{

"nbformat": 4,

"nbformat\_minor": 0,

"metadata": {

"colab": {

"name": "Machine\_Translation.ipynb",

"provenance": [],

"collapsed\_sections": [],

"toc\_visible": true

},

"kernelspec": {

"name": "python3",

"display\_name": "Python 3"

},

"accelerator": "GPU"

},

"cells": [

{

"cell\_type": "markdown",

"metadata": {

"id": "w3LV6bzWmCUN"

},

"source": [

"# Text Translation"

]

},

{

"cell\_type": "markdown",

"metadata": {

"id": "YJs3pIp0l-MW"

},

"source": [

"## Importing Library"

]

},

{

"cell\_type": "code",

"metadata": {

"id": "RrgHQcjk11rW"

},

"source": [

"# get the data at: http://www.manythings.org/anki/"

],

"execution\_count": null,

"outputs": []

},

{

"cell\_type": "code",

"metadata": {

"id": "HOZDC22\_16Br"

},

"source": [

"import os, sys\n",

"import numpy as np\n",

"import matplotlib.pyplot as plt\n",

"\n",

"from tensorflow.keras.preprocessing.text import Tokenizer\n",

"from tensorflow.keras.preprocessing.sequence import pad\_sequences\n",

"\n",

"from tensorflow.keras.utils import to\_categorical\n",

"\n",

"from tensorflow.keras.layers import Input, LSTM, GRU, Dense, Embedding\n",

"from tensorflow.keras.models import Model"

],

"execution\_count": null,

"outputs": []

},

{

"cell\_type": "markdown",

"metadata": {

"id": "mSBgMHajmHcl"

},

"source": [

"## Defining Configuration"

]

},

{

"cell\_type": "code",

"metadata": {

"id": "6mT9RpoT2DWv"

},

"source": [

"# some config\n",

"BATCH\_SIZE = 64 # Batch size for training.\n",

"EPOCHS = 100 # Number of epochs to train for.\n",

"LATENT\_DIM = 256 # Latent dimensionality of the encoding space.\n",

"NUM\_SAMPLES = 10000 # Number of samples to train on.\n",

"MAX\_SEQUENCE\_LENGTH = 100\n",

"MAX\_NUM\_WORDS = 20000\n",

"EMBEDDING\_DIM = 50"

],

"execution\_count": null,

"outputs": []

},

{

"cell\_type": "markdown",

"metadata": {

"id": "bDr83n2-mOaF"

},

"source": [

"## Loading Data and Data Preparation"

]

},

{

"cell\_type": "code",

"metadata": {

"id": "jA-1qBVE3Z6g",

"colab": {

"base\_uri": "https://localhost:8080/"

},

"outputId": "3c239d9b-f728-45a2-a36e-5188eef6adac"

},

"source": [

"from google.colab import drive\n",

"drive.mount('/content/drive', force\_remount=True)"

],

"execution\_count": null,

"outputs": [

{

"output\_type": "stream",

"text": [

"Mounted at /content/drive\n"

],

"name": "stdout"

}

]

},

{

"cell\_type": "code",

"metadata": {

"id": "bNh0ouPi4Gkf",

"colab": {

"base\_uri": "https://localhost:8080/"

},

"outputId": "c161a2d2-e6f5-4066-898d-1d67fcd81d87"

},

"source": [

"!unzip '/content/drive/My Drive/data/spa-eng.zip'"

],

"execution\_count": null,

"outputs": [

{

"output\_type": "stream",

"text": [

"Archive: /content/drive/My Drive/data/spa-eng.zip\n",

" inflating: \_about.txt \n",

" inflating: spa.txt \n"

],

"name": "stdout"

}

]

},

{

"cell\_type": "code",

"metadata": {

"id": "LVCIYaxo5L7H",

"colab": {

"base\_uri": "https://localhost:8080/"

},

"outputId": "c0d9df44-e626-407a-9853-f3bc98542def"

},

"source": [

"!unzip '/content/drive/My Drive/glove/glove.6B.50d.txt.zip'"

],

"execution\_count": null,

"outputs": [

{

"output\_type": "stream",

"text": [

"Archive: /content/drive/My Drive/glove/glove.6B.50d.txt.zip\n",

" inflating: glove.6B.50d.txt \n"

],

"name": "stdout"

}

]

},

{

"cell\_type": "code",

"metadata": {

"colab": {

"base\_uri": "https://localhost:8080/"

},

"id": "-2wNtqX1aTnN",

"outputId": "2b32cd4f-a43f-48a1-b012-ae8c12fe1a60"

},

"source": [

"!head -5 'spa.txt'"

],

"execution\_count": null,

"outputs": [

{

"output\_type": "stream",

"text": [

"Go.\tVe.\tCC-BY 2.0 (France) Attribution: tatoeba.org #2877272 (CM) & #4986655 (cueyayotl)\n",

"Go.\tVete.\tCC-BY 2.0 (France) Attribution: tatoeba.org #2877272 (CM) & #4986656 (cueyayotl)\n",

"Go.\tVaya.\tCC-BY 2.0 (France) Attribution: tatoeba.org #2877272 (CM) & #4986657 (cueyayotl)\n",

"Go.\tV√°yase.\tCC-BY 2.0 (France) Attribution: tatoeba.org #2877272 (CM) & #6586271 (arh)\n",

"Hi.\tHola.\tCC-BY 2.0 (France) Attribution: tatoeba.org #538123 (CM) & #431975 (Leono)\n"

],

"name": "stdout"

}

]

},

{

"cell\_type": "code",

"metadata": {

"id": "z9TyUbu7\_0Rj"

},

"source": [

"# Where we will store the data\n",

"input\_texts = [] # sentence in source language\n",

"target\_texts = [] # sentence in target language\n",

"target\_texts\_inputs = [] # sentence in target language offset by 1"

],

"execution\_count": null,

"outputs": []

},

{

"cell\_type": "code",

"metadata": {

"id": "4y8dq56Q224e"

},

"source": [

"t = 0\r\n",

"for line in open('spa.txt'):\r\n",

" # only keep a limited number of samples\r\n",

" t += 1\r\n",

" if t > NUM\_SAMPLES: # Number of samples is kept less as at the end of the document the length is big and with bigger length, we have to do more padding.\r\n",

" break\r\n",

"\r\n",

" # input and target are separated by tab\r\n",

" if '\\t' not in line:\r\n",

" continue\r\n",

"\r\n",

" # split up the input and translation\r\n",

" input\_text = line.rstrip().split('\\t')[0]\r\n",

" translation = line.rstrip().split('\\t')[1]\r\n",

"\r\n",

" # make the target input and output\r\n",

" # recall we'll be using teacher forcing\r\n",

" target\_text = translation + ' <end>'\r\n",

" target\_text\_input = '<start> ' + translation\r\n",

"\r\n",

" input\_texts.append(input\_text)\r\n",

" target\_texts.append(target\_text)\r\n",

" target\_texts\_inputs.append(target\_text\_input)"

],

"execution\_count": null,

"outputs": []

},

{

"cell\_type": "code",

"metadata": {

"id": "py8sFFmg2EQb",

"colab": {

"base\_uri": "https://localhost:8080/"

},

"outputId": "7ddabaed-9ab4-4d33-ddc7-165eb92610cd"

},

"source": [

"print(\"num samples:\", len(input\_texts))"

],

"execution\_count": null,

"outputs": [

{

"output\_type": "stream",

"text": [

"num samples: 10000\n"

],

"name": "stdout"

}

]

},

{

"cell\_type": "markdown",

"metadata": {

"id": "y3JvtKwVmVuN"

},

"source": [

"## Tokenization, conversion to integers and creating sequences"

]

},

{

"cell\_type": "markdown",

"metadata": {

"id": "xPh6wGB\_mcvW"

},

"source": [

"### Encoder: Input"

]

},

{

"cell\_type": "code",

"metadata": {

"id": "K0ZDtbcN2FMq"

},

"source": [

"# tokenize the inputs\n",

"tokenizer\_inputs = Tokenizer(num\_words=MAX\_NUM\_WORDS)\n",

"tokenizer\_inputs.fit\_on\_texts(input\_texts)\n",

"input\_sequences = tokenizer\_inputs.texts\_to\_sequences(input\_texts)"

],

"execution\_count": null,

"outputs": []

},

{

"cell\_type": "code",

"metadata": {

"id": "t2Ei5XjX2IDU",

"colab": {

"base\_uri": "https://localhost:8080/"

},

"outputId": "540935a6-fdf9-4352-bdfc-b62af44fdb0a"

},

"source": [

"# get the word to index mapping for input language\n",

"word2idx\_inputs = tokenizer\_inputs.word\_index\n",

"print('Found %s unique input tokens.' % len(word2idx\_inputs))"

],

"execution\_count": null,

"outputs": [

{

"output\_type": "stream",

"text": [

"Found 2337 unique input tokens.\n"

],

"name": "stdout"

}

]

},

{

"cell\_type": "code",

"metadata": {

"id": "QFQrWLWx2Io3",

"colab": {

"base\_uri": "https://localhost:8080/"

},

"outputId": "ad93155c-4605-48e1-f982-332eca40a2a0"

},

"source": [

"# determine maximum length input sequence\n",

"max\_len\_input = max(len(s) for s in input\_sequences)\n",

"max\_len\_input"

],

"execution\_count": null,

"outputs": [

{

"output\_type": "execute\_result",

"data": {

"text/plain": [

"5"

]

},

"metadata": {

"tags": []

},

"execution\_count": 12

}

]

},

{

"cell\_type": "markdown",

"metadata": {

"id": "FCHOUJOomgGQ"

},

"source": [

"### Decoder: Output"

]

},

{

"cell\_type": "code",

"metadata": {

"id": "Xk6GfIQ42JKs"

},

"source": [

"# tokenize the outputs\n",

"# don't filter out special characters\n",

"# otherwise <start> and <end> won't appear\n",

"\n",

"tokenizer\_outputs = Tokenizer(num\_words=MAX\_NUM\_WORDS, filters='')\n",

"tokenizer\_outputs.fit\_on\_texts(target\_texts + target\_texts\_inputs)\n",

"\n",

"target\_sequences = tokenizer\_outputs.texts\_to\_sequences(target\_texts)\n",

"target\_sequences\_inputs = tokenizer\_outputs.texts\_to\_sequences(target\_texts\_inputs)"

],

"execution\_count": null,

"outputs": []

},

{

"cell\_type": "code",

"metadata": {

"id": "AJtfeBMD2Jtc",

"colab": {

"base\_uri": "https://localhost:8080/"

},

"outputId": "f9c622de-3857-4ef3-a256-ad750b0a5cd2"

},

"source": [

"# get the word to index mapping for output language\n",

"\n",

"word2idx\_outputs = tokenizer\_outputs.word\_index\n",

"print('Found %s unique output tokens.' % len(word2idx\_outputs))"

],

"execution\_count": null,

"outputs": [

{

"output\_type": "stream",

"text": [

"Found 6316 unique output tokens.\n"

],

"name": "stdout"

}

]

},

{

"cell\_type": "code",

"metadata": {

"id": "CygmZukK2KQ4",

"colab": {

"base\_uri": "https://localhost:8080/"

},

"outputId": "048f7eb2-7ab4-4c51-b6b2-ca330219181c"

},

"source": [

"# store number of output words for later\n",

"# remember to add 1 since indexing starts at 1\n",

"num\_words\_output = len(word2idx\_outputs) + 1\n",

"\n",

"# determine maximum length output sequence\n",

"max\_len\_target = max(len(s) for s in target\_sequences)\n",

"max\_len\_target"

],

"execution\_count": null,

"outputs": [

{

"output\_type": "execute\_result",

"data": {

"text/plain": [

"9"

]

},

"metadata": {

"tags": []

},

"execution\_count": 15

}

]

},

{

"cell\_type": "markdown",

"metadata": {

"id": "gvIhJpAvnif\_"

},

"source": [

"### Padding Sequences"

]

},

{

"cell\_type": "code",

"metadata": {

"id": "P\_uGzoUc2Lg9",

"colab": {

"base\_uri": "https://localhost:8080/"

},

"outputId": "e4579950-1c53-4d86-fe69-6e590407f70e"

},

"source": [

"# pad the sequences\n",

"encoder\_inputs = pad\_sequences(input\_sequences, maxlen=max\_len\_input)\n",

"print(\"encoder\_inputs.shape:\", encoder\_inputs.shape)\n",

"print(\"encoder\_inputs[0]:\", encoder\_inputs[0])\n",

"\n",

"decoder\_inputs = pad\_sequences(target\_sequences\_inputs, maxlen=max\_len\_target, padding='post')\n",

"print(\"decoder\_inputs[0]:\", decoder\_inputs[0])\n",

"print(\"decoder\_inputs.shape:\", decoder\_inputs.shape)\n",

"\n",

"decoder\_targets = pad\_sequences(target\_sequences, maxlen=max\_len\_target, padding='post')"

],

"execution\_count": null,

"outputs": [

{

"output\_type": "stream",

"text": [

"encoder\_inputs.shape: (10000, 5)\n",

"encoder\_inputs[0]: [ 0 0 0 0 15]\n",

"decoder\_inputs[0]: [ 2 1468 0 0 0 0 0 0 0]\n",

"decoder\_inputs.shape: (10000, 9)\n"

],

"name": "stdout"

}

]

},

{

"cell\_type": "markdown",

"metadata": {

"id": "gHuwp5uDnmL-"

},

"source": [

"## Embedding -- Pre-train"

]

},

{

"cell\_type": "code",

"metadata": {

"id": "ZZZVas1T2MjW",

"colab": {

"base\_uri": "https://localhost:8080/"

},

"outputId": "8f656f7b-2b94-48b1-f44a-0f19ed3d780a"

},

"source": [

"# store all the pre-trained word vectors\n",

"print('Loading word vectors...')\n",

"word2vec = {}\n",

"with open(os.path.join('glove.6B.%sd.txt' % EMBEDDING\_DIM)) as f:\n",

" # is just a space-separated text file in the format:\n",

" # word vec[0] vec[1] vec[2] ...\n",

" for line in f:\n",

" values = line.split()\n",

" word = values[0]\n",

" vec = np.asarray(values[1:], dtype='float32')\n",

" word2vec[word] = vec\n",

"print('Found %s word vectors.' % len(word2vec))"

],

"execution\_count": null,

"outputs": [

{

"output\_type": "stream",

"text": [

"Loading word vectors...\n",

"Found 400000 word vectors.\n"

],

"name": "stdout"

}

]

},

{

"cell\_type": "markdown",

"metadata": {

"id": "yOT30Fe3nq1-"

},

"source": [

"### Embedding Matrix"

]

},

{

"cell\_type": "code",

"metadata": {

"id": "AddETm9L2OKH",

"colab": {

"base\_uri": "https://localhost:8080/"

},

"outputId": "c4e81537-7891-4a18-869e-7e6950f48ece"

},

"source": [

"# prepare embedding matrix\n",

"print('Filling pre-trained embeddings...')\n",

"num\_words = min(MAX\_NUM\_WORDS, len(word2idx\_inputs) + 1)\n",

"embedding\_matrix = np.zeros((num\_words, EMBEDDING\_DIM))\n",

"for word, i in word2idx\_inputs.items():\n",

" if i < MAX\_NUM\_WORDS:\n",

" embedding\_vector = word2vec.get(word)\n",

" if embedding\_vector is not None:\n",

" # words not found in embedding index will be all zeros.\n",

" embedding\_matrix[i] = embedding\_vector"

],

"execution\_count": null,

"outputs": [

{

"output\_type": "stream",

"text": [

"Filling pre-trained embeddings...\n"

],

"name": "stdout"

}

]

},

{

"cell\_type": "markdown",

"metadata": {

"id": "YMf9J8cQntMh"

},

"source": [

"### Embedding Layer"

]

},

{

"cell\_type": "code",

"metadata": {

"id": "JzmHOaXe2PPx"

},

"source": [

"# create embedding layer\n",

"embedding\_layer = Embedding(\n",

" num\_words,\n",

" EMBEDDING\_DIM,\n",

" weights=[embedding\_matrix],\n",

" input\_length=max\_len\_input\n",

")"

],

"execution\_count": null,

"outputs": []

},

{

"cell\_type": "markdown",

"metadata": {

"id": "0tvh4C9yqeqa"

},

"source": [

"## OHE Output"

]

},

{

"cell\_type": "code",

"metadata": {

"id": "Tko0ZG2R2P02"

},

"source": [

"# One-hot encoded version of the decoder target, as we have \n",

"# categorical cross entropy as the loss.\n",

"decoder\_targets\_one\_hot = np.zeros((len(input\_texts), max\_len\_target, num\_words\_output), dtype='float32')\n",

"\n",

"# assign the values\n",

"for i, d in enumerate(decoder\_targets):\n",

" for t, word in enumerate(d):\n",

" decoder\_targets\_one\_hot[i, t, word] = 1"

],

"execution\_count": null,

"outputs": []

},

{

"cell\_type": "markdown",

"metadata": {

"id": "LGuZ2dyqn-Yn"

},

"source": [

"## Model Building"

]

},

{

"cell\_type": "markdown",

"metadata": {

"id": "roLW7GBPoFnr"

},

"source": [

"### Encoder"

]

},

{

"cell\_type": "code",

"metadata": {

"id": "RG\_jAbxr2Qaq"

},

"source": [

"encoder\_inputs\_placeholder = Input(shape=(max\_len\_input,))\n",

"x = embedding\_layer(encoder\_inputs\_placeholder)\n",

"\n",

"encoder = LSTM(\n",

" LATENT\_DIM,\n",

" return\_state=True,\n",

")\n",

"\n",

"encoder\_outputs, h, c = encoder(x)\n",

"\n",

"# keep only the states to pass into decoder\n",

"encoder\_states = [h, c]\n"

],

"execution\_count": null,

"outputs": []

},

{

"cell\_type": "markdown",

"metadata": {

"id": "JtCZZ4FZoKDk"

},

"source": [

"### Decoder"

]

},

{

"cell\_type": "code",

"metadata": {

"id": "4x1\_ORCU2RQl"

},

"source": [

"# Set up the decoder, using [h, c] as initial state.\n",

"decoder\_inputs\_placeholder = Input(shape=(max\_len\_target,))\n",

"\n",

"# this word embedding will not use pre-trained vectors\n",

"# although you could\n",

"decoder\_embedding = Embedding(num\_words\_output, LATENT\_DIM)\n",

"\n",

"decoder\_inputs\_x = decoder\_embedding(decoder\_inputs\_placeholder)\n",

"\n",

"# since the decoder is a \"to-many\" model we want to have\n",

"# return\_sequences=True\n",

"decoder\_lstm = LSTM(\n",

" LATENT\_DIM,\n",

" return\_sequences=True,\n",

" return\_state=True,\n",

")\n",

"\n",

"decoder\_outputs, \_, \_ = decoder\_lstm(\n",

" decoder\_inputs\_x,\n",

" initial\_state=encoder\_states\n",

")\n",

"\n",

"# final dense layer for predictions\n",

"decoder\_dense = Dense(num\_words\_output, activation='softmax')\n",

"decoder\_outputs = decoder\_dense(decoder\_outputs)\n",

"\n",

"# Create the model object\n",

"model = Model([encoder\_inputs\_placeholder, decoder\_inputs\_placeholder], decoder\_outputs)"

],

"execution\_count": null,

"outputs": []

},

{

"cell\_type": "code",

"metadata": {

"colab": {

"base\_uri": "https://localhost:8080/"

},

"id": "nuPfdD9wVcVx",

"outputId": "b166a9ad-8c2b-4156-a4f5-6e09f096f5b6"

},

"source": [

"model.summary()"

],

"execution\_count": null,

"outputs": [

{

"output\_type": "stream",

"text": [

"Model: \"model\"\n",

"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n",

"Layer (type) Output Shape Param # Connected to \n",

"==================================================================================================\n",

"input\_1 (InputLayer) [(None, 5)] 0 \n",

"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n",

"input\_2 (InputLayer) [(None, 9)] 0 \n",

"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n",

"embedding (Embedding) (None, 5, 50) 116900 input\_1[0][0] \n",

"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n",

"embedding\_1 (Embedding) (None, 9, 256) 1617152 input\_2[0][0] \n",

"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n",

"lstm (LSTM) [(None, 256), (None, 314368 embedding[0][0] \n",

"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n",

"lstm\_1 (LSTM) [(None, 9, 256), (No 525312 embedding\_1[0][0] \n",

" lstm[0][1] \n",

" lstm[0][2] \n",

"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n",

"dense (Dense) (None, 9, 6317) 1623469 lstm\_1[0][0] \n",

"==================================================================================================\n",

"Total params: 4,197,201\n",

"Trainable params: 4,197,201\n",

"Non-trainable params: 0\n",

"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n"

],

"name": "stdout"

}

]

},

{

"cell\_type": "markdown",

"metadata": {

"id": "sw420uIPoZnG"

},

"source": [

"## Model Compiling"

]

},

{

"cell\_type": "code",

"metadata": {

"id": "tUQwiyZOAQjb"

},

"source": [

"# Compile the model and train it\n",

"model.compile(\n",

" optimizer='rmsprop',\n",

" loss='categorical\_crossentropy',\n",

" metrics=['accuracy']\n",

")"

],

"execution\_count": null,

"outputs": []

},

{

"cell\_type": "markdown",

"metadata": {

"id": "HeTdTf17ock\_"

},

"source": [

"## Model Fitting"

]

},

{

"cell\_type": "code",

"metadata": {

"id": "j9oR7fA4ASw3",

"colab": {

"base\_uri": "https://localhost:8080/"

},

"outputId": "6d3fa079-6612-4a02-d685-79b22c3ec52a"

},

"source": [

"r = model.fit(\n",

" [encoder\_inputs, decoder\_inputs], decoder\_targets\_one\_hot,\n",

" batch\_size=BATCH\_SIZE,\n",

" epochs=5, #EPOCHS,\n",

" validation\_split=0.2,\n",

")"

],

"execution\_count": null,

"outputs": [

{

"output\_type": "stream",

"text": [

"Epoch 1/5\n",

"125/125 [==============================] - 13s 33ms/step - loss: 3.4189 - accuracy: 0.6051 - val\_loss: 2.5863 - val\_accuracy: 0.6503\n",

"Epoch 2/5\n",

"125/125 [==============================] - 3s 26ms/step - loss: 2.0379 - accuracy: 0.7127 - val\_loss: 2.4423 - val\_accuracy: 0.6698\n",

"Epoch 3/5\n",

"125/125 [==============================] - 3s 26ms/step - loss: 1.8519 - accuracy: 0.7267 - val\_loss: 2.2760 - val\_accuracy: 0.6866\n",

"Epoch 4/5\n",

"125/125 [==============================] - 3s 26ms/step - loss: 1.6913 - accuracy: 0.7443 - val\_loss: 2.1812 - val\_accuracy: 0.7003\n",

"Epoch 5/5\n",

"125/125 [==============================] - 3s 26ms/step - loss: 1.5495 - accuracy: 0.7627 - val\_loss: 2.1197 - val\_accuracy: 0.7108\n"

],

"name": "stdout"

}

]

},

{

"cell\_type": "markdown",

"metadata": {

"id": "Gz4Iiffiog33"

},

"source": [

"## Plot Diagnostics"

]

},

{

"cell\_type": "code",

"metadata": {

"id": "5UK13HBjATk4",

"colab": {

"base\_uri": "https://localhost:8080/",

"height": 265

},

"outputId": "97d60608-fda8-4d68-cac9-be9d19765452"

},

"source": [

"# plot some data\n",

"plt.plot(r.history['loss'], label='loss')\n",

"plt.plot(r.history['val\_loss'], label='val\_loss')\n",

"plt.legend()\n",

"plt.show()"

],

"execution\_count": null,

"outputs": [

{

"output\_type": "display\_data",

"data": {

"image/png": "\n",

"text/plain": [

"<Figure size 432x288 with 1 Axes>"

]

},

"metadata": {

"tags": [],

"needs\_background": "light"

}

}

]

},

{

"cell\_type": "code",

"metadata": {

"id": "ZrCkUfCBAaXe",

"colab": {

"base\_uri": "https://localhost:8080/",

"height": 265

},

"outputId": "b4a20c38-320e-4fbd-bf8f-607a4cfddae9"

},

"source": [

"# accuracies\n",

"plt.plot(r.history['accuracy'], label='acc')\n",

"plt.plot(r.history['val\_accuracy'], label='val\_acc')\n",

"plt.legend()\n",

"plt.show()"

],

"execution\_count": null,

"outputs": [

{

"output\_type": "display\_data",

"data": {

"image/png": "\n",

"text/plain": [

"<Figure size 432x288 with 1 Axes>"

]

},

"metadata": {

"tags": [],

"needs\_background": "light"

}

}

]

},

{

"cell\_type": "markdown",

"metadata": {

"id": "VHPNtth2AcqK"

},

"source": [

"## Prediction"

]

},

{

"cell\_type": "markdown",

"metadata": {

"id": "NdJFOMwqozjm"

},

"source": [

"### Encoder"

]

},

{

"cell\_type": "code",

"metadata": {

"id": "xIOs3-ZqAb1a"

},

"source": [

"##### Make predictions #####\n",

"\n",

"# The encoder will be stand-alone\n",

"# From this we will get our initial decoder hidden state\n",

"encoder\_model = Model(encoder\_inputs\_placeholder, encoder\_states)"

],

"execution\_count": null,

"outputs": []

},

{

"cell\_type": "markdown",

"metadata": {

"id": "m6XJhUSfpF-o"

},

"source": [

"### Decoder"

]

},

{

"cell\_type": "code",

"metadata": {

"id": "VQKeWWNRAkP8"

},

"source": [

"decoder\_state\_input\_h = Input(shape=(LATENT\_DIM,))\n",

"decoder\_state\_input\_c = Input(shape=(LATENT\_DIM,))\n",

"\n",

"decoder\_states\_inputs = [decoder\_state\_input\_h, decoder\_state\_input\_c]\n",

"\n",

"decoder\_inputs\_single = Input(shape=(1,))\n",

"\n",

"decoder\_inputs\_single\_x = decoder\_embedding(decoder\_inputs\_single)\n",

"\n",

"decoder\_outputs, h, c = decoder\_lstm(\n",

" decoder\_inputs\_single\_x,\n",

" initial\_state=decoder\_states\_inputs\n",

")\n",

"\n",

"decoder\_states = [h, c]\n",

"\n",

"decoder\_outputs = decoder\_dense(decoder\_outputs)\n",

"\n",

"# The sampling model\n",

"# inputs: y(t-1), h(t-1), c(t-1)\n",

"# outputs: y(t), h(t), c(t)\n",

"decoder\_model = Model(\n",

" [decoder\_inputs\_single] + decoder\_states\_inputs, \n",

" [decoder\_outputs] + decoder\_states\n",

")\n"

],

"execution\_count": null,

"outputs": []

},

{

"cell\_type": "code",

"metadata": {

"id": "WCjzDIwnAl6k"

},

"source": [

"# map indexes back into real words\n",

"# so we can view the results\n",

"idx2word\_eng = {v:k for k, v in word2idx\_inputs.items()}\n",

"idx2word\_trans = {v:k for k, v in word2idx\_outputs.items()}"

],

"execution\_count": null,

"outputs": []

},

{

"cell\_type": "code",

"metadata": {

"id": "bEXM5U6MAm\_p"

},

"source": [

"def decode\_sequence(input\_seq):\n",

" # Encode the input as state vectors.\n",

" states\_value = encoder\_model.predict(input\_seq)\n",

"\n",

" # Generate empty target sequence of length 1.\n",

" target\_seq = np.zeros((1, 1))\n",

"\n",

" # Populate the first character of target sequence with the start character.\n",

" # NOTE: tokenizer lower-cases all words\n",

" target\_seq[0, 0] = word2idx\_outputs['<start>']\n",

"\n",

" # if we get this we break\n",

" eos = word2idx\_outputs['<end>']\n",

"\n",

" # Create the translation\n",

" output\_sentence = []\n",

" for \_ in range(max\_len\_target):\n",

" output\_tokens, h, c = decoder\_model.predict(\n",

" [target\_seq] + states\_value\n",

" )\n",

"\n",

" # Get next word\n",

" idx = np.argmax(output\_tokens[0, 0, :]) \n",

"\n",

" # End sentence of EOS\n",

" if eos == idx:\n",

" break\n",

"\n",

" word = ''\n",

" if idx > 0:\n",

" word = idx2word\_trans[idx]\n",

" output\_sentence.append(word)\n",

"\n",

" # Update the decoder input\n",

" # which is just the word just generated\n",

" target\_seq[0, 0] = idx\n",

"\n",

" # Update states\n",

" states\_value = [h, c]\n",

"\n",

" return ' '.join(output\_sentence)"

],

"execution\_count": null,

"outputs": []

},

{

"cell\_type": "code",

"metadata": {

"id": "JDQ2NkEAAq6k",

"colab": {

"base\_uri": "https://localhost:8080/",

"height": 153

},

"outputId": "c832b5bf-8d7c-46e1-b861-b515e638b74f"

},

"source": [

"while True:\n",

" # Do some test translations\n",

" i = np.random.choice(len(input\_texts))\n",

" input\_seq = encoder\_inputs[i:i+1]\n",

" translation = decode\_sequence(input\_seq)\n",

" print('-')\n",

" print('Input:', input\_texts[i])\n",

" print('Translation:', translation)\n",

"\n",

" ans = input(\"Continue? [Y/n]\")\n",

" if ans and ans.lower().startswith('n'):\n",

" break"

],

"execution\_count": null,

"outputs": [

{

"output\_type": "stream",

"text": [

"-\n",

"Input: Tom rocks.\n",

"Translation: tom se qued√≥ est√° de tom a√±os.\n",

"Continue? [Y/n]Y\n",

"-\n",

"Input: Just leave it.\n",

"Translation: ahora lo nuevo.\n",

"Continue? [Y/n]n\n"

],

"name": "stdout"

}

]

}

]

}