In [1]:

```
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import LSTM
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.callbacks import Callback
from tensorflow.keras.callbacks import TensorBoard
import sys
import os
```

In [2]:

```
from google.colab import files
```

In [ ]:

```
files.upload()
```

In [4]:

```
!ls
os.getcwd()
```

```
sample_data  wonderland.txt
```

Out[4]:

```
'/content'
```

Загрузка текста и в нижний регистр

In [5]:

```
filename = "wonderland.txt"
raw_text = open(filename).read() # не закрыть ли???
raw_text = raw_text.lower()
```

Создание набора уникальных символов и создание словаря символ - число

In [6]:

```
chars = sorted(list(set(raw_text)))
char_to_int = dict((c, i) for i, c in enumerate(chars))
```

In [7]:

```
n_chars = len(raw_text)
n_vocab = len(chars)
print("Total Characters: ", n_chars)
print("Total Vocab: ", n_vocab)
```

```
Total Characters:  144404
Total Vocab:  45
```

Деление на последовательности и перевод символов в целые числа

In [8]:

```
seq_length = 100
dataX = []
dataY = []
for i in range(0, n_chars - seq_length, 1):
 seq_in = raw_text[i:i + seq_length]
 seq_out = raw_text[i + seq_length]
 dataX.append([char_to_int[char] for char in seq_in])
 dataY.append(char_to_int[seq_out])
n_patterns = len(dataX)
print("Total Patterns: ", n_patterns)
```

```
Total Patterns:  144304
```

Преобразование список вход-х посл-й в форму **[**образцы**,** временные шаги**,** признаки**]** поменять масштаб в **0** до **1 +** преобразовать вх-е шаблоны по **hot-**кодированию **(46 0** и одна **'1')**

In [9]:

```
X = np.reshape(dataX, (n_patterns, seq_length, 1))   # reshape X to be [samples, time st
eps, features]
X = X / float(n_vocab)                                   # normalize
y = to_categorical(dataY)                       # one hot encode the output variable
int_to_char = dict((i, c) for i, c in enumerate(chars))
```

Определение модели

In [10]:

```
model = Sequential()
model.add(LSTM(256, input_shape=(X.shape[1], X.shape[2])))
model.add(Dropout(0.2))
model.add(Dense(y.shape[1], activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam')
```

In [11]:

```
def gen_text(model, epoch=0, console_log=False, seq_len=1000):
    # pick a random seed
    start = np.random.randint(0, len(dataX)-1)
    pattern = dataX[start]
    if console_log:
        print("Seed:")
        print("\"", ''.join([int_to_char[value] for value in pattern]), "\"")

    full_result = []
    # generate characters
    for i in range(seq_len):
        x = np.reshape(pattern, (1, len(pattern), 1))
        x = x / float(n_vocab)
        prediction = model.predict(x, verbose=0)
        index = np.argmax(prediction)
        result = int_to_char[index]
        full_result.append(result)
        seq_in = [int_to_char[value] for value in pattern]
        if console_log:
            #sys.stdout.write("On {} epoch:".format(epoch))
            sys.stdout.write(result)
        pattern.append(index)
        pattern = pattern[1:len(pattern)]
    print("\nDone.")
    return ''.join(full_result)
```

In [12]:

```
class TrainGeneratTextLogger(Callback):
    def on_epoch_end(self, epoch, logs=None):
        if not hasattr(self, 'freq'):
            self.freq = 2
        if not hasattr(self, 'save_file'):
            self.save_file = False
        if not hasattr(self, 'console_log'):
            self.console_log = True
        if not hasattr(self, 'seq_len'):
            self.seq_len = 1000

        if epoch % self.freq == 0:
            res = gen_text(self.model, epoch,
                           console_log=self.console_log,
                           seq_len=self.seq_len)
            if self.console_log:
                sys.stdout.write("Epoch {}:".format(epoch))
                sys.stdout.write(res)
            if self.save_file:
                with open("gen_text_ep_{}".format(epoch) + ".txt", "w") as file:
                    file.write(res)

    def set_freq(self, freq):
        self.freq = freq

    def set_save_file(self, save_file):
        self.save_file = save_file

    def set_console_log(self, console_log):
        self.console_log = console_log

    def set_seq_len(self, seq_len):
        self.seq_len = seq_len
```

Контрольные точки для весов **(Для val_acc** это должно быть **max,** для **val_loss-min ModelCheckpoint - сохранение весов после каждой эпохи, без затирания)**

In [13]:

```
filepath="weights-improvement-{epoch:02d}-{loss:.4f}.hdf5"
checkpoint = ModelCheckpoint(filepath, monitor='loss', verbose=1, save_best_only=True, m
ode='min')
```

In [14]:

```
train_log_cb = TrainGeneratTextLogger()
train_log_cb.set_freq(4)
train_log_cb.set_save_file(True)
train_log_cb.set_console_log(True)
train_log_cb.set_seq_len(1000)
```

In [15]:

```
callbacks = [
     checkpoint,
     TensorBoard(
        log_dir='my_log_dir',
        histogram_freq=1,
        embeddings_freq=1,
     ),
     train_log_cb
]
```

In [ ]:

```
model.fit(X, y, epochs=30, batch_size=128, callbacks=callbacks)
```

In [ ]:

```
!ls
model.summary()
```

In [ ]:

```
!tensorboard --logdir=my_log_dir
```

In [52]:

```
from google.colab import drive
```

```
drive.mount('/content/gdrive', force_remount=True)
```

```
Mounted at /content/gdrive
```

In [53]:

```
import shutil
shutil.make_archive('my_log_dir', 'zip', 'my_log_dir')
```

Out[53]:

```
'/content/my_log_dir.zip'
```

Генерируем текст

In [50]:

```
res_text = gen_text(model)
with open("gen_text_Finally_11" + ".txt", "w") as file:
    file.write(res_text)
```

```
Done.
```

Загружаем веса сети

In [54]:

```
from tensorflow.keras.models import load_model

model = load_model("weights-improvement-30-1.4626.hdf5")
res_text = gen_text(model)
with open("gen_text_Finally_22" + ".txt", "w") as file:
    file.write(res_text)
```

```
Done.
```