4)]          0
__________________________________________________________________________________________________
input_2 (InputLayer)            [(None, 4)]          0
__________________________________________________________________________________________________
embedding (Embedding)           (None, 4, 32)        272800      input_1[0][0]
__________________________________________________________________________________________________
embedding_1 (Embedding)         (None, 4, 32)        123488      input_2[0][0]
__________________________________________________________________________________________________
lstm (LSTM)                     [(None, 128), (None, 82432       embedding[0][0]
__________________________________________________________________________________________________
lstm_1 (LSTM)                   [(None, 4, 128), (No 82432       embedding_1[0][0]
                                                                 lstm[0][1]
                                                                 lstm[0][2]
__________________________________________________________________________________________________
dense (Dense)                   (None, 4, 3859)      497811      lstm_1[0][0]
==================================================================================================
Total params: 1,058,963
Trainable params: 1,058,963
Non-trainable params: 0
__________________________________________________________________________________________________
```

Prepare model for training

In [14]:
```python
# Optimization params
# -------------------
# Loss
loss = tf.keras.losses.SparseCategoricalCrossentropy()
# Learning rate
lr = 1e-3
optimizer = tf.keras.optimizers.Adam(learning_rate=lr)

# Validation metrics
# ------------------
metrics = ['accuracy']

# Compile Model
model.compile(optimizer=optimizer, loss=loss, metrics=metrics)
```

In [15]:
```python
import os
from datetime import datetime

cwd = os.getcwd()

exps_dir = os.path.join('/content/drive/My Drive/KerasRNN', 'translation_experiments')
if not os.path.exists(exps_dir):
    os.makedirs(exps_dir)

now = datetime.now().strftime('%b%d_%H-%M-%S')

exp_name = 'exp'

exp_dir = os.path.join(exps_dir, exp_name + '_' + str(now))
if not os.path.exists(exp_dir):
    os.makedirs(exp_dir)

callbacks = []

# Model checkpoint
# ----------------
ckpt_dir = os.path.join(exp_dir, 'ckpts')
if not os.path.exists(ckpt_dir):
    os.makedirs(ckpt_dir)

ckpt_callback = tf.keras.callbacks.ModelCheckpoint(filepath=os.path.join(ckpt_dir, 'cp_{epoch:02d}.ckpt'),
                                                   save_weights_only=True)  # False to save the model directly
callbacks.append(ckpt_callback)

# Visualize Learning on Tensorboard
# ---------------------------------
tb_dir = os.path.join(exp_dir, 'tb_logs')
if not os.path.exists(tb_dir):
    os.makedirs(tb_dir)

# By default shows losses and metrics for both training and validation
tb_callback = tf.keras.callbacks.TensorBoard(log_dir=tb_dir,
                                             profile_batch=0,
                                             histogram_freq=1)  # if 1 shows weights histograms
callbacks.append(tb_callback)

# Early Stopping
# --------------
early_stop = False
if early_stop:
    es_callback = tf.keras.callback.EarlyStopping(monitor='val_loss', patience=10)
    callbacks.append(es_callback)

## Model
model.fit([ita_encoder_inputs, eng_decoder_inputs],
          eng_outputs,
          epochs=100,
          batch_size=128,
          validation_split=0.2,
          callbacks=callbacks)

# How to visualize Tensorboard

# 1. tensorboard --logdir EXPERIMENTS_DIR --port PORT     <- from terminal
# 2. localhost:PORT   <- in your browser
```

```
Epoch 1/100
204/204 [==============================] - 28s 113ms/step - loss: 6.2915 - accuracy: 0.2422 - val_loss: 4.9048 - val_acc
uracy: 0.2878
```

```
Epoch 2/100
204/204 [==============================] - 21s 102ms/step - loss: 3.9382 - accuracy: 0.3221 - val_loss: 4.5691 - val_acc
uracy: 0.3451
Epoch 3/100
204/204 [==============================] - 21s 103ms/step - loss: 3.4704 - accuracy: 0.4294 - val_loss: 4.3518 - val_acc
uracy: 0.3882
Epoch 4/100
204/204 [==============================] - 21s 102ms/step - loss: 3.1697 - accuracy: 0.4632 - val_loss: 4.2485 - val_acc
uracy: 0.4080
Epoch 5/100
204/204 [==============================] - 21s 102ms/step - loss: 2.9207 - accuracy: 0.4957 - val_loss: 4.0910 - val_acc
uracy: 0.4435
Epoch 6/100
204/204 [==============================] - 21s 102ms/step - loss: 2.6697 - accuracy: 0.5335 - val_loss: 3.9836 - val_acc
uracy: 0.4630
Epoch 7/100
204/204 [==============================] - 21s 101ms/step - loss: 2.4682 - accuracy: 0.5579 - val_loss: 3.9546 - val_acc
uracy: 0.4682
Epoch 8/100
204/204 [==============================] - 21s 102ms/step - loss: 2.2908 - accuracy: 0.5787 - val_loss: 3.9104 - val_acc
uracy: 0.4741
Epoch 9/100
204/204 [==============================] - 21s 102ms/step - loss: 2.1219 - accuracy: 0.6010 - val_loss: 3.8740 - val_acc
uracy: 0.4789
Epoch 10/100
204/204 [==============================] - 21s 101ms/step - loss: 1.9805 - accuracy: 0.6175 - val_loss: 3.8369 - val_acc
uracy: 0.4846
Epoch 11/100
194/204 [===========================>..] - ETA: 0s - loss: 1.8506 - accuracy: 0.6342
```

Translation (inference)

In [19]:

```
# Uncomment this to load model (use this if you want to restore saved model)
# model.load_weights(os.path.join('/content/drive/My Drive/KerasRNN', 'translation_experiments/exp_Dec04_02-03-59', 'ckpts/cp-58.ckpt'))

# Modify the model such that the decoder takes predictions as inputs (no teacher forcing)

# ENCODER (remains the same)
# --------
encoder_model = tf.keras.Model(encoder_input, encoder_states)

# DECODER (modified)
# ------------------
decoder_state_input_h = tf.keras.Input(shape=[128])
decoder_state_input_c = tf.keras.Input(shape=[128])
decoder_state_inputs = [decoder_state_input_h, decoder_state_input_c]

decoder_input_single = tf.keras.Input(shape=[1])
decoder_input_single_embedding = decoder_embedding_layer(decoder_input_single)
decoder_outputs, h, c = decoder_lstm(decoder_input_single_embedding, initial_state=decoder_state_inputs)

decoder_states = [h, c]
decoder_outputs = decoder_dense(decoder_outputs)

decoder_model = tf.keras.Model([decoder_input_single] + decoder_state_inputs,
                               [decoder_outputs]+decoder_states)
```

In [20]:

```
# Translation utils
ita_itow = {v:k for k, v in ita_wtoi.items()}
eng_itow = {v:k for k, v in eng_wtoi.items()}

def translate(input_sentence):

    # Prepare input sentence
    input_tokenized = ita_tokenizer.texts_to_sequences([input_sentence])
    input_tokenized = pad_sequences(input_tokenized, maxlen=max_ita_length)

    # Get encoder state
    states_value = encoder_model.predict(input_tokenized)

    # Set first input '<sos>'
    curr_input = np.zeros([1, 1])  # bs x seq_length (1 x 1 at the beginning)
    curr_input[0, 0] = eng_wtoi['<sos>']
    eos = eng_wtoi['<eos>']

    output_sentence = []

    # Cycle until max_eng_length or until the '<eos>' is predicted
    for _ in range(max_eng_length):
        preds, h_, c_ = decoder_model.predict([curr_input]+states_value)
        word_id = np.argmax(preds[0, 0, :])

        if eos == word_id:
            break

        word = ''

        if word_id > 0:
            word = eng_itow[word_id]
            output_sentence.append(word)

        # Update next input with the predicted one
        curr_input[0, 0] = word_id
        # Update state
        states_value = [h_, c_]

    return ' '.join(output_sentence)
```

In [21]:

```
print(translate("Ciao"))
```

```
welcome
```

Topics: 2/2 text generator

It uses an LSTM to produce the next char. It does not work on words but on chars. It produces the most probable following char given an input

In [1]:
```
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

In [2]:
```
import os
import tensorflow as tf
import numpy as np

# Set the seed for random operations.
# This let our experiments to be reproducible.
SEED = 1234
tf.random.set_seed(SEED)
np.random.seed(SEED)

# Get current working directory
cwd = os.getcwd()
```

In [3]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [4]:
```
!cat /content/drive/My\ Drive/Cantico_di_Natale.txt  # Try to change the text with the one you prefer
```

```
Marley, prima di tutto, era morto. Niente dubbio su questo. Il registro mortuario portava le firme del prete, del chieri
co, dell'appaltatore delle pompe funebri e della persona che aveva guidato il mortoro. Scrooge vi aveva apposto la sua:
e il nome di Scrooge, su qualunque fogliaccio fosse scritto, valeva tant'oro. Il vecchio Marley era proprio morto per qu
anto è morto, come diciamo noi, un chiodo di porta.

Badiamo! [..]
```

Text generation - Next character prediction

Charles Dickens

In [ ]:
```
# Given the text we will:
# 1: divide the text into subsequences of N chars
# 2: build (input, target) pairs, where input is a sequence and target is the
#    same sequence shifted of 1 char
# 3: train a recurrent neural network to predict the next char after each input char
# 4: use the learned model to generate text
```

Dataset

In [5]:
```
# Prepare dataset
# ---------------
# Read full text
with open(os.path.join('/content/drive/My Drive/Cantico_di_Natale.txt'), 'r') as f:
    full_text = f.read()

full_text_length = len(full_text)
print('Full text length:', full_text_length)

# Create vocabulary
vocabulary = sorted(list(set(full_text)))

print('Number of unique characters:', len(vocabulary))
print(vocabulary)

# Dictionaries for char-to-int/int-to-char conversion
ctoi = {c:i for i, c in enumerate(vocabulary)}
itoc = {i:c for i, c in enumerate(vocabulary)}

# Create input-target pairs
# e.g., given an input sequence
# 'Hell' predict the next characters 'ello'
# Thus,
# extract from the full text sequences of length seq_length as x and
# the corresponding seq_length+1 character as target

# Define number of characters to be considered for the prediction
seq_length = 100

X = [] # will contain all the sequences
Y = [] # will contain for each sequence in X the next characters
X_enc = []
Y_enc = []
# Cycle over the full text
step = 1
for i in range(0, full_text_length - (seq_length), step):
    sequence = full_text[i:i+seq_length]
    target = full_text[i+1:i+seq_length+1]
    X.append(sequence)
    Y.append(target)
    X_enc.append([ctoi[c] for c in sequence])
    Y_enc.append([ctoi[c] for c in target])

X = np.array(X)
Y = np.array(Y)
X_enc = np.array(X_enc)
Y_enc = np.array(Y_enc)

print('Number of sequences in the dataset:', len(X))
```

```
Full text length: 158965
Number of unique characters: 69
['\n', ' ', '!', '"', '(', ')', ',', '-', '.', '2', '5', ':', ';', '?', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I',
'J', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'Z', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'k',
```

```
'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'y', 'z', 'È', 'à', 'è', 'é', 'ì', 'ò', 'ù', '’']
Number of sequences in the dataset: 158865
```

In [6]:
```
print("Input Sequence: {}".format(X[0]))
print("Target Sequence: {}".format(Y[0]))
```

```
Input Sequence: Marley, prima di tutto, era morto. Niente dubbio su questo. Il registro mortuario portava le firme d
Target Sequence: arley, prima di tutto, era morto. Niente dubbio su questo. Il registro mortuario portava le firme de
```

In [7]:
```
# Create data loaders
# -------------------
# Batch size
bs = 256

# Encode characters. Many ways, for example one-hot encoding.
def char_encode(x_, y_):
    return tf.one_hot(x_, len(vocabulary)), tf.one_hot(y_, len(vocabulary))

# Prepare input x to match recurrent layer input shape
# -> (bs, seq_length, input_size)

# Training
train_dataset = tf.data.Dataset.from_tensor_slices((X_enc, Y_enc))
train_dataset = train_dataset.shuffle(buffer_size=X_enc.shape[0])
train_dataset = train_dataset.map(char_encode)
train_dataset = train_dataset.batch(bs)
train_dataset = train_dataset.repeat()
```

Model

In [8]:
```
# Build Recurrent Neural Network
# ------------------------------

# We build two models, one for training and one for inference. The two models
# SHARE THE WEIGHTS, thus the inference model will use the learned weights after training

# Training and inference model differ only for the initialization of the state
# in the lstm. In the inference model the state of the lstm is obtained through
# input layers. In this way, we can provide the prediction at time t-1 as input
# at time t for the generation of text at inference time.

# Model architecture (2 stacked lstm layers): Input -> LSTM-1 -> LSTM-2 -> Dense

# Hidden size (state)
h_size = 128

# Model
input_x = tf.keras.Input(shape=(None, len(vocabulary)))

lstm1 = tf.keras.layers.LSTM(
    units=h_size, batch_input_shape=[None, None, len(vocabulary)],
    return_sequences=True, return_state=True, stateful=False)
lstm2 = tf.keras.layers.LSTM(
    units=h_size, return_sequences=True,
    return_state=True, stateful=False)
dense = tf.keras.layers.Dense(units=len(vocabulary), activation='softmax')

x, _, _ = lstm1(input_x)
x, _, _ = lstm2(x)
out = dense(x)

train_model = tf.keras.Model(
    inputs=input_x, outputs=out)

# Inference Model
h1_in = tf.keras.Input(shape=[h_size])
c1_in = tf.keras.Input(shape=[h_size])
h2_in = tf.keras.Input(shape=[h_size])
c2_in = tf.keras.Input(shape=[h_size])

x, h1, c1 = lstm1(input_x, initial_state=[h1_in, c1_in])
x, h2, c2 = lstm2(x, initial_state=[h2_in, c2_in])
out = dense(x)

inference_model = tf.keras.Model(
    inputs=[input_x, h1_in, c1_in, h2_in, c2_in],
    outputs=[out, h1, c1, h2, c2])
```

In [9]:
```
train_model.summary()

inference_model.summary()
#model.weights
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, None, 69)]        0
_________________________________________________________________
lstm (LSTM)                  [(None, None, 128), (None 101376
_________________________________________________________________
lstm_1 (LSTM)                [(None, None, 128), (None 131584
_________________________________________________________________
dense (Dense)                (None, None, 69)          8901
=================================================================
Total params: 241,861
Trainable params: 241,861
Non-trainable params: 0
_________________________________________________________________
Model: "model_1"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
```

```
input_1 (InputLayer)            [(None, None, 69)]   0
__________________________________________________________________________________________________
input_2 (InputLayer)            [(None, 128)]        0
__________________________________________________________________________________________________
input_3 (InputLayer)            [(None, 128)]        0
__________________________________________________________________________________________________
lstm (LSTM)                     [(None, None, 128),  101376      input_1[0][0]
                                                                 input_2[0][0]
                                                                 input_3[0][0]
__________________________________________________________________________________________________
input_4 (InputLayer)            [(None, 128)]        0
__________________________________________________________________________________________________
input_5 (InputLayer)            [(None, 128)]        0
__________________________________________________________________________________________________
lstm_1 (LSTM)                   [(None, None, 128),  131584      lstm[1][0]
                                                                 input_4[0][0]
                                                                 input_5[0][0]
__________________________________________________________________________________________________
dense (Dense)                   (None, None, 69)     8901        lstm_1[1][0]
==================================================================================================
Total params: 241,861
Trainable params: 241,861
Non-trainable params: 0
__________________________________________________________________________________________________
```

Prepare model for training

In [10]:
```
# Optimization params
# -------------------
# Loss
loss = tf.keras.losses.CategoricalCrossentropy()

# Learning rate
lr = 1e-3
optimizer = tf.keras.optimizers.Adam(learning_rate=lr)

# Validation metrics
# ------------------
metrics = ['accuracy']

# Compile Model
train_model.compile(optimizer=optimizer, loss=loss, metrics=metrics)
```

In [11]:
```
import os
from datetime import datetime

cwd = os.getcwd()

exps_dir = os.path.join('/content/drive/My Drive/KerasRNN', 'text_gen_experiments')
if not os.path.exists(exps_dir):
    os.makedirs(exps_dir)

now = datetime.now().strftime('%b%d_%H-%M-%S')

exp_name = 'exp'

exp_dir = os.path.join(exps_dir, exp_name + '_' + str(now))
if not os.path.exists(exp_dir):
    os.makedirs(exp_dir)

callbacks = []

# Model checkpoint
# ----------------
ckpt_dir = os.path.join(exp_dir, 'ckpts')
if not os.path.exists(ckpt_dir):
    os.makedirs(ckpt_dir)

ckpt_callback = tf.keras.callbacks.ModelCheckpoint(filepath=os.path.join(ckpt_dir, 'cp_{epoch:02d}.ckpt'),
                                                   save_weights_only=True)  # False to save the model directly
callbacks.append(ckpt_callback)

# Visualize Learning on Tensorboard
# ---------------------------------
tb_dir = os.path.join(exp_dir, 'tb_logs')
if not os.path.exists(tb_dir):
    os.makedirs(tb_dir)

# By default shows losses and metrics for both training and validation
tb_callback = tf.keras.callbacks.TensorBoard(log_dir=tb_dir,
                                             profile_batch=0,
                                             histogram_freq=1)  # if 1 shows weights histograms
callbacks.append(tb_callback)

# Early Stopping
# --------------
early_stop = False
if early_stop:
    es_callback = tf.keras.callback.EarlyStopping(monitor='val_loss', patience=10)
    callbacks.append(es_callback)

# How to visualize Tensorboard

# 1. tensorboard --logdir EXPERIMENTS_DIR --port PORT     <- from terminal
# 2. localhost:PORT   <- in your browser
```

In [12]:
```
# It is difficult to have a validation metric to evaluate quantitatively the quality
# of a generation, since we are generating new text. Thus, for example, we can
# evaluate qualitatively the generation performance by generating some text during validation.
# We can do it through a custom callback which, at the beginning of each epoch (on_epoch_begin),
# will take a list of test sequences and use the trained model to generate some text.
```

```
class TextGenerationCallback(tf.keras.callbacks.Callback):

  def __init__(self, inference_model, start_sequences,
               generation_length, temperature):
    super(TextGenerationCallback, self).__init__()

    self.inference_model = inference_model
    self.start_sequences = start_sequences
    self.generation_length = generation_length
    self.temperature = temperature

  def sample(self, pred, temperature=1.0):
    # Helper function to sample an index from a probability array
    # Temperature is a parameter that allows to scale the output of the network,
    # thus changing the level of 'exploration' in the output distribution. Try
    # yourself to play with this parameter and to see the difference in the generation.
    pred = np.asarray(pred).astype('float64')
    pred = np.log(pred) / temperature
    exp_pred = np.exp(pred)
    pred = exp_pred / np.sum(exp_pred)
    probas = np.random.multinomial(1, pred, 1)
    return np.argmax(probas)

  def generate_text(self, start_sequence, temperature):
    encoded_input = [ctoi[c] for c in start_sequence]
    encoded_input = tf.one_hot(encoded_input, len(vocabulary))
    encoded_input = tf.expand_dims(encoded_input, 0)

    generated_sequence = start_sequence

    in_states = [tf.zeros([1, h_size]) for _ in range(4)]

    for i in range(self.generation_length):
      output = self.inference_model.predict([encoded_input] + in_states)

      in_states = output[1:]
      pred = output[0][0, -1]
      # To get the final prediction we should use the argmax as usual but if we
      # do so, we will take always the most probable char, so we could incur into a generation loop.
      # Thus, instead of taking the max probability, we sample from the output distribution,
      # giving a chance to explore also other chars during the generation.
      # We do it with an helper function, that is the 'sample' one.
      pred = self.sample(pred, temperature)
      pred_char = itoc[pred]

      generated_sequence += pred_char

      encoded_input = tf.one_hot(pred, len(vocabulary))
      encoded_input = tf.reshape(encoded_input, [1, 1, len(vocabulary)])

    return generated_sequence

  def on_epoch_begin(self, epoch, logs):
    print("Epoch: {}".format(epoch))
    for start_seq in self.start_sequences:
      print("Starting Sequence: {}".format(start_seq))
      for temp in self.temperature:
        print("Temperature: {}".format(temp))
        generated_seq = self.generate_text(start_seq, temp)
        print(generated_seq)
    return
```

In [13]:
```
valid_callback = TextGenerationCallback(inference_model=inference_model,
                                        start_sequences=['Un giorno'], generation_length=100,
                                        temperature=[0.2, 0.5, 1.0, 1.2])
callbacks.append(valid_callback)
```

In [14]:
```
train_model.fit(x=train_dataset,
                epochs=100,  #### set repeat in training dataset
                steps_per_epoch=int(np.ceil(X_enc.shape[0] / bs)),
                callbacks=callbacks)
```

```
Epoch 1/100
Epoch: 0
Starting Sequence: Un giorno
Temperature: 0.2
Un giornovkSOWEuC-,L!rlLuù!MOÈs
CSs(ETZCCdkùùDvSdAs'Iwù5éodGÈWUk.yR5MaOZelNJZÈè.gEfay:èu!mpéubHkZ2ouudaòG-;Ml
Temperature: 0.5
Un giornoàZél vàVPppq?nr5,'kRéSè,PsHùrN élGvhVùveùI-(vJefBVD!Hà:hWVpUftE(Aé,?lLRò5àAA;QNGzJphdÈì:G5nCé!WU,t I
Temperature: 1.0
Un giornopféàJofg:EQUVvèRQT?IUNRnhTùvPoJisBgMOPò2SMU èTVQdopcz2w?r'. -P
TìsCÈsm;VyBèq!,2?i)yUJvA!à(Qep(qm auM
Temperature: 1.2
Un giornoNp.vbùDragEMVAè-hSèP(.PcGzwòp-f;R2yzèL;ì(wVk',UCOWmmaWf'ECSAFbwLCkDÈlùhkQCNf?àbd -òmfPU
yFAc5pwB,;FO
  6/621 [..............................] - ETA: 38s - loss: 4.1992 - accuracy: 0.1082WARNING:tensorflow:Callback method
`on_train_batch_end` is slow compared to the batch time (batch time: 0.0266s vs `on_train_batch_end` time: 0.0317s). Che
ck your callbacks.
621/621 [==============================] - 44s 26ms/step - loss: 2.8979 - accuracy: 0.2129
Epoch 2/100
Epoch: 1
Starting Sequence: Un giorno
Temperature: 0.2
Un giorno di sua di con pitto di sera di suno di suo di che la porto di la sore di conte di sere al pio di co
Temperature: 0.5
Un giorno de pio di porto de fore le sprose, le sconte di soletto i cona che con en a cosse cino con è parela
Temperature: 1.0
Un giorno di sufiò i tropero anrel litre. La no caccovera pittor a stofata sel hi endaszeto la senitto né so
Temperature: 1.2
Un giornoggio! L cone do veseno. Sun o ché qòrito al lasandii pidere, undo e è queste l'alalà, gnepe ana avet
621/621 [==============================] - 34s 26ms/step - loss: 1.9851 - accuracy: 0.3846
Epoch 3/100
Epoch: 2
Starting Sequence: Un giorno
```

```
Temperature: 0.2
Un giorno di statto alla statto di sore di suo che la spette di sua con la state la si pore di si fuore a con
Temperature: 0.5
Un giorno di sun al fecchio da stesto il veste quella statti una gli che sua costara cami che nella cancari d
Temperature: 1.0
Un giorno, scostandono. Alduosi glabbe detere quazza fornate, iganchio anghessa, lo Sproo sua gli porto dinqu
Temperature: 1.2
Un giorno, Fera elute ave angiole; si inchiava pio gagrizzali so con un gri afpote"
oss'ondbiata orr'vagone l
621/621 [==============================] - 34s 26ms/step - loss: 1.8042 - accuracy: 0.4386
Epoch 4/100
Epoch: 3
Starting Sequence: Un giorno
Temperature: 0.2
Un giorno di suo come si stato e se si seggiere in capo di Scrooge - disse Scrooge e si persa a cosa a cosa a
Temperature: 0.5
Un giorno e il porso che il volte lo stare il vorto il suo non che bisara con la suo del sobbeno a andito la
Temperature: 1.0
Un giorno spegna della raffico! -

Ha con logk Ma piritasté; me che tarore lo state, dute camverlo.

- Più se
Temperature: 1.2
Un giorno davrato quanno nevoa di rormusa sempo né più petterdori, l'ega come verfino. Volo te la drantato. F
621/621 [==============================] - 34s 26ms/step - loss: 1.6925 - accuracy: 0.4708
Epoch 5/100
Epoch: 4
Starting Sequence: Un giorno
Temperature: 0.2
Un giorno a compressi di capo di sole stesso se non si pare di quella con un faccio del visito di stanza alla
Temperature: 0.5
Un giorno a culito e sentina con assua che soliva alla più al suo mano un pano alla stesse di biano di scorte
Temperature: 1.0
Un giorno al si fite, mando. Il cirsi uni una grati i somipità, poltesso e che basciò che sgazzando, fecchi d
Temperature: 1.2
Un giorno, fenso suovo. Qui che (profrano per busciò dubbato sololita: la quanto se guiti sebbiusi di trobben
621/621 [==============================] - 34s 26ms/step - loss: 1.6082 - accuracy: 0.4959
Epoch 6/100
Epoch: 5
Starting Sequence: Un giorno
Temperature: 0.2
Un giorno di Scrooge di Scrooge di siano di tutto sulla stata per le parta di stato di casa di una dissero a
Temperature: 0.5
Un giorno di facer le parette di non si per sbalire della sarella della pirità di ogni stato il pasare a tarn
Temperature: 1.0
Un giorno di quel lento sulla strelche ebale al trava montroccelliditi seccoso si dotare, Ti per vituto malo,
Temperature: 1.2
Un giorno predensi.

- Lra peimo. Besa avrescollo, efottà affirito in quento Crottando? - Acconfinile, se, e
621/621 [==============================] - 34s 26ms/step - loss: 1.5361 - accuracy: 0.5180
Epoch 7/100
Epoch: 6
Starting Sequence: Un giorno
Temperature: 0.2
Un giorno di stato il suo fatto di casa di Scrooge di sol solo così sopra di fare che si sentiva a casa di ri
Temperature: 0.5
Un giorno della fare che nella sua bascia di fare che la spiglia; ma non sono farle in quale se dove da caper
Temperature: 1.0
Un giorno ligerte gli si chiale luccia piati, chiema: Ebbele così verre a mane. Bob, Scrooge fattaggazzo e go
Temperature: 1.2
Un giornore la bionvana, per una vecchia Scietella e là qualchi il bocnava si forseno sella saruta saffotta?"
621/621 [==============================] - 34s 27ms/step - loss: 1.4736 - accuracy: 0.5366
Epoch 8/100
Epoch: 7
Starting Sequence: Un giorno
Temperature: 0.2
Un giorno di quella stata per le stato in cortere la figlia di più freddi di quale di conto di solito di quel
Temperature: 0.5
Un giorno sempre in un bambino della stesse e sedere davanti in cillere il pensole si spiriti. Il nipote di t
Temperature: 1.0
Un giorno puttro solo quella soppresse s'ardervi quel cario le lentasse.

- gli mia sognir la meso una cotest
Temperature: 1.2
Un giorno; un mio buone primatanza non io nei volesse littuni, da catra matuta chi riveretta per la parso alm
621/621 [==============================] - 34s 27ms/step - loss: 1.4160 - accuracy: 0.5532
Epoch 9/100
Epoch: 8
Starting Sequence: Un giorno
Temperature: 0.2
Un giorno al morto come un po' di fanciulla, come se non si prese che solo in casa di Scrooge di Scrooge si s
Temperature: 0.5
Un giorno delle al fuoco per stato il pansola e di fu per vedere di tutto in altro commetto e il cappello. Sc
Temperature: 1.0
Un giorno al sposso. Ah, ah, badessero La sfragativa a c'è sariune."

Senza quando con Natale. Voi vostra rag
Temperature: 1.2
Un giorno alche ero consutino mondato un'altro...

- È Lai giocoscie, lintro, un suoro, mi pendato. Sai dispe
212/621 [=========>....................] - ETA: 10s - loss: 1.3688 - accuracy: 0.5670
```