Sequence to sequence implementation

There will be some functions that start with the word "grader" ex: grader_check_encoder(), grader_check_attention(), grader_onestepdecoder() etc, you should not change those function definition.

Every Grader function has to return True.

Note 1: There are many blogs on the attention mechanisum which might be misleading you, so do read the references completly and after that only please check the internet. The best things is to read the research papers and try to implement it on your own.

Note 2: To complete this assignment, the reference that are mentioned will be enough.

Note 3: If you are starting this assignment, you might have completed minimum of 20 assignment. If you are still not able to implement this algorithm you might have rushed in the previous assignments with out learning much and didn't spend your time productively.

Task -1: Simple Encoder and Decoder

Implement simple Encoder-Decoder model

1. Download the **Italian** to **English** translation dataset from [here (http://www.manythings.org/anki/ita-eng.zip)](http://www.manythings.org/anki/ita-eng.zip)
2. You will find **ita.txt** file in that ZIP, you can read that data using python and preprocess that data this way only:

```
Encoder input: "<start> vado a scuola <end>"
Decoder input: "<start> i am going school"
Decoder output: "i am going school <end>"
```

3. You have to implement a simple Encoder and Decoder architecture
4. Use BLEU score as metric to evaluate your model. You can use any loss function you need.
5. You have to use Tensorboard to plot the Graph, Scores and histograms of gradients.
6. a. Check the reference notebook
   b. [Resource 2 (https://medium.com/analytics-vidhya/understand-sequence-to-sequence-models-in-a-more-intuitive-way-1d517d8795bb)](https://medium.com/analytics-vidhya/understand-sequence-to-sequence-models-in-a-more-intuitive-way-1d517d8795bb)

In [ ]:
```
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
%matplotlib inline
import seaborn as sns
import pandas as pd
import re
import datetime
import tensorflow as tf
from tensorflow.keras.layers import Embedding, LSTM, Dense, TimeDistributed, Conc
atenate, dot
from tensorflow.keras.models import Model
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.callbacks import Callback, TensorBoard, EarlyStopping, Lear
ningRateScheduler, ReduceLROnPlateau, ModelCheckpoint
import numpy as np
```

In [ ]:
```
!wget http://www.manythings.org/anki/ita-eng.zip
!unzip ita-eng.zip
```

```
--2021-01-15 00:38:10--  http://www.manythings.org/anki/ita-eng.zip
Resolving www.manythings.org (www.manythings.org)... 104.21.55.222, 172.67.173.1
98, 2606:4700:3031::6815:37de, ...
Connecting to www.manythings.org (www.manythings.org)|104.21.55.222|:80... conne
cted.
HTTP request sent, awaiting response... 200 OK
Length: 7521114 (7.2M) [application/zip]
Saving to: 'ita-eng.zip'

ita-eng.zip         100%[===================>]   7.17M  10.4MB/s    in 0.7s

2021-01-15 00:38:11 (10.4 MB/s) - 'ita-eng.zip' saved [7521114/7521114]

Archive:  ita-eng.zip
  inflating: ita.txt
  inflating: _about.txt
```

Load the data

In [ ]:
```
with open('./ita.txt', 'r', encoding="utf8") as f:
    eng=[]
    ita=[]
    for i in f.readlines():
        eng.append(i.split("\t")[0])
        ita.append(i.split("\t")[1])


data = pd.DataFrame(data=list(zip(eng, ita)), columns=['english','italian'])
print(data.shape)
data
```

```
(343813, 2)
```

Out[ ]:

	english	italian
0	Hi.	Ciao!
1	Run!	Corri!
2	Run!	Corra!
3	Run!	Correte!
4	Who?	Chi?
...	...	...
343808	If you want to sound like a native speaker, yo...	Se vuoi sembrare un madrelingua, devi essere d...
343809	If you want to sound like a native speaker, yo...	Se vuoi sembrare un madrelingua, devi essere d...
343810	If someone who doesn't know your background sa...	Se qualcuno che non conosce il tuo background ...
343811	Doubtless there exists in this world precisely...	Senza dubbio esiste in questo mondo proprio la...
343812	Doubtless there exists in this world precisely...	Senza dubbio esiste in questo mondo proprio la...

343813 rows × 2 columns

Preprocess data

In [ ]:
```python
def decontractions(phrase):
    """decontracted takes text and convert contractions into natural form.
     ref: https://stackoverflow.com/questions/19790188/expanding-english-language
-contractions-in-python/47091490#47091490"""
    # specific
    phrase = re.sub(r"won\'t", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)
    phrase = re.sub(r"won\’t", "will not", phrase)
    phrase = re.sub(r"can\’t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)

    phrase = re.sub(r"n\’t", " not", phrase)
    phrase = re.sub(r"\’re", " are", phrase)
    phrase = re.sub(r"\’s", " is", phrase)
    phrase = re.sub(r"\’d", " would", phrase)
    phrase = re.sub(r"\’ll", " will", phrase)
    phrase = re.sub(r"\’t", " not", phrase)
    phrase = re.sub(r"\’ve", " have", phrase)
    phrase = re.sub(r"\’m", " am", phrase)

    return phrase

def preprocess(text):
    # convert all the text into lower letters
    # use this function to remove the contractions: https://gist.github.com/anand
borad/d410a49a493b56dace4f814ab5325bbd
    # remove all the spacial characters: except space ' '
    text = text.lower()
    text = decontractions(text)
    text = re.sub('[^A-Za-z0-9 ]+', '', text)
    return text

def preprocess_ita(text):
    # convert all the text into lower letters
    # remove the words betweent brakets ()
    # remove these characters: {'$', ')', '?', '"', '’', '.',  '°', '!', ';', '/',
"'", '€', '%', ':', ',', '('}
    # replace these spl characters with space: '\u200b', '\xa0', '-', '/'
    # we have found these characters after observing the data points, feel free t
o explore more and see if you can do find more
    # you are free to do more proprocessing
    # note that the model will learn better with better preprocessed data

    text = text.lower()
    text = decontractions(text)
    text = re.sub('[$)\?"’.°!;\'€%:,(/]', '', text)
    text = re.sub('\u200b', ' ', text)
    text = re.sub('\xa0', ' ', text)
    text = re.sub('-', ' ', text)
    return text


data['english'] = data['english'].apply(preprocess)
data['italian'] = data['italian'].apply(preprocess_ita)

data['italian_len'] = data['italian'].str.split().apply(len)
data = data[data['italian_len'] < 20]

data['english_len'] = data['english'].str.split().apply(len)
data = data[data['english_len'] < 20]
```

```
data.head()
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:64: SettingWithCopy
Warning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy
```

Out[ ]:

	english	italian	italian_len	english_len
0	hi	ciao	1	1
1	run	corri	1	1
2	run	corra	1	1
3	run	correte	1	1
4	who	chi	1	1

In [ ]:
```
data['italia_enc'] = '<start> '+data['italian']+' <end>'
```

In [ ]:
```
data['english_in'] = '<start> ' + data['english']
data['english_op'] = data['english'] + ' <end>'
```

In [ ]:
```
data = data.drop(['english','italian'], axis = 1)
data = data.drop(['italian_len','english_len'], axis = 1)
```

In [ ]:
```
data
```

Out[ ]:

	italia_enc	english_in	english_op
0	<start> ciao <end>	<start> hi	hi <end>
1	<start> corri <end>	<start> run	run <end>
2	<start> corra <end>	<start> run	run <end>
3	<start> correte <end>	<start> run	run <end>
4	<start> chi <end>	<start> who	who <end>
...	...	...	...
343471	<start> lorganizzazione mondiale della sanità ...	<start> the world health organization says alc...	the world health organization says alcohol abu...
343478	<start> è incline a guardare tutto dal punto d...	<start> he is inclined to look at everything f...	he is inclined to look at everything from the ...
343484	<start> charles moore ha creato il forth nel t...	<start> charles moore created forth in an atte...	charles moore created forth in an attempt to i...
343485	<start> charles moore creò il forth nel tentat...	<start> charles moore created forth in an atte...	charles moore created forth in an attempt to i...
343571	<start> lintelligenza è fondata nella capacità...	<start> intelligence is found in the capacity ...	intelligence is found in the capacity to recog...

343388 rows × 3 columns

In [ ]:
```
from sklearn.model_selection import train_test_split
train, validation = train_test_split(data, test_size=0.1)
```

In [ ]:
```
print(train.shape)
train
```

```
(309049, 3)
```

Out[ ]:

	italia_enc	english_in	english_op
256422	<start> che lingua stanno parlando <end>	<start> what language are they speaking	what language are they speaking <end>
340044	<start> vorremmo visitare nikko durante queste...	<start> we would like to visit nikko during th...	we would like to visit nikko during this summe...
50514	<start> dovremmo farlo <end>	<start> should we do that	should we do that <end>
88344	<start> ho preso lautobus sbagliato <end>	<start> i took the wrong bus	i took the wrong bus <end>
221761	<start> hai mai sentito parlare di nessie <end>	<start> have you ever heard of nessie	have you ever heard of nessie <end>
...	...	...	...
182816	<start> si prendeva cura della bambina <end>	<start> she took care of the child	she took care of the child <end>
324318	<start> lui è andato a pescare in un fiume vic...	<start> he went fishing in a river near the vi...	he went fishing in a river near the village <end>
17525	<start> controllate tutti <end>	<start> check everyone	check everyone <end>
161667	<start> deve parlare inglese <end>	<start> he needs to speak english	he needs to speak english <end>
315039	<start> io non so niente di questo progetto <end>	<start> i do not know anything about this project	i do not know anything about this project <end>

309049 rows × 3 columns

In [ ]:
```
print(validation.shape)
validation.head()
```

```
(34339, 3)
```

Out[ ]:

	italia_enc	english_in	english_op
258005	<start> il tuo inglese è migliorato molto <end>	<start> your english has improved a lot	your english has improved a lot <end>
132030	<start> mi sento male per quel tizio <end>	<start> i feel bad for that guy	i feel bad for that guy <end>
186107	<start> tom disse che non lo avrebbe fatto <end>	<start> tom said he would not do it	tom said he would not do it <end>
253635	<start> questo non ha nulla a che fare con tom...	<start> this has nothing to do with tom	this has nothing to do with tom <end>
231767	<start> quale materia vi piace di più <end>	<start> what subject do you like best	what subject do you like best <end>

In [ ]:
```
tokenizer_eng  = Tokenizer(filters='!"#$%&()*+,-./:;=?@[\\]^_`{|}~\t\n', oov_toke
n='UNK')
tokenizer_eng.fit_on_texts(train['english_in'].values)

tokenizer_ita = Tokenizer(filters='!"#$%&()*+,-./:;=?@[\\]^_`{|}~\t\n', oov_token
='UNK')
tokenizer_ita.fit_on_texts(train['italia_enc'].values)
```

In [ ]:
```
tokenizer_eng.word_index['<end>'] = max(tokenizer_eng.word_index.values())+1
```

In [ ]:
```
max(tokenizer_eng.word_index.values())
```

Out[ ]:
```
13127
```

In [ ]:
```
print(tokenizer_eng.index_word.keys())
```

```
dict_keys([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 2
0, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 4
0, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 6
0, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 8
0, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 1
00, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 1
16, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 1
32, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 1
48, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 1
64, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 1
80, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 1
96, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 2
12, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 2
28, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 2
44, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 2
60, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 2
76, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 2
92, 293, 294, 295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 305, 306, 307, 3
08, 309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 3
24, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 338, 339, 3
40, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 3
56, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 3
72, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 3
88, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400, 401, 402, 403, 4
04, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419, 4
20, 421, 422, 423, 424, 425, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435, 4
36, 437, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 4
52, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467, 4
68, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480, 481, 482, 483, 4
84, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 5
00, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510, 511, 512, 513, 514, 515, 5
16, 517, 518, 519, 520, 521, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 5
32, 533, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545, 546, 547, 5
48, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558, 559, 560, 561, 562, 563, 5
64, 565, 566, 567, 568, 569, 570, 571, 572, 573, 574, 575, 576, 577, 578, 579, 5
80, 581, 582, 583, 584, 585, 586, 587, 588, 589, 590, 591, 592, 593, 594, 595, 5
96, 597, 598, 599, 600, 601, 602, 603, 604, 605, 606, 607, 608, 609, 610, 611, 6
12, 613, 614, 615, 616, 617, 618, 619, 620, 621, 622, 623, 624, 625, 626, 627, 6
28, 629, 630, 631, 632, 633, 634, 635, 636, 637, 638, 639, 640, 641, 642, 643, 6
44, 645, 646, 647, 648, 649, 650, 651, 652, 653, 654, 655, 656, 657, 658, 659, 6
60, 661, 662, 663, 664, 665, 666, 667, 668, 669, 670, 671, 672, 673, 674, 675, 6
76, 677, 678, 679, 680, 681, 682, 683, 684, 685, 686, 687, 688, 689, 690, 691, 6
92, 693, 694, 695, 696, 697, 698, 699, 700, 701, 702, 703, 704, 705, 706, 707, 7
08, 709, 710, 711, 712, 713, 714, 715, 716, 717, 718, 719, 720, 721, 722, 723, 7
24, 725, 726, 727, 728, 729, 730, 731, 732, 733, 734, 735, 736, 737, 738, 739, 7
40, 741, 742, 743, 744, 745, 746, 747, 748, 749, 750, 751, 752, 753, 754, 755, 7
56, 757, 758, 759, 760, 761, 762, 763, 764, 765, 766, 767, 768, 769, 770, 771, 7
72, 773, 774, 775, 776, 777, 778, 779, 780, 781, 782, 783, 784, 785, 786, 787, 7
88, 789, 790, 791, 792, 793, 794, 795, 796, 797, 798, 799, 800, 801, 802, 803, 8
04, 805, 806, 807, 808, 809, 810, 811, 812, 813, 814, 815, 816, 817, 818, 819, 8
20, 821, 822, 823, 824, 825, 826, 827, 828, 829, 830, 831, 832, 833, 834, 835, 8
36, 837, 838, 839, 840, 841, 842, 843, 844, 845, 846, 847, 848, 849, 850, 851, 8
52, 853, 854, 855, 856, 857, 858, 859, 860, 861, 862, 863, 864, 865, 866, 867, 8
68, 869, 870, 871, 872, 873, 874, 875, 876, 877, 878, 879, 880, 881, 882, 883, 8
84, 885, 886, 887, 888, 889, 890, 891, 892, 893, 894, 895, 896, 897, 898, 899, 9
00, 901, 902, 903, 904, 905, 906, 907, 908, 909, 910, 911, 912, 913, 914, 915, 9
16, 917, 918, 919, 920, 921, 922, 923, 924, 925, 926, 927, 928, 929, 930, 931, 9
32, 933, 934, 935, 936, 937, 938, 939, 940, 941, 942, 943, 944, 945, 946, 947, 9
48, 949, 950, 951, 952, 953, 954, 955, 956, 957, 958, 959, 960, 961, 962, 963, 9
64, 965, 966, 967, 968, 969, 970, 971, 972, 973, 974, 975, 976, 977, 978, 979, 9
80, 981, 982, 983, 984, 985, 986, 987, 988, 989, 990, 991, 992, 993, 994, 995, 9
96, 997, 998, 999, 1000, 1001, 1002, 1003, 1004, 1005, 1006, 1007, 1008, 1009, 1
010, 1011, 1012, 1013, 1014, 1015, 1016, 1017, 1018, 1019, 1020, 1021, 1022, 102
3, 1024, 1025, 1026, 1027, 1028, 1029, 1030, 1031, 1032, 1033, 1034, 1035, 1036,
1037, 1038, 1039, 1040, 1041, 1042, 1043, 1044, 1045, 1046, 1047, 1048, 1049, 10
50, 1051, 1052, 1053, 1054, 1055, 1056, 1057, 1058, 1059, 1060, 1061, 1062, 106
3, 1064, 1065, 1066, 1067, 1068, 1069, 1070, 1071, 1072, 1073, 1074, 1075, 1076,
1077, 1078, 1079, 1080, 1081, 1082, 1083, 1084, 1085, 1086, 1087, 1088, 1089, 10
90, 1091, 1092, 1093, 1094, 1095, 1096, 1097, 1098, 1099, 1100, 1101, 1102, 110
```

```
3, 1104, 1105, 1106, 1107, 1108, 1109, 1110, 1111, 1112, 1113, 1114, 1115, 1116,
1117, 1118, 1119, 1120, 1121, 1122, 1123, 1124, 1125, 1126, 1127, 1128, 1129, 11
30, 1131, 1132, 1133, 1134, 1135, 1136, 1137, 1138, 1139, 1140, 1141, 1142, 114
3, 1144, 1145, 1146, 1147, 1148, 1149, 1150, 1151, 1152, 1153, 1154, 1155, 1156,
1157, 1158, 1159, 1160, 1161, 1162, 1163, 1164, 1165, 1166, 1167, 1168, 1169, 11
70, 1171, 1172, 1173, 1174, 1175, 1176, 1177, 1178, 1179, 1180, 1181, 1182, 118
3, 1184, 1185, 1186, 1187, 1188, 1189, 1190, 1191, 1192, 1193, 1194, 1195, 1196,
1197, 1198, 1199, 1200, 1201, 1202, 1203, 1204, 1205, 1206, 1207, 1208, 1209, 12
10, 1211, 1212, 1213, 1214, 1215, 1216, 1217, 1218, 1219, 1220, 1221, 1222, 122
3, 1224, 1225, 1226, 1227, 1228, 1229, 1230, 1231, 1232, 1233, 1234, 1235, 1236,
1237, 1238, 1239, 1240, 1241, 1242, 1243, 1244, 1245, 1246, 1247, 1248, 1249, 12
50, 1251, 1252, 1253, 1254, 1255, 1256, 1257, 1258, 1259, 1260, 1261, 1262, 126
3, 1264, 1265, 1266, 1267, 1268, 1269, 1270, 1271, 1272, 1273, 1274, 1275, 1276,
1277, 1278, 1279, 1280, 1281, 1282, 1283, 1284, 1285, 1286, 1287, 1288, 1289, 12
90, 1291, 1292, 1293, 1294, 1295, 1296, 1297, 1298, 1299, 1300, 1301, 1302, 130
3, 1304, 1305, 1306, 1307, 1308, 1309, 1310, 1311, 1312, 1313, 1314, 1315, 1316,
1317, 1318, 1319, 1320, 1321, 1322, 1323, 1324, 1325, 1326, 1327, 1328, 1329, 13
30, 1331, 1332, 1333, 1334, 1335, 1336, 1337, 1338, 1339, 1340, 1341, 1342, 134
3, 1344, 1345, 1346, 1347, 1348, 1349, 1350, 1351, 1352, 1353, 1354, 1355, 1356,
1357, 1358, 1359, 1360, 1361, 1362, 1363, 1364, 1365, 1366, 1367, 1368, 1369, 13
70, 1371, 1372, 1373, 1374, 1375, 1376, 1377, 1378, 1379, 1380, 1381, 1382, 138
3, 1384, 1385, 1386, 1387, 1388, 1389, 1390, 1391, 1392, 1393, 1394, 1395, 1396,
1397, 1398, 1399, 1400, 1401, 1402, 1403, 1404, 1405, 1406, 1407, 1408, 1409, 14
10, 1411, 1412, 1413, 1414, 1415, 1416, 1417, 1418, 1419, 1420, 1421, 1422, 142
3, 1424, 1425, 1426, 1427, 1428, 1429, 1430, 1431, 1432, 1433, 1434, 1435, 1436,
1437, 1438, 1439, 1440, 1441, 1442, 1443, 1444, 1445, 1446, 1447, 1448, 1449, 14
50, 1451, 1452, 1453, 1454, 1455, 1456, 1457, 1458, 1459, 1460, 1461, 1462, 146
3, 1464, 1465, 1466, 1467, 1468, 1469, 1470, 1471, 1472, 1473, 1474, 1475, 1476,
1477, 1478, 1479, 1480, 1481, 1482, 1483, 1484, 1485, 1486, 1487, 1488, 1489, 14
90, 1491, 1492, 1493, 1494, 1495, 1496, 1497, 1498, 1499, 1500, 1501, 1502, 150
3, 1504, 1505, 1506, 1507, 1508, 1509, 1510, 1511, 1512, 1513, 1514, 1515, 1516,
1517, 1518, 1519, 1520, 1521, 1522, 1523, 1524, 1525, 1526, 1527, 1528, 1529, 15
30, 1531, 1532, 1533, 1534, 1535, 1536, 1537, 1538, 1539, 1540, 1541, 1542, 154
3, 1544, 1545, 1546, 1547, 1548, 1549, 1550, 1551, 1552, 1553, 1554, 1555, 1556,
1557, 1558, 1559, 1560, 1561, 1562, 1563, 1564, 1565, 1566, 1567, 1568, 1569, 15
70, 1571, 1572, 1573, 1574, 1575, 1576, 1577, 1578, 1579, 1580, 1581, 1582, 158
3, 1584, 1585, 1586, 1587, 1588, 1589, 1590, 1591, 1592, 1593, 1594, 1595, 1596,
1597, 1598, 1599, 1600, 1601, 1602, 1603, 1604, 1605, 1606, 1607, 1608, 1609, 16
10, 1611, 1612, 1613, 1614, 1615, 1616, 1617, 1618, 1619, 1620, 1621, 1622, 162
3, 1624, 1625, 1626, 1627, 1628, 1629, 1630, 1631, 1632, 1633, 1634, 1635, 1636,
1637, 1638, 1639, 1640, 1641, 1642, 1643, 1644, 1645, 1646, 1647, 1648, 1649, 16
50, 1651, 1652, 1653, 1654, 1655, 1656, 1657, 1658, 1659, 1660, 1661, 1662, 166
3, 1664, 1665, 1666, 1667, 1668, 1669, 1670, 1671, 1672, 1673, 1674, 1675, 1676,
1677, 1678, 1679, 1680, 1681, 1682, 1683, 1684, 1685, 1686, 1687, 1688, 1689, 16
90, 1691, 1692, 1693, 1694, 1695, 1696, 1697, 1698, 1699, 1700, 1701, 1702, 170
3, 1704, 1705, 1706, 1707, 1708, 1709, 1710, 1711, 1712, 1713, 1714, 1715, 1716,
1717, 1718, 1719, 1720, 1721, 1722, 1723, 1724, 1725, 1726, 1727, 1728, 1729, 17
30, 1731, 1732, 1733, 1734, 1735, 1736, 1737, 1738, 1739, 1740, 1741, 1742, 174
3, 1744, 1745, 1746, 1747, 1748, 1749, 1750, 1751, 1752, 1753, 1754, 1755, 1756,
1757, 1758, 1759, 1760, 1761, 1762, 1763, 1764, 1765, 1766, 1767, 1768, 1769, 17
70, 1771, 1772, 1773, 1774, 1775, 1776, 1777, 1778, 1779, 1780, 1781, 1782, 178
3, 1784, 1785, 1786, 1787, 1788, 1789, 1790, 1791, 1792, 1793, 1794, 1795, 1796,
1797, 1798, 1799, 1800, 1801, 1802, 1803, 1804, 1805, 1806, 1807, 1808, 1809, 18
10, 1811, 1812, 1813, 1814, 1815, 1816, 1817, 1818, 1819, 1820, 1821, 1822, 182
3, 1824, 1825, 1826, 1827, 1828, 1829, 1830, 1831, 1832, 1833, 1834, 1835, 1836,
1837, 1838, 1839, 1840, 1841, 1842, 1843, 1844, 1845, 1846, 1847, 1848, 1849, 18
50, 1851, 1852, 1853, 1854, 1855, 1856, 1857, 1858, 1859, 1860, 1861, 1862, 186
3, 1864, 1865, 1866, 1867, 1868, 1869, 1870, 1871, 1872, 1873, 1874, 1875, 1876,
1877, 1878, 1879, 1880, 1881, 1882, 1883, 1884, 1885, 1886, 1887, 1888, 1889, 18
90, 1891, 1892, 1893, 1894, 1895, 1896, 1897, 1898, 1899, 1900, 1901, 1902, 190
3, 1904, 1905, 1906, 1907, 1908, 1909, 1910, 1911, 1912, 1913, 1914, 1915, 1916,
1917, 1918, 1919, 1920, 1921, 1922, 1923, 1924, 1925, 1926, 1927, 1928, 1929, 19
30, 1931, 1932, 1933, 1934, 1935, 1936, 1937, 1938, 1939, 1940, 1941, 1942, 194
3, 1944, 1945, 1946, 1947, 1948, 1949, 1950, 1951, 1952, 1953, 1954, 1955, 1956,
1957, 1958, 1959, 1960, 1961, 1962, 1963, 1964, 1965, 1966, 1967, 1968, 1969, 19
70, 1971, 1972, 1973, 1974, 1975, 1976, 1977, 1978, 1979, 1980, 1981, 1982, 198
3, 1984, 1985, 1986, 1987, 1988, 1989, 1990, 1991, 1992, 1993, 1994, 1995, 1996,
1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 20
10, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022, 202
3, 2024, 2025, 2026, 2027, 2028, 2029, 2030, 2031, 2032, 2033, 2034, 2035, 2036,
```

```
2037, 2038, 2039, 2040, 2041, 2042, 2043, 2044, 2045, 2046, 2047, 2048, 2049, 20
50, 2051, 2052, 2053, 2054, 2055, 2056, 2057, 2058, 2059, 2060, 2061, 2062, 206
3, 2064, 2065, 2066, 2067, 2068, 2069, 2070, 2071, 2072, 2073, 2074, 2075, 2076,
2077, 2078, 2079, 2080, 2081, 2082, 2083, 2084, 2085, 2086, 2087, 2088, 2089, 20
90, 2091, 2092, 2093, 2094, 2095, 2096, 2097, 2098, 2099, 2100, 2101, 2102, 210
3, 2104, 2105, 2106, 2107, 2108, 2109, 2110, 2111, 2112, 2113, 2114, 2115, 2116,
2117, 2118, 2119, 2120, 2121, 2122, 2123, 2124, 2125, 2126, 2127, 2128, 2129, 21
30, 2131, 2132, 2133, 2134, 2135, 2136, 2137, 2138, 2139, 2140, 2141, 2142, 214
3, 2144, 2145, 2146, 2147, 2148, 2149, 2150, 2151, 2152, 2153, 2154, 2155, 2156,
2157, 2158, 2159, 2160, 2161, 2162, 2163, 2164, 2165, 2166, 2167, 2168, 2169, 21
70, 2171, 2172, 2173, 2174, 2175, 2176, 2177, 2178, 2179, 2180, 2181, 2182, 218
3, 2184, 2185, 2186, 2187, 2188, 2189, 2190, 2191, 2192, 2193, 2194, 2195, 2196,
2197, 2198, 2199, 2200, 2201, 2202, 2203, 2204, 2205, 2206, 2207, 2208, 2209, 22
10, 2211, 2212, 2213, 2214, 2215, 2216, 2217, 2218, 2219, 2220, 2221, 2222, 222
3, 2224, 2225, 2226, 2227, 2228, 2229, 2230, 2231, 2232, 2233, 2234, 2235, 2236,
2237, 2238, 2239, 2240, 2241, 2242, 2243, 2244, 2245, 2246, 2247, 2248, 2249, 22
50, 2251, 2252, 2253, 2254, 2255, 2256, 2257, 2258, 2259, 2260, 2261, 2262, 226
3, 2264, 2265, 2266, 2267, 2268, 2269, 2270, 2271, 2272, 2273, 2274, 2275, 2276,
2277, 2278, 2279, 2280, 2281, 2282, 2283, 2284, 2285, 2286, 2287, 2288, 2289, 22
90, 2291, 2292, 2293, 2294, 2295, 2296, 2297, 2298, 2299, 2300, 2301, 2302, 230
3, 2304, 2305, 2306, 2307, 2308, 2309, 2310, 2311, 2312, 2313, 2314, 2315, 2316,
2317, 2318, 2319, 2320, 2321, 2322, 2323, 2324, 2325, 2326, 2327, 2328, 2329, 23
30, 2331, 2332, 2333, 2334, 2335, 2336, 2337, 2338, 2339, 2340, 2341, 2342, 234
3, 2344, 2345, 2346, 2347, 2348, 2349, 2350, 2351, 2352, 2353, 2354, 2355, 2356,
2357, 2358, 2359, 2360, 2361, 2362, 2363, 2364, 2365, 2366, 2367, 2368, 2369, 23
70, 2371, 2372, 2373, 2374, 2375, 2376, 2377, 2378, 2379, 2380, 2381, 2382, 238
3, 2384, 2385, 2386, 2387, 2388, 2389, 2390, 2391, 2392, 2393, 2394, 2395, 2396,
2397, 2398, 2399, 2400, 2401, 2402, 2403, 2404, 2405, 2406, 2407, 2408, 2409, 24
10, 2411, 2412, 2413, 2414, 2415, 2416, 2417, 2418, 2419, 2420, 2421, 2422, 242
3, 2424, 2425, 2426, 2427, 2428, 2429, 2430, 2431, 2432, 2433, 2434, 2435, 2436,
2437, 2438, 2439, 2440, 2441, 2442, 2443, 2444, 2445, 2446, 2447, 2448, 2449, 24
50, 2451, 2452, 2453, 2454, 2455, 2456, 2457, 2458, 2459, 2460, 2461, 2462, 246
3, 2464, 2465, 2466, 2467, 2468, 2469, 2470, 2471, 2472, 2473, 2474, 2475, 2476,
2477, 2478, 2479, 2480, 2481, 2482, 2483, 2484, 2485, 2486, 2487, 2488, 2489, 24
90, 2491, 2492, 2493, 2494, 2495, 2496, 2497, 2498, 2499, 2500, 2501, 2502, 250
3, 2504, 2505, 2506, 2507, 2508, 2509, 2510, 2511, 2512, 2513, 2514, 2515, 2516,
2517, 2518, 2519, 2520, 2521, 2522, 2523, 2524, 2525, 2526, 2527, 2528, 2529, 25
30, 2531, 2532, 2533, 2534, 2535, 2536, 2537, 2538, 2539, 2540, 2541, 2542, 254
3, 2544, 2545, 2546, 2547, 2548, 2549, 2550, 2551, 2552, 2553, 2554, 2555, 2556,
2557, 2558, 2559, 2560, 2561, 2562, 2563, 2564, 2565, 2566, 2567, 2568, 2569, 25
70, 2571, 2572, 2573, 2574, 2575, 2576, 2577, 2578, 2579, 2580, 2581, 2582, 258
3, 2584, 2585, 2586, 2587, 2588, 2589, 2590, 2591, 2592, 2593, 2594, 2595, 2596,
2597, 2598, 2599, 2600, 2601, 2602, 2603, 2604, 2605, 2606, 2607, 2608, 2609, 26
10, 2611, 2612, 2613, 2614, 2615, 2616, 2617, 2618, 2619, 2620, 2621, 2622, 262
3, 2624, 2625, 2626, 2627, 2628, 2629, 2630, 2631, 2632, 2633, 2634, 2635, 2636,
2637, 2638, 2639, 2640, 2641, 2642, 2643, 2644, 2645, 2646, 2647, 2648, 2649, 26
50, 2651, 2652, 2653, 2654, 2655, 2656, 2657, 2658, 2659, 2660, 2661, 2662, 266
3, 2664, 2665, 2666, 2667, 2668, 2669, 2670, 2671, 2672, 2673, 2674, 2675, 2676,
2677, 2678, 2679, 2680, 2681, 2682, 2683, 2684, 2685, 2686, 2687, 2688, 2689, 26
90, 2691, 2692, 2693, 2694, 2695, 2696, 2697, 2698, 2699, 2700, 2701, 2702, 270
3, 2704, 2705, 2706, 2707, 2708, 2709, 2710, 2711, 2712, 2713, 2714, 2715, 2716,
2717, 2718, 2719, 2720, 2721, 2722, 2723, 2724, 2725, 2726, 2727, 2728, 2729, 27
30, 2731, 2732, 2733, 2734, 2735, 2736, 2737, 2738, 2739, 2740, 2741, 2742, 274
3, 2744, 2745, 2746, 2747, 2748, 2749, 2750, 2751, 2752, 2753, 2754, 2755, 2756,
2757, 2758, 2759, 2760, 2761, 2762, 2763, 2764, 2765, 2766, 2767, 2768, 2769, 27
70, 2771, 2772, 2773, 2774, 2775, 2776, 2777, 2778, 2779, 2780, 2781, 2782, 278
3, 2784, 2785, 2786, 2787, 2788, 2789, 2790, 2791, 2792, 2793, 2794, 2795, 2796,
2797, 2798, 2799, 2800, 2801, 2802, 2803, 2804, 2805, 2806, 2807, 2808, 2809, 28
10, 2811, 2812, 2813, 2814, 2815, 2816, 2817, 2818, 2819, 2820, 2821, 2822, 282
3, 2824, 2825, 2826, 2827, 2828, 2829, 2830, 2831, 2832, 2833, 2834, 2835, 2836,
2837, 2838, 2839, 2840, 2841, 2842, 2843, 2844, 2845, 2846, 2847, 2848, 2849, 28
50, 2851, 2852, 2853, 2854, 2855, 2856, 2857, 2858, 2859, 2860, 2861, 2862, 286
3, 2864, 2865, 2866, 2867, 2868, 2869, 2870, 2871, 2872, 2873, 2874, 2875, 2876,
2877, 2878, 2879, 2880, 2881, 2882, 2883, 2884, 2885, 2886, 2887, 2888, 2889, 28
90, 2891, 2892, 2893, 2894, 2895, 2896, 2897, 2898, 2899, 2900, 2901, 2902, 290
3, 2904, 2905, 2906, 2907, 2908, 2909, 2910, 2911, 2912, 2913, 2914, 2915, 2916,
2917, 2918, 2919, 2920, 2921, 2922, 2923, 2924, 2925, 2926, 2927, 2928, 2929, 29
30, 2931, 2932, 2933, 2934, 2935, 2936, 2937, 2938, 2939, 2940, 2941, 2942, 294
3, 2944, 2945, 2946, 2947, 2948, 2949, 2950, 2951, 2952, 2953, 2954, 2955, 2956,
2957, 2958, 2959, 2960, 2961, 2962, 2963, 2964, 2965, 2966, 2967, 2968, 2969, 29
```

```
70, 2971, 2972, 2973, 2974, 2975, 2976, 2977, 2978, 2979, 2980, 2981, 2982, 298
3, 2984, 2985, 2986, 2987, 2988, 2989, 2990, 2991, 2992, 2993, 2994, 2995, 2996,
2997, 2998, 2999, 3000, 3001, 3002, 3003, 3004, 3005, 3006, 3007, 3008, 3009, 30
10, 3011, 3012, 3013, 3014, 3015, 3016, 3017, 3018, 3019, 3020, 3021, 3022, 302
3, 3024, 3025, 3026, 3027, 3028, 3029, 3030, 3031, 3032, 3033, 3034, 3035, 3036,
3037, 3038, 3039, 3040, 3041, 3042, 3043, 3044, 3045, 3046, 3047, 3048, 3049, 30
50, 3051, 3052, 3053, 3054, 3055, 3056, 3057, 3058, 3059, 3060, 3061, 3062, 306
3, 3064, 3065, 3066, 3067, 3068, 3069, 3070, 3071, 3072, 3073, 3074, 3075, 3076,
3077, 3078, 3079, 3080, 3081, 3082, 3083, 3084, 3085, 3086, 3087, 3088, 3089, 30
90, 3091, 3092, 3093, 3094, 3095, 3096, 3097, 3098, 3099, 3100, 3101, 3102, 310
3, 3104, 3105, 3106, 3107, 3108, 3109, 3110, 3111, 3112, 3113, 3114, 3115, 3116,
3117, 3118, 3119, 3120, 3121, 3122, 3123, 3124, 3125, 3126, 3127, 3128, 3129, 31
30, 3131, 3132, 3133, 3134, 3135, 3136, 3137, 3138, 3139, 3140, 3141, 3142, 314
3, 3144, 3145, 3146, 3147, 3148, 3149, 3150, 3151, 3152, 3153, 3154, 3155, 3156,
3157, 3158, 3159, 3160, 3161, 3162, 3163, 3164, 3165, 3166, 3167, 3168, 3169, 31
70, 3171, 3172, 3173, 3174, 3175, 3176, 3177, 3178, 3179, 3180, 3181, 3182, 318
3, 3184, 3185, 3186, 3187, 3188, 3189, 3190, 3191, 3192, 3193, 3194, 3195, 3196,
3197, 3198, 3199, 3200, 3201, 3202, 3203, 3204, 3205, 3206, 3207, 3208, 3209, 32
10, 3211, 3212, 3213, 3214, 3215, 3216, 3217, 3218, 3219, 3220, 3221, 3222, 322
3, 3224, 3225, 3226, 3227, 3228, 3229, 3230, 3231, 3232, 3233, 3234, 3235, 3236,
3237, 3238, 3239, 3240, 3241, 3242, 3243, 3244, 3245, 3246, 3247, 3248, 3249, 32
50, 3251, 3252, 3253, 3254, 3255, 3256, 3257, 3258, 3259, 3260, 3261, 3262, 326
3, 3264, 3265, 3266, 3267, 3268, 3269, 3270, 3271, 3272, 3273, 3274, 3275, 3276,
3277, 3278, 3279, 3280, 3281, 3282, 3283, 3284, 3285, 3286, 3287, 3288, 3289, 32
90, 3291, 3292, 3293, 3294, 3295, 3296, 3297, 3298, 3299, 3300, 3301, 3302, 330
3, 3304, 3305, 3306, 3307, 3308, 3309, 3310, 3311, 3312, 3313, 3314, 3315, 3316,
3317, 3318, 3319, 3320, 3321, 3322, 3323, 3324, 3325, 3326, 3327, 3328, 3329, 33
30, 3331, 3332, 3333, 3334, 3335, 3336, 3337, 3338, 3339, 3340, 3341, 3342, 334
3, 3344, 3345, 3346, 3347, 3348, 3349, 3350, 3351, 3352, 3353, 3354, 3355, 3356,
3357, 3358, 3359, 3360, 3361, 3362, 3363, 3364, 3365, 3366, 3367, 3368, 3369, 33
70, 3371, 3372, 3373, 3374, 3375, 3376, 3377, 3378, 3379, 3380, 3381, 3382, 338
3, 3384, 3385, 3386, 3387, 3388, 3389, 3390, 3391, 3392, 3393, 3394, 3395, 3396,
3397, 3398, 3399, 3400, 3401, 3402, 3403, 3404, 3405, 3406, 3407, 3408, 3409, 34
10, 3411, 3412, 3413, 3414, 3415, 3416, 3417, 3418, 3419, 3420, 3421, 3422, 342
3, 3424, 3425, 3426, 3427, 3428, 3429, 3430, 3431, 3432, 3433, 3434, 3435, 3436,
3437, 3438, 3439, 3440, 3441, 3442, 3443, 3444, 3445, 3446, 3447, 3448, 3449, 34
50, 3451, 3452, 3453, 3454, 3455, 3456, 3457, 3458, 3459, 3460, 3461, 3462, 346
3, 3464, 3465, 3466, 3467, 3468, 3469, 3470, 3471, 3472, 3473, 3474, 3475, 3476,
3477, 3478, 3479, 3480, 3481, 3482, 3483, 3484, 3485, 3486, 3487, 3488, 3489, 34
90, 3491, 3492, 3493, 3494, 3495, 3496, 3497, 3498, 3499, 3500, 3501, 3502, 350
3, 3504, 3505, 3506, 3507, 3508, 3509, 3510, 3511, 3512, 3513, 3514, 3515, 3516,
3517, 3518, 3519, 3520, 3521, 3522, 3523, 3524, 3525, 3526, 3527, 3528, 3529, 35
30, 3531, 3532, 3533, 3534, 3535, 3536, 3537, 3538, 3539, 3540, 3541, 3542, 354
3, 3544, 3545, 3546, 3547, 3548, 3549, 3550, 3551, 3552, 3553, 3554, 3555, 3556,
3557, 3558, 3559, 3560, 3561, 3562, 3563, 3564, 3565, 3566, 3567, 3568, 3569, 35
70, 3571, 3572, 3573, 3574, 3575, 3576, 3577, 3578, 3579, 3580, 3581, 3582, 358
3, 3584, 3585, 3586, 3587, 3588, 3589, 3590, 3591, 3592, 3593, 3594, 3595, 3596,
3597, 3598, 3599, 3600, 3601, 3602, 3603, 3604, 3605, 3606, 3607, 3608, 3609, 36
10, 3611, 3612, 3613, 3614, 3615, 3616, 3617, 3618, 3619, 3620, 3621, 3622, 362
3, 3624, 3625, 3626, 3627, 3628, 3629, 3630, 3631, 3632, 3633, 3634, 3635, 3636,
3637, 3638, 3639, 3640, 3641, 3642, 3643, 3644, 3645, 3646, 3647, 3648, 3649, 36
50, 3651, 3652, 3653, 3654, 3655, 3656, 3657, 3658, 3659, 3660, 3661, 3662, 366
3, 3664, 3665, 3666, 3667, 3668, 3669, 3670, 3671, 3672, 3673, 3674, 3675, 3676,
3677, 3678, 3679, 3680, 3681, 3682, 3683, 3684, 3685, 3686, 3687, 3688, 3689, 36
90, 3691, 3692, 3693, 3694, 3695, 3696, 3697, 3698, 3699, 3700, 3701, 3702, 370
3, 3704, 3705, 3706, 3707, 3708, 3709, 3710, 3711, 3712, 3713, 3714, 3715, 3716,
3717, 3718, 3719, 3720, 3721, 3722, 3723, 3724, 3725, 3726, 3727, 3728, 3729, 37
30, 3731, 3732, 3733, 3734, 3735, 3736, 3737, 3738, 3739, 3740, 3741, 3742, 374
3, 3744, 3745, 3746, 3747, 3748, 3749, 3750, 3751, 3752, 3753, 3754, 3755, 3756,
3757, 3758, 3759, 3760, 3761, 3762, 3763, 3764, 3765, 3766, 3767, 3768, 3769, 37
70, 3771, 3772, 3773, 3774, 3775, 3776, 3777, 3778, 3779, 3780, 3781, 3782, 378
3, 3784, 3785, 3786, 3787, 3788, 3789, 3790, 3791, 3792, 3793, 3794, 3795, 3796,
3797, 3798, 3799, 3800, 3801, 3802, 3803, 3804, 3805, 3806, 3807, 3808, 3809, 38
10, 3811, 3812, 3813, 3814, 3815, 3816, 3817, 3818, 3819, 3820, 3821, 3822, 382
3, 3824, 3825, 3826, 3827, 3828, 3829, 3830, 3831, 3832, 3833, 3834, 3835, 3836,
3837, 3838, 3839, 3840, 3841, 3842, 3843, 3844, 3845, 3846, 3847, 3848, 3849, 38
50, 3851, 3852, 3853, 3854, 3855, 3856, 3857, 3858, 3859, 3860, 3861, 3862, 386
3, 3864, 3865, 3866, 3867, 3868, 3869, 3870, 3871, 3872, 3873, 3874, 3875, 3876,
3877, 3878, 3879, 3880, 3881, 3882, 3883, 3884, 3885, 3886, 3887, 3888, 3889, 38
90, 3891, 3892, 3893, 3894, 3895, 3896, 3897, 3898, 3899, 3900, 3901, 3902, 390
```

3, 3904, 3905, 3906, 3907, 3908, 3909, 3910, 3911, 3912, 3913, 3914, 3915, 3916,
3917, 3918, 3919, 3920, 3921, 3922, 3923, 3924, 3925, 3926, 3927, 3928, 3929, 39
30, 3931, 3932, 3933, 3934, 3935, 3936, 3937, 3938, 3939, 3940, 3941, 3942, 394
3, 3944, 3945, 3946, 3947, 3948, 3949, 3950, 3951, 3952, 3953, 3954, 3955, 3956,
3957, 3958, 3959, 3960, 3961, 3962, 3963, 3964, 3965, 3966, 3967, 3968, 3969, 39
70, 3971, 3972, 3973, 3974, 3975, 3976, 3977, 3978, 3979, 3980, 3981, 3982, 398
3, 3984, 3985, 3986, 3987, 3988, 3989, 3990, 3991, 3992, 3993, 3994, 3995, 3996,
3997, 3998, 3999, 4000, 4001, 4002, 4003, 4004, 4005, 4006, 4007, 4008, 4009, 40
10, 4011, 4012, 4013, 4014, 4015, 4016, 4017, 4018, 4019, 4020, 4021, 4022, 402
3, 4024, 4025, 4026, 4027, 4028, 4029, 4030, 4031, 4032, 4033, 4034, 4035, 4036,
4037, 4038, 4039, 4040, 4041, 4042, 4043, 4044, 4045, 4046, 4047, 4048, 4049, 40
50, 4051, 4052, 4053, 4054, 4055, 4056, 4057, 4058, 4059, 4060, 4061, 4062, 406
3, 4064, 4065, 4066, 4067, 4068, 4069, 4070, 4071, 4072, 4073, 4074, 4075, 4076,
4077, 4078, 4079, 4080, 4081, 4082, 4083, 4084, 4085, 4086, 4087, 4088, 4089, 40
90, 4091, 4092, 4093, 4094, 4095, 4096, 4097, 4098, 4099, 4100, 4101, 4102, 410
3, 4104, 4105, 4106, 4107, 4108, 4109, 4110, 4111, 4112, 4113, 4114, 4115, 4116,
4117, 4118, 4119, 4120, 4121, 4122, 4123, 4124, 4125, 4126, 4127, 4128, 4129, 41
30, 4131, 4132, 4133, 4134, 4135, 4136, 4137, 4138, 4139, 4140, 4141, 4142, 414
3, 4144, 4145, 4146, 4147, 4148, 4149, 4150, 4151, 4152, 4153, 4154, 4155, 4156,
4157, 4158, 4159, 4160, 4161, 4162, 4163, 4164, 4165, 4166, 4167, 4168, 4169, 41
70, 4171, 4172, 4173, 4174, 4175, 4176, 4177, 4178, 4179, 4180, 4181, 4182, 418
3, 4184, 4185, 4186, 4187, 4188, 4189, 4190, 4191, 4192, 4193, 4194, 4195, 4196,
4197, 4198, 4199, 4200, 4201, 4202, 4203, 4204, 4205, 4206, 4207, 4208, 4209, 42
10, 4211, 4212, 4213, 4214, 4215, 4216, 4217, 4218, 4219, 4220, 4221, 4222, 422
3, 4224, 4225, 4226, 4227, 4228, 4229, 4230, 4231, 4232, 4233, 4234, 4235, 4236,
4237, 4238, 4239, 4240, 4241, 4242, 4243, 4244, 4245, 4246, 4247, 4248, 4249, 42
50, 4251, 4252, 4253, 4254, 4255, 4256, 4257, 4258, 4259, 4260, 4261, 4262, 426
3, 4264, 4265, 4266, 4267, 4268, 4269, 4270, 4271, 4272, 4273, 4274, 4275, 4276,
4277, 4278, 4279, 4280, 4281, 4282, 4283, 4284, 4285, 4286, 4287, 4288, 4289, 42
90, 4291, 4292, 4293, 4294, 4295, 4296, 4297, 4298, 4299, 4300, 4301, 4302, 430
3, 4304, 4305, 4306, 4307, 4308, 4309, 4310, 4311, 4312, 4313, 4314, 4315, 4316,
4317, 4318, 4319, 4320, 4321, 4322, 4323, 4324, 4325, 4326, 4327, 4328, 4329, 43
30, 4331, 4332, 4333, 4334, 4335, 4336, 4337, 4338, 4339, 4340, 4341, 4342, 434
3, 4344, 4345, 4346, 4347, 4348, 4349, 4350, 4351, 4352, 4353, 4354, 4355, 4356,
4357, 4358, 4359, 4360, 4361, 4362, 4363, 4364, 4365, 4366, 4367, 4368, 4369, 43
70, 4371, 4372, 4373, 4374, 4375, 4376, 4377, 4378, 4379, 4380, 4381, 4382, 438
3, 4384, 4385, 4386, 4387, 4388, 4389, 4390, 4391, 4392, 4393, 4394, 4395, 4396,
4397, 4398, 4399, 4400, 4401, 4402, 4403, 4404, 4405, 4406, 4407, 4408, 4409, 44
10, 4411, 4412, 4413, 4414, 4415, 4416, 4417, 4418, 4419, 4420, 4421, 4422, 442
3, 4424, 4425, 4426, 4427, 4428, 4429, 4430, 4431, 4432, 4433, 4434, 4435, 4436,
4437, 4438, 4439, 4440, 4441, 4442, 4443, 4444, 4445, 4446, 4447, 4448, 4449, 44
50, 4451, 4452, 4453, 4454, 4455, 4456, 4457, 4458, 4459, 4460, 4461, 4462, 446
3, 4464, 4465, 4466, 4467, 4468, 4469, 4470, 4471, 4472, 4473, 4474, 4475, 4476,
4477, 4478, 4479, 4480, 4481, 4482, 4483, 4484, 4485, 4486, 4487, 4488, 4489, 44
90, 4491, 4492, 4493, 4494, 4495, 4496, 4497, 4498, 4499, 4500, 4501, 4502, 450
3, 4504, 4505, 4506, 4507, 4508, 4509, 4510, 4511, 4512, 4513, 4514, 4515, 4516,
4517, 4518, 4519, 4520, 4521, 4522, 4523, 4524, 4525, 4526, 4527, 4528, 4529, 45
30, 4531, 4532, 4533, 4534, 4535, 4536, 4537, 4538, 4539, 4540, 4541, 4542, 454
3, 4544, 4545, 4546, 4547, 4548, 4549, 4550, 4551, 4552, 4553, 4554, 4555, 4556,
4557, 4558, 4559, 4560, 4561, 4562, 4563, 4564, 4565, 4566, 4567, 4568, 4569, 45
70, 4571, 4572, 4573, 4574, 4575, 4576, 4577, 4578, 4579, 4580, 4581, 4582, 458
3, 4584, 4585, 4586, 4587, 4588, 4589, 4590, 4591, 4592, 4593, 4594, 4595, 4596,
4597, 4598, 4599, 4600, 4601, 4602, 4603, 4604, 4605, 4606, 4607, 4608, 4609, 46
10, 4611, 4612, 4613, 4614, 4615, 4616, 4617, 4618, 4619, 4620, 4621, 4622, 462
3, 4624, 4625, 4626, 4627, 4628, 4629, 4630, 4631, 4632, 4633, 4634, 4635, 4636,
4637, 4638, 4639, 4640, 4641, 4642, 4643, 4644, 4645, 4646, 4647, 4648, 4649, 46
50, 4651, 4652, 4653, 4654, 4655, 4656, 4657, 4658, 4659, 4660, 4661, 4662, 466
3, 4664, 4665, 4666, 4667, 4668, 4669, 4670, 4671, 4672, 4673, 4674, 4675, 4676,
4677, 4678, 4679, 4680, 4681, 4682, 4683, 4684, 4685, 4686, 4687, 4688, 4689, 46
90, 4691, 4692, 4693, 4694, 4695, 4696, 4697, 4698, 4699, 4700, 4701, 4702, 470
3, 4704, 4705, 4706, 4707, 4708, 4709, 4710, 4711, 4712, 4713, 4714, 4715, 4716,
4717, 4718, 4719, 4720, 4721, 4722, 4723, 4724, 4725, 4726, 4727, 4728, 4729, 47
30, 4731, 4732, 4733, 4734, 4735, 4736, 4737, 4738, 4739, 4740, 4741, 4742, 474
3, 4744, 4745, 4746, 4747, 4748, 4749, 4750, 4751, 4752, 4753, 4754, 4755, 4756,
4757, 4758, 4759, 4760, 4761, 4762, 4763, 4764, 4765, 4766, 4767, 4768, 4769, 47
70, 4771, 4772, 4773, 4774, 4775, 4776, 4777, 4778, 4779, 4780, 4781, 4782, 478
3, 4784, 4785, 4786, 4787, 4788, 4789, 4790, 4791, 4792, 4793, 4794, 4795, 4796,
4797, 4798, 4799, 4800, 4801, 4802, 4803, 4804, 4805, 4806, 4807, 4808, 4809, 48
10, 4811, 4812, 4813, 4814, 4815, 4816, 4817, 4818, 4819, 4820, 4821, 4822, 482
3, 4824, 4825, 4826, 4827, 4828, 4829, 4830, 4831, 4832, 4833, 4834, 4835, 4836,

```
4837, 4838, 4839, 4840, 4841, 4842, 4843, 4844, 4845, 4846, 4847, 4848, 4849, 48
50, 4851, 4852, 4853, 4854, 4855, 4856, 4857, 4858, 4859, 4860, 4861, 4862, 486
3, 4864, 4865, 4866, 4867, 4868, 4869, 4870, 4871, 4872, 4873, 4874, 4875, 4876,
4877, 4878, 4879, 4880, 4881, 4882, 4883, 4884, 4885, 4886, 4887, 4888, 4889, 48
90, 4891, 4892, 4893, 4894, 4895, 4896, 4897, 4898, 4899, 4900, 4901, 4902, 490
3, 4904, 4905, 4906, 4907, 4908, 4909, 4910, 4911, 4912, 4913, 4914, 4915, 4916,
4917, 4918, 4919, 4920, 4921, 4922, 4923, 4924, 4925, 4926, 4927, 4928, 4929, 49
30, 4931, 4932, 4933, 4934, 4935, 4936, 4937, 4938, 4939, 4940, 4941, 4942, 494
3, 4944, 4945, 4946, 4947, 4948, 4949, 4950, 4951, 4952, 4953, 4954, 4955, 4956,
4957, 4958, 4959, 4960, 4961, 4962, 4963, 4964, 4965, 4966, 4967, 4968, 4969, 49
70, 4971, 4972, 4973, 4974, 4975, 4976, 4977, 4978, 4979, 4980, 4981, 4982, 498
3, 4984, 4985, 4986, 4987, 4988, 4989, 4990, 4991, 4992, 4993, 4994, 4995, 4996,
4997, 4998, 4999, 5000, 5001, 5002, 5003, 5004, 5005, 5006, 5007, 5008, 5009, 50
10, 5011, 5012, 5013, 5014, 5015, 5016, 5017, 5018, 5019, 5020, 5021, 5022, 502
3, 5024, 5025, 5026, 5027, 5028, 5029, 5030, 5031, 5032, 5033, 5034, 5035, 5036,
5037, 5038, 5039, 5040, 5041, 5042, 5043, 5044, 5045, 5046, 5047, 5048, 5049, 50
50, 5051, 5052, 5053, 5054, 5055, 5056, 5057, 5058, 5059, 5060, 5061, 5062, 506
3, 5064, 5065, 5066, 5067, 5068, 5069, 5070, 5071, 5072, 5073, 5074, 5075, 5076,
5077, 5078, 5079, 5080, 5081, 5082, 5083, 5084, 5085, 5086, 5087, 5088, 5089, 50
90, 5091, 5092, 5093, 5094, 5095, 5096, 5097, 5098, 5099, 5100, 5101, 5102, 510
3, 5104, 5105, 5106, 5107, 5108, 5109, 5110, 5111, 5112, 5113, 5114, 5115, 5116,
5117, 5118, 5119, 5120, 5121, 5122, 5123, 5124, 5125, 5126, 5127, 5128, 5129, 51
30, 5131, 5132, 5133, 5134, 5135, 5136, 5137, 5138, 5139, 5140, 5141, 5142, 514
3, 5144, 5145, 5146, 5147, 5148, 5149, 5150, 5151, 5152, 5153, 5154, 5155, 5156,
5157, 5158, 5159, 5160, 5161, 5162, 5163, 5164, 5165, 5166, 5167, 5168, 5169, 51
70, 5171, 5172, 5173, 5174, 5175, 5176, 5177, 5178, 5179, 5180, 5181, 5182, 518
3, 5184, 5185, 5186, 5187, 5188, 5189, 5190, 5191, 5192, 5193, 5194, 5195, 5196,
5197, 5198, 5199, 5200, 5201, 5202, 5203, 5204, 5205, 5206, 5207, 5208, 5209, 52
10, 5211, 5212, 5213, 5214, 5215, 5216, 5217, 5218, 5219, 5220, 5221, 5222, 522
3, 5224, 5225, 5226, 5227, 5228, 5229, 5230, 5231, 5232, 5233, 5234, 5235, 5236,
5237, 5238, 5239, 5240, 5241, 5242, 5243, 5244, 5245, 5246, 5247, 5248, 5249, 52
50, 5251, 5252, 5253, 5254, 5255, 5256, 5257, 5258, 5259, 5260, 5261, 5262, 526
3, 5264, 5265, 5266, 5267, 5268, 5269, 5270, 5271, 5272, 5273, 5274, 5275, 5276,
5277, 5278, 5279, 5280, 5281, 5282, 5283, 5284, 5285, 5286, 5287, 5288, 5289, 52
90, 5291, 5292, 5293, 5294, 5295, 5296, 5297, 5298, 5299, 5300, 5301, 5302, 530
3, 5304, 5305, 5306, 5307, 5308, 5309, 5310, 5311, 5312, 5313, 5314, 5315, 5316,
5317, 5318, 5319, 5320, 5321, 5322, 5323, 5324, 5325, 5326, 5327, 5328, 5329, 53
30, 5331, 5332, 5333, 5334, 5335, 5336, 5337, 5338, 5339, 5340, 5341, 5342, 534
3, 5344, 5345, 5346, 5347, 5348, 5349, 5350, 5351, 5352, 5353, 5354, 5355, 5356,
5357, 5358, 5359, 5360, 5361, 5362, 5363, 5364, 5365, 5366, 5367, 5368, 5369, 53
70, 5371, 5372, 5373, 5374, 5375, 5376, 5377, 5378, 5379, 5380, 5381, 5382, 538
3, 5384, 5385, 5386, 5387, 5388, 5389, 5390, 5391, 5392, 5393, 5394, 5395, 5396,
5397, 5398, 5399, 5400, 5401, 5402, 5403, 5404, 5405, 5406, 5407, 5408, 5409, 54
10, 5411, 5412, 5413, 5414, 5415, 5416, 5417, 5418, 5419, 5420, 5421, 5422, 542
3, 5424, 5425, 5426, 5427, 5428, 5429, 5430, 5431, 5432, 5433, 5434, 5435, 5436,
5437, 5438, 5439, 5440, 5441, 5442, 5443, 5444, 5445, 5446, 5447, 5448, 5449, 54
50, 5451, 5452, 5453, 5454, 5455, 5456, 5457, 5458, 5459, 5460, 5461, 5462, 546
3, 5464, 5465, 5466, 5467, 5468, 5469, 5470, 5471, 5472, 5473, 5474, 5475, 5476,
5477, 5478, 5479, 5480, 5481, 5482, 5483, 5484, 5485, 5486, 5487, 5488, 5489, 54
90, 5491, 5492, 5493, 5494, 5495, 5496, 5497, 5498, 5499, 5500, 5501, 5502, 550
3, 5504, 5505, 5506, 5507, 5508, 5509, 5510, 5511, 5512, 5513, 5514, 5515, 5516,
5517, 5518, 5519, 5520, 5521, 5522, 5523, 5524, 5525, 5526, 5527, 5528, 5529, 55
30, 5531, 5532, 5533, 5534, 5535, 5536, 5537, 5538, 5539, 5540, 5541, 5542, 554
3, 5544, 5545, 5546, 5547, 5548, 5549, 5550, 5551, 5552, 5553, 5554, 5555, 5556,
5557, 5558, 5559, 5560, 5561, 5562, 5563, 5564, 5565, 5566, 5567, 5568, 5569, 55
70, 5571, 5572, 5573, 5574, 5575, 5576, 5577, 5578, 5579, 5580, 5581, 5582, 558
3, 5584, 5585, 5586, 5587, 5588, 5589, 5590, 5591, 5592, 5593, 5594, 5595, 5596,
5597, 5598, 5599, 5600, 5601, 5602, 5603, 5604, 5605, 5606, 5607, 5608, 5609, 56
10, 5611, 5612, 5613, 5614, 5615, 5616, 5617, 5618, 5619, 5620, 5621, 5622, 562
3, 5624, 5625, 5626, 5627, 5628, 5629, 5630, 5631, 5632, 5633, 5634, 5635, 5636,
5637, 5638, 5639, 5640, 5641, 5642, 5643, 5644, 5645, 5646, 5647, 5648, 5649, 56
50, 5651, 5652, 5653, 5654, 5655, 5656, 5657, 5658, 5659, 5660, 5661, 5662, 566
3, 5664, 5665, 5666, 5667, 5668, 5669, 5670, 5671, 5672, 5673, 5674, 5675, 5676,
5677, 5678, 5679, 5680, 5681, 5682, 5683, 5684, 5685, 5686, 5687, 5688, 5689, 56
90, 5691, 5692, 5693, 5694, 5695, 5696, 5697, 5698, 5699, 5700, 5701, 5702, 570
3, 5704, 5705, 5706, 5707, 5708, 5709, 5710, 5711, 5712, 5713, 5714, 5715, 5716,
5717, 5718, 5719, 5720, 5721, 5722, 5723, 5724, 5725, 5726, 5727, 5728, 5729, 57
30, 5731, 5732, 5733, 5734, 5735, 5736, 5737, 5738, 5739, 5740, 5741, 5742, 574
3, 5744, 5745, 5746, 5747, 5748, 5749, 5750, 5751, 5752, 5753, 5754, 5755, 5756,
5757, 5758, 5759, 5760, 5761, 5762, 5763, 5764, 5765, 5766, 5767, 5768, 5769, 57
```

```
70, 5771, 5772, 5773, 5774, 5775, 5776, 5777, 5778, 5779, 5780, 5781, 5782, 578
3, 5784, 5785, 5786, 5787, 5788, 5789, 5790, 5791, 5792, 5793, 5794, 5795, 5796,
5797, 5798, 5799, 5800, 5801, 5802, 5803, 5804, 5805, 5806, 5807, 5808, 5809, 58
10, 5811, 5812, 5813, 5814, 5815, 5816, 5817, 5818, 5819, 5820, 5821, 5822, 582
3, 5824, 5825, 5826, 5827, 5828, 5829, 5830, 5831, 5832, 5833, 5834, 5835, 5836,
5837, 5838, 5839, 5840, 5841, 5842, 5843, 5844, 5845, 5846, 5847, 5848, 5849, 58
50, 5851, 5852, 5853, 5854, 5855, 5856, 5857, 5858, 5859, 5860, 5861, 5862, 586
3, 5864, 5865, 5866, 5867, 5868, 5869, 5870, 5871, 5872, 5873, 5874, 5875, 5876,
5877, 5878, 5879, 5880, 5881, 5882, 5883, 5884, 5885, 5886, 5887, 5888, 5889, 58
90, 5891, 5892, 5893, 5894, 5895, 5896, 5897, 5898, 5899, 5900, 5901, 5902, 590
3, 5904, 5905, 5906, 5907, 5908, 5909, 5910, 5911, 5912, 5913, 5914, 5915, 5916,
5917, 5918, 5919, 5920, 5921, 5922, 5923, 5924, 5925, 5926, 5927, 5928, 5929, 59
30, 5931, 5932, 5933, 5934, 5935, 5936, 5937, 5938, 5939, 5940, 5941, 5942, 594
3, 5944, 5945, 5946, 5947, 5948, 5949, 5950, 5951, 5952, 5953, 5954, 5955, 5956,
5957, 5958, 5959, 5960, 5961, 5962, 5963, 5964, 5965, 5966, 5967, 5968, 5969, 59
70, 5971, 5972, 5973, 5974, 5975, 5976, 5977, 5978, 5979, 5980, 5981, 5982, 598
3, 5984, 5985, 5986, 5987, 5988, 5989, 5990, 5991, 5992, 5993, 5994, 5995, 5996,
5997, 5998, 5999, 6000, 6001, 6002, 6003, 6004, 6005, 6006, 6007, 6008, 6009, 60
10, 6011, 6012, 6013, 6014, 6015, 6016, 6017, 6018, 6019, 6020, 6021, 6022, 602
3, 6024, 6025, 6026, 6027, 6028, 6029, 6030, 6031, 6032, 6033, 6034, 6035, 6036,
6037, 6038, 6039, 6040, 6041, 6042, 6043, 6044, 6045, 6046, 6047, 6048, 6049, 60
50, 6051, 6052, 6053, 6054, 6055, 6056, 6057, 6058, 6059, 6060, 6061, 6062, 606
3, 6064, 6065, 6066, 6067, 6068, 6069, 6070, 6071, 6072, 6073, 6074, 6075, 6076,
6077, 6078, 6079, 6080, 6081, 6082, 6083, 6084, 6085, 6086, 6087, 6088, 6089, 60
90, 6091, 6092, 6093, 6094, 6095, 6096, 6097, 6098, 6099, 6100, 6101, 6102, 610
3, 6104, 6105, 6106, 6107, 6108, 6109, 6110, 6111, 6112, 6113, 6114, 6115, 6116,
6117, 6118, 6119, 6120, 6121, 6122, 6123, 6124, 6125, 6126, 6127, 6128, 6129, 61
30, 6131, 6132, 6133, 6134, 6135, 6136, 6137, 6138, 6139, 6140, 6141, 6142, 614
3, 6144, 6145, 6146, 6147, 6148, 6149, 6150, 6151, 6152, 6153, 6154, 6155, 6156,
6157, 6158, 6159, 6160, 6161, 6162, 6163, 6164, 6165, 6166, 6167, 6168, 6169, 61
70, 6171, 6172, 6173, 6174, 6175, 6176, 6177, 6178, 6179, 6180, 6181, 6182, 618
3, 6184, 6185, 6186, 6187, 6188, 6189, 6190, 6191, 6192, 6193, 6194, 6195, 6196,
6197, 6198, 6199, 6200, 6201, 6202, 6203, 6204, 6205, 6206, 6207, 6208, 6209, 62
10, 6211, 6212, 6213, 6214, 6215, 6216, 6217, 6218, 6219, 6220, 6221, 6222, 622
3, 6224, 6225, 6226, 6227, 6228, 6229, 6230, 6231, 6232, 6233, 6234, 6235, 6236,
6237, 6238, 6239, 6240, 6241, 6242, 6243, 6244, 6245, 6246, 6247, 6248, 6249, 62
50, 6251, 6252, 6253, 6254, 6255, 6256, 6257, 6258, 6259, 6260, 6261, 6262, 626
3, 6264, 6265, 6266, 6267, 6268, 6269, 6270, 6271, 6272, 6273, 6274, 6275, 6276,
6277, 6278, 6279, 6280, 6281, 6282, 6283, 6284, 6285, 6286, 6287, 6288, 6289, 62
90, 6291, 6292, 6293, 6294, 6295, 6296, 6297, 6298, 6299, 6300, 6301, 6302, 630
3, 6304, 6305, 6306, 6307, 6308, 6309, 6310, 6311, 6312, 6313, 6314, 6315, 6316,
6317, 6318, 6319, 6320, 6321, 6322, 6323, 6324, 6325, 6326, 6327, 6328, 6329, 63
30, 6331, 6332, 6333, 6334, 6335, 6336, 6337, 6338, 6339, 6340, 6341, 6342, 634
3, 6344, 6345, 6346, 6347, 6348, 6349, 6350, 6351, 6352, 6353, 6354, 6355, 6356,
6357, 6358, 6359, 6360, 6361, 6362, 6363, 6364, 6365, 6366, 6367, 6368, 6369, 63
70, 6371, 6372, 6373, 6374, 6375, 6376, 6377, 6378, 6379, 6380, 6381, 6382, 638
3, 6384, 6385, 6386, 6387, 6388, 6389, 6390, 6391, 6392, 6393, 6394, 6395, 6396,
6397, 6398, 6399, 6400, 6401, 6402, 6403, 6404, 6405, 6406, 6407, 6408, 6409, 64
10, 6411, 6412, 6413, 6414, 6415, 6416, 6417, 6418, 6419, 6420, 6421, 6422, 642
3, 6424, 6425, 6426, 6427, 6428, 6429, 6430, 6431, 6432, 6433, 6434, 6435, 6436,
6437, 6438, 6439, 6440, 6441, 6442, 6443, 6444, 6445, 6446, 6447, 6448, 6449, 64
50, 6451, 6452, 6453, 6454, 6455, 6456, 6457, 6458, 6459, 6460, 6461, 6462, 646
3, 6464, 6465, 6466, 6467, 6468, 6469, 6470, 6471, 6472, 6473, 6474, 6475, 6476,
6477, 6478, 6479, 6480, 6481, 6482, 6483, 6484, 6485, 6486, 6487, 6488, 6489, 64
90, 6491, 6492, 6493, 6494, 6495, 6496, 6497, 6498, 6499, 6500, 6501, 6502, 650
3, 6504, 6505, 6506, 6507, 6508, 6509, 6510, 6511, 6512, 6513, 6514, 6515, 6516,
6517, 6518, 6519, 6520, 6521, 6522, 6523, 6524, 6525, 6526, 6527, 6528, 6529, 65
30, 6531, 6532, 6533, 6534, 6535, 6536, 6537, 6538, 6539, 6540, 6541, 6542, 654
3, 6544, 6545, 6546, 6547, 6548, 6549, 6550, 6551, 6552, 6553, 6554, 6555, 6556,
6557, 6558, 6559, 6560, 6561, 6562, 6563, 6564, 6565, 6566, 6567, 6568, 6569, 65
70, 6571, 6572, 6573, 6574, 6575, 6576, 6577, 6578, 6579, 6580, 6581, 6582, 658
3, 6584, 6585, 6586, 6587, 6588, 6589, 6590, 6591, 6592, 6593, 6594, 6595, 6596,
6597, 6598, 6599, 6600, 6601, 6602, 6603, 6604, 6605, 6606, 6607, 6608, 6609, 66
10, 6611, 6612, 6613, 6614, 6615, 6616, 6617, 6618, 6619, 6620, 6621, 6622, 662
3, 6624, 6625, 6626, 6627, 6628, 6629, 6630, 6631, 6632, 6633, 6634, 6635, 6636,
6637, 6638, 6639, 6640, 6641, 6642, 6643, 6644, 6645, 6646, 6647, 6648, 6649, 66
50, 6651, 6652, 6653, 6654, 6655, 6656, 6657, 6658, 6659, 6660, 6661, 6662, 666
3, 6664, 6665, 6666, 6667, 6668, 6669, 6670, 6671, 6672, 6673, 6674, 6675, 6676,
6677, 6678, 6679, 6680, 6681, 6682, 6683, 6684, 6685, 6686, 6687, 6688, 6689, 66
90, 6691, 6692, 6693, 6694, 6695, 6696, 6697, 6698, 6699, 6700, 6701, 6702, 670
```

```
3, 6704, 6705, 6706, 6707, 6708, 6709, 6710, 6711, 6712, 6713, 6714, 6715, 6716,
6717, 6718, 6719, 6720, 6721, 6722, 6723, 6724, 6725, 6726, 6727, 6728, 6729, 67
30, 6731, 6732, 6733, 6734, 6735, 6736, 6737, 6738, 6739, 6740, 6741, 6742, 674
3, 6744, 6745, 6746, 6747, 6748, 6749, 6750, 6751, 6752, 6753, 6754, 6755, 6756,
6757, 6758, 6759, 6760, 6761, 6762, 6763, 6764, 6765, 6766, 6767, 6768, 6769, 67
70, 6771, 6772, 6773, 6774, 6775, 6776, 6777, 6778, 6779, 6780, 6781, 6782, 678
3, 6784, 6785, 6786, 6787, 6788, 6789, 6790, 6791, 6792, 6793, 6794, 6795, 6796,
6797, 6798, 6799, 6800, 6801, 6802, 6803, 6804, 6805, 6806, 6807, 6808, 6809, 68
10, 6811, 6812, 6813, 6814, 6815, 6816, 6817, 6818, 6819, 6820, 6821, 6822, 682
3, 6824, 6825, 6826, 6827, 6828, 6829, 6830, 6831, 6832, 6833, 6834, 6835, 6836,
6837, 6838, 6839, 6840, 6841, 6842, 6843, 6844, 6845, 6846, 6847, 6848, 6849, 68
50, 6851, 6852, 6853, 6854, 6855, 6856, 6857, 6858, 6859, 6860, 6861, 6862, 686
3, 6864, 6865, 6866, 6867, 6868, 6869, 6870, 6871, 6872, 6873, 6874, 6875, 6876,
6877, 6878, 6879, 6880, 6881, 6882, 6883, 6884, 6885, 6886, 6887, 6888, 6889, 68
90, 6891, 6892, 6893, 6894, 6895, 6896, 6897, 6898, 6899, 6900, 6901, 6902, 690
3, 6904, 6905, 6906, 6907, 6908, 6909, 6910, 6911, 6912, 6913, 6914, 6915, 6916,
6917, 6918, 6919, 6920, 6921, 6922, 6923, 6924, 6925, 6926, 6927, 6928, 6929, 69
30, 6931, 6932, 6933, 6934, 6935, 6936, 6937, 6938, 6939, 6940, 6941, 6942, 694
3, 6944, 6945, 6946, 6947, 6948, 6949, 6950, 6951, 6952, 6953, 6954, 6955, 6956,
6957, 6958, 6959, 6960, 6961, 6962, 6963, 6964, 6965, 6966, 6967, 6968, 6969, 69
70, 6971, 6972, 6973, 6974, 6975, 6976, 6977, 6978, 6979, 6980, 6981, 6982, 698
3, 6984, 6985, 6986, 6987, 6988, 6989, 6990, 6991, 6992, 6993, 6994, 6995, 6996,
6997, 6998, 6999, 7000, 7001, 7002, 7003, 7004, 7005, 7006, 7007, 7008, 7009, 70
10, 7011, 7012, 7013, 7014, 7015, 7016, 7017, 7018, 7019, 7020, 7021, 7022, 702
3, 7024, 7025, 7026, 7027, 7028, 7029, 7030, 7031, 7032, 7033, 7034, 7035, 7036,
7037, 7038, 7039, 7040, 7041, 7042, 7043, 7044, 7045, 7046, 7047, 7048, 7049, 70
50, 7051, 7052, 7053, 7054, 7055, 7056, 7057, 7058, 7059, 7060, 7061, 7062, 706
3, 7064, 7065, 7066, 7067, 7068, 7069, 7070, 7071, 7072, 7073, 7074, 7075, 7076,
7077, 7078, 7079, 7080, 7081, 7082, 7083, 7084, 7085, 7086, 7087, 7088, 7089, 70
90, 7091, 7092, 7093, 7094, 7095, 7096, 7097, 7098, 7099, 7100, 7101, 7102, 710
3, 7104, 7105, 7106, 7107, 7108, 7109, 7110, 7111, 7112, 7113, 7114, 7115, 7116,
7117, 7118, 7119, 7120, 7121, 7122, 7123, 7124, 7125, 7126, 7127, 7128, 7129, 71
30, 7131, 7132, 7133, 7134, 7135, 7136, 7137, 7138, 7139, 7140, 7141, 7142, 714
3, 7144, 7145, 7146, 7147, 7148, 7149, 7150, 7151, 7152, 7153, 7154, 7155, 7156,
7157, 7158, 7159, 7160, 7161, 7162, 7163, 7164, 7165, 7166, 7167, 7168, 7169, 71
70, 7171, 7172, 7173, 7174, 7175, 7176, 7177, 7178, 7179, 7180, 7181, 7182, 718
3, 7184, 7185, 7186, 7187, 7188, 7189, 7190, 7191, 7192, 7193, 7194, 7195, 7196,
7197, 7198, 7199, 7200, 7201, 7202, 7203, 7204, 7205, 7206, 7207, 7208, 7209, 72
10, 7211, 7212, 7213, 7214, 7215, 7216, 7217, 7218, 7219, 7220, 7221, 7222, 722
3, 7224, 7225, 7226, 7227, 7228, 7229, 7230, 7231, 7232, 7233, 7234, 7235, 7236,
7237, 7238, 7239, 7240, 7241, 7242, 7243, 7244, 7245, 7246, 7247, 7248, 7249, 72
50, 7251, 7252, 7253, 7254, 7255, 7256, 7257, 7258, 7259, 7260, 7261, 7262, 726
3, 7264, 7265, 7266, 7267, 7268, 7269, 7270, 7271, 7272, 7273, 7274, 7275, 7276,
7277, 7278, 7279, 7280, 7281, 7282, 7283, 7284, 7285, 7286, 7287, 7288, 7289, 72
90, 7291, 7292, 7293, 7294, 7295, 7296, 7297, 7298, 7299, 7300, 7301, 7302, 730
3, 7304, 7305, 7306, 7307, 7308, 7309, 7310, 7311, 7312, 7313, 7314, 7315, 7316,
7317, 7318, 7319, 7320, 7321, 7322, 7323, 7324, 7325, 7326, 7327, 7328, 7329, 73
30, 7331, 7332, 7333, 7334, 7335, 7336, 7337, 7338, 7339, 7340, 7341, 7342, 734
3, 7344, 7345, 7346, 7347, 7348, 7349, 7350, 7351, 7352, 7353, 7354, 7355, 7356,
7357, 7358, 7359, 7360, 7361, 7362, 7363, 7364, 7365, 7366, 7367, 7368, 7369, 73
70, 7371, 7372, 7373, 7374, 7375, 7376, 7377, 7378, 7379, 7380, 7381, 7382, 738
3, 7384, 7385, 7386, 7387, 7388, 7389, 7390, 7391, 7392, 7393, 7394, 7395, 7396,
7397, 7398, 7399, 7400, 7401, 7402, 7403, 7404, 7405, 7406, 7407, 7408, 7409, 74
10, 7411, 7412, 7413, 7414, 7415, 7416, 7417, 7418, 7419, 7420, 7421, 7422, 742
3, 7424, 7425, 7426, 7427, 7428, 7429, 7430, 7431, 7432, 7433, 7434, 7435, 7436,
7437, 7438, 7439, 7440, 7441, 7442, 7443, 7444, 7445, 7446, 7447, 7448, 7449, 74
50, 7451, 7452, 7453, 7454, 7455, 7456, 7457, 7458, 7459, 7460, 7461, 7462, 746
3, 7464, 7465, 7466, 7467, 7468, 7469, 7470, 7471, 7472, 7473, 7474, 7475, 7476,
7477, 7478, 7479, 7480, 7481, 7482, 7483, 7484, 7485, 7486, 7487, 7488, 7489, 74
90, 7491, 7492, 7493, 7494, 7495, 7496, 7497, 7498, 7499, 7500, 7501, 7502, 750
3, 7504, 7505, 7506, 7507, 7508, 7509, 7510, 7511, 7512, 7513, 7514, 7515, 7516,
7517, 7518, 7519, 7520, 7521, 7522, 7523, 7524, 7525, 7526, 7527, 7528, 7529, 75
30, 7531, 7532, 7533, 7534, 7535, 7536, 7537, 7538, 7539, 7540, 7541, 7542, 754
3, 7544, 7545, 7546, 7547, 7548, 7549, 7550, 7551, 7552, 7553, 7554, 7555, 7556,
7557, 7558, 7559, 7560, 7561, 7562, 7563, 7564, 7565, 7566, 7567, 7568, 7569, 75
70, 7571, 7572, 7573, 7574, 7575, 7576, 7577, 7578, 7579, 7580, 7581, 7582, 758
3, 7584, 7585, 7586, 7587, 7588, 7589, 7590, 7591, 7592, 7593, 7594, 7595, 7596,
7597, 7598, 7599, 7600, 7601, 7602, 7603, 7604, 7605, 7606, 7607, 7608, 7609, 76
10, 7611, 7612, 7613, 7614, 7615, 7616, 7617, 7618, 7619, 7620, 7621, 7622, 762
3, 7624, 7625, 7626, 7627, 7628, 7629, 7630, 7631, 7632, 7633, 7634, 7635, 7636,
```

```
7637, 7638, 7639, 7640, 7641, 7642, 7643, 7644, 7645, 7646, 7647, 7648, 7649, 76
50, 7651, 7652, 7653, 7654, 7655, 7656, 7657, 7658, 7659, 7660, 7661, 7662, 766
3, 7664, 7665, 7666, 7667, 7668, 7669, 7670, 7671, 7672, 7673, 7674, 7675, 7676,
7677, 7678, 7679, 7680, 7681, 7682, 7683, 7684, 7685, 7686, 7687, 7688, 7689, 76
90, 7691, 7692, 7693, 7694, 7695, 7696, 7697, 7698, 7699, 7700, 7701, 7702, 770
3, 7704, 7705, 7706, 7707, 7708, 7709, 7710, 7711, 7712, 7713, 7714, 7715, 7716,
7717, 7718, 7719, 7720, 7721, 7722, 7723, 7724, 7725, 7726, 7727, 7728, 7729, 77
30, 7731, 7732, 7733, 7734, 7735, 7736, 7737, 7738, 7739, 7740, 7741, 7742, 774
3, 7744, 7745, 7746, 7747, 7748, 7749, 7750, 7751, 7752, 7753, 7754, 7755, 7756,
7757, 7758, 7759, 7760, 7761, 7762, 7763, 7764, 7765, 7766, 7767, 7768, 7769, 77
70, 7771, 7772, 7773, 7774, 7775, 7776, 7777, 7778, 7779, 7780, 7781, 7782, 778
3, 7784, 7785, 7786, 7787, 7788, 7789, 7790, 7791, 7792, 7793, 7794, 7795, 7796,
7797, 7798, 7799, 7800, 7801, 7802, 7803, 7804, 7805, 7806, 7807, 7808, 7809, 78
10, 7811, 7812, 7813, 7814, 7815, 7816, 7817, 7818, 7819, 7820, 7821, 7822, 782
3, 7824, 7825, 7826, 7827, 7828, 7829, 7830, 7831, 7832, 7833, 7834, 7835, 7836,
7837, 7838, 7839, 7840, 7841, 7842, 7843, 7844, 7845, 7846, 7847, 7848, 7849, 78
50, 7851, 7852, 7853, 7854, 7855, 7856, 7857, 7858, 7859, 7860, 7861, 7862, 786
3, 7864, 7865, 7866, 7867, 7868, 7869, 7870, 7871, 7872, 7873, 7874, 7875, 7876,
7877, 7878, 7879, 7880, 7881, 7882, 7883, 7884, 7885, 7886, 7887, 7888, 7889, 78
90, 7891, 7892, 7893, 7894, 7895, 7896, 7897, 7898, 7899, 7900, 7901, 7902, 790
3, 7904, 7905, 7906, 7907, 7908, 7909, 7910, 7911, 7912, 7913, 7914, 7915, 7916,
7917, 7918, 7919, 7920, 7921, 7922, 7923, 7924, 7925, 7926, 7927, 7928, 7929, 79
30, 7931, 7932, 7933, 7934, 7935, 7936, 7937, 7938, 7939, 7940, 7941, 7942, 794
3, 7944, 7945, 7946, 7947, 7948, 7949, 7950, 7951, 7952, 7953, 7954, 7955, 7956,
7957, 7958, 7959, 7960, 7961, 7962, 7963, 7964, 7965, 7966, 7967, 7968, 7969, 79
70, 7971, 7972, 7973, 7974, 7975, 7976, 7977, 7978, 7979, 7980, 7981, 7982, 798
3, 7984, 7985, 7986, 7987, 7988, 7989, 7990, 7991, 7992, 7993, 7994, 7995, 7996,
7997, 7998, 7999, 8000, 8001, 8002, 8003, 8004, 8005, 8006, 8007, 8008, 8009, 80
10, 8011, 8012, 8013, 8014, 8015, 8016, 8017, 8018, 8019, 8020, 8021, 8022, 802
3, 8024, 8025, 8026, 8027, 8028, 8029, 8030, 8031, 8032, 8033, 8034, 8035, 8036,
8037, 8038, 8039, 8040, 8041, 8042, 8043, 8044, 8045, 8046, 8047, 8048, 8049, 80
50, 8051, 8052, 8053, 8054, 8055, 8056, 8057, 8058, 8059, 8060, 8061, 8062, 806
3, 8064, 8065, 8066, 8067, 8068, 8069, 8070, 8071, 8072, 8073, 8074, 8075, 8076,
8077, 8078, 8079, 8080, 8081, 8082, 8083, 8084, 8085, 8086, 8087, 8088, 8089, 80
90, 8091, 8092, 8093, 8094, 8095, 8096, 8097, 8098, 8099, 8100, 8101, 8102, 810
3, 8104, 8105, 8106, 8107, 8108, 8109, 8110, 8111, 8112, 8113, 8114, 8115, 8116,
8117, 8118, 8119, 8120, 8121, 8122, 8123, 8124, 8125, 8126, 8127, 8128, 8129, 81
30, 8131, 8132, 8133, 8134, 8135, 8136, 8137, 8138, 8139, 8140, 8141, 8142, 814
3, 8144, 8145, 8146, 8147, 8148, 8149, 8150, 8151, 8152, 8153, 8154, 8155, 8156,
8157, 8158, 8159, 8160, 8161, 8162, 8163, 8164, 8165, 8166, 8167, 8168, 8169, 81
70, 8171, 8172, 8173, 8174, 8175, 8176, 8177, 8178, 8179, 8180, 8181, 8182, 818
3, 8184, 8185, 8186, 8187, 8188, 8189, 8190, 8191, 8192, 8193, 8194, 8195, 8196,
8197, 8198, 8199, 8200, 8201, 8202, 8203, 8204, 8205, 8206, 8207, 8208, 8209, 82
10, 8211, 8212, 8213, 8214, 8215, 8216, 8217, 8218, 8219, 8220, 8221, 8222, 822
3, 8224, 8225, 8226, 8227, 8228, 8229, 8230, 8231, 8232, 8233, 8234, 8235, 8236,
8237, 8238, 8239, 8240, 8241, 8242, 8243, 8244, 8245, 8246, 8247, 8248, 8249, 82
50, 8251, 8252, 8253, 8254, 8255, 8256, 8257, 8258, 8259, 8260, 8261, 8262, 826
3, 8264, 8265, 8266, 8267, 8268, 8269, 8270, 8271, 8272, 8273, 8274, 8275, 8276,
8277, 8278, 8279, 8280, 8281, 8282, 8283, 8284, 8285, 8286, 8287, 8288, 8289, 82
90, 8291, 8292, 8293, 8294, 8295, 8296, 8297, 8298, 8299, 8300, 8301, 8302, 830
3, 8304, 8305, 8306, 8307, 8308, 8309, 8310, 8311, 8312, 8313, 8314, 8315, 8316,
8317, 8318, 8319, 8320, 8321, 8322, 8323, 8324, 8325, 8326, 8327, 8328, 8329, 83
30, 8331, 8332, 8333, 8334, 8335, 8336, 8337, 8338, 8339, 8340, 8341, 8342, 834
3, 8344, 8345, 8346, 8347, 8348, 8349, 8350, 8351, 8352, 8353, 8354, 8355, 8356,
8357, 8358, 8359, 8360, 8361, 8362, 8363, 8364, 8365, 8366, 8367, 8368, 8369, 83
70, 8371, 8372, 8373, 8374, 8375, 8376, 8377, 8378, 8379, 8380, 8381, 8382, 838
3, 8384, 8385, 8386, 8387, 8388, 8389, 8390, 8391, 8392, 8393, 8394, 8395, 8396,
8397, 8398, 8399, 8400, 8401, 8402, 8403, 8404, 8405, 8406, 8407, 8408, 8409, 84
10, 8411, 8412, 8413, 8414, 8415, 8416, 8417, 8418, 8419, 8420, 8421, 8422, 842
3, 8424, 8425, 8426, 8427, 8428, 8429, 8430, 8431, 8432, 8433, 8434, 8435, 8436,
8437, 8438, 8439, 8440, 8441, 8442, 8443, 8444, 8445, 8446, 8447, 8448, 8449, 84
50, 8451, 8452, 8453, 8454, 8455, 8456, 8457, 8458, 8459, 8460, 8461, 8462, 846
3, 8464, 8465, 8466, 8467, 8468, 8469, 8470, 8471, 8472, 8473, 8474, 8475, 8476,
8477, 8478, 8479, 8480, 8481, 8482, 8483, 8484, 8485, 8486, 8487, 8488, 8489, 84
90, 8491, 8492, 8493, 8494, 8495, 8496, 8497, 8498, 8499, 8500, 8501, 8502, 850
3, 8504, 8505, 8506, 8507, 8508, 8509, 8510, 8511, 8512, 8513, 8514, 8515, 8516,
8517, 8518, 8519, 8520, 8521, 8522, 8523, 8524, 8525, 8526, 8527, 8528, 8529, 85
30, 8531, 8532, 8533, 8534, 8535, 8536, 8537, 8538, 8539, 8540, 8541, 8542, 854
3, 8544, 8545, 8546, 8547, 8548, 8549, 8550, 8551, 8552, 8553, 8554, 8555, 8556,
8557, 8558, 8559, 8560, 8561, 8562, 8563, 8564, 8565, 8566, 8567, 8568, 8569, 85
```

```
70, 8571, 8572, 8573, 8574, 8575, 8576, 8577, 8578, 8579, 8580, 8581, 8582, 858
3, 8584, 8585, 8586, 8587, 8588, 8589, 8590, 8591, 8592, 8593, 8594, 8595, 8596,
8597, 8598, 8599, 8600, 8601, 8602, 8603, 8604, 8605, 8606, 8607, 8608, 8609, 86
10, 8611, 8612, 8613, 8614, 8615, 8616, 8617, 8618, 8619, 8620, 8621, 8622, 862
3, 8624, 8625, 8626, 8627, 8628, 8629, 8630, 8631, 8632, 8633, 8634, 8635, 8636,
8637, 8638, 8639, 8640, 8641, 8642, 8643, 8644, 8645, 8646, 8647, 8648, 8649, 86
50, 8651, 8652, 8653, 8654, 8655, 8656, 8657, 8658, 8659, 8660, 8661, 8662, 866
3, 8664, 8665, 8666, 8667, 8668, 8669, 8670, 8671, 8672, 8673, 8674, 8675, 8676,
8677, 8678, 8679, 8680, 8681, 8682, 8683, 8684, 8685, 8686, 8687, 8688, 8689, 86
90, 8691, 8692, 8693, 8694, 8695, 8696, 8697, 8698, 8699, 8700, 8701, 8702, 870
3, 8704, 8705, 8706, 8707, 8708, 8709, 8710, 8711, 8712, 8713, 8714, 8715, 8716,
8717, 8718, 8719, 8720, 8721, 8722, 8723, 8724, 8725, 8726, 8727, 8728, 8729, 87
30, 8731, 8732, 8733, 8734, 8735, 8736, 8737, 8738, 8739, 8740, 8741, 8742, 874
3, 8744, 8745, 8746, 8747, 8748, 8749, 8750, 8751, 8752, 8753, 8754, 8755, 8756,
8757, 8758, 8759, 8760, 8761, 8762, 8763, 8764, 8765, 8766, 8767, 8768, 8769, 87
70, 8771, 8772, 8773, 8774, 8775, 8776, 8777, 8778, 8779, 8780, 8781, 8782, 878
3, 8784, 8785, 8786, 8787, 8788, 8789, 8790, 8791, 8792, 8793, 8794, 8795, 8796,
8797, 8798, 8799, 8800, 8801, 8802, 8803, 8804, 8805, 8806, 8807, 8808, 8809, 88
10, 8811, 8812, 8813, 8814, 8815, 8816, 8817, 8818, 8819, 8820, 8821, 8822, 882
3, 8824, 8825, 8826, 8827, 8828, 8829, 8830, 8831, 8832, 8833, 8834, 8835, 8836,
8837, 8838, 8839, 8840, 8841, 8842, 8843, 8844, 8845, 8846, 8847, 8848, 8849, 88
50, 8851, 8852, 8853, 8854, 8855, 8856, 8857, 8858, 8859, 8860, 8861, 8862, 886
3, 8864, 8865, 8866, 8867, 8868, 8869, 8870, 8871, 8872, 8873, 8874, 8875, 8876,
8877, 8878, 8879, 8880, 8881, 8882, 8883, 8884, 8885, 8886, 8887, 8888, 8889, 88
90, 8891, 8892, 8893, 8894, 8895, 8896, 8897, 8898, 8899, 8900, 8901, 8902, 890
3, 8904, 8905, 8906, 8907, 8908, 8909, 8910, 8911, 8912, 8913, 8914, 8915, 8916,
8917, 8918, 8919, 8920, 8921, 8922, 8923, 8924, 8925, 8926, 8927, 8928, 8929, 89
30, 8931, 8932, 8933, 8934, 8935, 8936, 8937, 8938, 8939, 8940, 8941, 8942, 894
3, 8944, 8945, 8946, 8947, 8948, 8949, 8950, 8951, 8952, 8953, 8954, 8955, 8956,
8957, 8958, 8959, 8960, 8961, 8962, 8963, 8964, 8965, 8966, 8967, 8968, 8969, 89
70, 8971, 8972, 8973, 8974, 8975, 8976, 8977, 8978, 8979, 8980, 8981, 8982, 898
3, 8984, 8985, 8986, 8987, 8988, 8989, 8990, 8991, 8992, 8993, 8994, 8995, 8996,
8997, 8998, 8999, 9000, 9001, 9002, 9003, 9004, 9005, 9006, 9007, 9008, 9009, 90
10, 9011, 9012, 9013, 9014, 9015, 9016, 9017, 9018, 9019, 9020, 9021, 9022, 902
3, 9024, 9025, 9026, 9027, 9028, 9029, 9030, 9031, 9032, 9033, 9034, 9035, 9036,
9037, 9038, 9039, 9040, 9041, 9042, 9043, 9044, 9045, 9046, 9047, 9048, 9049, 90
50, 9051, 9052, 9053, 9054, 9055, 9056, 9057, 9058, 9059, 9060, 9061, 9062, 906
3, 9064, 9065, 9066, 9067, 9068, 9069, 9070, 9071, 9072, 9073, 9074, 9075, 9076,
9077, 9078, 9079, 9080, 9081, 9082, 9083, 9084, 9085, 9086, 9087, 9088, 9089, 90
90, 9091, 9092, 9093, 9094, 9095, 9096, 9097, 9098, 9099, 9100, 9101, 9102, 910
3, 9104, 9105, 9106, 9107, 9108, 9109, 9110, 9111, 9112, 9113, 9114, 9115, 9116,
9117, 9118, 9119, 9120, 9121, 9122, 9123, 9124, 9125, 9126, 9127, 9128, 9129, 91
30, 9131, 9132, 9133, 9134, 9135, 9136, 9137, 9138, 9139, 9140, 9141, 9142, 914
3, 9144, 9145, 9146, 9147, 9148, 9149, 9150, 9151, 9152, 9153, 9154, 9155, 9156,
9157, 9158, 9159, 9160, 9161, 9162, 9163, 9164, 9165, 9166, 9167, 9168, 9169, 91
70, 9171, 9172, 9173, 9174, 9175, 9176, 9177, 9178, 9179, 9180, 9181, 9182, 918
3, 9184, 9185, 9186, 9187, 9188, 9189, 9190, 9191, 9192, 9193, 9194, 9195, 9196,
9197, 9198, 9199, 9200, 9201, 9202, 9203, 9204, 9205, 9206, 9207, 9208, 9209, 92
10, 9211, 9212, 9213, 9214, 9215, 9216, 9217, 9218, 9219, 9220, 9221, 9222, 922
3, 9224, 9225, 9226, 9227, 9228, 9229, 9230, 9231, 9232, 9233, 9234, 9235, 9236,
9237, 9238, 9239, 9240, 9241, 9242, 9243, 9244, 9245, 9246, 9247, 9248, 9249, 92
50, 9251, 9252, 9253, 9254, 9255, 9256, 9257, 9258, 9259, 9260, 9261, 9262, 926
3, 9264, 9265, 9266, 9267, 9268, 9269, 9270, 9271, 9272, 9273, 9274, 9275, 9276,
9277, 9278, 9279, 9280, 9281, 9282, 9283, 9284, 9285, 9286, 9287, 9288, 9289, 92
90, 9291, 9292, 9293, 9294, 9295, 9296, 9297, 9298, 9299, 9300, 9301, 9302, 930
3, 9304, 9305, 9306, 9307, 9308, 9309, 9310, 9311, 9312, 9313, 9314, 9315, 9316,
9317, 9318, 9319, 9320, 9321, 9322, 9323, 9324, 9325, 9326, 9327, 9328, 9329, 93
30, 9331, 9332, 9333, 9334, 9335, 9336, 9337, 9338, 9339, 9340, 9341, 9342, 934
3, 9344, 9345, 9346, 9347, 9348, 9349, 9350, 9351, 9352, 9353, 9354, 9355, 9356,
9357, 9358, 9359, 9360, 9361, 9362, 9363, 9364, 9365, 9366, 9367, 9368, 9369, 93
70, 9371, 9372, 9373, 9374, 9375, 9376, 9377, 9378, 9379, 9380, 9381, 9382, 938
3, 9384, 9385, 9386, 9387, 9388, 9389, 9390, 9391, 9392, 9393, 9394, 9395, 9396,
9397, 9398, 9399, 9400, 9401, 9402, 9403, 9404, 9405, 9406, 9407, 9408, 9409, 94
10, 9411, 9412, 9413, 9414, 9415, 9416, 9417, 9418, 9419, 9420, 9421, 9422, 942
3, 9424, 9425, 9426, 9427, 9428, 9429, 9430, 9431, 9432, 9433, 9434, 9435, 9436,
9437, 9438, 9439, 9440, 9441, 9442, 9443, 9444, 9445, 9446, 9447, 9448, 9449, 94
50, 9451, 9452, 9453, 9454, 9455, 9456, 9457, 9458, 9459, 9460, 9461, 9462, 946
3, 9464, 9465, 9466, 9467, 9468, 9469, 9470, 9471, 9472, 9473, 9474, 9475, 9476,
9477, 9478, 9479, 9480, 9481, 9482, 9483, 9484, 9485, 9486, 9487, 9488, 9489, 94
90, 9491, 9492, 9493, 9494, 9495, 9496, 9497, 9498, 9499, 9500, 9501, 9502, 950
```

```
3, 9504, 9505, 9506, 9507, 9508, 9509, 9510, 9511, 9512, 9513, 9514, 9515, 9516,
9517, 9518, 9519, 9520, 9521, 9522, 9523, 9524, 9525, 9526, 9527, 9528, 9529, 95
30, 9531, 9532, 9533, 9534, 9535, 9536, 9537, 9538, 9539, 9540, 9541, 9542, 954
3, 9544, 9545, 9546, 9547, 9548, 9549, 9550, 9551, 9552, 9553, 9554, 9555, 9556,
9557, 9558, 9559, 9560, 9561, 9562, 9563, 9564, 9565, 9566, 9567, 9568, 9569, 95
70, 9571, 9572, 9573, 9574, 9575, 9576, 9577, 9578, 9579, 9580, 9581, 9582, 958
3, 9584, 9585, 9586, 9587, 9588, 9589, 9590, 9591, 9592, 9593, 9594, 9595, 9596,
9597, 9598, 9599, 9600, 9601, 9602, 9603, 9604, 9605, 9606, 9607, 9608, 9609, 96
10, 9611, 9612, 9613, 9614, 9615, 9616, 9617, 9618, 9619, 9620, 9621, 9622, 962
3, 9624, 9625, 9626, 9627, 9628, 9629, 9630, 9631, 9632, 9633, 9634, 9635, 9636,
9637, 9638, 9639, 9640, 9641, 9642, 9643, 9644, 9645, 9646, 9647, 9648, 9649, 96
50, 9651, 9652, 9653, 9654, 9655, 9656, 9657, 9658, 9659, 9660, 9661, 9662, 966
3, 9664, 9665, 9666, 9667, 9668, 9669, 9670, 9671, 9672, 9673, 9674, 9675, 9676,
9677, 9678, 9679, 9680, 9681, 9682, 9683, 9684, 9685, 9686, 9687, 9688, 9689, 96
90, 9691, 9692, 9693, 9694, 9695, 9696, 9697, 9698, 9699, 9700, 9701, 9702, 970
3, 9704, 9705, 9706, 9707, 9708, 9709, 9710, 9711, 9712, 9713, 9714, 9715, 9716,
9717, 9718, 9719, 9720, 9721, 9722, 9723, 9724, 9725, 9726, 9727, 9728, 9729, 97
30, 9731, 9732, 9733, 9734, 9735, 9736, 9737, 9738, 9739, 9740, 9741, 9742, 974
3, 9744, 9745, 9746, 9747, 9748, 9749, 9750, 9751, 9752, 9753, 9754, 9755, 9756,
9757, 9758, 9759, 9760, 9761, 9762, 9763, 9764, 9765, 9766, 9767, 9768, 9769, 97
70, 9771, 9772, 9773, 9774, 9775, 9776, 9777, 9778, 9779, 9780, 9781, 9782, 978
3, 9784, 9785, 9786, 9787, 9788, 9789, 9790, 9791, 9792, 9793, 9794, 9795, 9796,
9797, 9798, 9799, 9800, 9801, 9802, 9803, 9804, 9805, 9806, 9807, 9808, 9809, 98
10, 9811, 9812, 9813, 9814, 9815, 9816, 9817, 9818, 9819, 9820, 9821, 9822, 982
3, 9824, 9825, 9826, 9827, 9828, 9829, 9830, 9831, 9832, 9833, 9834, 9835, 9836,
9837, 9838, 9839, 9840, 9841, 9842, 9843, 9844, 9845, 9846, 9847, 9848, 9849, 98
50, 9851, 9852, 9853, 9854, 9855, 9856, 9857, 9858, 9859, 9860, 9861, 9862, 986
3, 9864, 9865, 9866, 9867, 9868, 9869, 9870, 9871, 9872, 9873, 9874, 9875, 9876,
9877, 9878, 9879, 9880, 9881, 9882, 9883, 9884, 9885, 9886, 9887, 9888, 9889, 98
90, 9891, 9892, 9893, 9894, 9895, 9896, 9897, 9898, 9899, 9900, 9901, 9902, 990
3, 9904, 9905, 9906, 9907, 9908, 9909, 9910, 9911, 9912, 9913, 9914, 9915, 9916,
9917, 9918, 9919, 9920, 9921, 9922, 9923, 9924, 9925, 9926, 9927, 9928, 9929, 99
30, 9931, 9932, 9933, 9934, 9935, 9936, 9937, 9938, 9939, 9940, 9941, 9942, 994
3, 9944, 9945, 9946, 9947, 9948, 9949, 9950, 9951, 9952, 9953, 9954, 9955, 9956,
9957, 9958, 9959, 9960, 9961, 9962, 9963, 9964, 9965, 9966, 9967, 9968, 9969, 99
70, 9971, 9972, 9973, 9974, 9975, 9976, 9977, 9978, 9979, 9980, 9981, 9982, 998
3, 9984, 9985, 9986, 9987, 9988, 9989, 9990, 9991, 9992, 9993, 9994, 9995, 9996,
9997, 9998, 9999, 10000, 10001, 10002, 10003, 10004, 10005, 10006, 10007, 10008,
10009, 10010, 10011, 10012, 10013, 10014, 10015, 10016, 10017, 10018, 10019, 100
20, 10021, 10022, 10023, 10024, 10025, 10026, 10027, 10028, 10029, 10030, 10031,
10032, 10033, 10034, 10035, 10036, 10037, 10038, 10039, 10040, 10041, 10042, 100
43, 10044, 10045, 10046, 10047, 10048, 10049, 10050, 10051, 10052, 10053, 10054,
10055, 10056, 10057, 10058, 10059, 10060, 10061, 10062, 10063, 10064, 10065, 100
66, 10067, 10068, 10069, 10070, 10071, 10072, 10073, 10074, 10075, 10076, 10077,
10078, 10079, 10080, 10081, 10082, 10083, 10084, 10085, 10086, 10087, 10088, 100
89, 10090, 10091, 10092, 10093, 10094, 10095, 10096, 10097, 10098, 10099, 10100,
10101, 10102, 10103, 10104, 10105, 10106, 10107, 10108, 10109, 10110, 10111, 101
12, 10113, 10114, 10115, 10116, 10117, 10118, 10119, 10120, 10121, 10122, 10123,
10124, 10125, 10126, 10127, 10128, 10129, 10130, 10131, 10132, 10133, 10134, 101
35, 10136, 10137, 10138, 10139, 10140, 10141, 10142, 10143, 10144, 10145, 10146,
10147, 10148, 10149, 10150, 10151, 10152, 10153, 10154, 10155, 10156, 10157, 101
58, 10159, 10160, 10161, 10162, 10163, 10164, 10165, 10166, 10167, 10168, 10169,
10170, 10171, 10172, 10173, 10174, 10175, 10176, 10177, 10178, 10179, 10180, 101
81, 10182, 10183, 10184, 10185, 10186, 10187, 10188, 10189, 10190, 10191, 10192,
10193, 10194, 10195, 10196, 10197, 10198, 10199, 10200, 10201, 10202, 10203, 102
04, 10205, 10206, 10207, 10208, 10209, 10210, 10211, 10212, 10213, 10214, 10215,
10216, 10217, 10218, 10219, 10220, 10221, 10222, 10223, 10224, 10225, 10226, 102
27, 10228, 10229, 10230, 10231, 10232, 10233, 10234, 10235, 10236, 10237, 10238,
10239, 10240, 10241, 10242, 10243, 10244, 10245, 10246, 10247, 10248, 10249, 102
50, 10251, 10252, 10253, 10254, 10255, 10256, 10257, 10258, 10259, 10260, 10261,
10262, 10263, 10264, 10265, 10266, 10267, 10268, 10269, 10270, 10271, 10272, 102
73, 10274, 10275, 10276, 10277, 10278, 10279, 10280, 10281, 10282, 10283, 10284,
10285, 10286, 10287, 10288, 10289, 10290, 10291, 10292, 10293, 10294, 10295, 102
96, 10297, 10298, 10299, 10300, 10301, 10302, 10303, 10304, 10305, 10306, 10307,
10308, 10309, 10310, 10311, 10312, 10313, 10314, 10315, 10316, 10317, 10318, 103
19, 10320, 10321, 10322, 10323, 10324, 10325, 10326, 10327, 10328, 10329, 10330,
10331, 10332, 10333, 10334, 10335, 10336, 10337, 10338, 10339, 10340, 10341, 103
42, 10343, 10344, 10345, 10346, 10347, 10348, 10349, 10350, 10351, 10352, 10353,
10354, 10355, 10356, 10357, 10358, 10359, 10360, 10361, 10362, 10363, 10364, 103
65, 10366, 10367, 10368, 10369, 10370, 10371, 10372, 10373, 10374, 10375, 10376,
```

```
10377, 10378, 10379, 10380, 10381, 10382, 10383, 10384, 10385, 10386, 10387, 103
88, 10389, 10390, 10391, 10392, 10393, 10394, 10395, 10396, 10397, 10398, 10399,
10400, 10401, 10402, 10403, 10404, 10405, 10406, 10407, 10408, 10409, 10410, 104
11, 10412, 10413, 10414, 10415, 10416, 10417, 10418, 10419, 10420, 10421, 10422,
10423, 10424, 10425, 10426, 10427, 10428, 10429, 10430, 10431, 10432, 10433, 104
34, 10435, 10436, 10437, 10438, 10439, 10440, 10441, 10442, 10443, 10444, 10445,
10446, 10447, 10448, 10449, 10450, 10451, 10452, 10453, 10454, 10455, 10456, 104
57, 10458, 10459, 10460, 10461, 10462, 10463, 10464, 10465, 10466, 10467, 10468,
10469, 10470, 10471, 10472, 10473, 10474, 10475, 10476, 10477, 10478, 10479, 104
80, 10481, 10482, 10483, 10484, 10485, 10486, 10487, 10488, 10489, 10490, 10491,
10492, 10493, 10494, 10495, 10496, 10497, 10498, 10499, 10500, 10501, 10502, 105
03, 10504, 10505, 10506, 10507, 10508, 10509, 10510, 10511, 10512, 10513, 10514,
10515, 10516, 10517, 10518, 10519, 10520, 10521, 10522, 10523, 10524, 10525, 105
26, 10527, 10528, 10529, 10530, 10531, 10532, 10533, 10534, 10535, 10536, 10537,
10538, 10539, 10540, 10541, 10542, 10543, 10544, 10545, 10546, 10547, 10548, 105
49, 10550, 10551, 10552, 10553, 10554, 10555, 10556, 10557, 10558, 10559, 10560,
10561, 10562, 10563, 10564, 10565, 10566, 10567, 10568, 10569, 10570, 10571, 105
72, 10573, 10574, 10575, 10576, 10577, 10578, 10579, 10580, 10581, 10582, 10583,
10584, 10585, 10586, 10587, 10588, 10589, 10590, 10591, 10592, 10593, 10594, 105
95, 10596, 10597, 10598, 10599, 10600, 10601, 10602, 10603, 10604, 10605, 10606,
10607, 10608, 10609, 10610, 10611, 10612, 10613, 10614, 10615, 10616, 10617, 106
18, 10619, 10620, 10621, 10622, 10623, 10624, 10625, 10626, 10627, 10628, 10629,
10630, 10631, 10632, 10633, 10634, 10635, 10636, 10637, 10638, 10639, 10640, 106
41, 10642, 10643, 10644, 10645, 10646, 10647, 10648, 10649, 10650, 10651, 10652,
10653, 10654, 10655, 10656, 10657, 10658, 10659, 10660, 10661, 10662, 10663, 106
64, 10665, 10666, 10667, 10668, 10669, 10670, 10671, 10672, 10673, 10674, 10675,
10676, 10677, 10678, 10679, 10680, 10681, 10682, 10683, 10684, 10685, 10686, 106
87, 10688, 10689, 10690, 10691, 10692, 10693, 10694, 10695, 10696, 10697, 10698,
10699, 10700, 10701, 10702, 10703, 10704, 10705, 10706, 10707, 10708, 10709, 107
10, 10711, 10712, 10713, 10714, 10715, 10716, 10717, 10718, 10719, 10720, 10721,
10722, 10723, 10724, 10725, 10726, 10727, 10728, 10729, 10730, 10731, 10732, 107
33, 10734, 10735, 10736, 10737, 10738, 10739, 10740, 10741, 10742, 10743, 10744,
10745, 10746, 10747, 10748, 10749, 10750, 10751, 10752, 10753, 10754, 10755, 107
56, 10757, 10758, 10759, 10760, 10761, 10762, 10763, 10764, 10765, 10766, 10767,
10768, 10769, 10770, 10771, 10772, 10773, 10774, 10775, 10776, 10777, 10778, 107
79, 10780, 10781, 10782, 10783, 10784, 10785, 10786, 10787, 10788, 10789, 10790,
10791, 10792, 10793, 10794, 10795, 10796, 10797, 10798, 10799, 10800, 10801, 108
02, 10803, 10804, 10805, 10806, 10807, 10808, 10809, 10810, 10811, 10812, 10813,
10814, 10815, 10816, 10817, 10818, 10819, 10820, 10821, 10822, 10823, 10824, 108
25, 10826, 10827, 10828, 10829, 10830, 10831, 10832, 10833, 10834, 10835, 10836,
10837, 10838, 10839, 10840, 10841, 10842, 10843, 10844, 10845, 10846, 10847, 108
48, 10849, 10850, 10851, 10852, 10853, 10854, 10855, 10856, 10857, 10858, 10859,
10860, 10861, 10862, 10863, 10864, 10865, 10866, 10867, 10868, 10869, 10870, 108
71, 10872, 10873, 10874, 10875, 10876, 10877, 10878, 10879, 10880, 10881, 10882,
10883, 10884, 10885, 10886, 10887, 10888, 10889, 10890, 10891, 10892, 10893, 108
94, 10895, 10896, 10897, 10898, 10899, 10900, 10901, 10902, 10903, 10904, 10905,
10906, 10907, 10908, 10909, 10910, 10911, 10912, 10913, 10914, 10915, 10916, 109
17, 10918, 10919, 10920, 10921, 10922, 10923, 10924, 10925, 10926, 10927, 10928,
10929, 10930, 10931, 10932, 10933, 10934, 10935, 10936, 10937, 10938, 10939, 109
40, 10941, 10942, 10943, 10944, 10945, 10946, 10947, 10948, 10949, 10950, 10951,
10952, 10953, 10954, 10955, 10956, 10957, 10958, 10959, 10960, 10961, 10962, 109
63, 10964, 10965, 10966, 10967, 10968, 10969, 10970, 10971, 10972, 10973, 10974,
10975, 10976, 10977, 10978, 10979, 10980, 10981, 10982, 10983, 10984, 10985, 109
86, 10987, 10988, 10989, 10990, 10991, 10992, 10993, 10994, 10995, 10996, 10997,
10998, 10999, 11000, 11001, 11002, 11003, 11004, 11005, 11006, 11007, 11008, 110
09, 11010, 11011, 11012, 11013, 11014, 11015, 11016, 11017, 11018, 11019, 11020,
11021, 11022, 11023, 11024, 11025, 11026, 11027, 11028, 11029, 11030, 11031, 110
32, 11033, 11034, 11035, 11036, 11037, 11038, 11039, 11040, 11041, 11042, 11043,
11044, 11045, 11046, 11047, 11048, 11049, 11050, 11051, 11052, 11053, 11054, 110
55, 11056, 11057, 11058, 11059, 11060, 11061, 11062, 11063, 11064, 11065, 11066,
11067, 11068, 11069, 11070, 11071, 11072, 11073, 11074, 11075, 11076, 11077, 110
78, 11079, 11080, 11081, 11082, 11083, 11084, 11085, 11086, 11087, 11088, 11089,
11090, 11091, 11092, 11093, 11094, 11095, 11096, 11097, 11098, 11099, 11100, 111
01, 11102, 11103, 11104, 11105, 11106, 11107, 11108, 11109, 11110, 11111, 11112,
11113, 11114, 11115, 11116, 11117, 11118, 11119, 11120, 11121, 11122, 11123, 111
24, 11125, 11126, 11127, 11128, 11129, 11130, 11131, 11132, 11133, 11134, 11135,
11136, 11137, 11138, 11139, 11140, 11141, 11142, 11143, 11144, 11145, 11146, 111
47, 11148, 11149, 11150, 11151, 11152, 11153, 11154, 11155, 11156, 11157, 11158,
11159, 11160, 11161, 11162, 11163, 11164, 11165, 11166, 11167, 11168, 11169, 111
70, 11171, 11172, 11173, 11174, 11175, 11176, 11177, 11178, 11179, 11180, 11181,
```

```
11182, 11183, 11184, 11185, 11186, 11187, 11188, 11189, 11190, 11191, 11192, 111
93, 11194, 11195, 11196, 11197, 11198, 11199, 11200, 11201, 11202, 11203, 11204,
11205, 11206, 11207, 11208, 11209, 11210, 11211, 11212, 11213, 11214, 11215, 112
16, 11217, 11218, 11219, 11220, 11221, 11222, 11223, 11224, 11225, 11226, 11227,
11228, 11229, 11230, 11231, 11232, 11233, 11234, 11235, 11236, 11237, 11238, 112
39, 11240, 11241, 11242, 11243, 11244, 11245, 11246, 11247, 11248, 11249, 11250,
11251, 11252, 11253, 11254, 11255, 11256, 11257, 11258, 11259, 11260, 11261, 112
62, 11263, 11264, 11265, 11266, 11267, 11268, 11269, 11270, 11271, 11272, 11273,
11274, 11275, 11276, 11277, 11278, 11279, 11280, 11281, 11282, 11283, 11284, 112
85, 11286, 11287, 11288, 11289, 11290, 11291, 11292, 11293, 11294, 11295, 11296,
11297, 11298, 11299, 11300, 11301, 11302, 11303, 11304, 11305, 11306, 11307, 113
08, 11309, 11310, 11311, 11312, 11313, 11314, 11315, 11316, 11317, 11318, 11319,
11320, 11321, 11322, 11323, 11324, 11325, 11326, 11327, 11328, 11329, 11330, 113
31, 11332, 11333, 11334, 11335, 11336, 11337, 11338, 11339, 11340, 11341, 11342,
11343, 11344, 11345, 11346, 11347, 11348, 11349, 11350, 11351, 11352, 11353, 113
54, 11355, 11356, 11357, 11358, 11359, 11360, 11361, 11362, 11363, 11364, 11365,
11366, 11367, 11368, 11369, 11370, 11371, 11372, 11373, 11374, 11375, 11376, 113
77, 11378, 11379, 11380, 11381, 11382, 11383, 11384, 11385, 11386, 11387, 11388,
11389, 11390, 11391, 11392, 11393, 11394, 11395, 11396, 11397, 11398, 11399, 114
00, 11401, 11402, 11403, 11404, 11405, 11406, 11407, 11408, 11409, 11410, 11411,
11412, 11413, 11414, 11415, 11416, 11417, 11418, 11419, 11420, 11421, 11422, 114
23, 11424, 11425, 11426, 11427, 11428, 11429, 11430, 11431, 11432, 11433, 11434,
11435, 11436, 11437, 11438, 11439, 11440, 11441, 11442, 11443, 11444, 11445, 114
46, 11447, 11448, 11449, 11450, 11451, 11452, 11453, 11454, 11455, 11456, 11457,
11458, 11459, 11460, 11461, 11462, 11463, 11464, 11465, 11466, 11467, 11468, 114
69, 11470, 11471, 11472, 11473, 11474, 11475, 11476, 11477, 11478, 11479, 11480,
11481, 11482, 11483, 11484, 11485, 11486, 11487, 11488, 11489, 11490, 11491, 114
92, 11493, 11494, 11495, 11496, 11497, 11498, 11499, 11500, 11501, 11502, 11503,
11504, 11505, 11506, 11507, 11508, 11509, 11510, 11511, 11512, 11513, 11514, 115
15, 11516, 11517, 11518, 11519, 11520, 11521, 11522, 11523, 11524, 11525, 11526,
11527, 11528, 11529, 11530, 11531, 11532, 11533, 11534, 11535, 11536, 11537, 115
38, 11539, 11540, 11541, 11542, 11543, 11544, 11545, 11546, 11547, 11548, 11549,
11550, 11551, 11552, 11553, 11554, 11555, 11556, 11557, 11558, 11559, 11560, 115
61, 11562, 11563, 11564, 11565, 11566, 11567, 11568, 11569, 11570, 11571, 11572,
11573, 11574, 11575, 11576, 11577, 11578, 11579, 11580, 11581, 11582, 11583, 115
84, 11585, 11586, 11587, 11588, 11589, 11590, 11591, 11592, 11593, 11594, 11595,
11596, 11597, 11598, 11599, 11600, 11601, 11602, 11603, 11604, 11605, 11606, 116
07, 11608, 11609, 11610, 11611, 11612, 11613, 11614, 11615, 11616, 11617, 11618,
11619, 11620, 11621, 11622, 11623, 11624, 11625, 11626, 11627, 11628, 11629, 116
30, 11631, 11632, 11633, 11634, 11635, 11636, 11637, 11638, 11639, 11640, 11641,
11642, 11643, 11644, 11645, 11646, 11647, 11648, 11649, 11650, 11651, 11652, 116
53, 11654, 11655, 11656, 11657, 11658, 11659, 11660, 11661, 11662, 11663, 11664,
11665, 11666, 11667, 11668, 11669, 11670, 11671, 11672, 11673, 11674, 11675, 116
76, 11677, 11678, 11679, 11680, 11681, 11682, 11683, 11684, 11685, 11686, 11687,
11688, 11689, 11690, 11691, 11692, 11693, 11694, 11695, 11696, 11697, 11698, 116
99, 11700, 11701, 11702, 11703, 11704, 11705, 11706, 11707, 11708, 11709, 11710,
11711, 11712, 11713, 11714, 11715, 11716, 11717, 11718, 11719, 11720, 11721, 117
22, 11723, 11724, 11725, 11726, 11727, 11728, 11729, 11730, 11731, 11732, 11733,
11734, 11735, 11736, 11737, 11738, 11739, 11740, 11741, 11742, 11743, 11744, 117
45, 11746, 11747, 11748, 11749, 11750, 11751, 11752, 11753, 11754, 11755, 11756,
11757, 11758, 11759, 11760, 11761, 11762, 11763, 11764, 11765, 11766, 11767, 117
68, 11769, 11770, 11771, 11772, 11773, 11774, 11775, 11776, 11777, 11778, 11779,
11780, 11781, 11782, 11783, 11784, 11785, 11786, 11787, 11788, 11789, 11790, 117
91, 11792, 11793, 11794, 11795, 11796, 11797, 11798, 11799, 11800, 11801, 11802,
11803, 11804, 11805, 11806, 11807, 11808, 11809, 11810, 11811, 11812, 11813, 118
14, 11815, 11816, 11817, 11818, 11819, 11820, 11821, 11822, 11823, 11824, 11825,
11826, 11827, 11828, 11829, 11830, 11831, 11832, 11833, 11834, 11835, 11836, 118
37, 11838, 11839, 11840, 11841, 11842, 11843, 11844, 11845, 11846, 11847, 11848,
11849, 11850, 11851, 11852, 11853, 11854, 11855, 11856, 11857, 11858, 11859, 118
60, 11861, 11862, 11863, 11864, 11865, 11866, 11867, 11868, 11869, 11870, 11871,
11872, 11873, 11874, 11875, 11876, 11877, 11878, 11879, 11880, 11881, 11882, 118
83, 11884, 11885, 11886, 11887, 11888, 11889, 11890, 11891, 11892, 11893, 11894,
11895, 11896, 11897, 11898, 11899, 11900, 11901, 11902, 11903, 11904, 11905, 119
06, 11907, 11908, 11909, 11910, 11911, 11912, 11913, 11914, 11915, 11916, 11917,
11918, 11919, 11920, 11921, 11922, 11923, 11924, 11925, 11926, 11927, 11928, 119
29, 11930, 11931, 11932, 11933, 11934, 11935, 11936, 11937, 11938, 11939, 11940,
11941, 11942, 11943, 11944, 11945, 11946, 11947, 11948, 11949, 11950, 11951, 119
52, 11953, 11954, 11955, 11956, 11957, 11958, 11959, 11960, 11961, 11962, 11963,
11964, 11965, 11966, 11967, 11968, 11969, 11970, 11971, 11972, 11973, 11974, 119
75, 11976, 11977, 11978, 11979, 11980, 11981, 11982, 11983, 11984, 11985, 11986,
```

11987, 11988, 11989, 11990, 11991, 11992, 11993, 11994, 11995, 11996, 11997, 119
98, 11999, 12000, 12001, 12002, 12003, 12004, 12005, 12006, 12007, 12008, 12009,
12010, 12011, 12012, 12013, 12014, 12015, 12016, 12017, 12018, 12019, 12020, 120
21, 12022, 12023, 12024, 12025, 12026, 12027, 12028, 12029, 12030, 12031, 12032,
12033, 12034, 12035, 12036, 12037, 12038, 12039, 12040, 12041, 12042, 12043, 120
44, 12045, 12046, 12047, 12048, 12049, 12050, 12051, 12052, 12053, 12054, 12055,
12056, 12057, 12058, 12059, 12060, 12061, 12062, 12063, 12064, 12065, 12066, 120
67, 12068, 12069, 12070, 12071, 12072, 12073, 12074, 12075, 12076, 12077, 12078,
12079, 12080, 12081, 12082, 12083, 12084, 12085, 12086, 12087, 12088, 12089, 120
90, 12091, 12092, 12093, 12094, 12095, 12096, 12097, 12098, 12099, 12100, 12101,
12102, 12103, 12104, 12105, 12106, 12107, 12108, 12109, 12110, 12111, 12112, 121
13, 12114, 12115, 12116, 12117, 12118, 12119, 12120, 12121, 12122, 12123, 12124,
12125, 12126, 12127, 12128, 12129, 12130, 12131, 12132, 12133, 12134, 12135, 121
36, 12137, 12138, 12139, 12140, 12141, 12142, 12143, 12144, 12145, 12146, 12147,
12148, 12149, 12150, 12151, 12152, 12153, 12154, 12155, 12156, 12157, 12158, 121
59, 12160, 12161, 12162, 12163, 12164, 12165, 12166, 12167, 12168, 12169, 12170,
12171, 12172, 12173, 12174, 12175, 12176, 12177, 12178, 12179, 12180, 12181, 121
82, 12183, 12184, 12185, 12186, 12187, 12188, 12189, 12190, 12191, 12192, 12193,
12194, 12195, 12196, 12197, 12198, 12199, 12200, 12201, 12202, 12203, 12204, 122
05, 12206, 12207, 12208, 12209, 12210, 12211, 12212, 12213, 12214, 12215, 12216,
12217, 12218, 12219, 12220, 12221, 12222, 12223, 12224, 12225, 12226, 12227, 122
28, 12229, 12230, 12231, 12232, 12233, 12234, 12235, 12236, 12237, 12238, 12239,
12240, 12241, 12242, 12243, 12244, 12245, 12246, 12247, 12248, 12249, 12250, 122
51, 12252, 12253, 12254, 12255, 12256, 12257, 12258, 12259, 12260, 12261, 12262,
12263, 12264, 12265, 12266, 12267, 12268, 12269, 12270, 12271, 12272, 12273, 122
74, 12275, 12276, 12277, 12278, 12279, 12280, 12281, 12282, 12283, 12284, 12285,
12286, 12287, 12288, 12289, 12290, 12291, 12292, 12293, 12294, 12295, 12296, 122
97, 12298, 12299, 12300, 12301, 12302, 12303, 12304, 12305, 12306, 12307, 12308,
12309, 12310, 12311, 12312, 12313, 12314, 12315, 12316, 12317, 12318, 12319, 123
20, 12321, 12322, 12323, 12324, 12325, 12326, 12327, 12328, 12329, 12330, 12331,
12332, 12333, 12334, 12335, 12336, 12337, 12338, 12339, 12340, 12341, 12342, 123
43, 12344, 12345, 12346, 12347, 12348, 12349, 12350, 12351, 12352, 12353, 12354,
12355, 12356, 12357, 12358, 12359, 12360, 12361, 12362, 12363, 12364, 12365, 123
66, 12367, 12368, 12369, 12370, 12371, 12372, 12373, 12374, 12375, 12376, 12377,
12378, 12379, 12380, 12381, 12382, 12383, 12384, 12385, 12386, 12387, 12388, 123
89, 12390, 12391, 12392, 12393, 12394, 12395, 12396, 12397, 12398, 12399, 12400,
12401, 12402, 12403, 12404, 12405, 12406, 12407, 12408, 12409, 12410, 12411, 124
12, 12413, 12414, 12415, 12416, 12417, 12418, 12419, 12420, 12421, 12422, 12423,
12424, 12425, 12426, 12427, 12428, 12429, 12430, 12431, 12432, 12433, 12434, 124
35, 12436, 12437, 12438, 12439, 12440, 12441, 12442, 12443, 12444, 12445, 12446,
12447, 12448, 12449, 12450, 12451, 12452, 12453, 12454, 12455, 12456, 12457, 124
58, 12459, 12460, 12461, 12462, 12463, 12464, 12465, 12466, 12467, 12468, 12469,
12470, 12471, 12472, 12473, 12474, 12475, 12476, 12477, 12478, 12479, 12480, 124
81, 12482, 12483, 12484, 12485, 12486, 12487, 12488, 12489, 12490, 12491, 12492,
12493, 12494, 12495, 12496, 12497, 12498, 12499, 12500, 12501, 12502, 12503, 125
04, 12505, 12506, 12507, 12508, 12509, 12510, 12511, 12512, 12513, 12514, 12515,
12516, 12517, 12518, 12519, 12520, 12521, 12522, 12523, 12524, 12525, 12526, 125
27, 12528, 12529, 12530, 12531, 12532, 12533, 12534, 12535, 12536, 12537, 12538,
12539, 12540, 12541, 12542, 12543, 12544, 12545, 12546, 12547, 12548, 12549, 125
50, 12551, 12552, 12553, 12554, 12555, 12556, 12557, 12558, 12559, 12560, 12561,
12562, 12563, 12564, 12565, 12566, 12567, 12568, 12569, 12570, 12571, 12572, 125
73, 12574, 12575, 12576, 12577, 12578, 12579, 12580, 12581, 12582, 12583, 12584,
12585, 12586, 12587, 12588, 12589, 12590, 12591, 12592, 12593, 12594, 12595, 125
96, 12597, 12598, 12599, 12600, 12601, 12602, 12603, 12604, 12605, 12606, 12607,
12608, 12609, 12610, 12611, 12612, 12613, 12614, 12615, 12616, 12617, 12618, 126
19, 12620, 12621, 12622, 12623, 12624, 12625, 12626, 12627, 12628, 12629, 12630,
12631, 12632, 12633, 12634, 12635, 12636, 12637, 12638, 12639, 12640, 12641, 126
42, 12643, 12644, 12645, 12646, 12647, 12648, 12649, 12650, 12651, 12652, 12653,
12654, 12655, 12656, 12657, 12658, 12659, 12660, 12661, 12662, 12663, 12664, 126
65, 12666, 12667, 12668, 12669, 12670, 12671, 12672, 12673, 12674, 12675, 12676,
12677, 12678, 12679, 12680, 12681, 12682, 12683, 12684, 12685, 12686, 12687, 126
88, 12689, 12690, 12691, 12692, 12693, 12694, 12695, 12696, 12697, 12698, 12699,
12700, 12701, 12702, 12703, 12704, 12705, 12706, 12707, 12708, 12709, 12710, 127
11, 12712, 12713, 12714, 12715, 12716, 12717, 12718, 12719, 12720, 12721, 12722,
12723, 12724, 12725, 12726, 12727, 12728, 12729, 12730, 12731, 12732, 12733, 127
34, 12735, 12736, 12737, 12738, 12739, 12740, 12741, 12742, 12743, 12744, 12745,
12746, 12747, 12748, 12749, 12750, 12751, 12752, 12753, 12754, 12755, 12756, 127
57, 12758, 12759, 12760, 12761, 12762, 12763, 12764, 12765, 12766, 12767, 12768,
12769, 12770, 12771, 12772, 12773, 12774, 12775, 12776, 12777, 12778, 12779, 127
80, 12781, 12782, 12783, 12784, 12785, 12786, 12787, 12788, 12789, 12790, 12791,

```
12792, 12793, 12794, 12795, 12796, 12797, 12798, 12799, 12800, 12801, 12802, 128
03, 12804, 12805, 12806, 12807, 12808, 12809, 12810, 12811, 12812, 12813, 12814,
12815, 12816, 12817, 12818, 12819, 12820, 12821, 12822, 12823, 12824, 12825, 128
26, 12827, 12828, 12829, 12830, 12831, 12832, 12833, 12834, 12835, 12836, 12837,
12838, 12839, 12840, 12841, 12842, 12843, 12844, 12845, 12846, 12847, 12848, 128
49, 12850, 12851, 12852, 12853, 12854, 12855, 12856, 12857, 12858, 12859, 12860,
12861, 12862, 12863, 12864, 12865, 12866, 12867, 12868, 12869, 12870, 12871, 128
72, 12873, 12874, 12875, 12876, 12877, 12878, 12879, 12880, 12881, 12882, 12883,
12884, 12885, 12886, 12887, 12888, 12889, 12890, 12891, 12892, 12893, 12894, 128
95, 12896, 12897, 12898, 12899, 12900, 12901, 12902, 12903, 12904, 12905, 12906,
12907, 12908, 12909, 12910, 12911, 12912, 12913, 12914, 12915, 12916, 12917, 129
18, 12919, 12920, 12921, 12922, 12923, 12924, 12925, 12926, 12927, 12928, 12929,
12930, 12931, 12932, 12933, 12934, 12935, 12936, 12937, 12938, 12939, 12940, 129
41, 12942, 12943, 12944, 12945, 12946, 12947, 12948, 12949, 12950, 12951, 12952,
12953, 12954, 12955, 12956, 12957, 12958, 12959, 12960, 12961, 12962, 12963, 129
64, 12965, 12966, 12967, 12968, 12969, 12970, 12971, 12972, 12973, 12974, 12975,
12976, 12977, 12978, 12979, 12980, 12981, 12982, 12983, 12984, 12985, 12986, 129
87, 12988, 12989, 12990, 12991, 12992, 12993, 12994, 12995, 12996, 12997, 12998,
12999, 13000, 13001, 13002, 13003, 13004, 13005, 13006, 13007, 13008, 13009, 130
10, 13011, 13012, 13013, 13014, 13015, 13016, 13017, 13018, 13019, 13020, 13021,
13022, 13023, 13024, 13025, 13026, 13027, 13028, 13029, 13030, 13031, 13032, 130
33, 13034, 13035, 13036, 13037, 13038, 13039, 13040, 13041, 13042, 13043, 13044,
13045, 13046, 13047, 13048, 13049, 13050, 13051, 13052, 13053, 13054, 13055, 130
56, 13057, 13058, 13059, 13060, 13061, 13062, 13063, 13064, 13065, 13066, 13067,
13068, 13069, 13070, 13071, 13072, 13073, 13074, 13075, 13076, 13077, 13078, 130
79, 13080, 13081, 13082, 13083, 13084, 13085, 13086, 13087, 13088, 13089, 13090,
13091, 13092, 13093, 13094, 13095, 13096, 13097, 13098, 13099, 13100, 13101, 131
02, 13103, 13104, 13105, 13106, 13107, 13108, 13109, 13110, 13111, 13112, 13113,
13114, 13115, 13116, 13117, 13118, 13119, 13120, 13121, 13122, 13123, 13124, 131
25, 13126])
```

In [ ]:

```
```

In [ ]:

```
tokenizer_eng.index_word[max(tokenizer_eng.word_index.values())]= '<end>'
```

In [ ]:

```
train_italia = pad_sequences(tokenizer_ita.texts_to_sequences(train['italia_enc'
]), padding='post', maxlen=20)
validation_italia = pad_sequences(tokenizer_ita.texts_to_sequences(validation['it
alia_enc']), padding='post', maxlen=20)
```

In [ ]:

```
train_eng_in = pad_sequences(tokenizer_eng.texts_to_sequences(train['english_in'
]), padding='post', maxlen=20)
train_eng_op = pad_sequences(tokenizer_eng.texts_to_sequences(train['english_op'
]), padding='post', maxlen=20)

validation_eng_in = pad_sequences(tokenizer_eng.texts_to_sequences(validation['en
glish_in']), padding='post', maxlen=20)
validation_eng_op = pad_sequences(tokenizer_eng.texts_to_sequences(validation['en
glish_op']), padding='post', maxlen=20)
```

In [ ]:

```
validation_eng_in[0].shape
```

Out[ ]: (20,)

In [ ]:

```
validation_eng_op[0].shape
```

Out[ ]: (20,)

In [ ]:

```
train_eng_in[0].shape
```

Out[ ]: (20,)

In [ ]:

```
train_eng_op[0].shape
```

Out[ ]: (20,)

In [ ]:

```

```

Implement custom encoder decoder

Encoder

In [ ]:

```
class Encoder(tf.keras.Model):
    '''
    Encoder model -- That takes a input sequence and returns encoder-outputs,enco
der_final_state_h,encoder_final_state_c
    '''

    def __init__(self,inp_vocab_size,embedding_size,lstm_size,input_length):
        super().__init__()
        self.vocab_size = inp_vocab_size
        self.embedding_size = embedding_size
        self.lstm_units = lstm_size
        self.input_length = input_length
        #Initialize Embedding layer
        #Intialize Encoder LSTM layer
        self.embedding = Embedding(input_dim=self.vocab_size, output_dim=self.emb
edding_size,mask_zero=True,
                                  input_length=self.input_length, name='encoder_e
mbed_layer')
        self.lstm = LSTM(units=self.lstm_units, return_state=True, return_sequenc
es=True, name='encoder_LSTM')

    def call(self, input_sequence, states):
        '''
          This function takes a sequence input and the initial states of the enco
der.
          Pass the input_sequence input to the Embedding layer, Pass the embeddin
g layer ouput to encoder_lstm
          returns -- encoder_output, last time step's hidden and cell state
        '''
        # first generate the embeddings for input utterence
        input_sequence = self.embedding(input_sequence)
        # pass the embeddings to the LSTM
        all_h_state, encoder_final_h_state, encoder_final_c_state = self.lstm(inp
uts=input_sequence, initial_state=states)

        return all_h_state, encoder_final_h_state, encoder_final_c_state


    def initialize_states(self, batch_size):

        '''
          Given a batch size it will return intial hidden state and intial cell s
tate.
        If batch size is 32- Hidden state is zeros of size [32,lstm_units], cell
 state zeros is of size [32,lstm_units]
        '''

        if batch_size == None:
            return
        initializer = tf.keras.initializers.Zeros()
        initial_state_h = initializer(shape=(batch_size,self.lstm_units))
        initial_state_c = initializer(shape=(batch_size,self.lstm_units))

        return [initial_state_h, initial_state_c]
```

Grader function - 1

In [ ]:
```
def grader_check_encoder():
    '''
        vocab-size: Unique words of the input language,
        embedding_size: output embedding dimension for each word after embedding
 layer,
        lstm_size: Number of lstm units,
        input_length: Length of the input sentence,
        batch_size
    '''
    vocab_size=10
    embedding_size=20
    lstm_size=32
    input_length=10
    batch_size=16
    #Intialzing encoder
    encoder=Encoder(vocab_size,embedding_size,lstm_size,input_length)
    input_sequence=tf.random.uniform(shape=[batch_size,input_length],maxval=vocab
_size,minval=0,dtype=tf.int32)
    #Intializing encoder initial states
    initial_state=encoder.initialize_states(batch_size)

    encoder_output,state_h,state_c=encoder(input_sequence,initial_state)

    assert(encoder_output.shape==(batch_size,input_length,lstm_size) and state_h.
shape==(batch_size,lstm_size) and state_c.shape==(batch_size,lstm_size))
    return True
print(grader_check_encoder())
```

```
True
```

In [ ]:
```
class Decoder(tf.keras.Model):
    '''
    Encoder model -- That takes a input sequence and returns output sequence
    '''

    def __init__(self,out_vocab_size,embedding_size,lstm_size,input_length):
        super().__init__()
        self.vocab_size = out_vocab_size
        self.embedding_size = embedding_size
        self.lstm_units = lstm_size
        self.input_length = input_length
        #Initialize Embedding layer
        #Intialize Decoder LSTM layer
        #print(self.vocab_size)
        self.embedding = Embedding(input_dim=self.vocab_size, output_dim=self.emb
edding_size, mask_zero=True,
                                  input_length=self.input_length, name='decoder_e
mbed_layer')
        self.lstm = LSTM(units=self.lstm_units, return_state=True, return_sequenc
es=True, name='decoder_LSTM')


    def call(self,input_sequence,initial_states):
        '''
          This function takes a sequence input and the initial states of the enco
der.
          Pass the input_sequence input to the Embedding layer, Pass the embeddin
g layer ouput to decoder_lstm

          returns -- decoder_output,decoder_final_state_h,decoder_final_state_c
        '''
        # generate embeddings
        input_sequence = self.embedding(input_sequence)
        # pass the input to the LSTM
        all_h_state, decoder_final_h_state, decoder_final_c_state = self.lstm(inp
uts=input_sequence, initial_state=initial_states)

        return all_h_state, decoder_final_h_state, decoder_final_c_state
```

Grader function - 2

In [ ]:
```
def grader_decoder():
    '''
        out_vocab_size: Unique words of the target language,
        embedding_size: output embedding dimension for each word after embedding
 layer,
        dec_units: Number of lstm units in decoder,
        input_length: Length of the input sentence,
        batch_size


    '''
    out_vocab_size=13
    embedding_dim=12
    input_length=10
    dec_units=16
    batch_size=32

    target_sentences=tf.random.uniform(shape=(batch_size,input_length),maxval=10,
minval=0,dtype=tf.int32)
    encoder_output=tf.random.uniform(shape=[batch_size,input_length,dec_units])
    state_h=tf.random.uniform(shape=[batch_size,dec_units])
    state_c=tf.random.uniform(shape=[batch_size,dec_units])
    states=[state_h,state_c]
    decoder=Decoder(out_vocab_size, embedding_dim, dec_units,input_length )
    output,_,_=decoder(target_sentences, states)
    assert(output.shape==(batch_size,input_length,dec_units))
    return True

print(grader_decoder())
```

```
True
```

In [ ]:
```
class Encoder_decoder(Model):

    def __init__(self, inp_vocab_size, out_vocab_size, enc_embedding_size, dec_em
bedding_size, enc_lstm_size, dec_lstm_size, enc_input_length, dec_input_length):
        super().__init__()
        #Create encoder object
        #Create decoder object
        #Intialize Dense layer(out_vocab_size) with activation='softmax'

        self.enc_vocab_size = inp_vocab_size+1
        self.dec_vocab_size = out_vocab_size+1
        self.out_vocab_size = out_vocab_size
        self.enc_embedding_size = enc_embedding_size
        self.dec_embedding_size = dec_embedding_size
        self.enc_lstm_size = enc_lstm_size
        self.dec_lstm_size = dec_lstm_size
        self.enc_input_length = enc_input_length
        self.dec_input_length = dec_input_length

        self.encoder = Encoder(self.enc_vocab_size,self.enc_embedding_size,self.e
nc_lstm_size,self.enc_input_length)
        self.decoder = Decoder(self.dec_vocab_size,self.dec_embedding_size,self.d
ec_lstm_size,self.dec_input_length)
        self.decoder_dense=TimeDistributed( Dense(self.dec_vocab_size,activation=
'softmax'))

    def call(self, input_sequence):
        '''
        A. Pass the input sequence to Encoder layer -- Return encoder_output,enco
der_final_state_h,encoder_final_state_c
        B. Pass the target sequence to Decoder layer with intial states as encode
r_final_state_h,encoder_final_state_C
        C. Pass the decoder_outputs into Dense layer

        Return decoder_outputs
        '''

        enc_ip_sequence, dec_ip_sequence =  input_sequence[0], input_sequence[1]
        initial_state=self.encoder.initialize_states(enc_ip_sequence.shape[0])

        # passig the input to the encoder
        encoder_output,encoder_final_state_h,encoder_final_state_c=self.encoder(e
nc_ip_sequence, initial_state)
        # passing the encoder final states to the decoder
        decoder_output,decoder_final_state_h,decoder_final_state_c=self.decoder(d
ec_ip_sequence, [encoder_final_state_h, encoder_final_state_c])
        # passin the decoder output to the Time distributed dense layer
        dense_output= self.decoder_dense(decoder_output)

        return dense_output
```

In [ ]:
```
# class Dataset:
#     def __init__(self, data, tknizer_ita, tknizer_eng, enc_max_len, dec_max_le
n):
#         self.encoder_inps = data['italia_enc'].values
#         self.decoder_inps = data['english_in'].values
#         self.decoder_outs = data['english_op'].values
#         self.tknizer_eng = tknizer_eng
#         self.tknizer_ita = tknizer_ita
#         self.enc_max_len = enc_max_len
#         self.dec_max_len = dec_max_len

#     def __getitem__(self, i):
#         self.encoder_seq = self.tknizer_ita.texts_to_sequences([self.encoder_in
ps[i]]) # need to pass list of values
#         self.decoder_inp_seq = self.tknizer_eng.texts_to_sequences([self.decode
r_inps[i]])
#         self.decoder_out_seq = self.tknizer_eng.texts_to_sequences([self.decode
r_outs[i]])

#         self.encoder_seq = pad_sequences(self.encoder_seq, maxlen=self.enc_max_
len, dtype='int32', padding='post')
#         self.decoder_inp_seq = pad_sequences(self.decoder_inp_seq, maxlen=self.
dec_max_len, dtype='int32', padding='post')
#         self.decoder_out_seq = pad_sequences(self.decoder_out_seq, maxlen=self.
dec_max_len, dtype='int32', padding='post')
#         return self.encoder_seq, self.decoder_inp_seq, self.decoder_out_seq

#     def __len__(self): # your model.fit_gen requires this function
#         return len(self.encoder_inps)


# class Dataloder(tf.keras.utils.Sequence):
#     def __init__(self, dataset, batch_size=1):
#         super().__init__()
#         self.dataset = dataset
#         self.batch_size = batch_size
#         self.indexes = np.arange(len(self.dataset.encoder_inps))


#     def __getitem__(self, i):
#         start = i * self.batch_size
#         stop = (i + 1) * self.batch_size
#         data = []
#         for j in range(start, stop):
#             data.append(self.dataset[j])


#         batch = [np.squeeze(np.stack(samples, axis=1), axis=0) for samples in z
ip(*data)]
#         # we are creating data like ([italian, english_inp], english_out) these
are already converted into seq
#         return tuple([[batch[0],batch[1]],batch[2]])

#     def __len__(self):  # your model.fit_gen requires this function
#         return len(self.indexes) // self.batch_size

#     def on_epoch_end(self):

#         self.indexes = np.random.permutation(self.indexes)
```

In [ ]:
```
# train_dataset = Dataset(train, tokenizer_ita, tokenizer_eng, 20, 20)
# test_dataset  = Dataset(validation, tokenizer_ita, tokenizer_eng, 20, 20)
# #print(train_dataset)

# train_dataloader = Dataloder(train_dataset, batch_size=1024)
# test_dataloader = Dataloder(test_dataset, batch_size=1024)
```

In [ ]:
```
# train_data_set = [ (i,j,k) for i,j,k in zip(train_italia, train_eng_in, train_e
ng_op) ]
# validation_data_set = [ (i,j,k) for i,j,k in zip(validation_italia, validation_
eng_in, validation_eng_op) ]
# from numpy import random
# def generator(batch_size):
#     print(batch_size)
#   # Create empty arrays to contain batch of features and labels#
#     train_enc = np.zeros((batch_size, train_italia.shape[1]))
#     train_dec_in = np.zeros((batch_size, train_eng_in.shape[1]))
#     train_dec_op = np.zeros((batch_size, train_eng_op.shape[1]))


#     for i in range(batch_size):
#      # choose random index in features
#         index = random.choice(train_italia.shape[0],1)
#         train_enc[i] = train_italia[index]
#         train_dec_in[i] = train_eng_in[index]
#         train_dec_op[i] = train_eng_op[index]

#     return tuple([[train_enc,train_dec_in], train_dec_op])
```

In [ ]:
```
len(tokenizer_eng.word_index.keys())
```

Out[ ]: 13127

In [ ]:
```
log_dir="logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = TensorBoard(log_dir=log_dir,histogram_freq=1, write_graph=
True)
es = EarlyStopping(monitor='val_loss', mode='min', patience=3, verbose=1)
m_check = ModelCheckpoint(
    'best_model',
    monitor="loss",
    save_best_only=True,
    mode="auto",
    save_freq="epoch",
)
```

In [ ]:
```
# # Create an object of encoder_decoder Model class,
# # Compile the model and fit the model

enc_ip_length = train_italia[0].shape[0]
dec_ip_length = train_eng_in[0].shape[0]

train_steps=train_italia.shape[0]//1024
valid_steps=validation_italia.shape[0]//1024

model=Encoder_decoder(len(tokenizer_ita.word_index), len(tokenizer_eng.word_index
), 100, 100, 128, 128, enc_ip_length, dec_ip_length)

optimizer = tf.keras.optimizers.Adam()

model.compile(optimizer=optimizer,loss='sparse_categorical_crossentropy', metrics
=['accuracy'])

model.fit([train_italia,train_eng_in], train_eng_op, steps_per_epoch=train_steps,
callbacks = [m_check, tensorboard_callback, es], validation_data= ([validation_it
alia, validation_eng_in], validation_eng_op), batch_size=1024, epochs=50)
```

```
Epoch 1/50
301/301 [==============================] - 131s 415ms/step - loss: 2.2341 - accu
racy: 0.1759 - val_loss: 1.6165 - val_accuracy: 0.2278
Epoch 2/50
301/301 [==============================] - 123s 407ms/step - loss: 1.5577 - accu
racy: 0.2499 - val_loss: 1.4066 - val_accuracy: 0.3108
Epoch 3/50
301/301 [==============================] - 123s 407ms/step - loss: 1.3578 - accu
racy: 0.3296 - val_loss: 1.2366 - val_accuracy: 0.3686
Epoch 4/50
301/301 [==============================] - 122s 407ms/step - loss: 1.1960 - accu
racy: 0.3817 - val_loss: 1.1133 - val_accuracy: 0.4181
Epoch 5/50
301/301 [==============================] - 123s 407ms/step - loss: 1.0787 - accu
racy: 0.4350 - val_loss: 1.0150 - val_accuracy: 0.4629
Epoch 6/50
301/301 [==============================] - 123s 407ms/step - loss: 0.9803 - accu
racy: 0.4770 - val_loss: 0.9338 - val_accuracy: 0.4974
Epoch 7/50
301/301 [==============================] - 122s 406ms/step - loss: 0.8978 - accu
racy: 0.5135 - val_loss: 0.8582 - val_accuracy: 0.5340
Epoch 8/50
301/301 [==============================] - 122s 407ms/step - loss: 0.8190 - accu
racy: 0.5493 - val_loss: 0.7938 - val_accuracy: 0.5634
Epoch 9/50
301/301 [==============================] - 122s 407ms/step - loss: 0.7519 - accu
racy: 0.5795 - val_loss: 0.7385 - val_accuracy: 0.5865
Epoch 10/50
301/301 [==============================] - 122s 407ms/step - loss: 0.6923 - accu
racy: 0.6052 - val_loss: 0.6901 - val_accuracy: 0.6086
Epoch 11/50
301/301 [==============================] - 122s 407ms/step - loss: 0.6380 - accu
racy: 0.6295 - val_loss: 0.6447 - val_accuracy: 0.6299
Epoch 12/50
301/301 [==============================] - 123s 407ms/step - loss: 0.5900 - accu
racy: 0.6519 - val_loss: 0.6056 - val_accuracy: 0.6476
Epoch 13/50
301/301 [==============================] - 123s 407ms/step - loss: 0.5477 - accu
racy: 0.6717 - val_loss: 0.5694 - val_accuracy: 0.6669
Epoch 14/50
301/301 [==============================] - 123s 407ms/step - loss: 0.5089 - accu
racy: 0.6914 - val_loss: 0.5382 - val_accuracy: 0.6812
Epoch 15/50
301/301 [==============================] - 122s 407ms/step - loss: 0.4718 - accu
racy: 0.7099 - val_loss: 0.5094 - val_accuracy: 0.6966
Epoch 16/50
301/301 [==============================] - 123s 407ms/step - loss: 0.4389 - accu
racy: 0.7272 - val_loss: 0.4844 - val_accuracy: 0.7102
Epoch 17/50
301/301 [==============================] - 123s 408ms/step - loss: 0.4122 - accu
racy: 0.7416 - val_loss: 0.4624 - val_accuracy: 0.7234
Epoch 18/50
301/301 [==============================] - 123s 407ms/step - loss: 0.3833 - accu
racy: 0.7579 - val_loss: 0.4402 - val_accuracy: 0.7348
Epoch 19/50
301/301 [==============================] - 122s 407ms/step - loss: 0.3597 - accu
racy: 0.7709 - val_loss: 0.4226 - val_accuracy: 0.7444
Epoch 20/50
301/301 [==============================] - 122s 407ms/step - loss: 0.3376 - accu
racy: 0.7834 - val_loss: 0.4046 - val_accuracy: 0.7552
Epoch 21/50
301/301 [==============================] - 122s 407ms/step - loss: 0.3190 - accu
racy: 0.7942 - val_loss: 0.3902 - val_accuracy: 0.7649
Epoch 22/50
301/301 [==============================] - 122s 407ms/step - loss: 0.2996 - accu
racy: 0.8055 - val_loss: 0.3776 - val_accuracy: 0.7707
Epoch 23/50
301/301 [==============================] - 122s 407ms/step - loss: 0.2839 - accu
racy: 0.8146 - val_loss: 0.3642 - val_accuracy: 0.7789
```

```
Epoch 24/50
301/301 [==============================] - 123s 407ms/step - loss: 0.2685 - accu
racy: 0.8236 - val_loss: 0.3534 - val_accuracy: 0.7846
Epoch 25/50
301/301 [==============================] - 122s 407ms/step - loss: 0.2557 - accu
racy: 0.8309 - val_loss: 0.3437 - val_accuracy: 0.7907
Epoch 26/50
301/301 [==============================] - 123s 407ms/step - loss: 0.2422 - accu
racy: 0.8392 - val_loss: 0.3347 - val_accuracy: 0.7952
Epoch 27/50
301/301 [==============================] - 122s 406ms/step - loss: 0.2311 - accu
racy: 0.8459 - val_loss: 0.3284 - val_accuracy: 0.7998
Epoch 28/50
301/301 [==============================] - 122s 406ms/step - loss: 0.2200 - accu
racy: 0.8526 - val_loss: 0.3209 - val_accuracy: 0.8049
Epoch 29/50
301/301 [==============================] - 122s 406ms/step - loss: 0.2101 - accu
racy: 0.8588 - val_loss: 0.3137 - val_accuracy: 0.8084
Epoch 30/50
301/301 [==============================] - 122s 407ms/step - loss: 0.2038 - accu
racy: 0.8631 - val_loss: 0.3071 - val_accuracy: 0.8134
Epoch 31/50
301/301 [==============================] - 122s 406ms/step - loss: 0.1927 - accu
racy: 0.8694 - val_loss: 0.3022 - val_accuracy: 0.8172
Epoch 32/50
301/301 [==============================] - 122s 407ms/step - loss: 0.1854 - accu
racy: 0.8745 - val_loss: 0.2972 - val_accuracy: 0.8200
Epoch 33/50
301/301 [==============================] - 122s 407ms/step - loss: 0.1780 - accu
racy: 0.8788 - val_loss: 0.2927 - val_accuracy: 0.8229
Epoch 34/50
301/301 [==============================] - 122s 406ms/step - loss: 0.1728 - accu
racy: 0.8824 - val_loss: 0.2889 - val_accuracy: 0.8253
Epoch 35/50
301/301 [==============================] - 122s 406ms/step - loss: 0.1655 - accu
racy: 0.8868 - val_loss: 0.2846 - val_accuracy: 0.8281
Epoch 36/50
301/301 [==============================] - 122s 406ms/step - loss: 0.1610 - accu
racy: 0.8898 - val_loss: 0.2815 - val_accuracy: 0.8302
Epoch 37/50
301/301 [==============================] - 122s 405ms/step - loss: 0.1543 - accu
racy: 0.8937 - val_loss: 0.2779 - val_accuracy: 0.8330
Epoch 38/50
301/301 [==============================] - 122s 405ms/step - loss: 0.1493 - accu
racy: 0.8967 - val_loss: 0.2757 - val_accuracy: 0.8347
Epoch 39/50
301/301 [==============================] - 122s 405ms/step - loss: 0.1453 - accu
racy: 0.8995 - val_loss: 0.2716 - val_accuracy: 0.8376
Epoch 40/50
301/301 [==============================] - 122s 404ms/step - loss: 0.1398 - accu
racy: 0.9031 - val_loss: 0.2690 - val_accuracy: 0.8395
Epoch 41/50
301/301 [==============================] - 122s 405ms/step - loss: 0.1358 - accu
racy: 0.9056 - val_loss: 0.2670 - val_accuracy: 0.8412
Epoch 42/50
301/301 [==============================] - 122s 405ms/step - loss: 0.1313 - accu
racy: 0.9083 - val_loss: 0.2661 - val_accuracy: 0.8406
Epoch 43/50
301/301 [==============================] - 122s 405ms/step - loss: 0.1272 - accu
racy: 0.9111 - val_loss: 0.2635 - val_accuracy: 0.8435
Epoch 44/50
301/301 [==============================] - 122s 405ms/step - loss: 0.1242 - accu
racy: 0.9129 - val_loss: 0.2620 - val_accuracy: 0.8453
Epoch 45/50
301/301 [==============================] - 122s 405ms/step - loss: 0.1195 - accu
racy: 0.9164 - val_loss: 0.2588 - val_accuracy: 0.8471
Epoch 46/50
301/301 [==============================] - 122s 405ms/step - loss: 0.1165 - accu
racy: 0.9179 - val_loss: 0.2578 - val_accuracy: 0.8482
Epoch 47/50
```

```
301/301 [==============================] - 122s 405ms/step - loss: 0.1124 - accu
racy: 0.9210 - val_loss: 0.2561 - val_accuracy: 0.8499
Epoch 48/50
301/301 [==============================] - 122s 405ms/step - loss: 0.1095 - accu
racy: 0.9229 - val_loss: 0.2544 - val_accuracy: 0.8519
Epoch 49/50
301/301 [==============================] - 122s 405ms/step - loss: 0.1078 - accu
racy: 0.9241 - val_loss: 0.2530 - val_accuracy: 0.8527
Epoch 50/50
301/301 [==============================] - 122s 405ms/step - loss: 0.1047 - accu
racy: 0.9257 - val_loss: 0.2520 - val_accuracy: 0.8532
```

Out[ ]:
```
<tensorflow.python.keras.callbacks.History at 0x7f90d1fe4518>
```

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

```
Model: "encoder_decoder_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
encoder_6 (Encoder)          multiple                  2822648   
_________________________________________________________________
decoder_6 (Decoder)          multiple                  1430048   
_________________________________________________________________
time_distributed_5 (TimeDist multiple                  1693512   
=================================================================
Total params: 5,946,208
Trainable params: 5,946,208
Non-trainable params: 0
_________________________________________________________________
```

In [79]:
```
%load_ext tensorboard
```

In [80]:
```
%tensorboard --logdir /content/logs/fit/20210115-011400
```

In [ ]:
```
def predict(input_sentence):
    '''
    A. Given input sentence, convert the sentence into integers using tokenizer u
sed earlier
    B. Pass the input_sequence to encoder. we get encoder_outputs, last time step
hidden and cell state
    C. Initialize index of <start> as input to decoder. and encoder final states
 as input_states to decoder
    D. till we reach max_length of decoder or till the model predicted word <end
>:
         predicted_out,state_h,state_c=model.layers[1](dec_input,states)
         pass the predicted_out to the dense layer
         update the states=[state_h,state_c]
         And get the index of the word with maximum probability of the dense laye
r output, using the tokenizer(word index) get the word and then store it in a str
ing.
         Update the input_to_decoder with current predictions
    F. Return the predicted sentence
    '''

    # padding the input
    padded_input_sentences = pad_sequences(tokenizer_ita.texts_to_sequences(input
_sentence), padding='post', maxlen=20)

    # generating initial states for LSTM layers
    initializer = tf.keras.initializers.Zeros()
    initial_state_h = initializer(shape=(padded_input_sentences.shape[0],128))
    initial_state_c = initializer(shape=(padded_input_sentences.shape[0],128))

    # passing the input to the encoder layer along with the states
    encoder_output,encoder_final_state_h,encoder_final_state_c = model.layers[0](
padded_input_sentences,[initial_state_h, initial_state_c])
    # grouping the encoder final states to be passed to the decoder
    decoder_ip_states = [encoder_final_state_h,encoder_final_state_c]

    word_count = 0
    output_sentence =''
    # initial input to be passed to the decoder
    decoder_ip_sequence = np.array([[tokenizer_eng.word_index['<start>']]])

    while True:

        # passing the input sequence and states to the decoder
        decoder_output, decoder_final_state_h, decoder_final_state_c = model.laye
rs[1](decoder_ip_sequence, decoder_ip_states)
        # passing the decoder output to the dense layer
        word_prob = model.layers[2](decoder_output)
        # getting the index of maximum probability
        idx = np.argmax(word_prob,-1)
        # finding the word at the index generated in previous step

        word = tokenizer_eng.index_word[idx[0][0]]

        if word_count>=20 or word=='<end>':
            break
        else:
            # increasing the word count
            word_count+=1
            # concat the output word with the output sentence
            output_sentence += ' '+word
            # tokenize the current word output by the decoder to again pass it as
input for further prediction
            decoder_ip_sequence= np.array([[tokenizer_eng.word_index[word.strip
()]]])
            # group the final states of decoder in order to pass them as input st
ate for next input
            decoder_ip_states = [decoder_final_state_h,decoder_final_state_c]

    return output_sentence.strip()
```

In [ ]:
```
validation.head()
```

Out[ ]:

	italia_enc	english_in	english_op
258005	<start> il tuo inglese è migliorato molto <end>	<start> your english has improved a lot	your english has improved a lot <end>
132030	<start> mi sento male per quel tizio <end>	<start> i feel bad for that guy	i feel bad for that guy <end>
186107	<start> tom disse che non lo avrebbe fatto <end>	<start> tom said he would not do it	tom said he would not do it <end>
253635	<start> questo non ha nulla a che fare con tom...	<start> this has nothing to do with tom	this has nothing to do with tom <end>
231767	<start> quale materia vi piace di più <end>	<start> what subject do you like best	what subject do you like best <end>

In [ ]:
```
# Predict on 1000 random sentences on test data and calculate the average BLEU sc
ore of these sentences.
# https://www.nltk.org/_modules/nltk/translate/bleu_score.html

import nltk.translate.bleu_score as bleu

bleu_score= 0

italian_sentence = validation['italia_enc'][:1000]
english_conversion = [ sentence.strip('<start>').strip(' ') for sentence in valid
ation['english_in'][:1000]]
english_conversion
for sentence, actual_translation in zip(italian_sentence,english_conversion):
    # the original

    # traslated using model
    translation = predict([sentence])
    # print(sentence)
    # print(actual_translation)
    # print(translation)
    bleu_score += bleu.sentence_bleu(actual_translation.split(), translation.spli
t())

print(bleu_score/1000)
```

```
/usr/local/lib/python3.6/dist-packages/nltk/translate/bleu_score.py:490: UserWar
ning:
Corpus/Sentence contains 0 counts of 2-gram overlaps.
BLEU scores might be undesirable; use SmoothingFunction().
  warnings.warn(_msg)

0.2541836128200154
```

In [ ]:
```
print('italian sentence =>', 'tom disse che non lo avrebbe fatto')
print('actual translation =>', 'tom said he would not do it')
print('predicted translation =>', predict(['<start> tom disse che non lo avrebbe
 fatto <end>']))
```

```
italian sentence => tom disse che non lo avrebbe fatto
actual translation => tom said he would not do it
predicted translation => tom said he would not do that
```

Task -2: Including Attention mechanisum

1. Use the preprocessed data from Task-1
2. You have to implement an Encoder and Decoder architecture with attention as discussed in the reference notebook.
   - Encoder - with 1 layer LSTM
   - Decoder - with 1 layer LSTM
   - attention - (Please refer the [**reference notebook** (https://drive.google.com/file/d/1z_bnc-3aubKawbR6q8wyI6Mh5ho2R1aZ/view?usp=sharing)](https://drive.google.com/file/d/1z_bnc-3aubKawbR6q8wyI6Mh5ho2R1aZ/view?usp=sharing) to know more about the attention mechanism.)
3. In Global attention, we have 3 types of scoring functions(as discussed in the reference notebook). As a part of this assignment **you need to create 3 models for each scoring function**

   Here, score is referred as a *content-based* function for which we consider three different alternatives:

   $$\text{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s) = \begin{cases} \boldsymbol{h}_t^\top \bar{\boldsymbol{h}}_s & \textit{dot} \\ \boldsymbol{h}_t^\top \boldsymbol{W_a} \bar{\boldsymbol{h}}_s & \textit{general} \\ \boldsymbol{v}_a^\top \tanh\left(\boldsymbol{W_a}[\boldsymbol{h}_t; \bar{\boldsymbol{h}}_s]\right) & \textit{concat} \end{cases}$$

   - In model 1 you need to implemnt "dot" score function
   - In model 2 you need to implemnt "general" score function
   - In model 3 you need to implemnt "concat" score function.

   Please do add the markdown titles for each model so that we can have a better look at the code and verify.
4. It is mandatory to train the model with simple model.fit() only, Donot train the model with custom GradientTape()
5. Using attention weights, you can plot the attention plots, please plot those for 2-3 examples. You can check about those in [this (https://www.tensorflow.org/tutorials/text/nmt_with_attention#translate)](https://www.tensorflow.org/tutorials/text/nmt_with_attention#translate)
6. The attention layer has to be written by yourself only. The main objective of this assignment is to read and implement a paper on yourself so please do it yourself.
7. Please implement the class **onestepdecoder** as mentioned in the assignment instructions.
8. You can use any tf.Keras highlevel API's to build and train the models. Check the reference notebook for better understanding.
9. Use BLEU score as metric to evaluate your model. You can use any loss function you need.
10. You have to use Tensorboard to plot the Graph, Scores and histograms of gradients.
11. Resources: a. Check the reference notebook b. [Resource 1 (https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/)](https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/) c. [Resource 2 (https://www.tensorflow.org/tutorials/text/nmt_with_attention)](https://www.tensorflow.org/tutorials/text/nmt_with_attention) d. [Resource 3 (https://stackoverflow.com/questions/44238154/what-is-the-difference-between-luong-attention-and-bahdanau-attention#:~:text=Luong%20attention%20used%20top%20hidden,hidden%20state%20at%20time%20t.)](https://stackoverflow.com/questions/44238154/what-is-the-difference-between-luong-attention-and-bahdanau-attention#:~:text=Luong%20attention%20used%20top%20hidden,hidden%20state%20at%20time%20t.)

Implement custom encoder decoder and attention layers

Encoder

In [ ]:
```
class Encoder(tf.keras.Model):
    '''
    Encoder model -- That takes a input sequence and returns output sequence
    '''

    def __init__(self,inp_vocab_size,embedding_size,lstm_size,input_length):

        #Initialize Embedding layer
        #Intialize Encoder LSTM layer
        super().__init__()
        self.vocab_size = inp_vocab_size
        self.embedding_size = embedding_size
        self.lstm_units = lstm_size
        self.input_length = input_length
        #Initialize Embedding layer
        #Intialize Encoder LSTM layer
        self.embedding = Embedding(input_dim=self.vocab_size, output_dim=self.emb
edding_size,
                                   input_length=self.input_length, mask_zero=True,
name='encoder_embed_layer')
        # pass the embeddings and states to LSTM
        self.lstm = LSTM(units=self.lstm_units, return_state=True, return_sequenc
es=True, name='encoder_LSTM')

    def call(self,input_sequence,states):
        '''
          This function takes a sequence input and the initial states of the enco
der.
          Pass the input_sequence input to the Embedding layer, Pass the embeddin
g layer ouput to encoder_lstm
          returns -- All encoder_outputs, last time steps hidden and cell state
        '''
        # generate the embeddings for input sequence
        input_sequence = self.embedding(input_sequence)
        # pass the input sequece to the LSTM
        all_h_state, encoder_final_h_state, encoder_final_c_state = self.lstm(inp
uts=input_sequence, initial_state=states)

        return all_h_state, encoder_final_h_state, encoder_final_c_state

    def initialize_states(self,batch_size):
        '''
        Given a batch size it will return intial hidden state and intial cell sta
te.
        If batch size is 32- Hidden state is zeros of size [32,lstm_units], cell
 state zeros is of size [32,lstm_units]
        '''
        if batch_size==None:
            return
        initializer = tf.keras.initializers.Zeros()
        initial_state_h = initializer(shape=(batch_size,self.lstm_units))
        initial_state_c = initializer(shape=(batch_size,self.lstm_units))
        return [initial_state_h, initial_state_c]
```

Grader function - 1

In [ ]:
```
def grader_check_encoder():

    '''
        vocab-size: Unique words of the input language,
        embedding_size: output embedding dimension for each word after embedding
 layer,
        lstm_size: Number of lstm units in encoder,
        input_length: Length of the input sentence,
        batch_size
    '''

    vocab_size=10
    embedding_size=20
    lstm_size=32
    input_length=10
    batch_size=16
    encoder=Encoder(vocab_size,embedding_size,lstm_size,input_length)
    input_sequence=tf.random.uniform(shape=[batch_size,input_length],maxval=vocab
_size,minval=0,dtype=tf.int32)
    initial_state=encoder.initialize_states(batch_size)
    encoder_output,state_h,state_c=encoder(input_sequence,initial_state)

    assert(encoder_output.shape==(batch_size,input_length,lstm_size) and state_h.
shape==(batch_size,lstm_size) and state_c.shape==(batch_size,lstm_size))
    return True
print(grader_check_encoder())
```

```
True
```

Attention

In [ ]:
```
class Attention(tf.keras.layers.Layer):
    '''
    Class the calculates score based on the scoring_function using Bahdanu attent
ion mechanism.
    '''
    def __init__(self,scoring_function, att_units):
        super().__init__()
        self.initializer = tf.keras.initializers.Zeros()
        # Please go through the reference notebook and research paper to complete
the scoring functions
        self.scoring_function = scoring_function
        self.att_units = att_units
        self.initializer = tf.keras.initializers.Zeros()
        if self.scoring_function=='dot':

            # Intialize variables needed for Dot score function here
            return
        if scoring_function == 'general':
          # Intialize variables needed for General score function here
            self.W = tf.Variable(self.initializer(shape=(self.att_units,self.att_
units)))
            return
        elif scoring_function == 'concat':
            # Intialize variables needed for Concat score function here
            self.e_dense = Dense(self.att_units)
            self.d_dense = Dense(self.att_units)
            self.op_dense = Dense(1)


    def call(self,decoder_hidden_state,encoder_output):
        '''
          Attention mechanism takes two inputs current step -- decoder_hidden_sta
te and all the encoder_outputs.
          * Based on the scoring function we will find the score or similarity be
tween decoder_hidden_state and encoder_output.
        Multiply the score function with your encoder_outputs to get the context
 vector.
        Function returns context vector and attention weights(softmax - scores)
        '''

        if self.scoring_function == 'dot':
            # Implement Dot score function here

            # performing the dot product b/w hidden state of decoder add all the
 encoder states
            similarities = dot([decoder_hidden_state, encoder_output], axes=[1,2
], normalize=False)
            # passing the similarities to the softmax in order to get the attenti
on weights
            attention_weights = tf.keras.activations.softmax(similarities)
            # multiplying the attention weights with respective encoder states an
d adding them
            context_vector = dot([attention_weights, encoder_output], axes=1)
            # reshaping the attention weights vector
            attention_weights = tf.reshape(attention_weights, shape=(tf.shape(att
ention_weights)[0],tf.shape(attention_weights)[1],1))

            return context_vector, attention_weights

        elif self.scoring_function == 'general':
            # Implement General score function here

            # multiply encoder outputs with the matrix of shape (att_units x att_
units)
            output = tf.linalg.matmul(encoder_output, self.W)
            # reshapig the decoder hidden state
            decoder_hidden_state = tf.reshape(decoder_hidden_state, shape=(tf.sha
pe(decoder_hidden_state)[0],tf.shape(decoder_hidden_state)[1],1))
```

```
            # multiplying the output generated in previous state with decoder hid
den state
            similarities = tf.linalg.matmul(output,decoder_hidden_state)
            # passing the similarities to dense layer to generatee attention
            attention_weights = tf.keras.activations.softmax(similarities, axis=1
)
            # generating the context vector
            context_vector = tf.linalg.matmul(tf.reshape(attention_weights,shape
=(tf.shape(attention_weights)[0],1,tf.shape(attention_weights)[1])),encoder_outpu
t)

            return tf.compat.v1.squeeze(context_vector, axis=1), attention_weight
s


        elif self.scoring_function == 'concat':
            # Implement General score function here

            # passing the encoder output and decoder hidden state to separate den
se layer and then adding the output
            addition_op = tf.keras.layers.Add()([self.d_dense(decoder_hidden_stat
e), self.e_dense(encoder_output)])
            # apply the tah on the output in previous states
            tanh_op = tf.keras.activations.tanh(addition_op)
            # passing each of the vectors to the dense layer to generate a single
value i-e similarities
            similarities = self.op_dense(tanh_op)
            # passing the similarities to the dense layer
            attention_weights = tf.keras.activations.softmax(similarities, axis=1
)
            # multiply attention weights with respective encoder output vectors a
nd generate the context vector
            context_vector = tf.linalg.matmul(tf.reshape(attention_weights,shape
= (tf.shape(attention_weights)[0],1,tf.shape(attention_weights)[1])),encoder_outp
ut)
            return tf.compat.v1.squeeze(context_vector, axis=1), attention_weight
s
```

Grader function - 2

In [ ]:
```
def grader_check_attention(scoring_fun):

    '''
        att_units: Used in matrix multiplications for scoring functions,
        input_length: Length of the input sentence,
        batch_size
    '''

    input_length=10
    batch_size=16
    att_units=32

    state_h=tf.random.uniform(shape=[batch_size,att_units])
    encoder_output=tf.random.uniform(shape=[batch_size,input_length,att_units])
    attention=Attention(scoring_fun,att_units)
    context_vector,attention_weights=attention(state_h,encoder_output)
    assert(context_vector.shape==(batch_size,att_units) and attention_weights.sha
pe==(batch_size,input_length,1))
    return True
print(grader_check_attention('dot'))
print(grader_check_attention('general'))
print(grader_check_attention('concat'))
```

```
True
True
True
```

OneStepDecoder

In [ ]:
```
class One_Step_Decoder(tf.keras.Model):
    def __init__(self,tar_vocab_size, embedding_dim, input_length, dec_units ,sco
re_fun ,att_units):
        super().__init__()
      # Initialize decoder embedding layer, LSTM and any other objects needed
        self.vocab_size = tar_vocab_size
        self.embedding_dim = embedding_dim
        self.input_length = input_length
        self.dec_units = dec_units
        self.score_fun = score_fun
        self.att_units = att_units
        self.embedding = Embedding(input_dim=self.vocab_size, output_dim=self.emb
edding_dim,
                                   input_length=1, mask_zero=True, name='decoder_e
mbed_layer')
        self.lstm = LSTM(units=self.dec_units, return_state=True, return_sequence
s=True, name='decoder_LSTM')
        self.attention = Attention(score_fun,att_units)
        #self.concat = Concatenate(axis=1)
        self.dense = Dense(self.vocab_size)

    def call(self, input_to_decoder, encoder_output, state_h, state_c):
        '''
          One step decoder mechanisim step by step:
         A. Pass the input_to_decoder to the embedding layer and then get the ou
tput(batch_size,1,embedding_dim)
         B. Using the encoder_output and decoder hidden state, compute the conte
xt vector.
         C. Concat the context vector with the step A output
         D. Pass the Step-C output to LSTM/GRU and get the decoder output and st
ates(hidden and cell state)
         E. Pass the decoder output to dense layer(vocab size) and store the res
ult into output.
         F. Return the states from step D, output from Step E, attention weights
from Step -B
        '''

        # generating embeddings for the input
        embedded_decoder_input = self.embedding(input_to_decoder)

        # getting the context vector and attention weights
        context_vector, attention_weights = self.attention(state_h,encoder_output
)

        # concat the decoder input and the cotext vector
        decoder_ip_lstm = tf.concat([context_vector,tf.reshape(embedded_decoder_i
nput, shape=(-1,embedded_decoder_input.shape[2]))],axis=1)

        # reshaping the decoder input
        decoder_ip_lstm = tf.reshape(decoder_ip_lstm, shape=(-1, 1, decoder_ip_ls
tm.shape[1]))
        # passing the decoder input to LSTM along with the states

        decoder_all_h_state, decoder_final_h_state, decoder_final_c_state= self.l
stm(decoder_ip_lstm, [state_h, state_c])
        # passing the LSTM output to dense layer

        vocab_selected = self.dense(decoder_all_h_state)

        return tf.compat.v1.squeeze(vocab_selected), decoder_final_h_state, decod
er_final_c_state , attention_weights, context_vector
```

Grader function - 3

In [ ]:
```
def grader_onestepdecoder(score_fun):

    '''
        tar_vocab_size: Unique words of the target language,
        embedding_dim: output embedding dimension for each word after embedding l
ayer,
        dec_units: Number of lstm units in decoder,
        att_units: Used in matrix multiplications for scoring functions in attent
ion class,
        input_length: Length of the target sentence,
        batch_size


    '''

    tar_vocab_size=13
    embedding_dim=12
    input_length=10
    dec_units=16
    att_units=16
    batch_size=32
    onestepdecoder=One_Step_Decoder(tar_vocab_size, embedding_dim, input_length,
dec_units ,score_fun ,att_units)
    input_to_decoder=tf.random.uniform(shape=(batch_size,1),maxval=10,minval=0,dt
ype=tf.int32)
    encoder_output=tf.random.uniform(shape=[batch_size,input_length,dec_units])
    state_h=tf.random.uniform(shape=[batch_size,dec_units])
    state_c=tf.random.uniform(shape=[batch_size,dec_units])
    output,state_h,state_c,attention_weights,context_vector=onestepdecoder(input_
to_decoder,encoder_output,state_h,state_c)
    assert(output.shape==(batch_size,tar_vocab_size))
    assert(state_h.shape==(batch_size,dec_units))
    assert(state_c.shape==(batch_size,dec_units))
    assert(attention_weights.shape==(batch_size,input_length,1))
    assert(context_vector.shape==(batch_size,dec_units))
    return True

print(grader_onestepdecoder('dot'))
print(grader_onestepdecoder('general'))
print(grader_onestepdecoder('concat'))
```

```
True
True
True
```

Decoder

In [ ]:
```python
class Decoder(tf.keras.layers.Layer):
    def __init__(self,out_vocab_size, embedding_dim, input_length, dec_units ,score_fun ,att_units):
        super(Decoder, self).__init__()
      #Intialize necessary variables and create an object from the class onestepdecoder
        self.onestepdecoder = One_Step_Decoder(out_vocab_size, embedding_dim, input_length, dec_units ,score_fun ,att_units)
        return

    def loop_cond(self, output_words, timestep, input_to_decoder, encoder_output, decoder_hidden_state, decoder_cell_state):
        return tf.less(timestep, input_to_decoder.shape[1])

    def loop_body(self, output_words, timestep, input_to_decoder, encoder_output, decoder_hidden_state, decoder_cell_state):


        output,decoder_hidden_state, decoder_cell_state,attention_weights,context_vector = self.onestepdecoder(tf.slice(input_to_decoder, [0, timestep], [tf.shape(input_to_decoder)[0], 1]),encoder_output,decoder_hidden_state,decoder_cell_state)

        output_words = output_words.write(timestep,output)

        timestep+=1

        return output_words, timestep, input_to_decoder, encoder_output, decoder_hidden_state, decoder_cell_state

    @tf.function
    def call(self, input_to_decoder,encoder_output,decoder_hidden_state,decoder_cell_state):

        #Initialize an empty Tensor array, that will store the outputs at each and every time step
        #Create a tensor array as shown in the reference notebook

        #Iterate till the length of the decoder input
            # Call onestepdecoder for each token in decoder_input
            # Store the output in tensorarray
        # Return the tensor array

        # initialize the output array to store output
        output_word = tf.TensorArray(tf.float32, size = input_to_decoder.shape[1])

        for timestep in tf.range(0,input_to_decoder.shape[1]):
            # calling the onestep decoder for each word in decoder input sentence.
            output, decoder_hidden_state, decoder_cell_state, attention_weights, context_vector = self.onestepdecoder(tf.slice(input_to_decoder, [0, timestep], [tf.shape(input_to_decoder)[0], 1]),encoder_output,decoder_hidden_state,decoder_cell_state)
            # storing the output word in the array of output words
            output_word = output_word.write(timestep,output)

        output_word = tf.transpose(output_word.stack(), [1,0,2])

        return output_word
```

Grader function - 4

In [ ]:
```
def grader_decoder(score_fun):

    '''
        out_vocab_size: Unique words of the target language,
        embedding_dim: output embedding dimension for each word after embedding l
ayer,
        dec_units: Number of lstm units in decoder,
        att_units: Used in matrix multiplications for scoring functions in attent
ion class,
        input_length: Length of the target sentence,
        batch_size

    '''
    out_vocab_size=13
    embedding_dim=12
    input_length=11
    dec_units=16
    att_units=16
    batch_size=32

    target_sentences=tf.random.uniform(shape=(batch_size,input_length),maxval=10,
minval=0,dtype=tf.int32)
    encoder_output=tf.random.uniform(shape=[batch_size,input_length,dec_units])
    state_h=tf.random.uniform(shape=[batch_size,dec_units])
    state_c=tf.random.uniform(shape=[batch_size,dec_units])

    decoder=Decoder(out_vocab_size, embedding_dim, input_length, dec_units ,score
_fun ,att_units)
    output=decoder(target_sentences,encoder_output, state_h, state_c)
    assert(output.shape==(batch_size,input_length,out_vocab_size))
    return True

print(grader_decoder('dot'))
print(grader_decoder('general'))
print(grader_decoder('concat'))
```

```
True
WARNING:tensorflow:5 out of the last 5 calls to <function Decoder.call at 0x7f90
cdfa07b8> triggered tf.function retracing. Tracing is expensive and the excessiv
e number of tracings could be due to (1) creating @tf.function repeatedly in a l
oop, (2) passing tensors with different shapes, (3) passing Python objects inste
ad of tensors. For (1), please define your @tf.function outside of the loop. For
(2), @tf.function has experimental_relax_shapes=True option that relaxes argumen
t shapes that can avoid unnecessary retracing. For (3), please refer to https://
www.tensorflow.org/guide/function#controlling_retracing and https://www.tensorfl
ow.org/api_docs/python/tf/function for  more details.
True
WARNING:tensorflow:6 out of the last 6 calls to <function Decoder.call at 0x7f90
cdb0dc80> triggered tf.function retracing. Tracing is expensive and the excessiv
e number of tracings could be due to (1) creating @tf.function repeatedly in a l
oop, (2) passing tensors with different shapes, (3) passing Python objects inste
ad of tensors. For (1), please define your @tf.function outside of the loop. For
(2), @tf.function has experimental_relax_shapes=True option that relaxes argumen
t shapes that can avoid unnecessary retracing. For (3), please refer to https://
www.tensorflow.org/guide/function#controlling_retracing and https://www.tensorfl
ow.org/api_docs/python/tf/function for  more details.
True
```

Encoder Decoder model

In [ ]:
```
class encoder_decoder(tf.keras.Model):

    def __init__(self, inp_vocab_size, out_vocab_size, enc_embedding_size, dec_em
bedding_size, enc_lstm_size, dec_lstm_size, enc_input_length, dec_input_length, s
core_fun, att_units):
        super().__init__()
        #Intialize objects from encoder decoder
        self.enc_vocab_size = inp_vocab_size+1
        self.dec_vocab_size = out_vocab_size+1
        self.out_vocab_size = out_vocab_size
        self.enc_embedding_size = enc_embedding_size
        self.dec_embedding_size = dec_embedding_size
        self.enc_lstm_size = enc_lstm_size
        self.dec_lstm_size = dec_lstm_size
        self.enc_input_length = enc_input_length
        self.dec_input_length = dec_input_length
        self.score_fun = score_fun
        self.att_units = att_units

        self.encoder = Encoder(self.enc_vocab_size,self.enc_embedding_size,self.e
nc_lstm_size,self.enc_input_length)
        self.decoder = Decoder(self.dec_vocab_size,self.dec_embedding_size,self.d
ec_input_length,self.dec_lstm_size,self.score_fun,self.att_units)

        self.decoder_dense=TimeDistributed( Dense(self.dec_vocab_size))

    def call(self,input_sequence):

        #Intialize encoder states, Pass the encoder_sequence to the embedding lay
er
        # Decoder initial states are encoder final states, Initialize it accordin
gly
        # Pass the decoder sequence,encoder_output,decoder states to Decoder
        # return the decoder output

        # separating the encoder input sequence ad decoder input sequence
        enc_ip_sequence, dec_ip_sequence =  input_sequence[0], input_sequence[1]
        # initializing the encoder input states
        initial_state=self.encoder.initialize_states(enc_ip_sequence.shape[0])
        # passing the input sequence and states to the encoder
        encoder_output,encoder_final_state_h,encoder_final_state_c=self.encoder(e
nc_ip_sequence, initial_state)
        # calling the decoder
        decoder_output=self.decoder(dec_ip_sequence, encoder_output,encoder_final
_state_h, encoder_final_state_c)

        return decoder_output
```

Custom loss function

In [ ]:
```
def custom_lossfunction(real,pred):
    # Custom loss function that will not consider the loss for padded zeros.
    # Refer https://www.tensorflow.org/tutorials/text/nmt_with_attention#define_t
he_optimizer_and_the_loss_function
    loss_object = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True,
reduction='none')

    mask = tf.math.logical_not(tf.math.equal(real, 0))
    loss_ = loss_object(real, pred)

    mask = tf.cast(mask, dtype=loss_.dtype)
    loss_ *= mask

    return tf.reduce_mean(loss_)
```

Training

Implement dot function here.

DOT FUNCTION

In [ ]:
```
log_dir="logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = TensorBoard(log_dir=log_dir,histogram_freq=1, write_graph=
True)
es = EarlyStopping(monitor='val_loss', mode='min', patience=3, verbose=1)

best_dot_model = ModelCheckpoint(
    'best_dot_model',
    monitor="val_loss",
    verbose=0,
    save_best_only=False)
```

In [77]:
```
# Implement teacher forcing while training your model. You can do it two ways.
# Prepare your data, encoder_input,decoder_input and decoder_output
# if decoder input is
# <start> Hi how are you
# decoder output should be
# Hi How are you <end>
# i.e when you have send <start>-- decoder predicted Hi, 'Hi' decoder predicted
 'How' .. e.t.c

# or

# model.fit([train_ita,train_eng],train_eng[:,1:]..)
# Note: If you follow this approach some grader functions might return false and
 this is fine.

enc_ip_length = train_italia[0].shape[0]
dec_ip_length = train_eng_in[0].shape[0]

train_steps=train_italia.shape[0]//1024
valid_steps=validation_italia.shape[0]//1024

test_data = ([validation_italia, validation_eng_in], validation_eng_op)

model_dot=encoder_decoder(len(tokenizer_ita.word_index), len(tokenizer_eng.word_i
ndex), 100, 100, 128, 128, enc_ip_length, dec_ip_length, 'dot', 20)

optimizer = tf.keras.optimizers.Adam()

model_dot.compile(optimizer=optimizer,loss=custom_lossfunction, metrics=["accurac
y"])

model_dot.fit([train_italia,train_eng_in], train_eng_op, steps_per_epoch=train_st
eps, validation_data= test_data, batch_size=1024, callbacks = [es, tensorboard_ca
llback, best_dot_model], epochs=50)
```

```
Epoch 1/50
301/301 [==============================] - 176s 564ms/step - loss: 2.2094 - accu
racy: 0.0561 - val_loss: 1.6907 - val_accuracy: 0.0702
Epoch 2/50
301/301 [==============================] - 167s 554ms/step - loss: 1.6591 - accu
racy: 0.0742 - val_loss: 1.5078 - val_accuracy: 0.0979
Epoch 3/50
301/301 [==============================] - 167s 555ms/step - loss: 1.4396 - accu
racy: 0.1068 - val_loss: 1.2696 - val_accuracy: 0.1259
Epoch 4/50
301/301 [==============================] - 166s 551ms/step - loss: 1.2239 - accu
racy: 0.1326 - val_loss: 1.1228 - val_accuracy: 0.1477
Epoch 5/50
301/301 [==============================] - 165s 549ms/step - loss: 1.0798 - accu
racy: 0.1532 - val_loss: 1.0113 - val_accuracy: 0.1631
Epoch 6/50
301/301 [==============================] - 163s 541ms/step - loss: 0.9649 - accu
racy: 0.1693 - val_loss: 0.9154 - val_accuracy: 0.1771
Epoch 7/50
301/301 [==============================] - 165s 549ms/step - loss: 0.8650 - accu
racy: 0.1847 - val_loss: 0.8307 - val_accuracy: 0.1908
Epoch 8/50
301/301 [==============================] - 166s 553ms/step - loss: 0.7763 - accu
racy: 0.1983 - val_loss: 0.7461 - val_accuracy: 0.2040
Epoch 9/50
301/301 [==============================] - 167s 556ms/step - loss: 0.6888 - accu
racy: 0.2129 - val_loss: 0.6666 - val_accuracy: 0.2178
Epoch 10/50
301/301 [==============================] - 167s 555ms/step - loss: 0.6104 - accu
racy: 0.2261 - val_loss: 0.5974 - val_accuracy: 0.2292
Epoch 11/50
301/301 [==============================] - 169s 560ms/step - loss: 0.5427 - accu
racy: 0.2383 - val_loss: 0.5413 - val_accuracy: 0.2399
Epoch 12/50
301/301 [==============================] - 170s 565ms/step - loss: 0.4826 - accu
racy: 0.2489 - val_loss: 0.4919 - val_accuracy: 0.2484
Epoch 13/50
301/301 [==============================] - 169s 563ms/step - loss: 0.4315 - accu
racy: 0.2577 - val_loss: 0.4524 - val_accuracy: 0.2553
Epoch 14/50
301/301 [==============================] - 170s 565ms/step - loss: 0.3880 - accu
racy: 0.2655 - val_loss: 0.4182 - val_accuracy: 0.2611
Epoch 15/50
301/301 [==============================] - 168s 558ms/step - loss: 0.3519 - accu
racy: 0.2719 - val_loss: 0.3915 - val_accuracy: 0.2659
Epoch 16/50
301/301 [==============================] - 168s 558ms/step - loss: 0.3203 - accu
racy: 0.2774 - val_loss: 0.3669 - val_accuracy: 0.2704
Epoch 17/50
301/301 [==============================] - 168s 558ms/step - loss: 0.2922 - accu
racy: 0.2819 - val_loss: 0.3474 - val_accuracy: 0.2737
Epoch 18/50
301/301 [==============================] - 168s 558ms/step - loss: 0.2698 - accu
racy: 0.2862 - val_loss: 0.3309 - val_accuracy: 0.2768
Epoch 19/50
301/301 [==============================] - 168s 558ms/step - loss: 0.2496 - accu
racy: 0.2901 - val_loss: 0.3165 - val_accuracy: 0.2794
Epoch 20/50
301/301 [==============================] - 168s 557ms/step - loss: 0.2318 - accu
racy: 0.2931 - val_loss: 0.3041 - val_accuracy: 0.2815
Epoch 21/50
301/301 [==============================] - 167s 553ms/step - loss: 0.2154 - accu
racy: 0.2963 - val_loss: 0.2937 - val_accuracy: 0.2837
Epoch 22/50
301/301 [==============================] - 166s 553ms/step - loss: 0.2037 - accu
racy: 0.2987 - val_loss: 0.2847 - val_accuracy: 0.2852
Epoch 23/50
301/301 [==============================] - 169s 563ms/step - loss: 0.1910 - accu
racy: 0.3007 - val_loss: 0.2779 - val_accuracy: 0.2868
```

```
Epoch 24/50
301/301 [==============================] - 169s 561ms/step - loss: 0.1789 - accu
racy: 0.3031 - val_loss: 0.2700 - val_accuracy: 0.2884
Epoch 25/50
301/301 [==============================] - 165s 548ms/step - loss: 0.1715 - accu
racy: 0.3047 - val_loss: 0.2633 - val_accuracy: 0.2894
Epoch 26/50
301/301 [==============================] - 165s 549ms/step - loss: 0.1604 - accu
racy: 0.3066 - val_loss: 0.2586 - val_accuracy: 0.2903
Epoch 27/50
301/301 [==============================] - 167s 555ms/step - loss: 0.1535 - accu
racy: 0.3082 - val_loss: 0.2532 - val_accuracy: 0.2916
Epoch 28/50
301/301 [==============================] - 165s 550ms/step - loss: 0.1465 - accu
racy: 0.3094 - val_loss: 0.2501 - val_accuracy: 0.2921
Epoch 29/50
301/301 [==============================] - 166s 550ms/step - loss: 0.1393 - accu
racy: 0.3109 - val_loss: 0.2456 - val_accuracy: 0.2934
Epoch 30/50
301/301 [==============================] - 169s 561ms/step - loss: 0.1340 - accu
racy: 0.3119 - val_loss: 0.2421 - val_accuracy: 0.2940
Epoch 31/50
301/301 [==============================] - 165s 548ms/step - loss: 0.1273 - accu
racy: 0.3128 - val_loss: 0.2399 - val_accuracy: 0.2944
Epoch 32/50
301/301 [==============================] - 164s 546ms/step - loss: 0.1227 - accu
racy: 0.3143 - val_loss: 0.2372 - val_accuracy: 0.2952
Epoch 33/50
301/301 [==============================] - 164s 545ms/step - loss: 0.1176 - accu
racy: 0.3150 - val_loss: 0.2348 - val_accuracy: 0.2959
Epoch 34/50
301/301 [==============================] - 163s 541ms/step - loss: 0.1185 - accu
racy: 0.3149 - val_loss: 0.2324 - val_accuracy: 0.2964
Epoch 35/50
301/301 [==============================] - 163s 542ms/step - loss: 0.1100 - accu
racy: 0.3168 - val_loss: 0.2300 - val_accuracy: 0.2972
Epoch 36/50
301/301 [==============================] - 163s 542ms/step - loss: 0.1055 - accu
racy: 0.3179 - val_loss: 0.2284 - val_accuracy: 0.2975
Epoch 37/50
301/301 [==============================] - 163s 543ms/step - loss: 0.1025 - accu
racy: 0.3185 - val_loss: 0.2271 - val_accuracy: 0.2980
Epoch 38/50
301/301 [==============================] - 162s 539ms/step - loss: 0.0990 - accu
racy: 0.3192 - val_loss: 0.2252 - val_accuracy: 0.2984
Epoch 39/50
301/301 [==============================] - 164s 544ms/step - loss: 0.0961 - accu
racy: 0.3199 - val_loss: 0.2248 - val_accuracy: 0.2986
Epoch 40/50
301/301 [==============================] - 164s 545ms/step - loss: 0.0926 - accu
racy: 0.3206 - val_loss: 0.2238 - val_accuracy: 0.2990
Epoch 41/50
301/301 [==============================] - 164s 543ms/step - loss: 0.0902 - accu
racy: 0.3208 - val_loss: 0.2222 - val_accuracy: 0.2995
Epoch 42/50
301/301 [==============================] - 163s 542ms/step - loss: 0.0876 - accu
racy: 0.3218 - val_loss: 0.2223 - val_accuracy: 0.2996
Epoch 43/50
301/301 [==============================] - 163s 540ms/step - loss: 0.0853 - accu
racy: 0.3218 - val_loss: 0.2216 - val_accuracy: 0.3000
Epoch 44/50
301/301 [==============================] - 163s 542ms/step - loss: 0.0833 - accu
racy: 0.3228 - val_loss: 0.2208 - val_accuracy: 0.3002
Epoch 45/50
301/301 [==============================] - 164s 545ms/step - loss: 0.0806 - accu
racy: 0.3230 - val_loss: 0.2203 - val_accuracy: 0.3004
Epoch 46/50
301/301 [==============================] - 163s 542ms/step - loss: 0.0785 - accu
racy: 0.3234 - val_loss: 0.2196 - val_accuracy: 0.3009
Epoch 47/50
```

```
301/301 [==============================] - 163s 543ms/step - loss: 0.0760 - accu
racy: 0.3237 - val_loss: 0.2192 - val_accuracy: 0.3010
Epoch 48/50
301/301 [==============================] - 163s 543ms/step - loss: 0.0739 - accu
racy: 0.3243 - val_loss: 0.2187 - val_accuracy: 0.3014
Epoch 49/50
301/301 [==============================] - 164s 545ms/step - loss: 0.0730 - accu
racy: 0.3248 - val_loss: 0.2198 - val_accuracy: 0.3012
Epoch 50/50
301/301 [==============================] - 164s 545ms/step - loss: 0.0707 - accu
racy: 0.3250 - val_loss: 0.2191 - val_accuracy: 0.3016
```

Out[77]:
```
<tensorflow.python.keras.callbacks.History at 0x7f90cd558ef0>
```

In [114]:
```
%tensorboard --logdir /content/logs/fit/20210115-031159
```

Inference

Plot attention weights

In [83]:
```
def plot_attention(attention, sentence, predicted_sentence):
    #Refer: https://www.tensorflow.org/tutorials/text/nmt_with_attention#translat
e
    fig = plt.figure(figsize=(10,10))
    ax = fig.add_subplot(1, 1, 1)
    ax.matshow(attention, cmap='viridis')

    fontdict = {'fontsize': 14}

    ax.set_xticklabels([''] + sentence, fontdict=fontdict, rotation=90)
    ax.set_yticklabels([''] + predicted_sentence, fontdict=fontdict)

    ax.xaxis.set_major_locator(ticker.MultipleLocator(1))
    ax.yaxis.set_major_locator(ticker.MultipleLocator(1))

    plt.show()
```

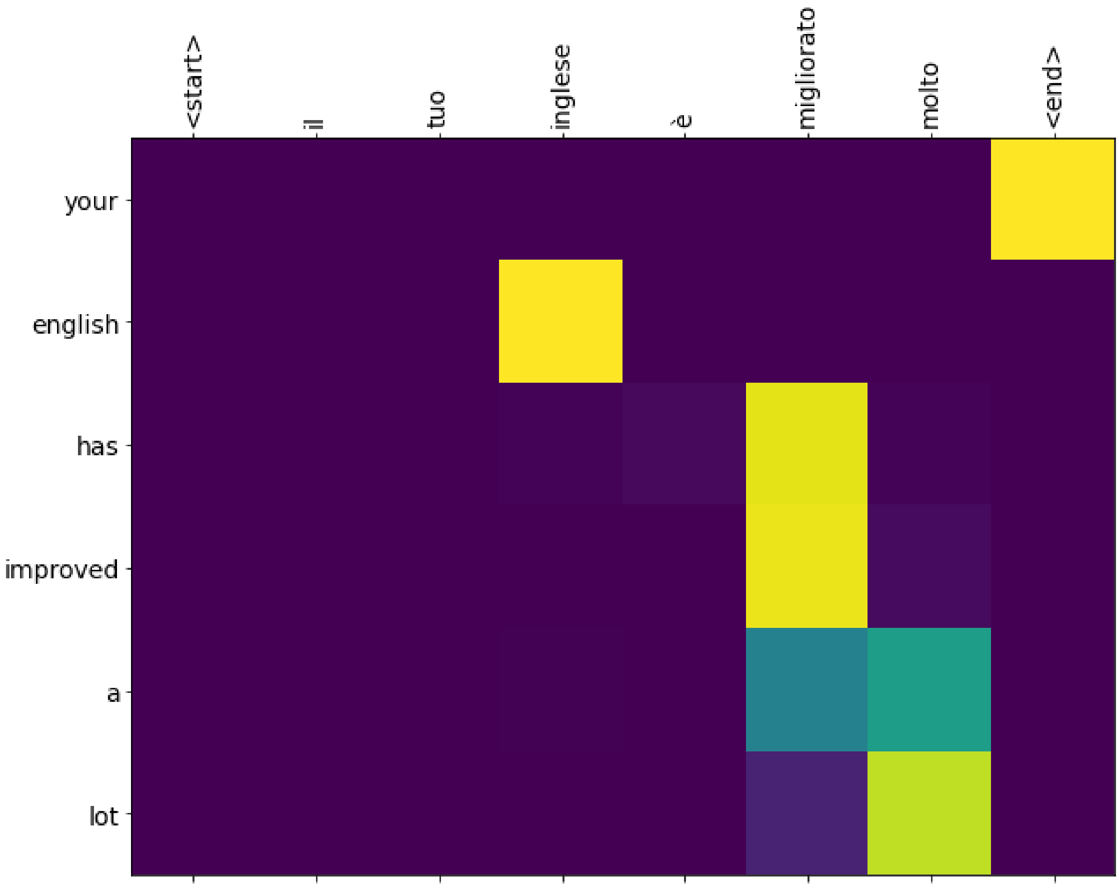
Predict the sentence translation

In [110]:
```
def predict_with_attention(input_sentence, model, plot_attention_flag):


    '''
    A. Given input sentence, convert the sentence into integers using tokenizer u
sed earlier
    B. Pass the input_sequence to encoder. we get encoder_outputs, last time step
hidden and cell state
    C. Initialize index of <start> as input to decoder. and encoder final states
 as input_states to onestepdecoder.
    D. till we reach max_length of decoder or till the model predicted word <end
>:
         predictions, input_states, attention_weights = model.layers[1].onestepde
coder(input_to_decoder, encoder_output, input_states)
         Save the attention weights
         And get the word using the tokenizer(word index) and then store it in a
 string.
    E. Call plot_attention(#params)
    F. Return the predicted sentence
    '''
    attention_plot = np.zeros((model.dec_input_length, model.enc_input_length))

    padded_input_sentences = pad_sequences(tokenizer_ita.texts_to_sequences([inpu
t_sentence]), padding='post', maxlen=20)
    initializer = tf.keras.initializers.Zeros()
    initial_state_h = initializer(shape=(padded_input_sentences.shape[0],model.en
c_lstm_size))
    initial_state_c = initializer(shape=(padded_input_sentences.shape[0],model.en
c_lstm_size))

    encoder_output,encoder_final_state_h,encoder_final_state_c = model.layers[0](
padded_input_sentences,[initial_state_h, initial_state_c])

    word_count = 0
    output_sentence =''
    decoder_ip_sequence = np.array([[tokenizer_eng.word_index['<start>']]])
    decoder_h_state, decoder_c_state = encoder_final_state_h,encoder_final_state_
c

    while True:

        decoder_output, decoder_h_state, decoder_c_state, attention_weights,conte
xt_vector = model.layers[1].onestepdecoder(decoder_ip_sequence, encoder_output, d
ecoder_h_state, decoder_c_state)
        idx = np.argmax(decoder_output,-1) #getting word index with max prob

        word = tokenizer_eng.index_word[idx]

        if word_count>=20 or word=='<end>':
            break
        else:

            output_sentence += ' '+word
            decoder_ip_sequence= np.array([[tokenizer_eng.word_index[word.strip
()]]])
            attention_weights = tf.reshape(attention_weights, (-1, ))
            attention_plot[word_count] = attention_weights.numpy()
            word_count+=1



    attention_plot = attention_plot[:len(output_sentence.strip().split(' ')), :le
n(input_sentence.split(' '))]
    if plot_attention_flag:
      plot_attention(attention_plot, input_sentence.split(' '), output_sentence.s
trip().split(' '))

    return output_sentence.strip()
```

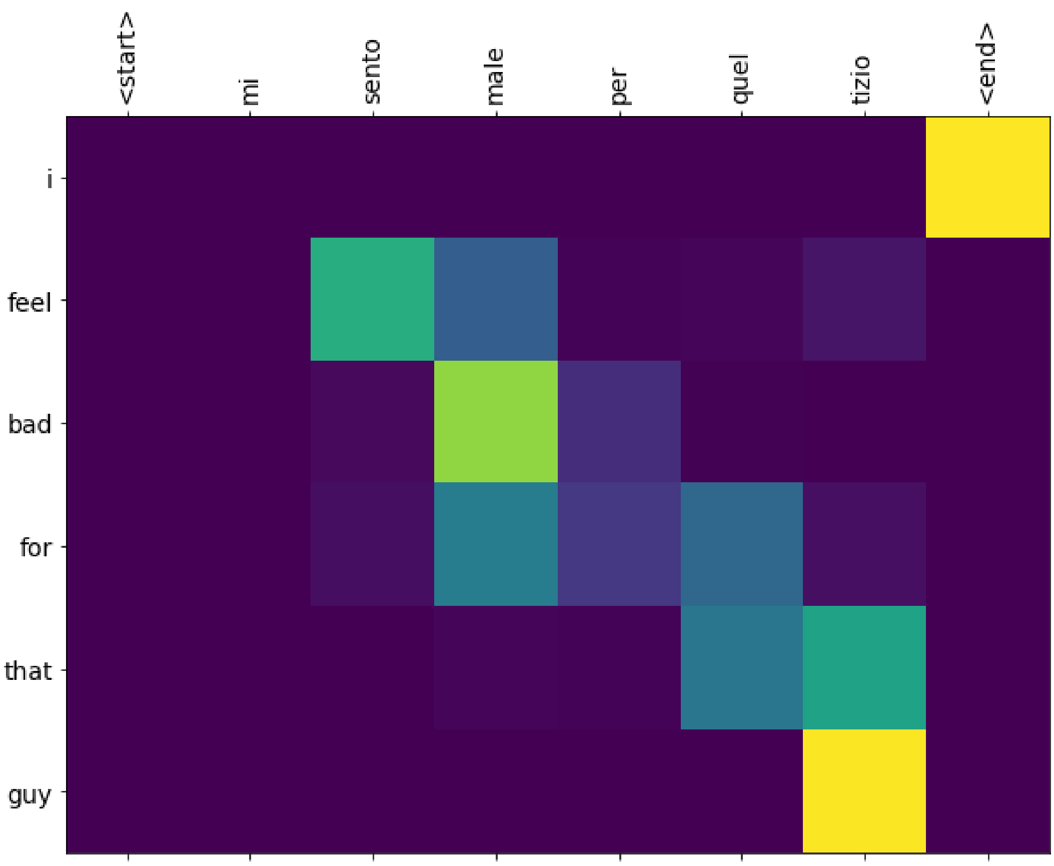
In [111]:
```
bleu_score= 0

italian_sentence = validation['italia_enc'][:1000]
english_conversion = [ sentence.lstrip('<start>').strip(' ') for sentence in vali
dation['english_in'][:1000]]
english_conversion
for sentence, actual_translation in zip(italian_sentence,english_conversion):
    # the original

    # traslated using model
    translation = predict_with_attention(sentence, model_dot, False)
    # print(sentence)
    # print(actual_translation)
    # print(translation)
    bleu_score += bleu.sentence_bleu(actual_translation.split(), translation.spli
t())

print(bleu_score/1000)
```

```
/usr/local/lib/python3.6/dist-packages/nltk/translate/bleu_score.py:490: UserWar
ning:
Corpus/Sentence contains 0 counts of 2-gram overlaps.
BLEU scores might be undesirable; use SmoothingFunction().
  warnings.warn(_msg)

0.25351676810888724
```

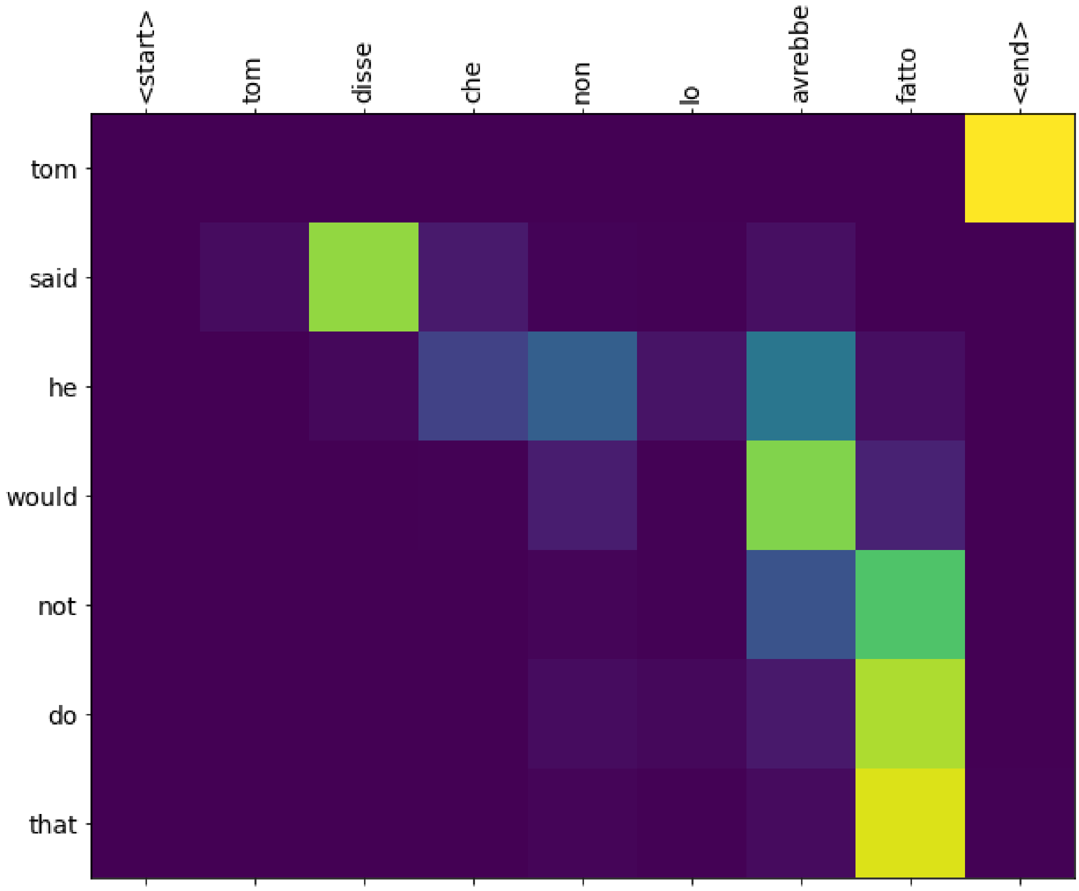
In [112]:
```
bleu_score= 0

italian_sentence = validation['italia_enc'][:10]
english_conversion = [ sentence.lstrip('<start>').strip(' ') for sentence in vali
dation['english_in'][:10]]
english_conversion
for sentence, actual_translation in zip(italian_sentence,english_conversion):
    # the original

    # traslated using model
    translation = predict_with_attention(sentence, model_dot, True)
    print(' italian sentence ',sentence)
    print(' actual english translation ',actual_translation)
    print(' predicted english translation ', translation)
    #bleu_score += bleu.sentence_bleu(actual_translation.split(), translation.spl
it())
```



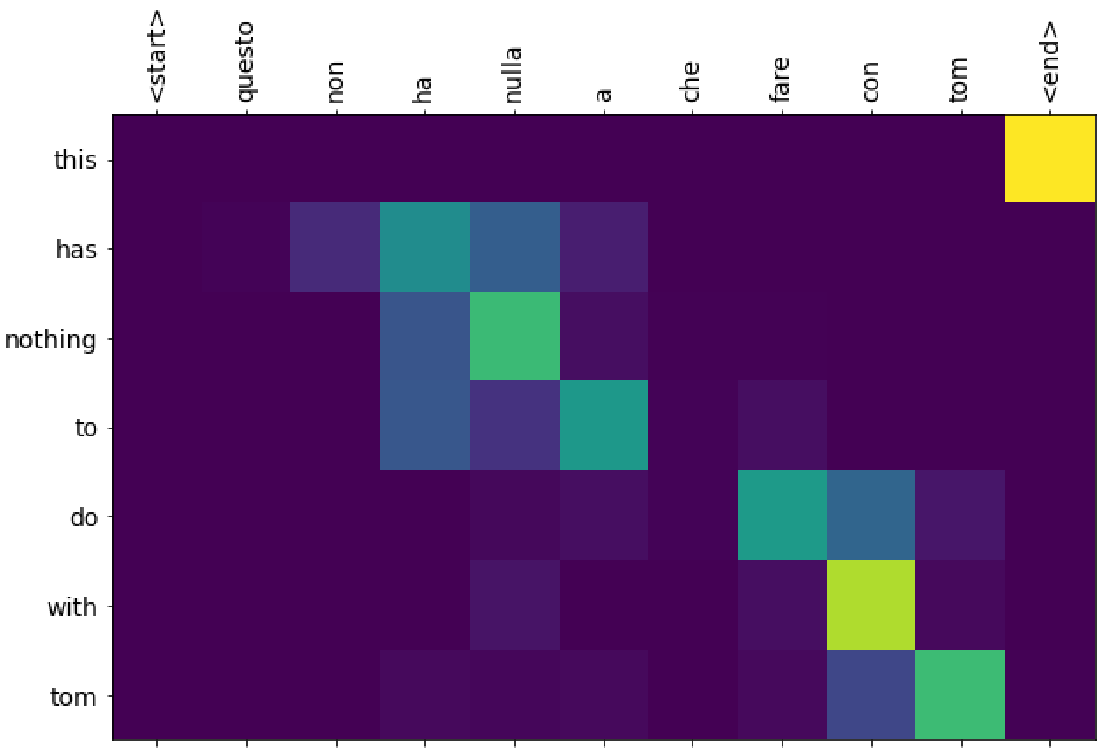

```
italian sentence  <start> il tuo inglese è migliorato molto <end>
actual english translation  your english has improved a lot
predicted english translation  your english has improved a lot
```



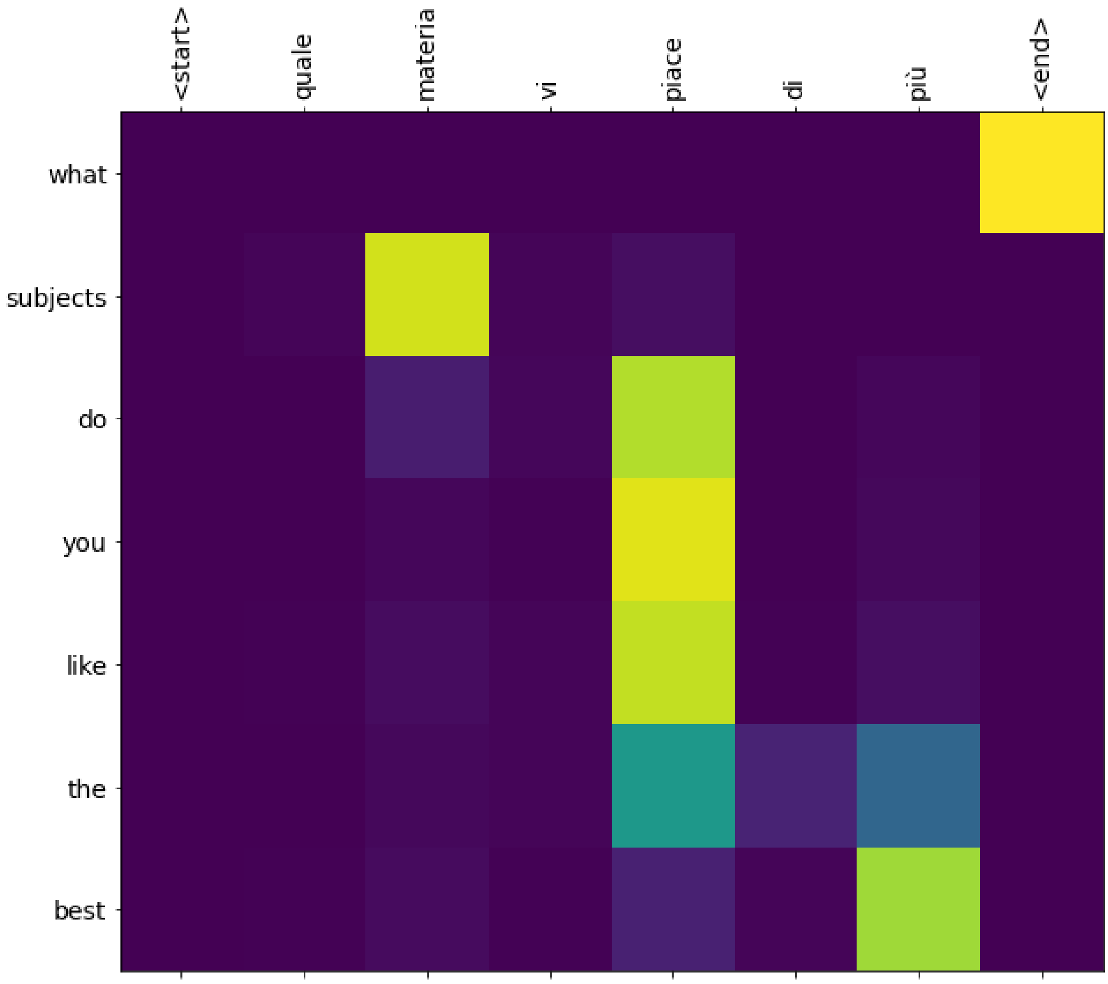

```
italian sentence  <start> mi sento male per quel tizio <end>
actual english translation  i feel bad for that guy
predicted english translation  i feel bad for that guy
```



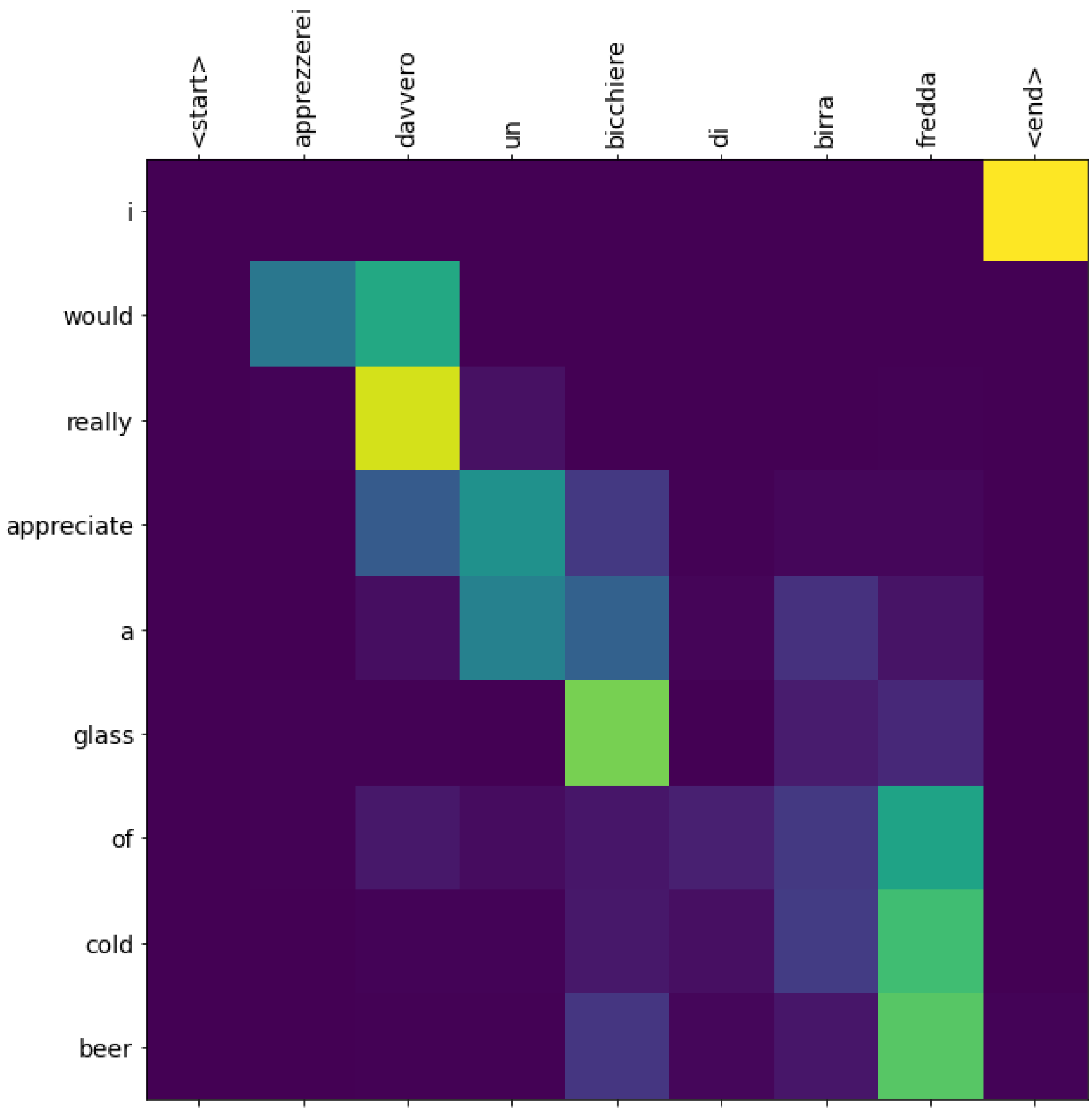

```
italian sentence  <start> tom disse che non lo avrebbe fatto <end>
actual english translation  tom said he would not do it
predicted english translation  tom said he would not do that
```



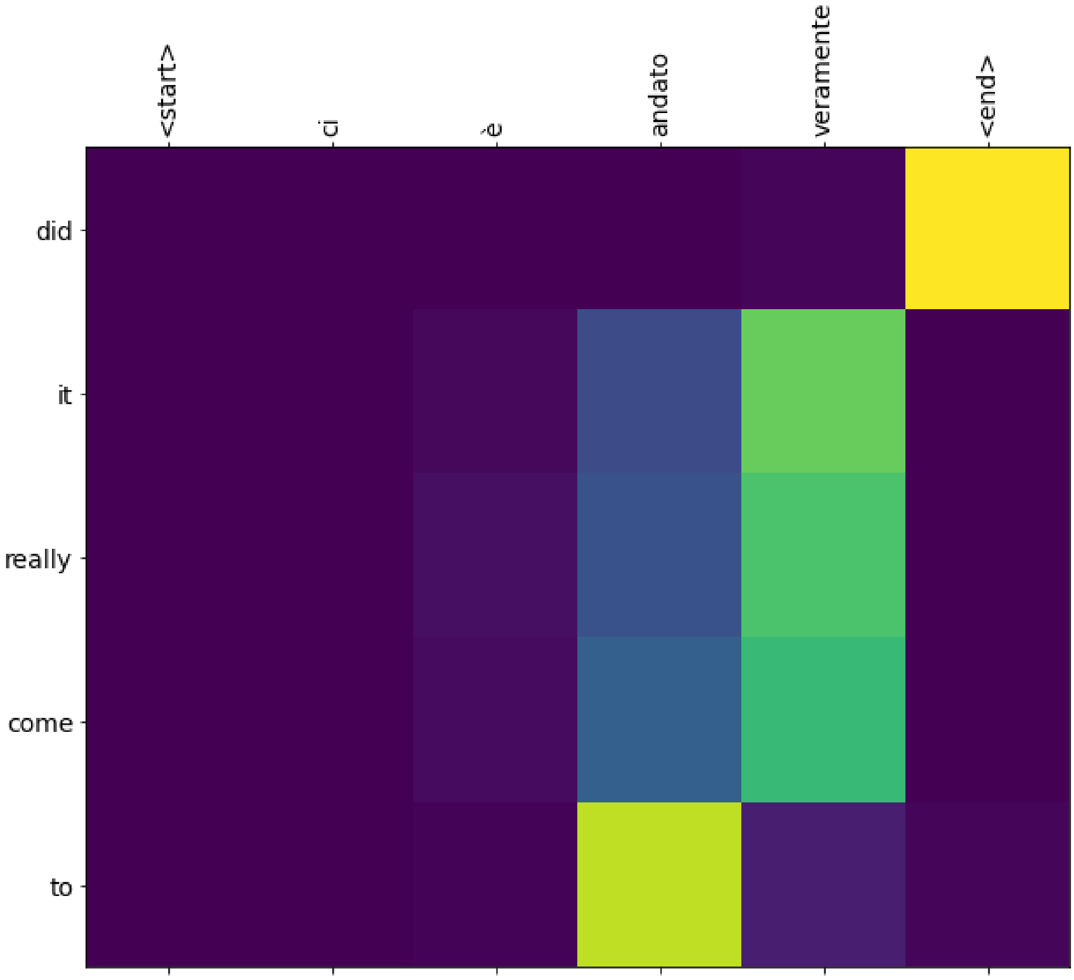

```
italian sentence  <start> questo non ha nulla a che fare con tom <end>
actual english translation  this has nothing to do with tom
predicted english translation  this has nothing to do with tom
```



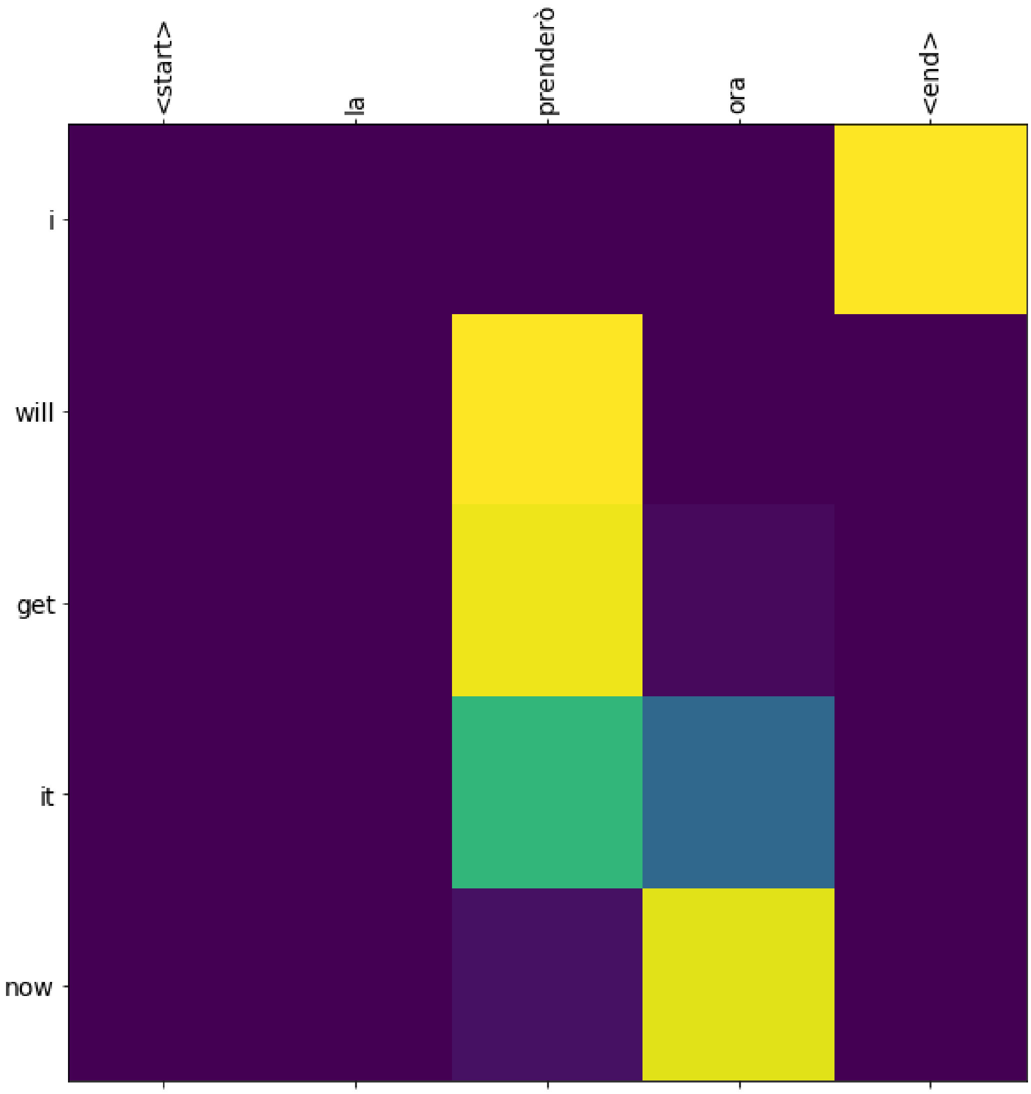

```
italian sentence  <start> quale materia vi piace di più <end>
actual english translation  what subject do you like best
predicted english translation  what subjects do you like the best
```



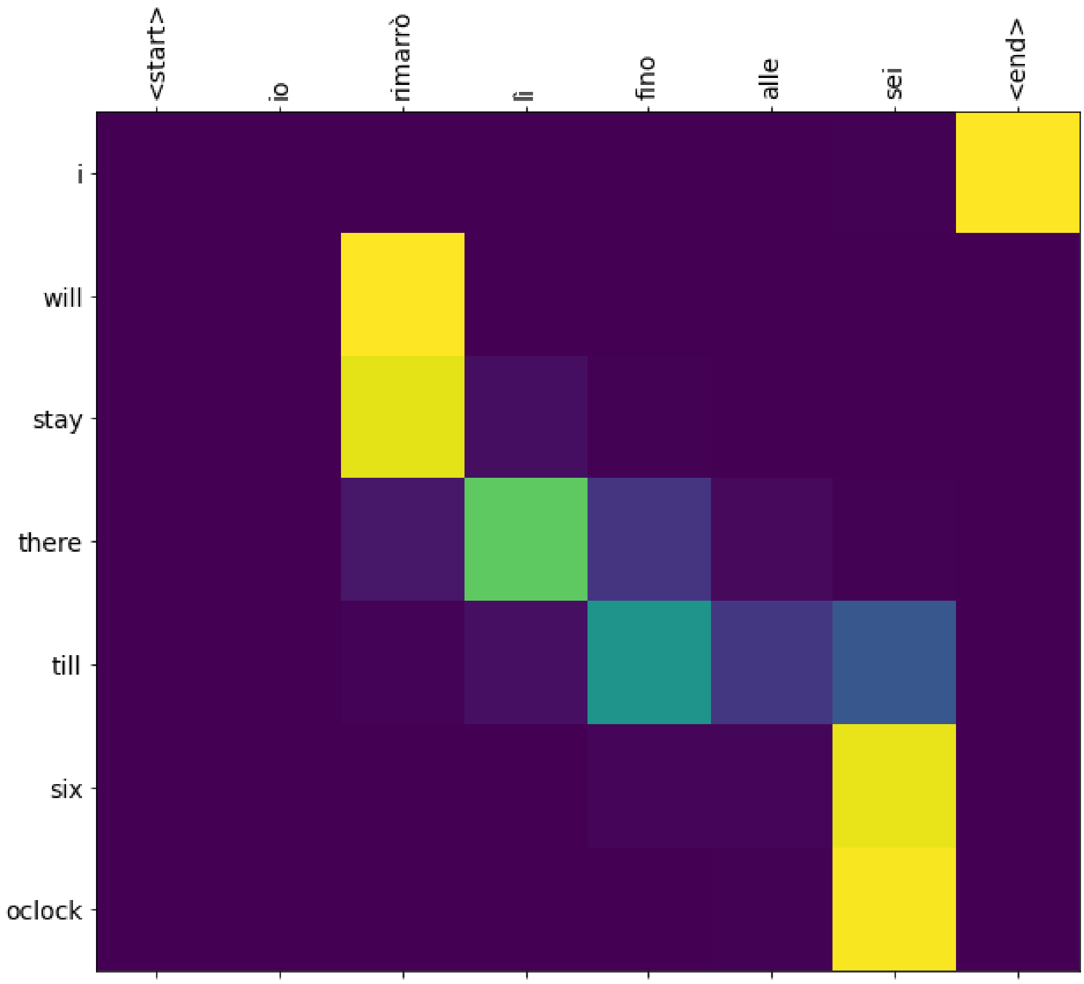

```
 italian sentence  <start> apprezzerei davvero un bicchiere di birra fredda <end
>
 actual english translation  i would really appreciate a glass of cold beer
 predicted english translation  i would really appreciate a glass of cold beer
```



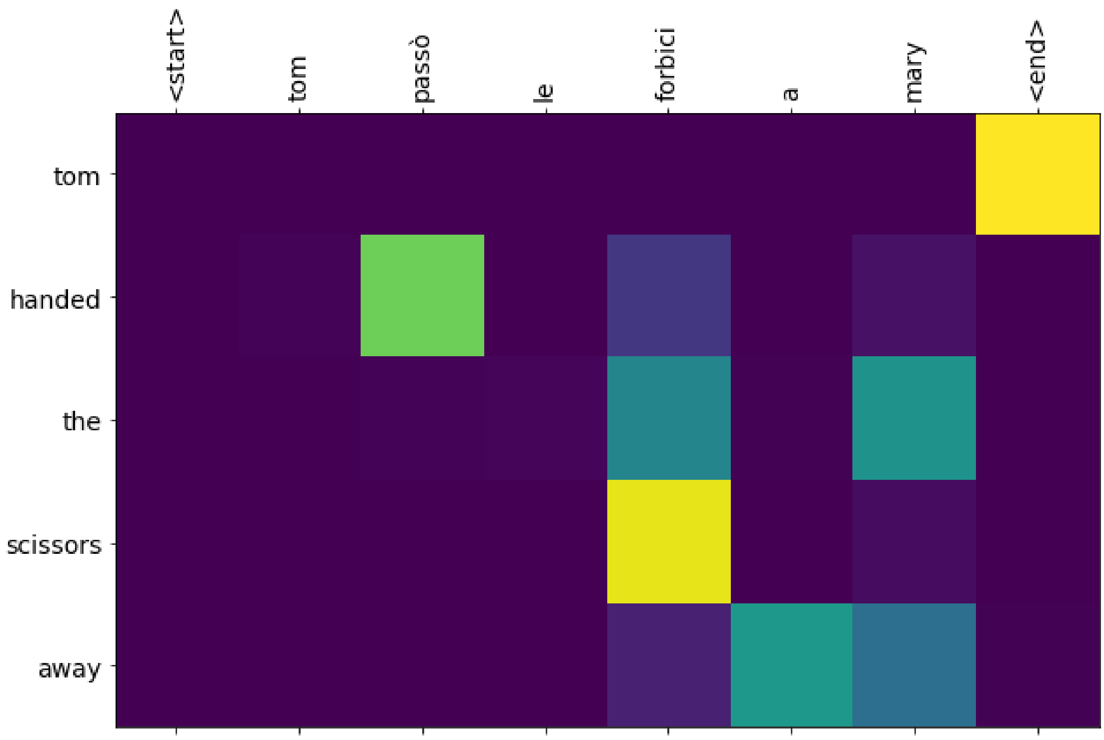

```
italian sentence  <start> ci è andato veramente <end>
actual english translation  did you really go there
predicted english translation  did it really come to
```




```
italian sentence  <start> la prenderò ora <end>
actual english translation  i will take it now
predicted english translation  i will get it now
```




```
italian sentence  <start> io rimarrò lì fino alle sei <end>
actual english translation  i will stay there till six oclock
predicted english translation  i will stay there till six oclock
```




```
italian sentence  <start> tom passò le forbici a mary <end>
actual english translation  tom handed mary the scissors
predicted english translation  tom handed the scissors away
```

Calculate BLEU score

In [96]:
```
#Create an object of your custom model.
#Compile and train your model on dot scoring function.
# Visualize few sentences randomly in Test data
# Predict on 1000 random sentences on test data and calculate the average BLEU sc
ore of these sentences.
# https://www.nltk.org/_modules/nltk/translate/bleu_score.html

#Sample example
# import nltk.translate.bleu_score as bleu
# reference = ['i am groot'.split(),] # the original
# translation = 'it is ship'.split() # trasilated using model
# print('BLEU score: {}'.format(bleu.sentence_bleu(reference, translation)))
```

Repeat the same steps for General scoring function

In [ ]:
```
#Compile and train your model on general scoring function.
# Visualize few sentences randomly in Test data
# Predict on 1000 random sentences on test data and calculate the average BLEU sc
ore of these sentences.
# https://www.nltk.org/_modules/nltk/translate/bleu_score.html
```

GENERAL FUNCTION

In [118]:
```
log_dir="logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = TensorBoard(log_dir=log_dir,histogram_freq=1, write_graph=
True)
es = EarlyStopping(monitor='val_loss', mode='min', patience=3, verbose=1)

best_general_model = ModelCheckpoint(
    'best_general_model',
    monitor="val_loss",
    verbose=0,
    save_best_only=False)
```

In [120]:
```
enc_ip_length = train_italia[0].shape[0]
dec_ip_length = train_eng_in[0].shape[0]

train_steps=train_italia.shape[0]//1024
valid_steps=validation_italia.shape[0]//1024

test_data = ([validation_italia, validation_eng_in], validation_eng_op)

model_general = encoder_decoder(len(tokenizer_ita.word_index), len(tokenizer_eng.
word_index), 100, 100, 128, 128, enc_ip_length, dec_ip_length, 'general', 128)

optimizer = tf.keras.optimizers.Adam()

model_general.compile(optimizer=optimizer,loss=custom_lossfunction, metrics=["acc
uracy"])

model_general.fit([train_italia,train_eng_in], train_eng_op, steps_per_epoch=trai
n_steps, validation_data= test_data, batch_size=1024, callbacks = [es, tensorboar
d_callback, best_general_model], epochs=50)
```

```
Epoch 1/5
301/301 [==============================] - 183s 590ms/step - loss: 2.2110 - accu
racy: 0.0564 - val_loss: 1.6997 - val_accuracy: 0.0696
Epoch 2/5
301/301 [==============================] - 174s 579ms/step - loss: 1.6684 - accu
racy: 0.0733 - val_loss: 1.5216 - val_accuracy: 0.0949
Epoch 3/5
301/301 [==============================] - 175s 580ms/step - loss: 1.4508 - accu
racy: 0.1063 - val_loss: 1.3004 - val_accuracy: 0.1226
Epoch 4/5
301/301 [==============================] - 174s 577ms/step - loss: 1.2490 - accu
racy: 0.1291 - val_loss: 1.1570 - val_accuracy: 0.1406
Epoch 5/5
301/301 [==============================] - 174s 578ms/step - loss: 1.1107 - accu
racy: 0.1479 - val_loss: 1.0500 - val_accuracy: 0.1568
```

Out[120]:
```
<tensorflow.python.keras.callbacks.History at 0x7f9051e75898>
```

Repeat the same steps for Concat scoring function

In [ ]:
```
#Compile and train your model on concat scoring function.
# Visualize few sentences randomly in Test data
# Predict on 1000 random sentences on test data and calculate the average BLEU sc
ore of these sentences.
# https://www.nltk.org/_modules/nltk/translate/bleu_score.html
```

In [ ]:
```
# Write your observations on each of the scoring function
```

CONCAT FUNCTION

In [81]:
```
log_dir="logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = TensorBoard(log_dir=log_dir,histogram_freq=1, write_graph=
True)
es = EarlyStopping(monitor='val_loss', mode='min', patience=3, verbose=1)

best_concat_model = ModelCheckpoint(
    'best_concat_model',
    monitor="val_loss",
    verbose=0,
    save_best_only=False)
```

In [82]:

```
enc_ip_length = train_italia[0].shape[0]
dec_ip_length = train_eng_in[0].shape[0]

train_steps=train_italia.shape[0]//1024
valid_steps=validation_italia.shape[0]//1024

test_data = ([validation_italia, validation_eng_in], validation_eng_op)

model_concat = encoder_decoder(len(tokenizer_ita.word_index), len(tokenizer_eng.word_index), 100, 100, 128, 128, enc_ip_length, dec_ip_length, 'concat', 20)

optimizer = tf.keras.optimizers.Adam()

model_concat.compile(optimizer=optimizer,loss=custom_lossfunction, metrics=["accuracy"])

model_concat.fit([train_italia,train_eng_in], train_eng_op, steps_per_epoch=train_steps, validation_data= test_data, batch_size=1024, callbacks = [es, tensorboard_callback, best_concat_model], epochs=50)
```

```
Epoch 1/50
301/301 [==============================] - 192s 619ms/step - loss: 2.2090 - accu
racy: 0.0563 - val_loss: 1.6762 - val_accuracy: 0.0737
Epoch 2/50
301/301 [==============================] - 183s 607ms/step - loss: 1.6352 - accu
racy: 0.0773 - val_loss: 1.4708 - val_accuracy: 0.1030
Epoch 3/50
301/301 [==============================] - 183s 609ms/step - loss: 1.4084 - accu
racy: 0.1112 - val_loss: 1.2515 - val_accuracy: 0.1286
Epoch 4/50
301/301 [==============================] - 183s 607ms/step - loss: 1.1997 - accu
racy: 0.1357 - val_loss: 1.1050 - val_accuracy: 0.1488
Epoch 5/50
301/301 [==============================] - 183s 607ms/step - loss: 1.0523 - accu
racy: 0.1576 - val_loss: 0.9767 - val_accuracy: 0.1687
Epoch 6/50
301/301 [==============================] - 183s 607ms/step - loss: 0.9247 - accu
racy: 0.1773 - val_loss: 0.8631 - val_accuracy: 0.1856
Epoch 7/50
301/301 [==============================] - 183s 608ms/step - loss: 0.8076 - accu
racy: 0.1952 - val_loss: 0.7555 - val_accuracy: 0.2027
Epoch 8/50
301/301 [==============================] - 183s 608ms/step - loss: 0.6987 - accu
racy: 0.2123 - val_loss: 0.6577 - val_accuracy: 0.2183
Epoch 9/50
301/301 [==============================] - 183s 609ms/step - loss: 0.6005 - accu
racy: 0.2288 - val_loss: 0.5750 - val_accuracy: 0.2332
Epoch 10/50
301/301 [==============================] - 183s 608ms/step - loss: 0.5170 - accu
racy: 0.2433 - val_loss: 0.5078 - val_accuracy: 0.2449
Epoch 11/50
301/301 [==============================] - 183s 608ms/step - loss: 0.4483 - accu
racy: 0.2552 - val_loss: 0.4537 - val_accuracy: 0.2545
Epoch 12/50
301/301 [==============================] - 182s 605ms/step - loss: 0.3924 - accu
racy: 0.2651 - val_loss: 0.4109 - val_accuracy: 0.2618
Epoch 13/50
301/301 [==============================] - 183s 608ms/step - loss: 0.3472 - accu
racy: 0.2729 - val_loss: 0.3777 - val_accuracy: 0.2680
Epoch 14/50
301/301 [==============================] - 184s 610ms/step - loss: 0.3106 - accu
racy: 0.2795 - val_loss: 0.3482 - val_accuracy: 0.2733
Epoch 15/50
301/301 [==============================] - 183s 608ms/step - loss: 0.2791 - accu
racy: 0.2852 - val_loss: 0.3261 - val_accuracy: 0.2771
Epoch 16/50
301/301 [==============================] - 184s 610ms/step - loss: 0.2549 - accu
racy: 0.2894 - val_loss: 0.3090 - val_accuracy: 0.2802
Epoch 17/50
301/301 [==============================] - 184s 611ms/step - loss: 0.2318 - accu
racy: 0.2933 - val_loss: 0.2942 - val_accuracy: 0.2831
Epoch 18/50
301/301 [==============================] - 183s 609ms/step - loss: 0.2142 - accu
racy: 0.2964 - val_loss: 0.2801 - val_accuracy: 0.2855
Epoch 19/50
301/301 [==============================] - 182s 605ms/step - loss: 0.1981 - accu
racy: 0.2998 - val_loss: 0.2693 - val_accuracy: 0.2874
Epoch 20/50
301/301 [==============================] - 184s 611ms/step - loss: 0.1846 - accu
racy: 0.3019 - val_loss: 0.2605 - val_accuracy: 0.2891
Epoch 21/50
301/301 [==============================] - 183s 607ms/step - loss: 0.1726 - accu
racy: 0.3043 - val_loss: 0.2522 - val_accuracy: 0.2909
Epoch 22/50
301/301 [==============================] - 183s 608ms/step - loss: 0.1627 - accu
racy: 0.3062 - val_loss: 0.2472 - val_accuracy: 0.2917
Epoch 23/50
301/301 [==============================] - 183s 607ms/step - loss: 0.1527 - accu
racy: 0.3084 - val_loss: 0.2396 - val_accuracy: 0.2931
```

```
Epoch 24/50
301/301 [==============================] - 182s 605ms/step - loss: 0.1436 - accu
racy: 0.3095 - val_loss: 0.2346 - val_accuracy: 0.2942
Epoch 25/50
301/301 [==============================] - 182s 606ms/step - loss: 0.1371 - accu
racy: 0.3108 - val_loss: 0.2300 - val_accuracy: 0.2952
Epoch 26/50
301/301 [==============================] - 182s 604ms/step - loss: 0.1294 - accu
racy: 0.3125 - val_loss: 0.2268 - val_accuracy: 0.2959
Epoch 27/50
301/301 [==============================] - 182s 606ms/step - loss: 0.1245 - accu
racy: 0.3142 - val_loss: 0.2272 - val_accuracy: 0.2955
Epoch 28/50
301/301 [==============================] - 181s 601ms/step - loss: 0.1187 - accu
racy: 0.3147 - val_loss: 0.2220 - val_accuracy: 0.2969
Epoch 29/50
301/301 [==============================] - 181s 601ms/step - loss: 0.1130 - accu
racy: 0.3159 - val_loss: 0.2187 - val_accuracy: 0.2978
Epoch 30/50
301/301 [==============================] - 181s 600ms/step - loss: 0.1081 - accu
racy: 0.3172 - val_loss: 0.2151 - val_accuracy: 0.2987
Epoch 31/50
301/301 [==============================] - 181s 601ms/step - loss: 0.1037 - accu
racy: 0.3177 - val_loss: 0.2127 - val_accuracy: 0.2992
Epoch 32/50
301/301 [==============================] - 181s 601ms/step - loss: 0.1006 - accu
racy: 0.3184 - val_loss: 0.2116 - val_accuracy: 0.2996
Epoch 33/50
301/301 [==============================] - 181s 602ms/step - loss: 0.0958 - accu
racy: 0.3194 - val_loss: 0.2100 - val_accuracy: 0.2999
Epoch 34/50
301/301 [==============================] - 181s 603ms/step - loss: 0.0921 - accu
racy: 0.3204 - val_loss: 0.2073 - val_accuracy: 0.3006
Epoch 35/50
301/301 [==============================] - 183s 608ms/step - loss: 0.0897 - accu
racy: 0.3210 - val_loss: 0.2078 - val_accuracy: 0.3008
Epoch 36/50
301/301 [==============================] - 182s 604ms/step - loss: 0.0865 - accu
racy: 0.3218 - val_loss: 0.2065 - val_accuracy: 0.3012
Epoch 37/50
301/301 [==============================] - 183s 609ms/step - loss: 0.0832 - accu
racy: 0.3223 - val_loss: 0.2064 - val_accuracy: 0.3014
Epoch 38/50
301/301 [==============================] - 185s 615ms/step - loss: 0.0802 - accu
racy: 0.3230 - val_loss: 0.2043 - val_accuracy: 0.3019
Epoch 39/50
301/301 [==============================] - 185s 614ms/step - loss: 0.0779 - accu
racy: 0.3236 - val_loss: 0.2036 - val_accuracy: 0.3022
Epoch 40/50
301/301 [==============================] - 184s 611ms/step - loss: 0.0761 - accu
racy: 0.3239 - val_loss: 0.2038 - val_accuracy: 0.3022
Epoch 41/50
301/301 [==============================] - 184s 610ms/step - loss: 0.0729 - accu
racy: 0.3244 - val_loss: 0.2013 - val_accuracy: 0.3030
Epoch 42/50
301/301 [==============================] - 183s 609ms/step - loss: 0.0707 - accu
racy: 0.3248 - val_loss: 0.2024 - val_accuracy: 0.3028
Epoch 43/50
301/301 [==============================] - 184s 610ms/step - loss: 0.0692 - accu
racy: 0.3252 - val_loss: 0.2013 - val_accuracy: 0.3033
Epoch 44/50
301/301 [==============================] - 184s 612ms/step - loss: 0.0674 - accu
racy: 0.3259 - val_loss: 0.2007 - val_accuracy: 0.3037
Epoch 45/50
301/301 [==============================] - 185s 613ms/step - loss: 0.0660 - accu
racy: 0.3260 - val_loss: 0.2015 - val_accuracy: 0.3034
Epoch 46/50
301/301 [==============================] - 184s 613ms/step - loss: 0.0640 - accu
racy: 0.3264 - val_loss: 0.2019 - val_accuracy: 0.3037
Epoch 47/50
```

```
301/301 [==============================] - 183s 609ms/step - loss: 0.0615 - accu
racy: 0.3270 - val_loss: 0.2022 - val_accuracy: 0.3038
Epoch 00047: early stopping
```

Out[82]:

```
<tensorflow.python.keras.callbacks.History at 0x7f90cc5e16d8>
```

In [115]:

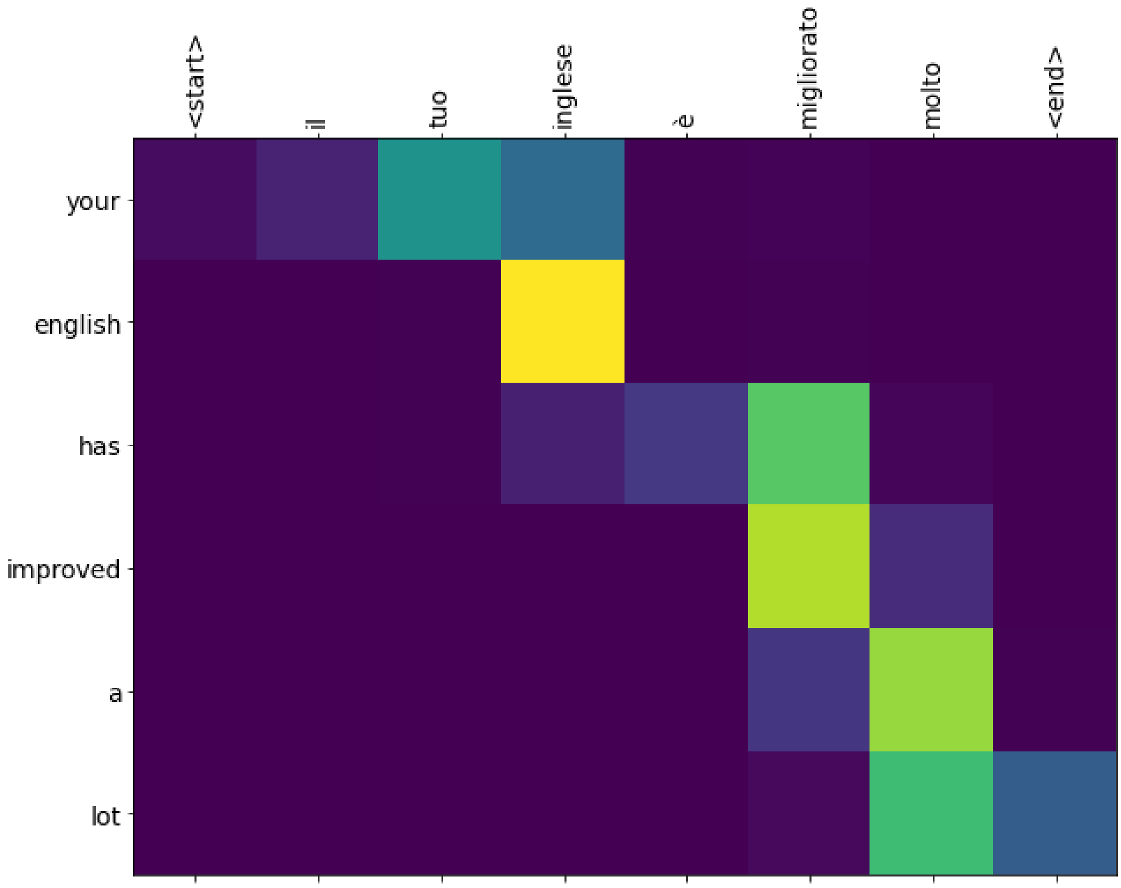
```
%tensorboard --logdir /content/logs/fit/20210115-055934
```

In [100]:

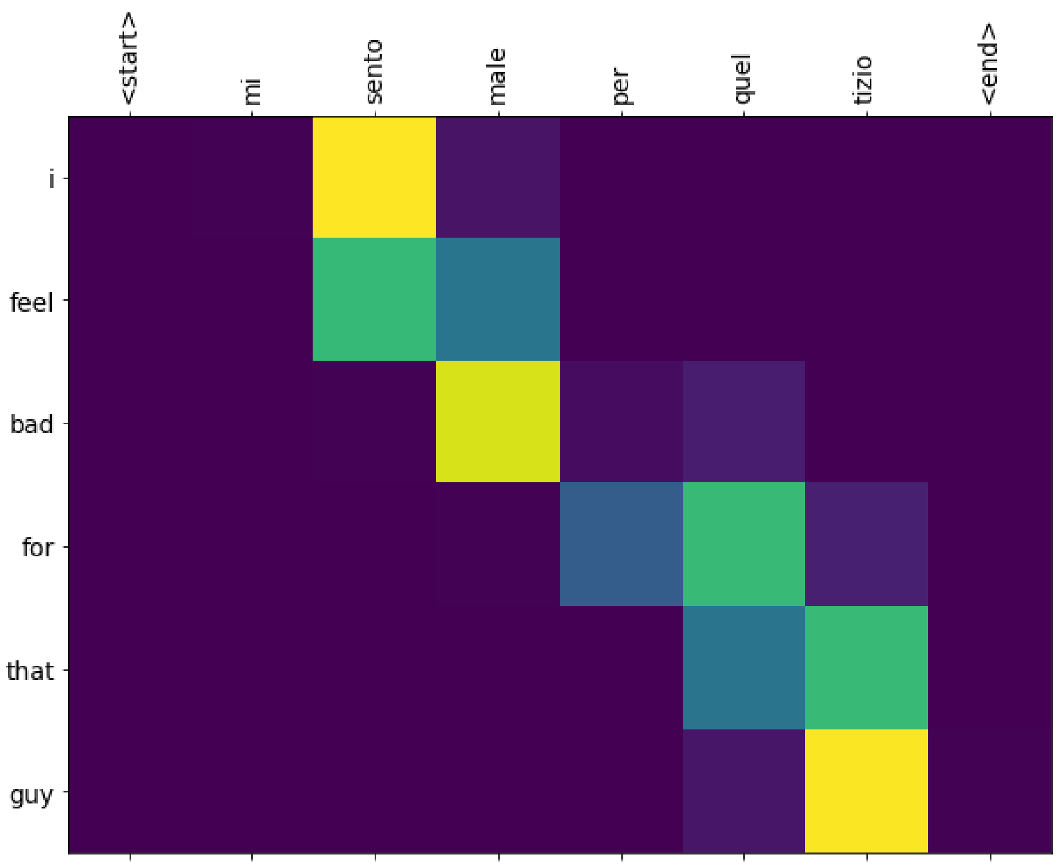
```
bleu_score= 0

italian_sentence = validation['italia_enc'][:1000]
english_conversion = [ sentence.lstrip('<start>').strip(' ') for sentence in vali
dation['english_in'][:1000]]
english_conversion
for sentence, actual_translation in zip(italian_sentence,english_conversion):
    # the original

    # traslated using model
    translation = predict_with_attention(sentence, model_concat, False)
    # print(sentence)
    # print(actual_translation)
    # print(translation)
    bleu_score += bleu.sentence_bleu(actual_translation.split(), translation.spli
t())

print(bleu_score/1000)
```

```
/usr/local/lib/python3.6/dist-packages/nltk/translate/bleu_score.py:490: UserWar
ning:
Corpus/Sentence contains 0 counts of 2-gram overlaps.
BLEU scores might be undesirable; use SmoothingFunction().
  warnings.warn(_msg)

0.25274042206807473
```

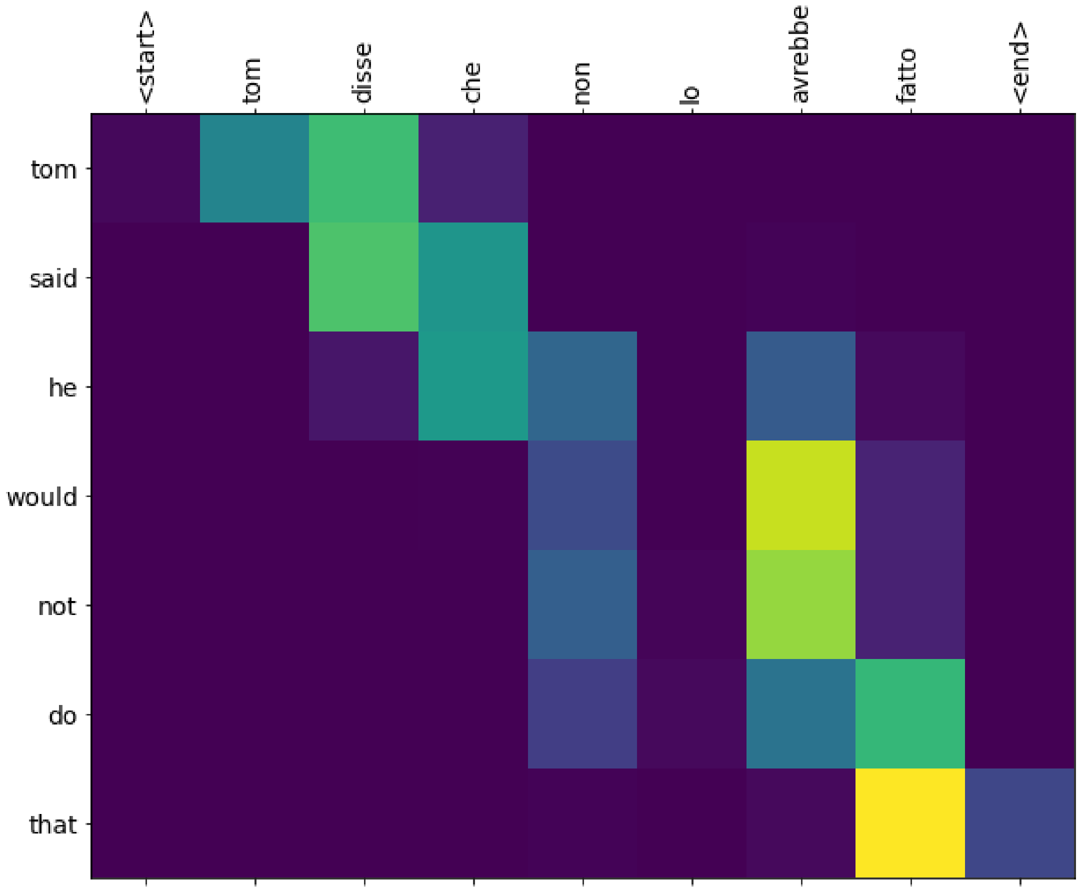
In [113]:
```
bleu_score= 0

italian_sentence = validation['italia_enc'][:10]
english_conversion = [ sentence.lstrip('<start>').strip(' ') for sentence in vali
dation['english_in'][:10]]
english_conversion
for sentence, actual_translation in zip(italian_sentence,english_conversion):
    # the original

    # traslated using model
    translation = predict_with_attention(sentence, model_concat, True)
    print(' italian sentence ',sentence)
    print(' actual english translation ',actual_translation)
    print(' predicted english translation ', translation)
    #bleu_score += bleu.sentence_bleu(actual_translation.split(), translation.spl
it())
```



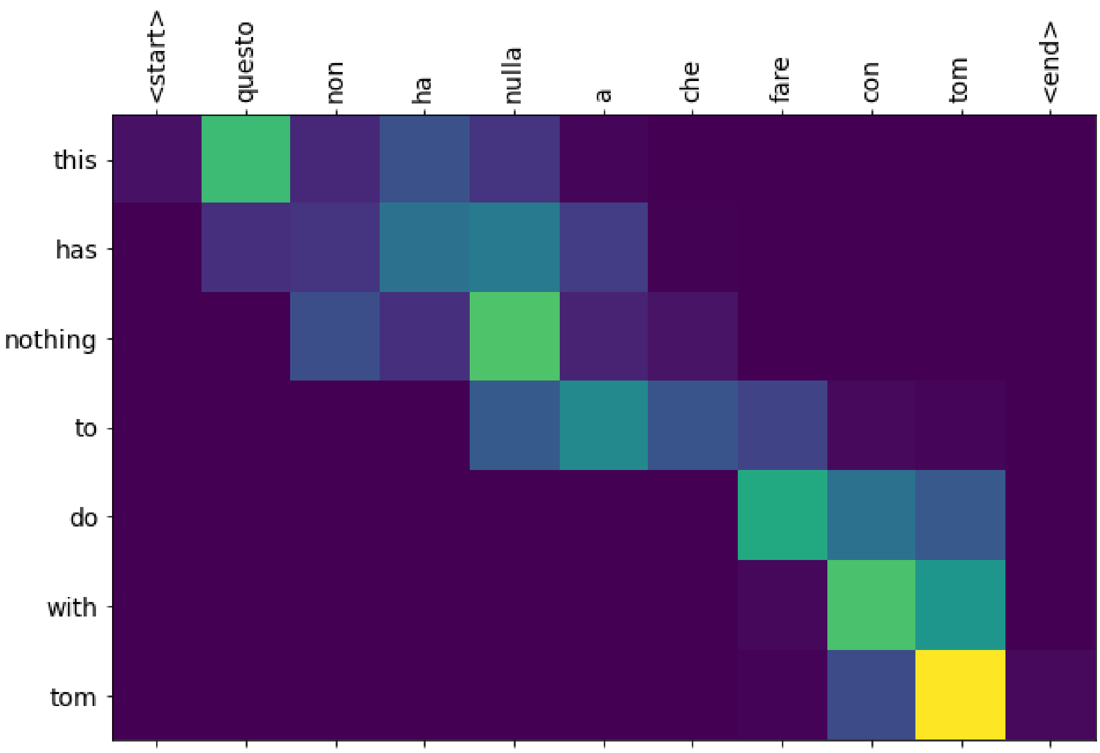

```
italian sentence  <start> il tuo inglese è migliorato molto <end>
actual english translation  your english has improved a lot
predicted english translation  your english has improved a lot
```



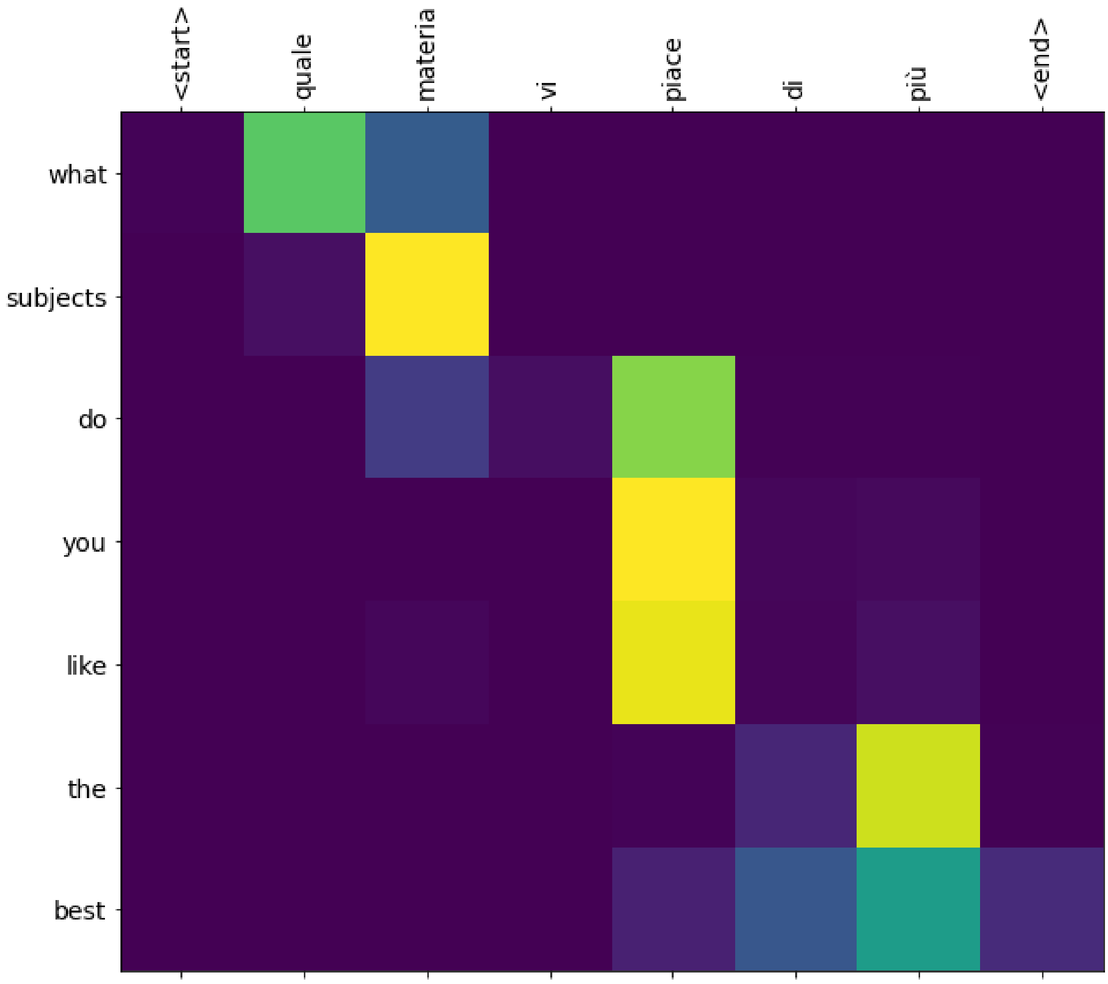

```
italian sentence  <start> mi sento male per quel tizio <end>
actual english translation  i feel bad for that guy
predicted english translation  i feel bad for that guy
```



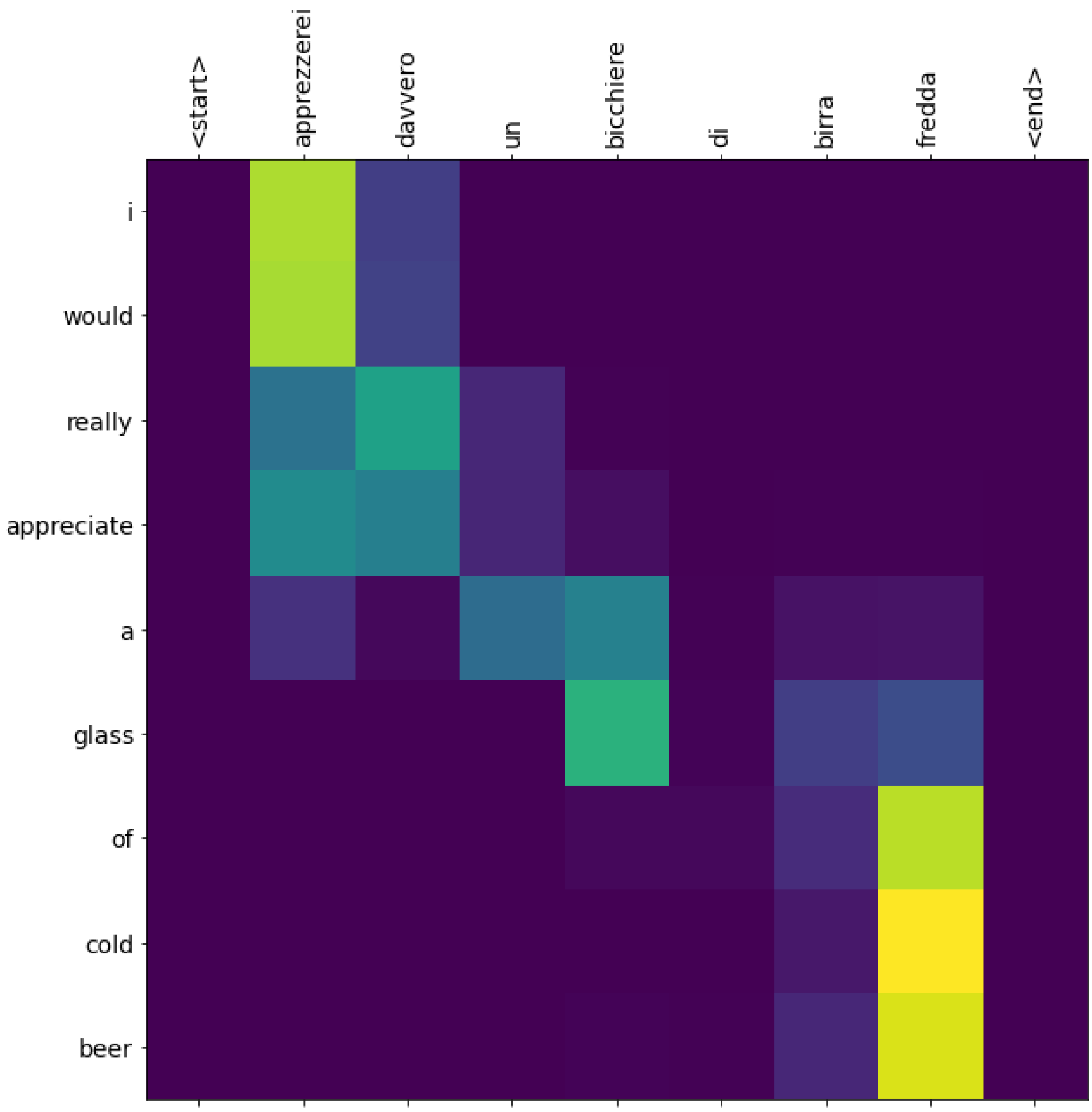

```
italian sentence  <start> tom disse che non lo avrebbe fatto <end>
actual english translation  tom said he would not do it
predicted english translation  tom said he would not do that
```



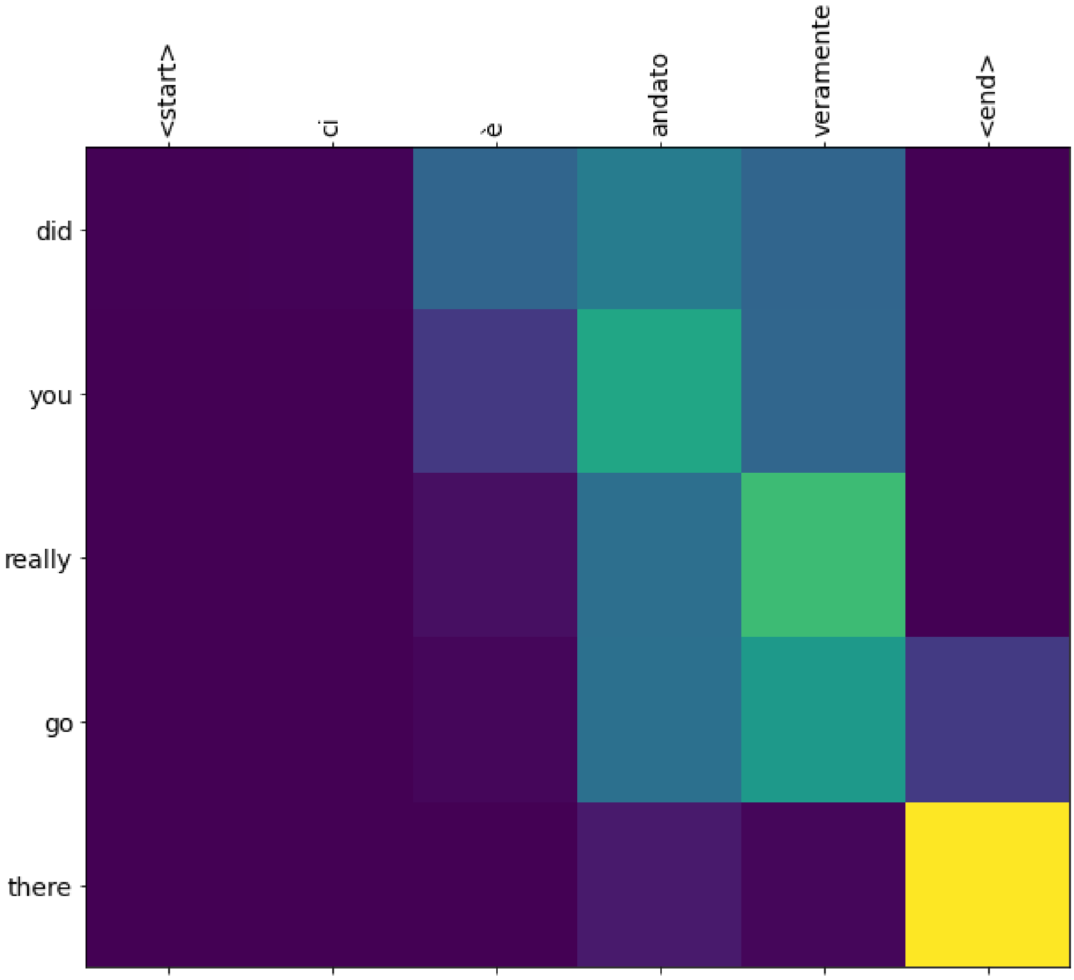

```
italian sentence  <start> questo non ha nulla a che fare con tom <end>
actual english translation  this has nothing to do with tom
predicted english translation  this has nothing to do with tom
```



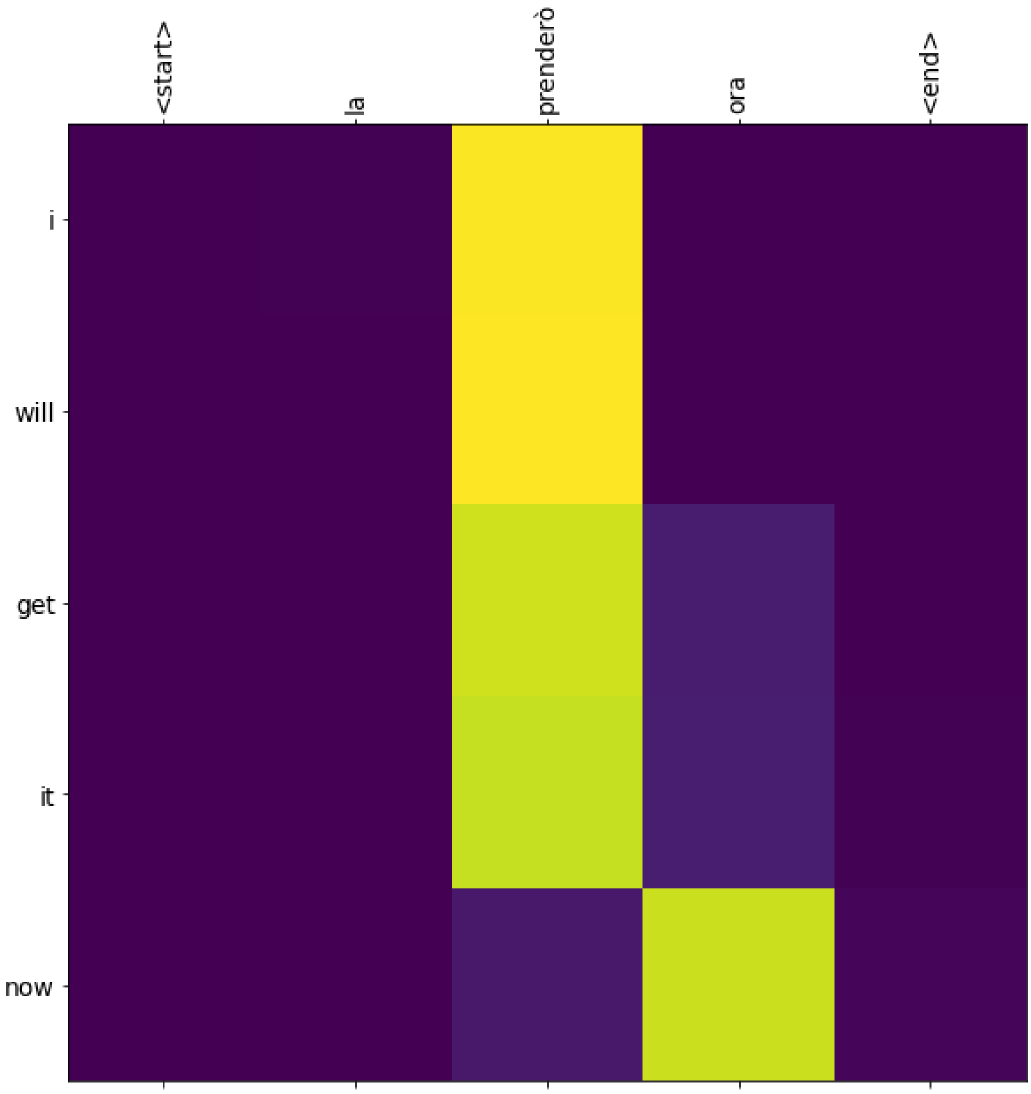

```
italian sentence  <start> quale materia vi piace di più <end>
actual english translation  what subject do you like best
predicted english translation  what subjects do you like the best
```



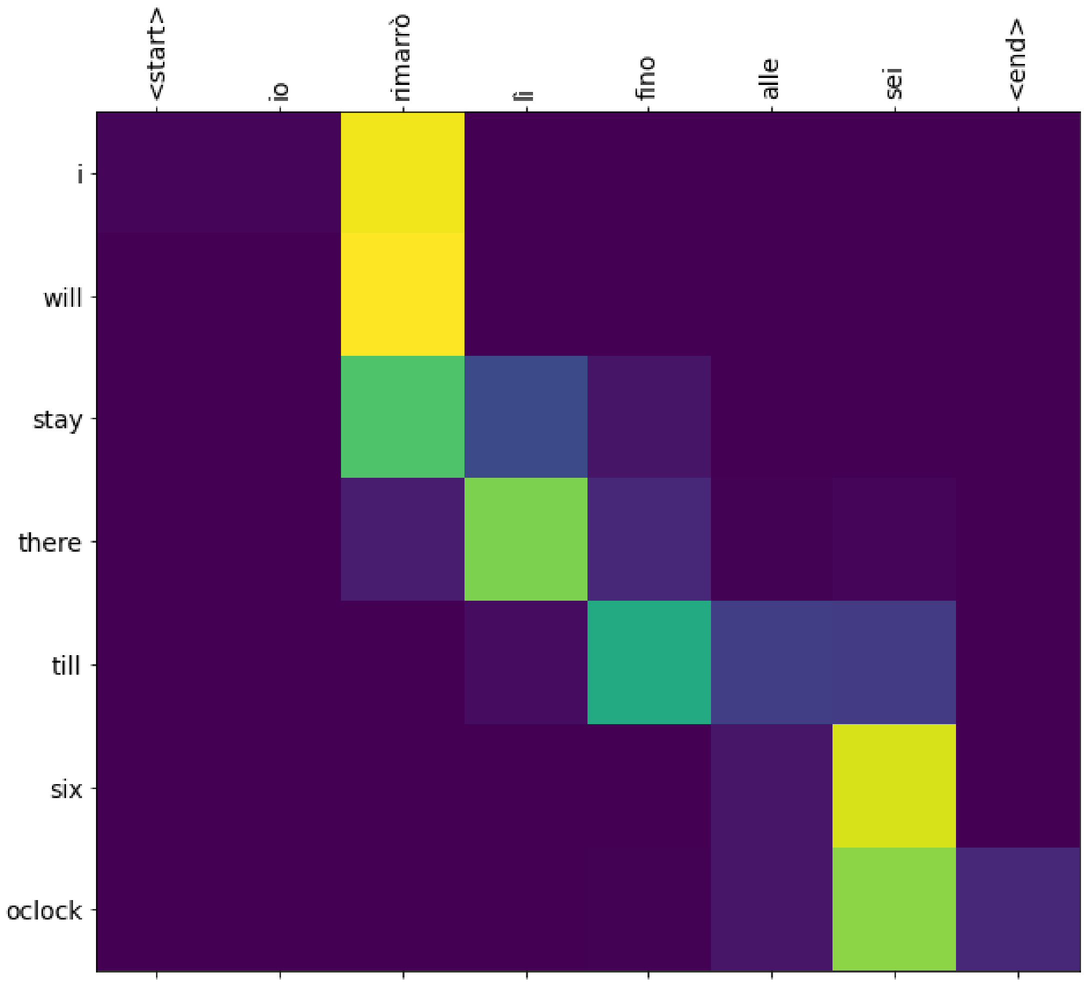

```
italian sentence  <start> apprezzerei davvero un bicchiere di birra fredda <end
>
actual english translation  i would really appreciate a glass of cold beer
predicted english translation  i would really appreciate a glass of cold beer
```



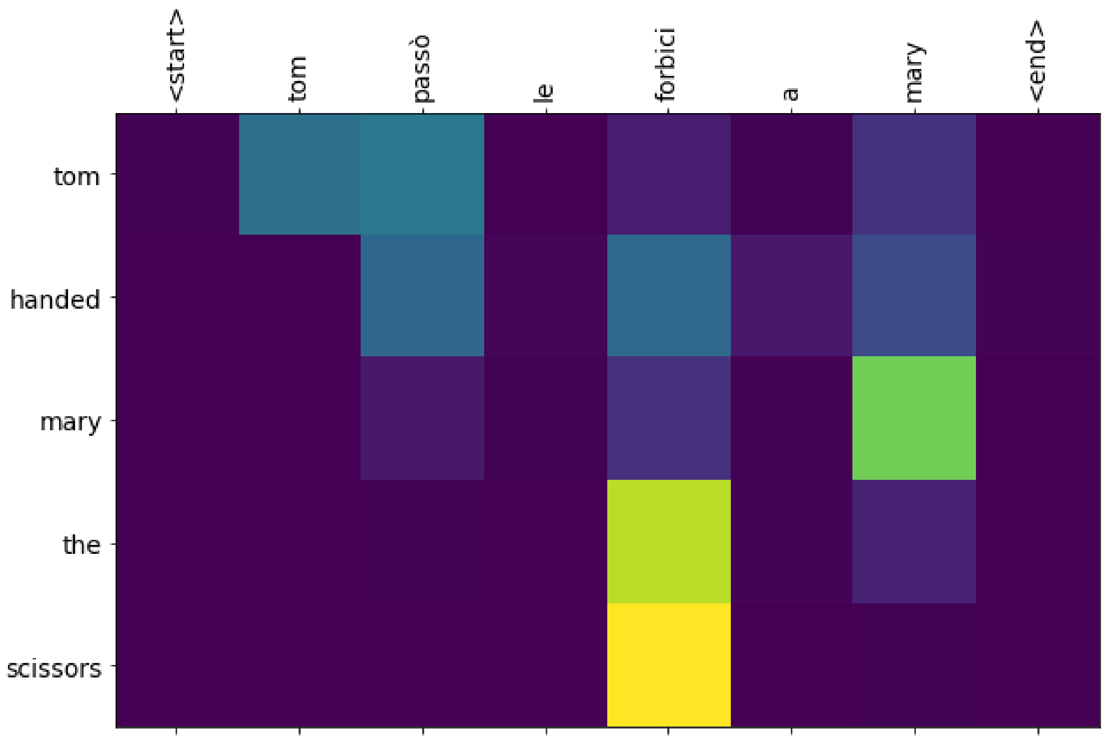

```
italian sentence  <start> ci è andato veramente <end>
actual english translation  did you really go there
predicted english translation  did you really go there
```




```
italian sentence  <start> la prenderò ora <end>
actual english translation  i will take it now
predicted english translation  i will get it now
```




```
italian sentence  <start> io rimarrò lì fino alle sei <end>
actual english translation  i will stay there till six oclock
predicted english translation  i will stay there till six oclock
```




```
italian sentence  <start> tom passò le forbici a mary <end>
actual english translation  tom handed mary the scissors
predicted english translation  tom handed mary the scissors
```

In [ ]:

Observation: The general function takes more time to train as compared to other functions, since there are more trainable parameters.

In [ ]: