MyBERT-local-machine

July 11, 2021

[1]:
```
%cd ~/content/
%rm -rf ~/content/ActionPrediction4CA
%rm -rf ~/content/ActionPredictionBERT
!git clone  --branch colab_exe https://github.com/jmcrav/ActionPrediction4CA.git
```

```
/home/gian/content
Clone in 'ActionPrediction4CA' in corso…
remote: Enumerating objects: 290, done.
remote: Counting objects: 100% (290/290), done.
remote: Compressing objects: 100% (206/206), done.
remote: Total 290 (delta 137), reused 220 (delta 77), pack-reused 0
Ricezione degli oggetti: 100% (290/290), 42.47 MiB | 10.48 MiB/s, fatto.
Risoluzione dei delta: 100% (137/137), fatto.
```

[2]:
```
%mkdir ~/content/ActionPredictionBERT ActionPredictionBERT/input_data␣
 ↪ActionPredictionBERT/extr_output
%cd ~/content/ActionPrediction4CA/tools
%mv extract_actions_fashion.py ~/content/ActionPredictionBERT
%mv action_evaluation.py ~/content/ActionPredictionBERT

%cd ~/content/ActionPrediction4CA/data/simmc_fashion/
%mv fashion_train_dials.json fashion_dev_dials.json␣
 ↪fashion_teststd_dials_public.json fashion_metadata.json␣
 ↪fashion_devtest_dials.json ~/content/ActionPredictionBERT/input_data
%cd ~/content/
%rm -rf ./ActionPrediction4CA/
```

```
/home/gian/content/ActionPrediction4CA/tools
/home/gian/content/ActionPrediction4CA/data/simmc_fashion
/home/gian/content
```

[3]:
```
%cd ~/content/ActionPredictionBERT/
!python extract_actions_fashion.py --json_path="./input_data/
 ↪fashion_train_dials.json ./input_data/fashion_dev_dials.json ./input_data/
 ↪fashion_devtest_dials.json" --save_root="./extr_output"  --metadata_path="./
 ↪fashion_metadata.json"
```

```
/home/gian/content/ActionPredictionBERT
```

```
Reading: ./input_data/fashion_train_dials.json
Dialogue task Id missing: 3406
Dialogue task Id missing: 3969
Dialogue task Id missing: 4847
Dialogue task Id missing: 321
Dialogue task Id missing: 3455
Dialogue task Id missing: 3414
Saving: ./extr_output/fashion_train_dials_api_calls.json
Reading: ./input_data/fashion_dev_dials.json
Dialogue task Id missing: 2117
Saving: ./extr_output/fashion_dev_dials_api_calls.json
Reading: ./input_data/fashion_devtest_dials.json
Dialogue task Id missing: 9308
Saving: ./extr_output/fashion_devtest_dials_api_calls.json
```

[4]:
```
#!pip install markdown
#!pip install transformers
#!pip install pandas
#!pip install torch
```

[5]:
```
#prima parte del fashion_train_dials
import json
import pandas as pd
with open ('./input_data/fashion_train_dials.json',"r") as f:
   data= json.load(f)

result=[]
row ={}
for k in data:
  row[k] = data[k]
# []
#prima parte del fashion_train_dials
import json
import pandas as pd
with open ('./input_data/fashion_train_dials.json',"r") as f:
   data= json.load(f)

result=[]
row ={}
for k in data:
  row[k] = data[k]
# []

dialogue_data = pd.json_normalize(row['dialogue_data'])
type(dialogue_data)
# dialogue = dialogue_data["dialogue"]
# for x in dialogue.head(1):
```

```python
#   display(x)
# #dialogue.head(1)
dialogue_data.head()



dialogue_data = pd.json_normalize(row['dialogue_data'])
type(dialogue_data)
# dialogue = dialogue_data["dialogue"]
# for x in dialogue.head(1):
#   display(x)
# #dialogue.head(1)
dialogue_data.head()
```

[5]:

```
                                            dialogue  dialogue_idx    domains  \
0  [{'belief_state': [{'act': 'DA:ASK:CHECK:CLOTH…          3094  [fashion]
1  [{'belief_state': [{'act': 'DA:INFORM:PREFER:C…           822  [fashion]
2  [{'belief_state': [{'act': 'DA:REQUEST:GET:CLO…          7411  [fashion]
3  [{'belief_state': [{'act': 'DA:INFORM:DISPREFE…          7029  [fashion]
4  [{'belief_state': [{'act': 'DA:INFORM:DISPREFE…          1506  [fashion]

   dialogue_task_id  dialogue_coref_map.1426  dialogue_coref_map.1429  \
0            1785.0                      0.0                      1.0
1            1720.0                      NaN                      NaN
2            2038.0                      NaN                      NaN
3            2011.0                      NaN                      NaN
4            1686.0                      NaN                      NaN

   dialogue_coref_map.708  dialogue_coref_map.712  dialogue_coref_map.2401  \
0                     NaN                     NaN                      NaN
1                     0.0                     1.0                      NaN
2                     NaN                     NaN                      4.0
3                     NaN                     NaN                      NaN
4                     NaN                     NaN                      NaN

   dialogue_coref_map.2402  …  dialogue_coref_map.2335  \
0                      NaN  …                      NaN
1                      NaN  …                      NaN
2                      0.0  …                      NaN
3                      NaN  …                      NaN
4                      NaN  …                      NaN

   dialogue_coref_map.713  dialogue_coref_map.1507  dialogue_coref_map.1509  \
0                     NaN                      NaN                      NaN
1                     NaN                      NaN                      NaN
2                     NaN                      NaN                      NaN
3                     NaN                      NaN                      NaN
```

```
4                   NaN                      NaN                      NaN

   dialogue_coref_map.949  dialogue_coref_map.1137  dialogue_coref_map.1872  \
0                     NaN                      NaN                      NaN
1                     NaN                      NaN                      NaN
2                     NaN                      NaN                      NaN
3                     NaN                      NaN                      NaN
4                     NaN                      NaN                      NaN

   dialogue_coref_map.1873  dialogue_coref_map.1753  dialogue_coref_map.834
0                      NaN                      NaN                     NaN
1                      NaN                      NaN                     NaN
2                      NaN                      NaN                     NaN
3                      NaN                      NaN                     NaN
4                      NaN                      NaN                     NaN

[5 rows x 1648 columns]
```

[6]:
```python
#seconda parte del fashion_train_dials
task_mapping = pd.json_normalize(row['task_mapping'])
task_mapping.head()
```

[6]:
```
   task_id                                          image_ids  focus_image  \
0     2042  [2441, 2442, 2443, 2444, 2445, 2446, 2447, 244…         2441
1     2041  [2431, 2432, 2433, 2434, 2435, 2436, 2437, 243…         2431
2     2040  [2421, 2422, 2423, 2424, 2425, 2426, 2427, 242…         2421
3     2039  [2411, 2412, 2413, 2414, 2415, 2416, 2417, 241…         2411
4     2038  [2401, 2402, 2403, 2404, 2405, 2406, 2407, 240…         2401

        memory_images                             database_images
0  [2442, 2443, 2444]  [2445, 2446, 2447, 2448, 2449, 2450]
1  [2432, 2433, 2434]  [2435, 2436, 2437, 2438, 2439, 2440]
2  [2422, 2423, 2424]  [2425, 2426, 2427, 2428, 2429, 2430]
3  [2412, 2413, 2414]  [2415, 2416, 2417, 2418, 2419, 2420]
4  [2402, 2403, 2404]  [2405, 2406, 2407, 2408, 2409, 2410]
```

[7]:
```python
import pandas as pd
dev_dials_api = pd.read_json('./extr_output/fashion_dev_dials_api_calls.json')
dev_dials_api.head()
```

[7]:
```
   dialog_id                                            actions  \
0       4146  [{'turn_idx': 0, 'action': 'None', 'action_sup…
1       4260  [{'turn_idx': 0, 'action': 'SpecifyInfo', 'act…
2       8022  [{'turn_idx': 0, 'action': 'SearchDatabase', '…
3       4992  [{'turn_idx': 0, 'action': 'None', 'action_sup…
4       5606  [{'turn_idx': 0, 'action': 'None', 'action_sup…
```

```
                                  focus_images
0  [1646, 1646, 1646, 1649, 1649, 1649, 1649]
1                    [2161, 2161, 2161, 2161]
2        [1971, 1972, 1972, 1972, 1977, 1978]
3              [1931, 1931, 1936, 1936, 1936]
4              [1931, 1931, 1931, 1931, 1931]
```

[8]:
```
import pandas as pd
devtest_dials_api = pd.read_json('./extr_output/fashion_devtest_dials_api_calls.
↪json')
devtest_dials_api.head()
```

[8]:
```
   dialog_id                                            actions  \
0       2494  [{'turn_idx': 0, 'action': 'SearchDatabase', '…
1       3731  [{'turn_idx': 0, 'action': 'SearchDatabase', '…
2       8546  [{'turn_idx': 0, 'action': 'SpecifyInfo', 'act…
3       5590  [{'turn_idx': 0, 'action': 'SearchDatabase', '…
4       5452  [{'turn_idx': 0, 'action': 'SpecifyInfo', 'act…

                    focus_images
0  [1836, 1841, 1841, 1841, 1841]
1  [1676, 1681, 1681, 1683, 1683]
2  [840, 840, 840, 849, 849, 843]
3  [1616, 1618, 1618, 1618, 1618]
4  [2231, 2231, 2231, 2236, 2236]
```

[9]:
```
import pandas as pd
import json

def createDataframe(json_file):
  with open(json_file) as f:
    dictftdac = json.load(f)

  data = []

  for e in dictftdac:
    dialog_id = e['dialog_id']
    actions = e['actions']
    focus_images = e['focus_images']

    for a in actions:

      turn_idx = a['turn_idx']
      action = a['action']
      action_supervision = a['action_supervision']
      transcript = a['transcript']
      transcript_annotated = a['transcript_annotated']
```

```
        system_transcript = a['system_transcript']
        system_transcript_annotated = a['system_transcript_annotated']

        row = {
            "dialog_id" : dialog_id,
            'turn_idx' : turn_idx,
            'action' : action,
            'action_supervision' : action_supervision,
            'focus_images' : focus_images,
            'transcript': transcript,
            'transcript_annotated': transcript_annotated,
            'system_transcript': system_transcript,
            'system_transcript_annotated':system_transcript_annotated,
            'previous_transcript': "",
            'previous_system_transcript': ""
        }
        if (action_supervision != None):
          if 'focus' in action_supervision:
            acsf = {'focus':action_supervision['focus']}
          else:
            acsf = {'focus':None}

          if 'attributes' in action_supervision:
            acsa = {'attributes':action_supervision['attributes']}
          else:
            acsa = {'attributes':[]}
        else:
            acsf = {'focus':None}
            acsa = {'attributes':[]}


        row.update(acsf)
        row.update(acsa)

        data.append(row)

    # Conservo id turno e risposta sistema per provare a implementare una␣
  ↪soluzione articolata
    df = pd.
  ↪DataFrame(data,columns=['dialog_id','turn_idx','transcript','action','attributes',␣
  ↪'system_transcript','transcript_annotated','system_transcript_annotated','previous_transcri

    return df
```

[10]:
```
df_training = createDataframe('./extr_output/fashion_train_dials_api_calls.
  ↪json')
print("Training: ",len(df_training)," elementi")
```

```
Training:  21196  elementi
```

[11]:
```
df_validation = createDataframe('./extr_output/fashion_dev_dials_api_calls.
↪json')
print("Validation: ",len(df_validation)," elementi")
```

```
Validation:  3513  elementi
```

[12]:
```
df_test = createDataframe('./extr_output/fashion_devtest_dials_api_calls.json')
print("Test: ",len(df_test)," elementi")
```

```
Test:  5397  elementi
```

[13]:
```
use_next = True
```

[14]:
```
#Training
df_training.sort_values(by=['dialog_id', 'turn_idx'])
for i in range(1,(len(df_training))):
  if(i<(len(df_training)) and  df_training['dialog_id'][i] ==␣
↪df_training['dialog_id'][i-1]):
    df_training.loc[i,'previous_transcript'] = df_training['transcript'][i-1]
    df_training.loc[i,'previous_system_transcript'] =␣
↪df_training['system_transcript'][i-1]

#Validation
df_validation.sort_values(by=['dialog_id', 'turn_idx'])
for i in range(1,(len(df_validation))):
  if(i<(len(df_validation)) and  df_validation['dialog_id'][i] ==␣
↪df_validation['dialog_id'][i-1]):
    df_validation.loc[i,'previous_transcript'] =␣
↪df_validation['transcript'][i-1]
    df_validation.loc[i,'previous_system_transcript'] =␣
↪df_validation['system_transcript'][i-1]

#Evaluation
df_test.sort_values(by=['dialog_id', 'turn_idx'])
for i in range(1,(len(df_test))):
  if(i<(len(df_test)) and  df_test['dialog_id'][i] ==␣
↪df_test['dialog_id'][i-1]):
    df_test.loc[i,'previous_transcript'] = df_test['transcript'][i-1]
    df_test.loc[i,'previous_system_transcript'] =␣
↪df_test['system_transcript'][i-1]
```

[15]:
```
from transformers import BertTokenizer

# Load the BERT tokenizer.
print('Loading BERT tokenizer...')
```

```
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased',␣
↪do_lower_case=True)
```

```
Loading BERT tokenizer…
```

[16]:
```
transcripts_tr = df_training.transcript.values
previous_transcript_tr = df_training.previous_transcript.values
previous_system_transcript_tr = df_training.previous_system_transcript.values
action_labels_tr = df_training.action.values
attributes_labels_tr=df_training.attributes.values

print ("TRAINING DATA:")
# Print the original sentence.
print(' Original: ', transcripts_tr[0])

# Print the sentence split into tokens.
print('Tokenized: ', tokenizer.tokenize(transcripts_tr[0]))

# Print the sentence mapped to token ids.
print('Token IDs: ', tokenizer.convert_tokens_to_ids(tokenizer.
↪tokenize(transcripts_tr[0])))
```

```
TRAINING DATA:
 Original:  Is there a pattern on this one? It's hard to see in the image.
Tokenized:  ['is', 'there', 'a', 'pattern', 'on', 'this', 'one', '?', 'it', "'",
's', 'hard', 'to', 'see', 'in', 'the', 'image', '.']
Token IDs:  [2003, 2045, 1037, 5418, 2006, 2023, 2028, 1029, 2009, 1005, 1055,
2524, 2000, 2156, 1999, 1996, 3746, 1012]
```

[17]:
```
for k in range(0,10):
  print(f"T: {transcripts_tr[k]} | PT: {previous_transcript_tr[k]} | PST:␣
↪{previous_system_transcript_tr[k]}")
```

```
T: Is there a pattern on this one? It's hard to see in the image. | PT:  | PST:
T: That's fancy. Do you have anything in warmer colors like yellow or red? | PT:
Is there a pattern on this one? It's hard to see in the image. | PST: I don't
have any information on the pattern, but it has pointelle embellishments.
T: Yeah, that sounds good. | PT: That's fancy. Do you have anything in warmer
colors like yellow or red? | PST: I have a crew neck sweater in red, would you
like to see it?
T: Oh, I love that. Please tell me you have a small. | PT: Yeah, that sounds
good. | PST: This is $187 from Downtown Stylists with a 3.62 rating.
T: Yes, please! Thank you for your help with this | PT: Oh, I love that. Please
tell me you have a small. | PST: It does come in small, shall I put one in your
cart?
T: How nice! Does this come in other colors? | PT:  | PST:
T: Oh well.  Can you show me a dress that comes in red? | PT: How nice! Does
this come in other colors? | PST: No, I'm sorry, It comes only in blue.
```

```
T: Cute! Do these come in Small? | PT: Oh well.  Can you show me a dress that
comes in red? | PST: This dress comes in many colors, including a bright red and
a pinkish-red. What do you think?
T: Awesome. Would you add a red one in S to my cart please? | PT: Cute! Do these
come in Small? | PST: Yes, they do!
T: That's all. Thanks! | PT: Awesome. Would you add a red one in S to my cart
please? | PST: The red one is in your cart. Is there anything else I can find
for you?
```

[18]:

```
transcripts_vd = df_validation.transcript.values
previous_transcript_vd = df_validation.previous_transcript.values
previous_system_transcript_vd = df_validation.previous_system_transcript.values
action_labels_vd = df_validation.action.values
attributes_labels_vd=df_validation.attributes.values
dialog_ids_vd = df_validation.dialog_id.values
turn_idxs_vd = df_validation.turn_idx.values

print ("VALIDATION DATA:")

# Print the original sentence.
print(' Original: ', transcripts_vd[0])

# Print the sentence split into tokens.
print('Tokenized: ', tokenizer.tokenize(transcripts_vd[0]))

# Print the sentence mapped to token ids.
print('Token IDs: ', tokenizer.convert_tokens_to_ids(tokenizer.
 ↪tokenize(transcripts_vd[0])))

# Print the dialog ids.
print(f"Dialog IDs: {dialog_ids_vd[0:20]}")

# Print the turn idxs.
print(f"Turn IDs: {turn_idxs_vd[0:20]}")
```

```
VALIDATION DATA:
 Original:  What's the price of this sweater compared to the other blue and gray
one I looked at?
Tokenized:  ['what', "'", 's', 'the', 'price', 'of', 'this', 'sweater',
'compared', 'to', 'the', 'other', 'blue', 'and', 'gray', 'one', 'i', 'looked',
'at', '?']
Token IDs:  [2054, 1005, 1055, 1996, 3976, 1997, 2023, 14329, 4102, 2000, 1996,
2060, 2630, 1998, 3897, 2028, 1045, 2246, 2012, 1029]
Dialog IDs: [4146 4146 4146 4146 4146 4146 4146 4260 4260 4260 4260 8022 8022
8022
 8022 8022 8022 4992 4992 4992]
Turn IDs: [0 1 2 3 4 5 6 0 1 2 3 0 1 2 3 4 5 0 1 2]
```

[19]:
```
transcripts_tst = df_test.transcript.values
previous_transcript_tst = df_test.previous_transcript.values
previous_system_transcript_tst = df_test.previous_system_transcript.values
action_labels_tst = df_test.action.values
attributes_labels_tst=df_test.attributes.values
dialog_ids_tst = df_test.dialog_id.values
turn_idxs_tst = df_test.turn_idx.values

print ("EVALUATION DATA:")

# Print the original sentence.
print(' Original: ', transcripts_tst[0])

# Print the sentence split into tokens.
print('Tokenized: ', tokenizer.tokenize(transcripts_tst[0]))

# Print the sentence mapped to token ids.
print('Token IDs: ', tokenizer.convert_tokens_to_ids(tokenizer.
 ↪tokenize(transcripts_tst[0])))

# Print the dialog ids.
print(f"Dialog IDs: {dialog_ids_tst[0:20]}")

# Print the turn idxs.
print(f"Turn IDs: {turn_idxs_tst[0:20]}")
```

```
EVALUATION DATA:
 Original:  That looks a little too light for what I need, do you have something
else with a high customer rating?
Tokenized:  ['that', 'looks', 'a', 'little', 'too', 'light', 'for', 'what', 'i',
'need', ',', 'do', 'you', 'have', 'something', 'else', 'with', 'a', 'high',
'customer', 'rating', '?']
Token IDs:  [2008, 3504, 1037, 2210, 2205, 2422, 2005, 2054, 1045, 2342, 1010,
2079, 2017, 2031, 2242, 2842, 2007, 1037, 2152, 8013, 5790, 1029]
Dialog IDs: [2494 2494 2494 2494 2494 3731 3731 3731 3731 3731 8546 8546 8546
8546
 8546 8546 5590 5590 5590 5590]
Turn IDs: [0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 5 0 1 2 3]
```

[20]:
```
max_len_tr = 0

# For every sentence...
for i in range(0,len(transcripts_tr)):

    # Tokenize the text and add `[CLS]` and `[SEP]` tokens.

    if (previous_transcript_tr[i] != "" and use_next):
```

```
        input_ids = tokenizer.encode(previous_transcript_tr[i]+ " " +␣
↪previous_system_transcript_tr[i],transcripts_tr[i], add_special_tokens=True)
    else:
        input_ids = tokenizer.encode(transcripts_tr[i], add_special_tokens=True)

    # Update the maximum sentence length.
    max_len_tr = max(max_len_tr, len(input_ids))

print('Max transcript length for training: ', max_len_tr)
```

```
Max transcript length for training:  177
```

[21]:
```
max_len_vd = 0

# For every sentence...
for i in range(0,len(transcripts_vd)):

    # Tokenize the text and add `[CLS]` and `[SEP]` tokens.
    if (previous_transcript_vd[i] != "" and use_next):
        input_ids = tokenizer.encode(previous_transcript_vd[i]+ " " +␣
↪previous_system_transcript_vd[i],transcripts_vd[i], add_special_tokens=True)
    else:
        input_ids = tokenizer.encode(transcripts_vd[i], add_special_tokens=True)

    # Update the maximum sentence length.
    max_len_vd = max(max_len_vd, len(input_ids))

print('Max transcript length for validation: ', max_len_vd)
```

```
Max transcript length for validation:  133
```

[22]:
```
max_len_tst = 0

#non sono sicuro che il controllo della lunghezza vada fatto anche sul test␣
↪set, dopo la performance non è determinata
#dalla conoscenza del test set?
#è anche vero che in teoria per far funzionare BERT bisogna dargli in pasto dei␣
↪dati tokenizzati, quindi in un caso reale il nostro
#model non potrebbe prendere in ingresso del testo non trattato. Nel dubbio ho␣
↪controllato le dimensioni

for i in range(0,len(transcripts_tst)):
    # Tokenize the text and add `[CLS]` and `[SEP]` tokens.
    if (previous_transcript_tst[i] != "" and use_next):
        input_ids = tokenizer.encode(previous_transcript_tst[i]+ " " +␣
↪previous_system_transcript_tst[i],transcripts_tst[i],␣
↪add_special_tokens=True)
```

```
    else:
      input_ids = tokenizer.encode(transcripts_tst[i], add_special_tokens=True)

    # Update the maximum sentence length.
    max_len_tst = max(max_len_tst, len(input_ids))

print("Max transcript length for evaluation: ",max_len_tst)
```

```
Max transcript length for evaluation:  150
```

[23]:
```
max_len = max(max_len_tr, max_len_vd, max_len_tst)

# if (max_len_tr >= max_len_vd):
#   max_len = max_len_tr
# else:
#   max_len = max_len_vd
# if (max_len_tst >= max_len):
#   max_len = max_len_tst

print("La massima lunghezza dei token da gestire è quindi ",max_len)
```

```
La massima lunghezza dei token da gestire è quindi  177
```

[24]:
```
from sklearn.preprocessing import MultiLabelBinarizer
import numpy as np

mlb = MultiLabelBinarizer()
attributes_labels_all = np.concatenate((attributes_labels_tr,␣
 ↪attributes_labels_vd,attributes_labels_tst), axis=None)
attr_yt = mlb.fit_transform(attributes_labels_all)
print(attr_yt[0:15])
print(mlb.inverse_transform(attr_yt[3].reshape(1, -1)))
print(mlb.classes_)
print(f"Totale: {len(attr_yt)}, Training: {len(attributes_labels_tr)},␣
 ↪Validation: {len(attributes_labels_vd)}, Evaluation:␣
 ↪{len(attributes_labels_tst)}")
attributes_labels_tr_vect = attr_yt[0:len(attributes_labels_tr)]
attributes_labels_vd_vect = attr_yt[len(attributes_labels_tr):
 ↪(len(attributes_labels_tr)+len(attributes_labels_vd))]
attributes_labels_tst_vect =␣
 ↪attr_yt[(len(attributes_labels_tr)+len(attributes_labels_vd)):]
print(f"Training: {len(attributes_labels_tr_vect)}, Validation:␣
 ↪{len(attributes_labels_vd_vect)}, Evaluation:␣
 ↪{len(attributes_labels_tst_vect)}")
```

```
[[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]]
[('availableSizes',)]
['ageRange' 'amountInStock' 'availableSizes' 'brand' 'clothingCategory'
 'clothingStyle' 'color' 'customerRating' 'dressStyle' 'embellishment'
 'forGender' 'forOccasion' 'hasPart' 'hemLength' 'hemStyle' 'info'
 'jacketStyle' 'madeIn' 'material' 'necklineStyle' 'pattern' 'price'
 'sequential' 'size' 'skirtLength' 'skirtStyle' 'sleeveLength'
 'sleeveStyle' 'soldBy' 'sweaterStyle' 'waistStyle' 'warmthRating'
 'waterResistance']
Totale: 30106, Training: 21196, Validation: 3513, Evaluation: 5397
Training: 21196, Validation: 3513, Evaluation: 5397
```

[25]:

```python
import torch
import tensorflow as tf
torch.backends.cudnn.benchmark = True
torch.backends.cudnn.enabled = True
```

[26]:

```python
# Tokenize all of the sentences and map the tokens to thier word IDs.

#dobbiamo convertire le nostre lables da string a valori numerici, usiamo il
metodo fornito da sklearn

#TRAINING DATASET
from sklearn import preprocessing

le = preprocessing.LabelEncoder()
action_labels_encoded_tr = le.fit_transform(action_labels_tr)

input_ids_tr = []
attention_masks_tr = []
print(f"{len(df_training)} records to encode.")
# For every sentence...
for i in range(0,len(df_training)):
    # `encode_plus` will:
    #   (1) Tokenize the sentence.
```

```
    #   (2) Prepend the `[CLS]` token to the start.
    #   (3) Append the `[SEP]` token to the end.
    #   (4) Map tokens to their IDs.
    #   (5) Pad or truncate the sentence to `max_length`
    #   (6) Create attention masks for [PAD] tokens.

  if (previous_transcript_tr[i] != "" and use_next):
    encoded_dict = tokenizer.encode_plus(
                        previous_transcript_tr[i]+ " " +␣
↪previous_system_transcript_tr[i],  # Sentence to encode.
                        transcripts_tr[i], #next sentece to encode
                        add_special_tokens = True, # Add '[CLS]' and '[SEP]'
                        truncation = True,
                        max_length = max_len,           # Pad & truncate all␣
↪sentences.
                        pad_to_max_length = True,
                        return_attention_mask = True,   # Construct attn. masks.
                        return_tensors = 'pt',     # Return pytorch tensors.
                    )
  else:
    encoded_dict = tokenizer.encode_plus(
                        transcripts_tr[i],  # Sentence to encode.
                        add_special_tokens = True, # Add '[CLS]' and '[SEP]'
                        truncation = True,
                        max_length = max_len,           # Pad & truncate all␣
↪sentences.
                        pad_to_max_length = True,
                        return_attention_mask = True,   # Construct attn. masks.
                        return_tensors = 'pt',     # Return pytorch tensors.
                    )



  # Add the encoded sentence to the list.
  input_ids_tr.append(encoded_dict['input_ids'])

  # And its attention mask (simply differentiates padding from non-padding).
  attention_masks_tr.append(encoded_dict['attention_mask'])

# Convert the lists into tensors.
input_ids_tr = torch.cat(input_ids_tr, dim=0)
attention_masks_tr = torch.cat(attention_masks_tr, dim=0)
labels_actions_tr = torch.tensor(action_labels_encoded_tr)
labels_attributes_tr = torch.tensor(attributes_labels_tr_vect)

# Print sentence 0, now as a list of IDs.
print ("TRAINING : ")
```

```
if (use_next):
  print('Original: ', transcripts_tr[0])
else:
  print('Original: ', transcripts_tr[0])
print('Token IDs:', input_ids_tr[0])
```

```
21196 records to encode.
```

```
/home/gian/anaconda3/envs/testcuda1/lib/python3.8/site-
packages/transformers/tokenization_utils_base.py:2126: FutureWarning: The
`pad_to_max_length` argument is deprecated and will be removed in a future
version, use `padding=True` or `padding='longest'` to pad to the longest
sequence in the batch, or use `padding='max_length'` to pad to a max length. In
this case, you can give a specific length with `max_length` (e.g.
`max_length=45`) or leave max_length to None to pad to the maximal input size of
the model (e.g. 512 for Bert).
  warnings.warn(
```

```
TRAINING :
Original:  Is there a pattern on this one? It's hard to see in the image.
Token IDs: tensor([ 101, 2003, 2045, 1037, 5418, 2006, 2023, 2028, 1029, 2009,
1005, 1055,
        2524, 2000, 2156, 1999, 1996, 3746, 1012,  102,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0])
```

[27]:
```
# Tokenize all of the sentences and map the tokens to thier word IDs.

#dobbiamo convertire le nostre lables da string a valori numerici, usiamo il␣
↪metodo fornito da sklearn

#VALIDATION DATASET
from sklearn import preprocessing

le = preprocessing.LabelEncoder()
action_labels_encoded_vd = le.fit_transform(action_labels_vd)
```

```python
input_ids_vd = []
attention_masks_vd = []
print(f"{len(df_validation)} records to encode.")
# For every sentence...
for i in range(0,len(df_validation)):
  # `encode_plus` will:
  #   (1) Tokenize the sentence.
  #   (2) Prepend the `[CLS]` token to the start.
  #   (3) Append the `[SEP]` token to the end.
  #   (4) Map tokens to their IDs.
  #   (5) Pad or truncate the sentence to `max_length`
  #   (6) Create attention masks for [PAD] tokens.

  if (previous_transcript_vd[i] != "" and use_next):
    encoded_dict = tokenizer.encode_plus(
                        previous_transcript_vd[i]+ " " +␣
 ↪previous_system_transcript_vd[i],  # Sentence to encode.
                        transcripts_vd[i], #next sentece to encode
                        add_special_tokens = True, # Add '[CLS]' and '[SEP]'
                        truncation = True,
                        max_length = max_len,           # Pad & truncate all␣
 ↪sentences.
                        pad_to_max_length = True,
                        return_attention_mask = True,   # Construct attn. masks.
                        return_tensors = 'pt',     # Return pytorch tensors.
                    )
  else:
    encoded_dict = tokenizer.encode_plus(
                        transcripts_vd[i],  # Sentence to encode.
                        add_special_tokens = True, # Add '[CLS]' and '[SEP]'
                        truncation = True,
                        max_length = max_len,           # Pad & truncate all␣
 ↪sentences.
                        pad_to_max_length = True,
                        return_attention_mask = True,   # Construct attn. masks.
                        return_tensors = 'pt',     # Return pytorch tensors.
                    )

  # Add the encoded sentence to the list.
  input_ids_vd.append(encoded_dict['input_ids'])

  # And its attention mask (simply differentiates padding from non-padding).
  attention_masks_vd.append(encoded_dict['attention_mask'])

# Convert the lists into tensors.
input_ids_vd = torch.cat(input_ids_vd, dim=0)
attention_masks_vd = torch.cat(attention_masks_vd, dim=0)
```

```python
labels_actions_vd = torch.tensor(action_labels_encoded_vd)
labels_attributes_vd = torch.tensor(attributes_labels_vd_vect)
# Check warning:
# /usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:43: UserWarning:␣
↪To copy construct from a tensor, it is recommended to use sourceTensor.
↪clone().detach() or sourceTensor.clone().detach().requires_grad_(True),␣
↪rather than torch.tensor(sourceTensor).
dialog_ids_vd = torch.tensor(dialog_ids_vd)
turn_idxs_vd = torch.tensor(turn_idxs_vd)

# Print sentence 0, now as a list of IDs.
print ("VALIDATION : ")
if (use_next):
  print('Original: ', transcripts_vd[0])
else:
  print('Original: ', transcripts_vd[0])
print('Token IDs:', input_ids_vd[0])
print(f"Dialog IDs: {dialog_ids_vd[0:20]}")
print(f"Turn IDXs: {turn_idxs_vd[0:20]}")
```

```
3513 records to encode.
VALIDATION :
Original:  What's the price of this sweater compared to the other blue and gray
one I looked at?
Token IDs: tensor([  101,  2054,  1005,  1055,  1996,  3976,  1997,  2023,
14329,  4102,
         2000,  1996,  2060,  2630,  1998,  3897,  2028,  1045,  2246,  2012,
         1029,   102,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0])
Dialog IDs: tensor([4146, 4146, 4146, 4146, 4146, 4146, 4146, 4260, 4260, 4260,
4260, 8022,
        8022, 8022, 8022, 8022, 8022, 4992, 4992, 4992])
Turn IDXs: tensor([0, 1, 2, 3, 4, 5, 6, 0, 1, 2, 3, 0, 1, 2, 3, 4, 5, 0, 1, 2])
```

[28]:
```
# Tokenize all of the sentences and map the tokens to thier word IDs.

#dobbiamo convertire le nostre lables da string a valori numerici, usiamo il␣
↪metodo fornito da sklearn

#VALIDATION DATASET
from sklearn import preprocessing

le = preprocessing.LabelEncoder()
action_labels_encoded_tst = le.fit_transform(action_labels_tst)

input_ids_tst = []
attention_masks_tst = []
print(f"{len(df_test)} records to encode.")
# For every sentence...
for i in range(0,len(df_test)):
# for t in transcripts_tst:
    # `encode_plus` will:
    #   (1) Tokenize the sentence.
    #   (2) Prepend the `[CLS]` token to the start.
    #   (3) Append the `[SEP]` token to the end.
    #   (4) Map tokens to their IDs.
    #   (5) Pad or truncate the sentence to `max_length`
    #   (6) Create attention masks for [PAD] tokens.

  #Aggiungere "and False" PER UTILIZZARE sempre la tokenizzazione senza␣
↪concatenazione
  if (previous_transcript_tst[i] != "" and use_next):
    encoded_dict = tokenizer.encode_plus(
                        previous_transcript_tst[i]+ " " +␣
↪previous_system_transcript_tst[i],  # Sentence to encode.
                        transcripts_tst[i], #next sentece to encode
                        add_special_tokens = True, # Add '[CLS]' and '[SEP]'
                        truncation = True,
                        max_length = max_len,           # Pad & truncate all␣
↪sentences.
                        pad_to_max_length = True,
                        return_attention_mask = True,   # Construct attn. masks.
                        return_tensors = 'pt',     # Return pytorch tensors.
                   )
  else:
    encoded_dict = tokenizer.encode_plus(
                        transcripts_tst[i],  # Sentence to encode.
                        add_special_tokens = True, # Add '[CLS]' and '[SEP]'
                        truncation = True,
                        max_length = max_len,           # Pad & truncate all␣
↪sentences.
```

```
                        pad_to_max_length = True,
                        return_attention_mask = True,   # Construct attn. masks.
                        return_tensors = 'pt',     # Return pytorch tensors.
                   )

    # Add the encoded sentence to the list.
    input_ids_tst.append(encoded_dict['input_ids'])

    # And its attention mask (simply differentiates padding from non-padding).
    attention_masks_tst.append(encoded_dict['attention_mask'])

# Convert the lists into tensors.
input_ids_tst = torch.cat(input_ids_tst, dim=0)
attention_masks_tst = torch.cat(attention_masks_tst, dim=0)
labels_actions_tst = torch.tensor(action_labels_encoded_tst)
labels_attributes_tst = torch.tensor(attributes_labels_tst_vect)
# Check warning:
# /usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:43: UserWarning:␣
 ↪To copy construct from a tensor, it is recommended to use sourceTensor.
 ↪clone().detach() or sourceTensor.clone().detach().requires_grad_(True),␣
 ↪rather than torch.tensor(sourceTensor).
dialog_ids_tst = torch.tensor(dialog_ids_tst)
turn_idxs_tst = torch.tensor(turn_idxs_tst)

# Print sentence 0, now as a list of IDs.
print ("Evaluation : ")
if (use_next):
  print('Original: ', transcripts_tst[0])
else:
  print('Original: ', transcripts_tst[0])
print('Token IDs:', input_ids_tst[0])
print(f"Dialog IDs: {dialog_ids_tst[0:20]}")
print(f"Turn IDXs: {turn_idxs_tst[0:20]}")
```

```
5397 records to encode.
Evaluation :
Original:  That looks a little too light for what I need, do you have something
else with a high customer rating?
Token IDs: tensor([ 101, 2008, 3504, 1037, 2210, 2205, 2422, 2005, 2054, 1045,
2342, 1010,
        2079, 2017, 2031, 2242, 2842, 2007, 1037, 2152, 8013, 5790, 1029,  102,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
```

```
            0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
            0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
            0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
            0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
            0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
            0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
            0,    0,    0,    0,    0,    0,    0,    0,    0])
Dialog IDs: tensor([2494, 2494, 2494, 2494, 2494, 3731, 3731, 3731, 3731, 3731,
8546, 8546,
        8546, 8546, 8546, 8546, 5590, 5590, 5590, 5590])
Turn IDXs: tensor([0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 5, 0, 1, 2, 3])
```

[29]:

```python
from torch.utils.data import TensorDataset, random_split
# Combine the training inputs into a TensorDataset.
#labels_tr = {'actions': labels_actions_tr, 'attributes': labels_attributes_tr}
#labels_vd = {'actions': labels_actions_vd, 'attributes': labels_attributes_vd}

train_dataset = TensorDataset(input_ids_tr, attention_masks_tr,␣
 ↪labels_actions_tr, labels_attributes_tr)
val_dataset = TensorDataset(input_ids_vd, attention_masks_vd,␣
 ↪labels_actions_vd, labels_attributes_vd, dialog_ids_vd, turn_idxs_vd)
tst_dataset = TensorDataset(input_ids_tst, attention_masks_tst,␣
 ↪labels_actions_tst, labels_attributes_tst, dialog_ids_tst, turn_idxs_tst)

print('{:>5,} training samples'.format(len(train_dataset)))
print('{:>5,} validation samples'.format(len(val_dataset)))
print('{:>5,} evaluation samples'.format(len(tst_dataset)))
```

```
21,196 training samples
3,513 validation samples
5,397 evaluation samples
```

[30]:

```python
# Check evaluation TensorDataset content
tst_dataset[0:10]
```

[30]:

```
(tensor([[ 101, 2008, 3504,  …,    0,    0,    0],
         [ 101, 2008, 3504,  …,    0,    0,    0],
         [ 101, 2040, 5617,  …,    0,    0,    0],
         …,
         [ 101, 2821, 1045,  …,    0,    0,    0],
         [ 101, 4086, 1010,  …,    0,    0,    0],
         [ 101, 1045, 2066,  …,    0,    0,    0]]),
 tensor([[1, 1, 1,  …, 0, 0, 0],
         [1, 1, 1,  …, 0, 0, 0],
         [1, 1, 1,  …, 0, 0, 0],
         …,
         [1, 1, 1,  …, 0, 0, 0],
```

```
         [1, 1, 1,  …, 0, 0, 0],
         [1, 1, 1,  …, 0, 0, 0]]),
 tensor([2, 4, 4, 0, 1, 2, 1, 2, 0, 1]),
 tensor([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
          0, 0, 0, 0, 0, 0, 0, 0, 0],
         [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
          0, 0, 0, 0, 0, 0, 0, 0, 0],
         [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
          0, 0, 0, 0, 0, 0, 0, 0, 0],
         [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
          0, 0, 0, 0, 0, 0, 0, 0, 0],
         [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
          0, 0, 0, 0, 0, 0, 0, 0, 0],
         [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
          0, 0, 0, 0, 0, 0, 0, 0, 0],
         [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
          0, 0, 0, 0, 0, 0, 0, 0, 0],
         [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
          0, 0, 0, 0, 0, 0, 0, 0, 0],
         [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
          0, 0, 0, 0, 0, 0, 0, 0, 0],
         [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
          0, 0, 0, 0, 0, 0, 0, 0, 0]]),
 tensor([2494, 2494, 2494, 2494, 2494, 3731, 3731, 3731, 3731, 3731]),
 tensor([0, 1, 2, 3, 4, 0, 1, 2, 3, 4]))
```

[31]:

```python
# Tell PyTorch to use the GPU.
device = torch.device("cuda")

print('There are %d GPU(s) available.' % torch.cuda.device_count())

print('We will use the GPU:', torch.cuda.get_device_name(0))
```

```
There are 1 GPU(s) available.
We will use the GPU: NVIDIA GeForce RTX 2060
```

[32]:
```
from torch.utils.data import DataLoader, RandomSampler, SequentialSampler

# The DataLoader needs to know our batch size for training, so we specify it
# here. For fine-tuning BERT on a specific task, the authors recommend a batch
# size of 16 or 32.
# With size 32 GeForce RTX 2060 with 6GB run out of memory
batch_size = 12

# Create the DataLoaders for our training and validation sets.
# We'll take training samples in random order.

train_dataloader = DataLoader(
            train_dataset,  # The training samples.
            sampler = RandomSampler(train_dataset), # Select batches randomly
            batch_size = batch_size # Trains with this batch size.
        )

# For validation the order doesn't matter, so we'll just read them sequentially.
validation_dataloader = DataLoader(
            val_dataset, # The validation samples.
            sampler = SequentialSampler(val_dataset), # Pull out batches␣
 ↪sequentially.
            batch_size = batch_size # Evaluate with this batch size.
        )

#ho controllato nel colab su cui ci basiamo, anche lui usa un Sequential␣
 ↪Sampler per il dataset di evaluation
evaluation_dataloader = DataLoader(
            tst_dataset, # The validation samples.
            sampler = SequentialSampler(tst_dataset), # Pull out batches␣
 ↪sequentially.
            batch_size = batch_size # Evaluate with this batch size.
        )
```

[33]:
```
#DA SISTEMARE
from transformers import BertModel
from  torch import  nn

class CustomBERTModel(nn.Module):

  def __init__(self):

    super(CustomBERTModel, self).__init__()
    self.bert = BertModel.from_pretrained("bert-base-uncased")
    ### New layers:
    self.linear_intermedio = nn.Linear(768, 256)
```

```
    #provare ad aggiungere ulteriori layer intermedi per ridurre le dimensioni␣
↪fino ad arrivare all'output richiesto
    self.linear_actions = nn.Linear(256, 5)
    self.linear_attributes = nn.Linear(256, len(mlb.classes_)) #num attributi?

  def forward(self, ids, mask):
    #controllare che l'output non rappresenti solo lo stato interno dovuto al␣
↪token CLS
    output = self.bert(ids,attention_mask=mask)
    # print(f"Type output{type(output)}")
    # for p in output:
    #   print(p)
    #   print(type(output[p]))
    #   print(output[p])

    #prendiamo il campo last_hidden_state dall'oggetto output; last hidden␣
↪state rappresenta il tensore
    #in uscita dallo step di forward del BertModel
    last_hidden_state_output = output["last_hidden_state"]
    # last_hidden_state has the following shape: (batch_size, sequence_length,␣
↪768)
    #stiamo passando solo il token CLS ai layer successivi
    linear_output_intermedio = self.linear_intermedio(last_hidden_state_output[:
↪,0,:].view(-1,768))
    # linear_output_intermedio = self.linear_intermedio(pooled_output)

    linear_output_actions = self.linear_actions(linear_output_intermedio)
    # linear_output_actions = self.sftmx(linear_output_actions)
    # linear_output_actions = nn.functional.softmax(linear_output_actions)
    # Test sigmoid for increasing perplexity performance
    linear_output_actions = torch.sigmoid(linear_output_actions)
    linear_output_attributes = self.linear_attributes(linear_output_intermedio)
    # linear_output_attributes = self.sig(linear_output_attributes)
    linear_output_attributes = torch.sigmoid(linear_output_attributes)

    return {'actions': linear_output_actions, 'attributes':␣
↪linear_output_attributes}
```

[34]:

```
#test istanziazione del custom model
model = CustomBERTModel()

# model.bert.config
model.cuda()
```

```
Some weights of the model checkpoint at bert-base-uncased were not used when
initializing BertModel: ['cls.predictions.transform.dense.weight',
'cls.seq_relationship.bias', 'cls.predictions.decoder.weight',
```

```
'cls.predictions.transform.LayerNorm.weight',
'cls.predictions.transform.dense.bias',
'cls.predictions.transform.LayerNorm.bias', 'cls.predictions.bias',
'cls.seq_relationship.weight']
- This IS expected if you are initializing BertModel from the checkpoint of a
model trained on another task or with another architecture (e.g. initializing a
BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertModel from the checkpoint of
a model that you expect to be exactly identical (initializing a
BertForSequenceClassification model from a BertForSequenceClassification model).
```

[34]:
```
CustomBERTModel(
  (bert): BertModel(
    (embeddings): BertEmbeddings(
      (word_embeddings): Embedding(30522, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (token_type_embeddings): Embedding(2, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): BertEncoder(
      (layer): ModuleList(
        (0): BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (1): BertLayer(
          (attention): BertAttention(
```

```
      (self): BertSelfAttention(
        (query): Linear(in_features=768, out_features=768, bias=True)
        (key): Linear(in_features=768, out_features=768, bias=True)
        (value): Linear(in_features=768, out_features=768, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
      (output): BertSelfOutput(
        (dense): Linear(in_features=768, out_features=768, bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
    (intermediate): BertIntermediate(
      (dense): Linear(in_features=768, out_features=3072, bias=True)
    )
    (output): BertOutput(
      (dense): Linear(in_features=3072, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (2): BertLayer(
    (attention): BertAttention(
      (self): BertSelfAttention(
        (query): Linear(in_features=768, out_features=768, bias=True)
        (key): Linear(in_features=768, out_features=768, bias=True)
        (value): Linear(in_features=768, out_features=768, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
      (output): BertSelfOutput(
        (dense): Linear(in_features=768, out_features=768, bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
    (intermediate): BertIntermediate(
      (dense): Linear(in_features=768, out_features=3072, bias=True)
    )
    (output): BertOutput(
      (dense): Linear(in_features=3072, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (3): BertLayer(
    (attention): BertAttention(
      (self): BertSelfAttention(
```

```
          (query): Linear(in_features=768, out_features=768, bias=True)
          (key): Linear(in_features=768, out_features=768, bias=True)
          (value): Linear(in_features=768, out_features=768, bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (output): BertSelfOutput(
          (dense): Linear(in_features=768, out_features=768, bias=True)
          (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
      (intermediate): BertIntermediate(
        (dense): Linear(in_features=768, out_features=3072, bias=True)
      )
      (output): BertOutput(
        (dense): Linear(in_features=3072, out_features=768, bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
    (4): BertLayer(
      (attention): BertAttention(
        (self): BertSelfAttention(
          (query): Linear(in_features=768, out_features=768, bias=True)
          (key): Linear(in_features=768, out_features=768, bias=True)
          (value): Linear(in_features=768, out_features=768, bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (output): BertSelfOutput(
          (dense): Linear(in_features=768, out_features=768, bias=True)
          (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
      (intermediate): BertIntermediate(
        (dense): Linear(in_features=768, out_features=3072, bias=True)
      )
      (output): BertOutput(
        (dense): Linear(in_features=3072, out_features=768, bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
    (5): BertLayer(
      (attention): BertAttention(
        (self): BertSelfAttention(
          (query): Linear(in_features=768, out_features=768, bias=True)
```

```
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
(6): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
(7): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
```

```
        (value): Linear(in_features=768, out_features=768, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
      (output): BertSelfOutput(
        (dense): Linear(in_features=768, out_features=768, bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
    (intermediate): BertIntermediate(
      (dense): Linear(in_features=768, out_features=3072, bias=True)
    )
    (output): BertOutput(
      (dense): Linear(in_features=3072, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (8): BertLayer(
    (attention): BertAttention(
      (self): BertSelfAttention(
        (query): Linear(in_features=768, out_features=768, bias=True)
        (key): Linear(in_features=768, out_features=768, bias=True)
        (value): Linear(in_features=768, out_features=768, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
      (output): BertSelfOutput(
        (dense): Linear(in_features=768, out_features=768, bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
    (intermediate): BertIntermediate(
      (dense): Linear(in_features=768, out_features=3072, bias=True)
    )
    (output): BertOutput(
      (dense): Linear(in_features=3072, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (9): BertLayer(
    (attention): BertAttention(
      (self): BertSelfAttention(
        (query): Linear(in_features=768, out_features=768, bias=True)
        (key): Linear(in_features=768, out_features=768, bias=True)
        (value): Linear(in_features=768, out_features=768, bias=True)
```

```
        (dropout): Dropout(p=0.1, inplace=False)
      )
      (output): BertSelfOutput(
        (dense): Linear(in_features=768, out_features=768, bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
    (intermediate): BertIntermediate(
      (dense): Linear(in_features=768, out_features=3072, bias=True)
    )
    (output): BertOutput(
      (dense): Linear(in_features=3072, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (10): BertLayer(
    (attention): BertAttention(
      (self): BertSelfAttention(
        (query): Linear(in_features=768, out_features=768, bias=True)
        (key): Linear(in_features=768, out_features=768, bias=True)
        (value): Linear(in_features=768, out_features=768, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
      (output): BertSelfOutput(
        (dense): Linear(in_features=768, out_features=768, bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
    (intermediate): BertIntermediate(
      (dense): Linear(in_features=768, out_features=3072, bias=True)
    )
    (output): BertOutput(
      (dense): Linear(in_features=3072, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (11): BertLayer(
    (attention): BertAttention(
      (self): BertSelfAttention(
        (query): Linear(in_features=768, out_features=768, bias=True)
        (key): Linear(in_features=768, out_features=768, bias=True)
        (value): Linear(in_features=768, out_features=768, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
```

```
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
    (pooler): BertPooler(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (activation): Tanh()
    )
  )
  (linear_intermedio): Linear(in_features=768, out_features=256, bias=True)
  (linear_actions): Linear(in_features=256, out_features=5, bias=True)
  (linear_attributes): Linear(in_features=256, out_features=33, bias=True)
)
```

[35]:
```
# Get all of the model's parameters as a list of tuples.
params = list(model.named_parameters())

print('The BERT model has {:} different named parameters.\n'.
 ↪format(len(params)))

print('==== Embedding Layer ====\n')

for p in params[0:5]:
    print("{:<55} {:>12}".format(p[0], str(tuple(p[1].size()))))

print('\n==== First Transformer ====\n')

for p in params[5:21]:
    print("{:<55} {:>12}".format(p[0], str(tuple(p[1].size()))))

print('\n==== Output Layer ====\n')

for p in params[-4:]:
```

```
    print("{:<55} {:>12}".format(p[0], str(tuple(p[1].size()))))
```

```
The BERT model has 205 different named parameters.

==== Embedding Layer ====

bert.embeddings.word_embeddings.weight                  (30522, 768)
bert.embeddings.position_embeddings.weight                (512, 768)
bert.embeddings.token_type_embeddings.weight                (2, 768)
bert.embeddings.LayerNorm.weight                              (768,)
bert.embeddings.LayerNorm.bias                                (768,)

==== First Transformer ====

bert.encoder.layer.0.attention.self.query.weight          (768, 768)
bert.encoder.layer.0.attention.self.query.bias                (768,)
bert.encoder.layer.0.attention.self.key.weight            (768, 768)
bert.encoder.layer.0.attention.self.key.bias                  (768,)
bert.encoder.layer.0.attention.self.value.weight          (768, 768)
bert.encoder.layer.0.attention.self.value.bias                (768,)
bert.encoder.layer.0.attention.output.dense.weight        (768, 768)
bert.encoder.layer.0.attention.output.dense.bias              (768,)
bert.encoder.layer.0.attention.output.LayerNorm.weight        (768,)
bert.encoder.layer.0.attention.output.LayerNorm.bias          (768,)
bert.encoder.layer.0.intermediate.dense.weight           (3072, 768)
bert.encoder.layer.0.intermediate.dense.bias                 (3072,)
bert.encoder.layer.0.output.dense.weight                 (768, 3072)
bert.encoder.layer.0.output.dense.bias                        (768,)
bert.encoder.layer.0.output.LayerNorm.weight                  (768,)
bert.encoder.layer.0.output.LayerNorm.bias                    (768,)

==== Output Layer ====

linear_actions.weight                                       (5, 256)
linear_actions.bias                                             (5,)
linear_attributes.weight                                   (33, 256)
linear_attributes.bias                                         (33,)
```

[36]:
```
from transformers import AdamW
# Note: AdamW is a class from the huggingface library (as opposed to pytorch)
# I believe the 'W' stands for 'Weight Decay fix"
optimizer = AdamW(model.parameters(),
                  lr = 5e-5, # args.learning_rate - default is 5e-5
                  eps = 1e-8 # args.adam_epsilon  - default is 1e-8.
                )
```

[37]:
```
from transformers import get_linear_schedule_with_warmup

# Number of training epochs. The BERT authors recommend between 2 and 4.
# We chose to run for 4, but we'll see later that this may be over-fitting the
# training data.
epochs = 4

# Total number of training steps is [number of batches] x [number of epochs].
# (Note that this is not the same as the number of training samples).
total_steps = len(train_dataloader) * epochs

# Create the learning rate scheduler.
scheduler = get_linear_schedule_with_warmup(optimizer,
                                            num_warmup_steps = 0, # Default␣
 ↪value in run_glue.py
                                            num_training_steps = total_steps)
```

[38]:
```
import numpy as np

# Function to calculate the accuracy of our predictions vs labels
def flat_accuracy_actions(preds, labels):
    #print(f"[FA] preds: {preds} / labels: {labels}")
    #print(f"[FA-Actions] {type(preds)} {type(labels)}")
    pred_flat = np.argmax(preds, axis=1).flatten()
    labels_flat = labels.flatten()
    return {'matched': np.sum(pred_flat == labels_flat), 'counts':␣
 ↪len(labels_flat)}

def flat_accuracy_attributes(preds, labels):
  #print(f"[FA-Attributess] {type(preds)} {type(labels)}")
  tot_preds = preds.shape[0]
  preds_int = np.rint(preds)
  tot_eq = 0
  for i in range(tot_preds):
    comparison = preds_int[i] == labels[i]
    if comparison.all():
      tot_eq += 1
  return {'matched': tot_eq, 'counts' : tot_preds}
```

[39]:
```
import time
import datetime

def format_time(elapsed):
    '''
    Takes a time in seconds and returns a string hh:mm:ss
    '''
    # Round to the nearest second.
```

```
    elapsed_rounded = int(round((elapsed)))

    # Format as hh:mm:ss
    return str(datetime.timedelta(seconds=elapsed_rounded))
```

[40]:
```
from torch import nn
# Loss function definition
def MyBERT_loss(logits, actions_labels, attributes_labels):
  actions_logits = logits['actions']
  attributes_logits = logits['attributes']
  loss_actions_fn = nn.CrossEntropyLoss()
  loss_attributes_fn = nn.BCELoss()
  loss_actions = loss_actions_fn(actions_logits, actions_labels)
  loss_attributes = loss_attributes_fn(attributes_logits, attributes_labels.
 ↪float())
  return loss_actions + loss_attributes
```

[41]:
```
#from GPUtil import showUtilization as gpu_usage
```

[42]:
```
import random
import numpy as np
import action_evaluation as evaluation
import json
from  GPUtil import showUtilization as gpu_usage
with open('./extr_output/fashion_dev_dials_api_calls.json') as f:
  dev_dials = json.load(f)

# This training code is based on the `run_glue.py` script here:
# https://github.com/huggingface/transformers/blob/
 ↪5bfcd0485ece086ebcbed2d008813037968a9e58/examples/run_glue.py#L128

# Set the seed value all over the place to make this reproducible.
seed_val = 24

random.seed(seed_val)
np.random.seed(seed_val)
torch.manual_seed(seed_val)
torch.cuda.manual_seed_all(seed_val)

# We'll store a number of quantities such as training and validation loss,
# validation accuracy, and timings.
training_stats = []

# Measure the total training time for the whole run.
total_t0 = time.time()

test_batch = []
```

```
# For each epoch...
for epoch_i in range(0, epochs):

    # ========================================
    #               Training
    # ========================================

    # Perform one full pass over the training set.

    print("")
    print('======== Epoch {:} / {:} ========'.format(epoch_i + 1, epochs))
    print('Training...')

    # Measure how long the training epoch takes.
    t0 = time.time()

    # Reset the total loss for this epoch.
    total_train_loss = 0

    # Put the model into training mode. Don't be mislead--the call to
    # `train` just changes the *mode*, it doesn't *perform* the training.
    # `dropout` and `batchnorm` layers behave differently during training
    # vs. test (source: https://stackoverflow.com/questions/51433378/
 ↪what-does-model-train-do-in-pytorch)
    print("GPU before train")
    gpu_usage()

    model.train()
    print("GPU after train")
    gpu_usage()

    # For each batch of training data...
    for step, batch in enumerate(train_dataloader):

        # Progress update every 40 batches.
        if step % 40 == 0 and not step == 0:
            # Calculate elapsed time in minutes.
            elapsed = format_time(time.time() - t0)

            # Report progress.
            print('  Batch {:>5,}  of  {:>5,}.    Elapsed: {:}.'.format(step,␣
 ↪len(train_dataloader), elapsed))
            #DEBUG -- da levare
            #break
            if step == 40 or step % 400 == 0:
                gpu_usage()
```

```
        # Unpack this training batch from our dataloader.
        #
        # As we unpack the batch, we'll also copy each tensor to the GPU using␣
↪the
        # `to` method.
        #
        # `batch` contains three pytorch tensors:
        #   [0]: input ids
        #   [1]: attention masks
        #   [2]: actions labels
        #   [3]: attributes labels
        b_input_ids = batch[0].to(device)
        b_input_mask = batch[1].to(device)
        b_labels_actions = batch[2].to(device)
        b_labels_attributes = batch[3].to(device)

        # Always clear any previously calculated gradients before performing a
        # backward pass. PyTorch doesn't do this automatically because
        # accumulating the gradients is "convenient while training RNNs".
        # (source: https://stackoverflow.com/questions/48001598/
↪why-do-we-need-to-call-zero-grad-in-pytorch)
        model.zero_grad()

        # Perform a forward pass (evaluate the model on this training batch).
        # In PyTorch, calling `model` will in turn call the model's `forward`
        # function and pass down the arguments. The `forward` function is
        # documented here:
        # https://huggingface.co/transformers/model_doc/bert.
↪html#bertforsequenceclassification
        # The results are returned in a results object, documented here:
        # https://huggingface.co/transformers/main_classes/output.
↪html#transformers.modeling_outputs.SequenceClassifierOutput
        # Specifically, we'll get the loss (because we provided labels) and the
        # "logits"--the model outputs prior to activation.
        result = model(b_input_ids,
                       mask=b_input_mask)

        loss = MyBERT_loss(result, b_labels_actions, b_labels_attributes)

        # Accumulate the training loss over all of the batches so that we can
        # calculate the average loss at the end. `loss` is a Tensor containing a
        # single value; the `.item()` function just returns the Python value
        # from the tensor.
        total_train_loss += loss.item()

        # Perform a backward pass to calculate the gradients.
```

```
        loss.backward()

        # Clip the norm of the gradients to 1.0.
        # This is to help prevent the "exploding gradients" problem.from␣
↪transformers import BertModel, BertConfig
        torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)

        # Update parameters and take a step using the computed gradient.
        # The optimizer dictates the "update rule"--how the parameters are
        # modified based on their gradients, the learning rate, etc.
        optimizer.step()

        # Update the learning rate.
        scheduler.step()

    print(f"End of epoch {epoch_i}")
    gpu_usage()

    # Calculate the average loss over all of the batches.
    avg_train_loss = total_train_loss / len(train_dataloader)

    # Measure how long this epoch took.
    training_time = format_time(time.time() - t0)

    print("")
    print("  Average training loss: {0:.2f}".format(avg_train_loss))
    print("  Training epcoh took: {:}".format(training_time))

    # ========================================
    #               Validation
    # ========================================
    # After the completion of each training epoch, measure our performance on
    # our validation set.

    print("")
    print("Running Validation...")

    t0 = time.time()

    # Put the model in evaluation mode--the dropout layers behave differently
    # during evaluation.mlb.inverse_transform(attr_yt[3].reshape(1, -1))
    model.eval()

    # Tracking variables
    total_eval_accuracy_classification = { 'matched': 0, 'counts': 0}
    total_eval_accuracy_multilabel = { 'matched': 0, 'counts': 0}
    total_eval_loss = 0
```

```python
nb_eval_steps = 0

batch_number = 0

# Dictionary for action_evaluation
model_actions = {}

# Evaluate data for one epoch
for batch in validation_dataloader:

    batch_number += 1

    # Unpack this training batch from our dataloader.
    #
    # As we unpack the batch, we'll also copy each tensor to the GPU using
    # the `to` method.
    #
    # `batch` contains three pytorch tensors:
    #   [0]: input ids
    #   [1]: attention masks
    #   [2]: labels
    b_input_ids = batch[0].to(device)
    b_input_mask = batch[1].to(device)
    b_labels_actions = batch[2].to(device)
    b_labels_attributes = batch[3].to(device)
    b_dialog_ids = batch[4].to(device).detach().cpu().numpy()
    b_turn_idxs = batch[5].to(device).detach().cpu().numpy()

    # Tell pytorch not to bother with constructing the compute graph during
    # the forward pass, since this is only needed for backprop (training).
    with torch.no_grad():

        # Forward pass, calculate logit predictions.
        # token_type_ids is the same as the "segment ids", which
        # differentiates sentence 1 and 2 in 2-sentence tasks.
        result = model(b_input_ids,
                       mask=b_input_mask)

    # Get the loss and "logits" output by the model. The "logits" are the
    # output values prior to applying an activation function like the
    # softmax.
    loss = MyBERT_loss(result, b_labels_actions, b_labels_attributes)

    # Accumulate the validation loss.
    total_eval_loss += loss.item()

    # Move logits and labels to CPU
```

```
        # logits = logits.detach().cpu().numpy()
        # label_ids = b_labels.to('cpu').numpy()


        actions_logits_foracc=result['actions'].detach().cpu().numpy()
        attributes_logits_foracc=result['attributes'].detach().cpu().numpy()
        actions_labels_foracc= b_labels_actions.to('cpu').numpy()
        attributes_labels_foracc =b_labels_attributes.to('cpu').numpy()

        #TODO: definire la nostra funzione di accuracy

        # Calculate the accuracy for this batch of test sentences, and
        # accumulate it over all batches.
        accuracy_classification = flat_accuracy_actions(actions_logits_foracc,␣
↪actions_labels_foracc)
        accuracy_multilabel =␣
↪flat_accuracy_attributes(attributes_logits_foracc, attributes_labels_foracc)

        total_eval_accuracy_classification['matched'] +=␣
↪accuracy_classification['matched']
        total_eval_accuracy_classification['counts'] +=␣
↪accuracy_classification['counts']
        total_eval_accuracy_multilabel['matched'] +=␣
↪accuracy_multilabel['matched']
        total_eval_accuracy_multilabel['counts'] +=␣
↪accuracy_multilabel['counts']
        # Salvo dati elaborazione batch per debug/analisi
        test_batch.append({
            'ephoc' : epoch_i + 1,
            'batchnum' : batch_number,
            'actions_logits' : actions_logits_foracc,
            'actions_labels' : actions_labels_foracc,
            'attributes_logits' : attributes_logits_foracc,
            'attributes_labels' : attributes_labels_foracc,
            'accuracy_classification' : accuracy_classification,
            'accuracy_multilabel' : accuracy_multilabel,
        })

        # Fill dictionary for action_evaluation
        for el_i in range(len(actions_logits_foracc)):
          dialog_id = b_dialog_ids[el_i]
          action_log_prob = {}
          for act_i in range(len(actions_logits_foracc[el_i])):
            #todo: controllare che la probabilità predetta sia in scala␣
↪logaritmica (?? potrebbe essere fonte di errori)
```

```
            action_log_prob[le.classes_[act_i]] = np.
↪log(actions_logits_foracc[el_i][act_i])
          #attributes = {}
          attributes = []
          #attributes_list = np.rint(attributes_logits_foracc[el_i])
          attributes_list = np.array(attributes_logits_foracc[el_i])
          for attr in range(len(attributes_list)):
            attribute = mlb.classes_[attr]
            #attributes[mlb.classes_[attr]] = attributes_list[attr]
            if attributes_list[attr] >= 0.5:
              attributes.append(attribute)
          prediction = {
              'action': le.classes_[np.argmax(actions_logits_foracc[el_i])],
              'action_log_prob': action_log_prob,
              'attributes': {'attributes': attributes},
              'turn_id': b_turn_idxs[el_i]
          }
          if dialog_id in model_actions:
            model_actions[dialog_id]['predictions'].append(prediction)
          else:
            predictions = list()
            predictions.append(prediction)
            model_actions[dialog_id] = {
                'dialog_id': dialog_id,
                'predictions': predictions
            }


    # Report the final accuracy for this validation

    #avg_val_accuracy_classification = total_eval_accuracy_classification /␣
↪len(validation_dataloader)
    #avg_val_accuracy_multilabel = total_eval_accuracy_multilabel /␣
↪len(validation_dataloader)
    avg_val_accuracy_classification =␣
↪total_eval_accuracy_classification['matched'] /␣
↪total_eval_accuracy_classification['counts']
    avg_val_accuracy_multilabel = total_eval_accuracy_multilabel['matched'] /␣
↪total_eval_accuracy_multilabel['counts']
    print("  Accuracy for classification (actions): {0:.4f}".
↪format(avg_val_accuracy_classification))
    print("  Accuracy for multilabel-classification (attributes): {0:.4f}".
↪format(avg_val_accuracy_multilabel))

    # Reference implementation: evaluation of action prediction along with␣
↪attributes
```

```python
    metrics = evaluation.evaluate_action_prediction(dev_dials, model_actions.
 ↪values())
    # print("model_actions passed to the evaluator:")
    # for v in model_actions.values():
    #   print(v)
    print("***************************************")
    print("Reference evaluation metrics:")
    print(metrics)

    # Calculate the average loss over all of the batches.
    avg_val_loss = total_eval_loss / len(validation_dataloader)

    # Measure how long the validation run took.
    validation_time = format_time(time.time() - t0)

    print("  Validation Loss: {0:.4f}".format(avg_val_loss))
    print("  Validation took: {:}".format(validation_time))

    # Record all statistics from this epoch.
    training_stats.append(
        {
            'epoch': epoch_i + 1,
            'Training Loss': avg_train_loss,
            'Valid. Loss': avg_val_loss,
            'Valid. Accur. class.': avg_val_accuracy_classification,
            'Valid. Accur. mult.label': avg_val_accuracy_multilabel,
            'Training Time': training_time,
            'Validation Time': validation_time
        }
    )

print("")
print("Training complete!")

print("Total training took {:} (h:mm:ss)".format(format_time(time.
 ↪time()-total_t0)))
```

```
======== Epoch 1 / 4 ========
Training…
GPU before train
| ID | GPU | MEM |
------------------
|  0 |  4% | 29% |
GPU after train
| ID | GPU | MEM |
------------------
```

```
|  0 |  4% | 29% |
  Batch    40  of  1,767.    Elapsed: 0:00:12.
| ID | GPU | MEM |
------------------
|  0 | 99% | 88% |
  Batch    80  of  1,767.    Elapsed: 0:00:23.
  Batch   120  of  1,767.    Elapsed: 0:00:34.
  Batch   160  of  1,767.    Elapsed: 0:00:46.
  Batch   200  of  1,767.    Elapsed: 0:00:57.
  Batch   240  of  1,767.    Elapsed: 0:01:08.
  Batch   280  of  1,767.    Elapsed: 0:01:20.
  Batch   320  of  1,767.    Elapsed: 0:01:31.
  Batch   360  of  1,767.    Elapsed: 0:01:42.
  Batch   400  of  1,767.    Elapsed: 0:01:54.
| ID | GPU | MEM |
------------------
|  0 | 99% | 87% |
  Batch   440  of  1,767.    Elapsed: 0:02:05.
  Batch   480  of  1,767.    Elapsed: 0:02:17.
  Batch   520  of  1,767.    Elapsed: 0:02:29.
  Batch   560  of  1,767.    Elapsed: 0:02:41.
  Batch   600  of  1,767.    Elapsed: 0:02:53.
  Batch   640  of  1,767.    Elapsed: 0:03:05.
  Batch   680  of  1,767.    Elapsed: 0:03:16.
  Batch   720  of  1,767.    Elapsed: 0:03:28.
  Batch   760  of  1,767.    Elapsed: 0:03:40.
  Batch   800  of  1,767.    Elapsed: 0:03:52.
| ID | GPU | MEM |
------------------
|  0 | 99% | 89% |
  Batch   840  of  1,767.    Elapsed: 0:04:04.
  Batch   880  of  1,767.    Elapsed: 0:04:16.
  Batch   920  of  1,767.    Elapsed: 0:04:28.
  Batch   960  of  1,767.    Elapsed: 0:04:40.
  Batch 1,000  of  1,767.    Elapsed: 0:04:52.
  Batch 1,040  of  1,767.    Elapsed: 0:05:04.
  Batch 1,080  of  1,767.    Elapsed: 0:05:16.
  Batch 1,120  of  1,767.    Elapsed: 0:05:28.
  Batch 1,160  of  1,767.    Elapsed: 0:05:40.
  Batch 1,200  of  1,767.    Elapsed: 0:05:52.
| ID | GPU | MEM |
------------------
|  0 | 99% | 90% |
  Batch 1,240  of  1,767.    Elapsed: 0:06:04.
  Batch 1,280  of  1,767.    Elapsed: 0:06:15.
  Batch 1,320  of  1,767.    Elapsed: 0:06:27.
  Batch 1,360  of  1,767.    Elapsed: 0:06:38.
  Batch 1,400  of  1,767.    Elapsed: 0:06:50.
```

```
  Batch 1,440  of  1,767.    Elapsed: 0:07:02.
  Batch 1,480  of  1,767.    Elapsed: 0:07:13.
  Batch 1,520  of  1,767.    Elapsed: 0:07:25.
  Batch 1,560  of  1,767.    Elapsed: 0:07:37.
  Batch 1,600  of  1,767.    Elapsed: 0:07:48.
| ID | GPU | MEM |
------------------
|  0 | 99% | 88% |
  Batch 1,640  of  1,767.    Elapsed: 0:08:00.
  Batch 1,680  of  1,767.    Elapsed: 0:08:13.
  Batch 1,720  of  1,767.    Elapsed: 0:08:24.
  Batch 1,760  of  1,767.    Elapsed: 0:08:36.
End of epoch 0
| ID | GPU | MEM |
------------------
|  0 | 99% | 90% |

  Average training loss: 1.13
  Training epcoh took: 0:08:38

Running Validation…
  Accuracy for classification (actions): 0.8451
  Accuracy for multilabel-classification (attributes): 0.8785
#Instances evaluated API: 3513
***************************************
Reference evaluation metrics:
{'action_accuracy': 0.8451465983489894, 'action_perplexity': 2.061376839726222,
'attribute_accuracy': 0.6787099391419993, 'confusion_matrix': array([[4.720e+02,
4.600e+01, 7.000e+00, 1.000e+00, 1.600e+01],
       [1.000e+01, 6.530e+02, 2.600e+01, 4.000e+00, 9.000e+00],
       [1.000e+01, 1.500e+02, 5.090e+02, 4.700e+01, 1.500e+01],
       [0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00],
       [0.000e+00, 5.900e+01, 8.200e+01, 6.200e+01, 1.335e+03]])}
  Validation Loss: 1.0664
  Validation took: 0:00:28

======== Epoch 2 / 4 ========
Training…
GPU before train
| ID | GPU | MEM |
------------------
|  0 | 98% | 89% |
GPU after train
| ID | GPU | MEM |
------------------
|  0 | 98% | 89% |
  Batch    40  of  1,767.    Elapsed: 0:00:12.
| ID | GPU  | MEM |
```

```
-------------------
|  0 | 100% | 90% |
  Batch    80  of  1,767.    Elapsed: 0:00:24.
  Batch   120  of  1,767.    Elapsed: 0:00:35.
  Batch   160  of  1,767.    Elapsed: 0:00:47.
  Batch   200  of  1,767.    Elapsed: 0:00:59.
  Batch   240  of  1,767.    Elapsed: 0:01:11.
  Batch   280  of  1,767.    Elapsed: 0:01:23.
  Batch   320  of  1,767.    Elapsed: 0:01:35.
  Batch   360  of  1,767.    Elapsed: 0:01:47.
  Batch   400  of  1,767.    Elapsed: 0:01:59.
| ID | GPU | MEM |
------------------
|  0 | 99% | 91% |
  Batch   440  of  1,767.    Elapsed: 0:02:11.
  Batch   480  of  1,767.    Elapsed: 0:02:22.
  Batch   520  of  1,767.    Elapsed: 0:02:34.
  Batch   560  of  1,767.    Elapsed: 0:02:46.
  Batch   600  of  1,767.    Elapsed: 0:02:58.
  Batch   640  of  1,767.    Elapsed: 0:03:09.
  Batch   680  of  1,767.    Elapsed: 0:03:21.
  Batch   720  of  1,767.    Elapsed: 0:03:32.
  Batch   760  of  1,767.    Elapsed: 0:03:44.
  Batch   800  of  1,767.    Elapsed: 0:03:55.
| ID | GPU | MEM |
------------------
|  0 | 95% | 89% |
  Batch   840  of  1,767.    Elapsed: 0:04:07.
  Batch   880  of  1,767.    Elapsed: 0:04:18.
  Batch   920  of  1,767.    Elapsed: 0:04:30.
  Batch   960  of  1,767.    Elapsed: 0:04:42.
  Batch 1,000  of  1,767.    Elapsed: 0:04:53.
  Batch 1,040  of  1,767.    Elapsed: 0:05:05.
  Batch 1,080  of  1,767.    Elapsed: 0:05:16.
  Batch 1,120  of  1,767.    Elapsed: 0:05:28.
  Batch 1,160  of  1,767.    Elapsed: 0:05:39.
  Batch 1,200  of  1,767.    Elapsed: 0:05:51.
| ID | GPU | MEM |
------------------
|  0 | 99% | 89% |
  Batch 1,240  of  1,767.    Elapsed: 0:06:03.
  Batch 1,280  of  1,767.    Elapsed: 0:06:15.
  Batch 1,320  of  1,767.    Elapsed: 0:06:26.
  Batch 1,360  of  1,767.    Elapsed: 0:06:38.
  Batch 1,400  of  1,767.    Elapsed: 0:06:49.
  Batch 1,440  of  1,767.    Elapsed: 0:07:02.
  Batch 1,480  of  1,767.    Elapsed: 0:07:14.
  Batch 1,520  of  1,767.    Elapsed: 0:07:26.
```

```
  Batch 1,560  of  1,767.    Elapsed: 0:07:37.
  Batch 1,600  of  1,767.    Elapsed: 0:07:49.
| ID | GPU  | MEM |
-------------------
|  0 | 100% | 90% |
  Batch 1,640  of  1,767.    Elapsed: 0:08:01.
  Batch 1,680  of  1,767.    Elapsed: 0:08:12.
  Batch 1,720  of  1,767.    Elapsed: 0:08:25.
  Batch 1,760  of  1,767.    Elapsed: 0:08:36.
End of epoch 1
| ID | GPU | MEM |
------------------
|  0 | 99% | 89% |

  Average training loss: 1.06
  Training epcoh took: 0:08:38

Running Validation…
  Accuracy for classification (actions): 0.8488
  Accuracy for multilabel-classification (attributes): 0.9209
#Instances evaluated API: 3513
***************************************
Reference evaluation metrics:
{'action_accuracy': 0.8488471391972673, 'action_perplexity': 2.3182883517974853,
'attribute_accuracy': 0.7356263777185521, 'confusion_matrix': array([[4.680e+02,
3.600e+01, 4.000e+00, 0.000e+00, 7.000e+00],
       [1.400e+01, 6.790e+02, 3.700e+01, 1.000e+01, 1.000e+01],
       [9.000e+00, 1.370e+02, 4.840e+02, 4.100e+01, 7.000e+00],
       [0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00],
       [1.000e+00, 5.600e+01, 9.900e+01, 6.300e+01, 1.351e+03]])}
  Validation Loss: 1.0584
  Validation took: 0:00:27

======== Epoch 3 / 4 ========
Training…
GPU before train
| ID | GPU | MEM |
------------------
|  0 | 97% | 88% |
GPU after train
| ID | GPU | MEM |
------------------
|  0 | 97% | 88% |
  Batch    40  of  1,767.    Elapsed: 0:00:12.
| ID | GPU | MEM |
------------------
|  0 | 99% | 88% |
  Batch    80  of  1,767.    Elapsed: 0:00:24.
```

```
  Batch   120  of  1,767.    Elapsed: 0:00:35.
  Batch   160  of  1,767.    Elapsed: 0:00:47.
  Batch   200  of  1,767.    Elapsed: 0:00:59.
  Batch   240  of  1,767.    Elapsed: 0:01:10.
  Batch   280  of  1,767.    Elapsed: 0:01:22.
  Batch   320  of  1,767.    Elapsed: 0:01:33.
  Batch   360  of  1,767.    Elapsed: 0:01:45.
  Batch   400  of  1,767.    Elapsed: 0:01:56.
| ID | GPU | MEM |
------------------
|  0 | 99% | 88% |
  Batch   440  of  1,767.    Elapsed: 0:02:08.
  Batch   480  of  1,767.    Elapsed: 0:02:20.
  Batch   520  of  1,767.    Elapsed: 0:02:31.
  Batch   560  of  1,767.    Elapsed: 0:02:43.
  Batch   600  of  1,767.    Elapsed: 0:02:54.
  Batch   640  of  1,767.    Elapsed: 0:03:06.
  Batch   680  of  1,767.    Elapsed: 0:03:17.
  Batch   720  of  1,767.    Elapsed: 0:03:29.
  Batch   760  of  1,767.    Elapsed: 0:03:40.
  Batch   800  of  1,767.    Elapsed: 0:03:52.
| ID | GPU  | MEM |
-------------------
|  0 | 100% | 89% |
  Batch   840  of  1,767.    Elapsed: 0:04:04.
  Batch   880  of  1,767.    Elapsed: 0:04:16.
  Batch   920  of  1,767.    Elapsed: 0:04:28.
  Batch   960  of  1,767.    Elapsed: 0:04:40.
  Batch 1,000  of  1,767.    Elapsed: 0:04:52.
  Batch 1,040  of  1,767.    Elapsed: 0:05:04.
  Batch 1,080  of  1,767.    Elapsed: 0:05:15.
  Batch 1,120  of  1,767.    Elapsed: 0:05:27.
  Batch 1,160  of  1,767.    Elapsed: 0:05:38.
  Batch 1,200  of  1,767.    Elapsed: 0:05:50.
| ID | GPU | MEM |
------------------
|  0 | 99% | 89% |
  Batch 1,240  of  1,767.    Elapsed: 0:06:02.
  Batch 1,280  of  1,767.    Elapsed: 0:06:14.
  Batch 1,320  of  1,767.    Elapsed: 0:06:25.
  Batch 1,360  of  1,767.    Elapsed: 0:06:37.
  Batch 1,400  of  1,767.    Elapsed: 0:06:49.
  Batch 1,440  of  1,767.    Elapsed: 0:07:00.
  Batch 1,480  of  1,767.    Elapsed: 0:07:12.
  Batch 1,520  of  1,767.    Elapsed: 0:07:24.
  Batch 1,560  of  1,767.    Elapsed: 0:07:36.
  Batch 1,600  of  1,767.    Elapsed: 0:07:48.
| ID | GPU | MEM |
```

```
------------------
|  0 | 99% | 89% |
  Batch 1,640  of  1,767.    Elapsed: 0:07:59.
  Batch 1,680  of  1,767.    Elapsed: 0:08:11.
  Batch 1,720  of  1,767.    Elapsed: 0:08:23.
  Batch 1,760  of  1,767.    Elapsed: 0:08:35.
End of epoch 2
| ID | GPU | MEM |
------------------
|  0 | 99% | 89% |

  Average training loss: 1.04
  Training epcoh took: 0:08:37

Running Validation…
  Accuracy for classification (actions): 0.8514
  Accuracy for multilabel-classification (attributes): 0.9243
#Instances evaluated API: 3513
***************************************
Reference evaluation metrics:
{'action_accuracy': 0.8514090520922288, 'action_perplexity': 2.4830429491963275,
'attribute_accuracy': 0.7453504514504714, 'confusion_matrix': array([[ 468.,
36.,    6.,    0.,    4.],
       [  10.,  695.,   70.,   14.,   16.],
       [   8.,  125.,  483.,   45.,   10.],
       [   0.,    0.,    0.,    0.,    0.],
       [   6.,   52.,   65.,   55., 1345.]])}
  Validation Loss: 1.0541
  Validation took: 0:00:27

======== Epoch 4 / 4 ========
Training…
GPU before train
| ID | GPU | MEM |
------------------
|  0 | 98% | 89% |
GPU after train
| ID | GPU | MEM |
------------------
|  0 | 98% | 89% |
  Batch    40  of  1,767.    Elapsed: 0:00:12.
| ID | GPU | MEM |
------------------
|  0 | 99% | 89% |
  Batch    80  of  1,767.    Elapsed: 0:00:24.
  Batch   120  of  1,767.    Elapsed: 0:00:36.
  Batch   160  of  1,767.    Elapsed: 0:00:47.
  Batch   200  of  1,767.    Elapsed: 0:00:59.
```

```
  Batch   240  of  1,767.    Elapsed: 0:01:11.
  Batch   280  of  1,767.    Elapsed: 0:01:23.
  Batch   320  of  1,767.    Elapsed: 0:01:35.
  Batch   360  of  1,767.    Elapsed: 0:01:46.
  Batch   400  of  1,767.    Elapsed: 0:01:58.
| ID | GPU | MEM |
------------------
|  0 | 99% | 89% |
  Batch   440  of  1,767.    Elapsed: 0:02:10.
  Batch   480  of  1,767.    Elapsed: 0:02:22.
  Batch   520  of  1,767.    Elapsed: 0:02:34.
  Batch   560  of  1,767.    Elapsed: 0:02:46.
  Batch   600  of  1,767.    Elapsed: 0:02:57.
  Batch   640  of  1,767.    Elapsed: 0:03:09.
  Batch   680  of  1,767.    Elapsed: 0:03:21.
  Batch   720  of  1,767.    Elapsed: 0:03:33.
  Batch   760  of  1,767.    Elapsed: 0:03:44.
  Batch   800  of  1,767.    Elapsed: 0:03:56.
| ID | GPU | MEM |
------------------
|  0 | 99% | 89% |
  Batch   840  of  1,767.    Elapsed: 0:04:08.
  Batch   880  of  1,767.    Elapsed: 0:04:20.
  Batch   920  of  1,767.    Elapsed: 0:04:31.
  Batch   960  of  1,767.    Elapsed: 0:04:43.
  Batch 1,000  of  1,767.    Elapsed: 0:04:55.
  Batch 1,040  of  1,767.    Elapsed: 0:05:06.
  Batch 1,080  of  1,767.    Elapsed: 0:05:18.
  Batch 1,120  of  1,767.    Elapsed: 0:05:29.
  Batch 1,160  of  1,767.    Elapsed: 0:05:41.
  Batch 1,200  of  1,767.    Elapsed: 0:05:53.
| ID | GPU | MEM |
------------------
|  0 | 99% | 89% |
  Batch 1,240  of  1,767.    Elapsed: 0:06:04.
  Batch 1,280  of  1,767.    Elapsed: 0:06:16.
  Batch 1,320  of  1,767.    Elapsed: 0:06:28.
  Batch 1,360  of  1,767.    Elapsed: 0:06:39.
  Batch 1,400  of  1,767.    Elapsed: 0:06:51.
  Batch 1,440  of  1,767.    Elapsed: 0:07:03.
  Batch 1,480  of  1,767.    Elapsed: 0:07:15.
  Batch 1,520  of  1,767.    Elapsed: 0:07:26.
  Batch 1,560  of  1,767.    Elapsed: 0:07:38.
  Batch 1,600  of  1,767.    Elapsed: 0:07:50.
| ID | GPU | MEM |
------------------
|  0 | 99% | 89% |
  Batch 1,640  of  1,767.    Elapsed: 0:08:02.
```

```
  Batch 1,680  of  1,767.    Elapsed: 0:08:14.
  Batch 1,720  of  1,767.    Elapsed: 0:08:25.
  Batch 1,760  of  1,767.    Elapsed: 0:08:37.
End of epoch 3
| ID | GPU | MEM |
------------------
|  0 | 99% | 89% |

  Average training loss: 1.03
  Training epcoh took: 0:08:39

Running Validation…
  Accuracy for classification (actions): 0.8565
  Accuracy for multilabel-classification (attributes): 0.9280
#Instances evaluated API: 3513
***************************************
Reference evaluation metrics:
{'action_accuracy': 0.856532877882152, 'action_perplexity': 2.810755603660233,
'attribute_accuracy': 0.7616486252087795, 'confusion_matrix': array([[ 469.,
33.,    4.,    0.,    4.],
       [   8.,  696.,   54.,   10.,   15.],
       [  10.,  133.,  502.,   48.,   14.],
       [   0.,    0.,    0.,    0.,    0.],
       [   5.,   46.,   64.,   56., 1342.]])}
  Validation Loss: 1.0542
  Validation took: 0:00:27

Training complete!
Total training took 0:36:21 (h:mm:ss)
```

[43]:

```python
#Prediction on test set
#quale modello gli viene passato? da controllare se BERT da solo riesce a␣
↪tenere traccia del modello che ha dato l'epoca migliore



with open('./extr_output/fashion_devtest_dials_api_calls.json') as f:
  devtest_dials = json.load(f)

# Tracking variables
total_eval_accuracy_classification = { 'matched': 0, 'counts': 0}
total_eval_accuracy_multilabel = { 'matched': 0, 'counts': 0}

model_actions = {}
# Put model in evaluation mode
model.eval()

for batch in evaluation_dataloader:
```

```
    # Unpack this training batch from our dataloader.
    #
    # As we unpack the batch, we'll also copy each tensor to the GPU using
    # the `to` method.
    #
    # `batch` contains three pytorch tensors:
    #   [0]: input ids
    #   [1]: attention masks
    #   [2]: labels
    b_input_ids = batch[0].to(device)
    b_input_mask = batch[1].to(device)
    b_labels_actions = batch[2].to(device)
    b_labels_attributes = batch[3].to(device)
    b_dialog_ids = batch[4].to(device).detach().cpu().numpy()
    b_turn_idxs = batch[5].to(device).detach().cpu().numpy()

    # Tell pytorch not to bother with constructing the compute graph during
    # the forward pass, since this is only needed for backprop (training).
    with torch.no_grad():
        # Forward pass, calculate logit predictions.
        # token_type_ids is the same as the "segment ids", which
        # differentiates sentence 1 and 2 in 2-sentence tasks.
        result = model(b_input_ids,mask=b_input_mask)


    actions_logits_foracc=result['actions'].detach().cpu().numpy()
    attributes_logits_foracc=result['attributes'].detach().cpu().numpy()
    actions_labels_foracc= b_labels_actions.to('cpu').numpy()
    attributes_labels_foracc =b_labels_attributes.to('cpu').numpy()

    # Calculate the accuracy for this batch of test sentences, and
    # accumulate it over all batches.
    accuracy_classification = flat_accuracy_actions(actions_logits_foracc,␣
↪actions_labels_foracc)
    accuracy_multilabel = flat_accuracy_attributes(attributes_logits_foracc,␣
↪attributes_labels_foracc)

    total_eval_accuracy_classification['matched'] +=␣
↪accuracy_classification['matched']
    total_eval_accuracy_classification['counts'] +=␣
↪accuracy_classification['counts']
    total_eval_accuracy_multilabel['matched'] += accuracy_multilabel['matched']
    total_eval_accuracy_multilabel['counts'] += accuracy_multilabel['counts']


    # Fill dictionary for action_evaluation
```

```
    for el_i in range(len(actions_logits_foracc)):
      dialog_id = b_dialog_ids[el_i]
      action_log_prob = {}
      for act_i in range(len(actions_logits_foracc[el_i])):
        #todo: controllare che la probabilità predetta sia in scala logaritmica␣
↪(?? potrebbe essere fonte di errori)
        action_log_prob[le.classes_[act_i]] = np.
↪log(actions_logits_foracc[el_i][act_i])
      #attributes = {}
      attributes = []
      #attributes_list = np.rint(attributes_logits_foracc[el_i])
      attributes_list = np.array(attributes_logits_foracc[el_i])
      for attr in range(len(attributes_list)):
        attribute = mlb.classes_[attr]
        #attributes[mlb.classes_[attr]] = attributes_list[attr]
        if attributes_list[attr] >= 0.5:
          attributes.append(attribute)
      prediction = {
          'action': le.classes_[np.argmax(actions_logits_foracc[el_i])],
          'action_log_prob': action_log_prob,
          'attributes': {'attributes': attributes},
          'turn_id': b_turn_idxs[el_i]
      }
      if dialog_id in model_actions:
        model_actions[dialog_id]['predictions'].append(prediction)
      else:
        predictions = list()
        predictions.append(prediction)
        model_actions[dialog_id] = {
            'dialog_id': dialog_id,
            'predictions': predictions
        }


# Report the final accuracy for this validation

#avg_val_accuracy_classification = total_eval_accuracy_classification /␣
↪len(validation_dataloader)
#avg_val_accuracy_multilabel = total_eval_accuracy_multilabel /␣
↪len(validation_dataloader)
avg_val_accuracy_classification = total_eval_accuracy_classification['matched']␣
↪/ total_eval_accuracy_classification['counts']
avg_val_accuracy_multilabel = total_eval_accuracy_multilabel['matched'] /␣
↪total_eval_accuracy_multilabel['counts']
print("  Accuracy for classification (actions): {0:.4f}".
↪format(avg_val_accuracy_classification))
```

```
print("  Accuracy for multilabel-classification (attributes): {0:.4f}".
 ↪format(avg_val_accuracy_multilabel))

# Reference implementation: evaluation of action prediction along with␣
 ↪attributes
metrics = evaluation.evaluate_action_prediction(devtest_dials, model_actions.
 ↪values())
# print("model_actions passed to the evaluator:")
# for v in model_actions.values():
#   print(v)
print("*************************************")
print("Reference evaluation metrics:")
print(metrics)
```

```
  Accuracy for classification (actions): 0.8425
  Accuracy for multilabel-classification (attributes): 0.9200
#Instances evaluated API: 5397
***************************************
Reference evaluation metrics:
{'action_accuracy': 0.8425050954233834, 'action_perplexity': 3.1634650211610884,
'attribute_accuracy': 0.7370522617939476, 'confusion_matrix': array([[ 747.,
53.,    8.,    4.,   13.],
       [  26., 1058.,   91.,   15.,   32.],
       [  12.,  179.,  727.,   88.,   28.],
       [   0.,    0.,    0.,    0.,    0.],
       [   8.,   96.,  118.,   79., 2015.]])}
```

[44]:
```
import pandas as pd
```

[45]:
```
# Convert test data to dataframe
df_test = pd.DataFrame(data = test_batch)
df_test.head()
```

[45]:
```
   ephoc  batchnum                                      actions_logits  \
0      1         1  [[0.0004531708, 0.99980944, 0.0025453316, 0.00…
1      1         2  [[0.00021071802, 0.9996798, 0.0008664634, 0.00…
2      1         3  [[0.0007729576, 0.0009535144, 0.0010184883, 0…
3      1         4  [[0.99967086, 0.0019957258, 0.0006378222, 0.00…
4      1         5  [[7.279184e-05, 0.00031680457, 0.00024304114, …

                          actions_labels  \
0  [1, 4, 2, 4, 0, 1, 1, 4, 4, 0, 1, 2]
1  [1, 4, 2, 2, 0, 1, 2, 1, 1, 1, 1, 4]
2  [4, 1, 1, 1, 4, 2, 1, 2, 0, 1, 1, 1]
3  [0, 0, 1, 2, 0, 1, 4, 4, 0, 1, 1, 4]
4  [1, 4, 2, 4, 0, 3, 1, 1, 2, 4, 1, 1]
```

```
                                   attributes_logits  \
0  [[0.00041479623, 0.0002373658, 0.001900391, 0…
1  [[0.00058873027, 0.00036660058, 0.0032797686, …
2  [[0.00016869778, 0.00028577558, 0.90405023, 0…
3  [[0.0009995383, 0.0007337396, 0.0064695496, 0…
4  [[0.00013425016, 0.00026139984, 0.040037967, 0…

                                   attributes_labels  \
0  [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,…
1  [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,…
2  [[0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,…
3  [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,…
4  [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,…

         accuracy_classification            accuracy_multilabel
0  {'matched': 10, 'counts': 12}  {'matched': 11, 'counts': 12}
1   {'matched': 9, 'counts': 12}  {'matched': 11, 'counts': 12}
2   {'matched': 8, 'counts': 12}  {'matched': 11, 'counts': 12}
3   {'matched': 9, 'counts': 12}  {'matched': 11, 'counts': 12}
4   {'matched': 9, 'counts': 12}  {'matched': 11, 'counts': 12}
```

[46]:
```python
# Display floats with two decimal places.
pd.set_option('precision', 2)

# Create a DataFrame from our training statistics.
df_stats = pd.DataFrame(data=training_stats)

# Use the 'epoch' as the row index.
df_stats = df_stats.set_index('epoch')

# A hack to force the column headers to wrap.
#df = df.style.set_table_styles([dict(selector="th",props=[('max-width',␣
↪'70px')])])

# Display the table.
df_stats
```

[46]:
```
       Training Loss  Valid. Loss  Valid. Accur. class.  \
epoch
1               1.13         1.07                  0.85
2               1.06         1.06                  0.85
3               1.04         1.05                  0.85
4               1.03         1.05                  0.86

       Valid. Accur. mult.label Training Time Validation Time
epoch
1                          0.88       0:08:38         0:00:28
```

```
2                          0.92       0:08:38          0:00:27
3                          0.92       0:08:37          0:00:27
4                          0.93       0:08:39          0:00:27
```

[47]:
```python
import time

# Objects serialization
timestr = time.strftime("%Y%m%d-%H%M%S")
testdata_filename = f"testdata-{timestr}"
stats_filename = f"stats-{timestr}"
#outtest = open(testdata_filename, "wb")
#outstats = open(stats_filename, "wb")
#pk.dump(obj=df_test, file=outtest)
#outtest.close()
#pk.dump(obj=df_stats, file=outstats)
#outstats.close()
df_test.to_pickle(testdata_filename)
df_stats.to_pickle(stats_filename)
```

[48]:
```python
import pandas as pd

# Test reimport data
df_stats_reload = pd.read_pickle(stats_filename)
df_test_reload = pd.read_pickle(testdata_filename)

print(testdata_filename)
print(df_stats_reload.head())
print(df_test_reload.head())
```

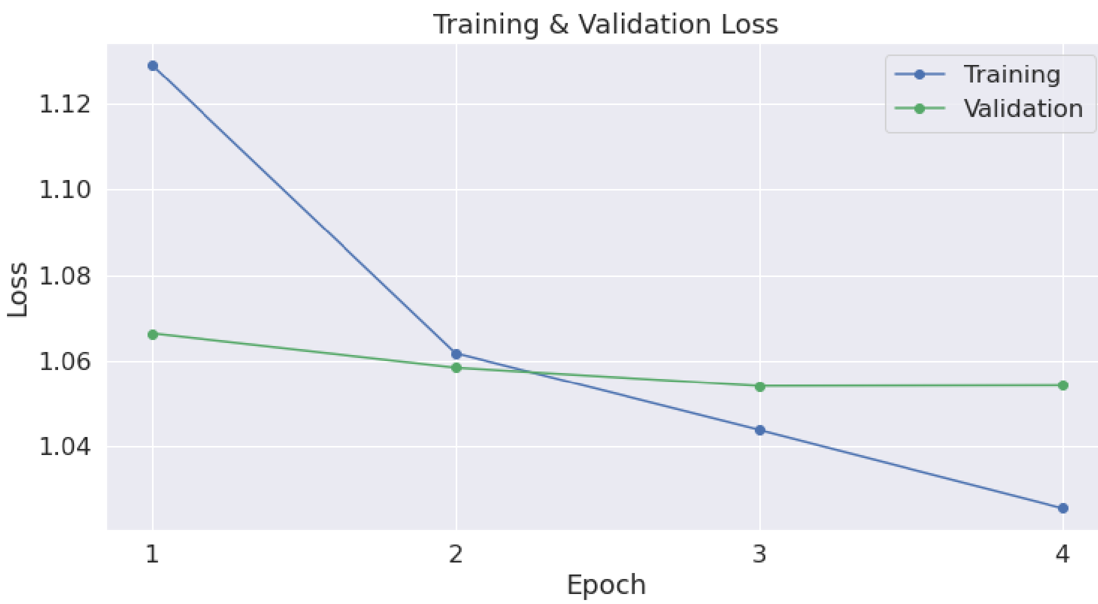
```
testdata-20210711-100903
       Training Loss  Valid. Loss  Valid. Accur. class.  \
epoch
1               1.13         1.07                  0.85
2               1.06         1.06                  0.85
3               1.04         1.05                  0.85
4               1.03         1.05                  0.86

       Valid. Accur. mult.label Training Time Validation Time
epoch
1                          0.88       0:08:38         0:00:28
2                          0.92       0:08:38         0:00:27
3                          0.92       0:08:37         0:00:27
4                          0.93       0:08:39         0:00:27
   ephoc  batchnum                                     actions_logits  \
0      1         1  [[0.0004531708, 0.99980944, 0.0025453316, 0.00…
1      1         2  [[0.00021071802, 0.9996798, 0.0008664634, 0.00…
2      1         3  [[0.0007729576, 0.0009535144, 0.0010184883, 0…
```

```
3      1         4  [[0.99967086, 0.0019957258, 0.0006378222, 0.00…
4      1         5  [[7.279184e-05, 0.00031680457, 0.00024304114, …

                          actions_labels  \
0  [1, 4, 2, 4, 0, 1, 1, 4, 4, 0, 1, 2]
1  [1, 4, 2, 2, 0, 1, 2, 1, 1, 1, 1, 4]
2  [4, 1, 1, 1, 4, 2, 1, 2, 0, 1, 1, 1]
3  [0, 0, 1, 2, 0, 1, 4, 4, 0, 1, 1, 4]
4  [1, 4, 2, 4, 0, 3, 1, 1, 2, 4, 1, 1]

                                    attributes_logits  \
0  [[0.00041479623, 0.0002373658, 0.001900391, 0…
1  [[0.00058873027, 0.00036660058, 0.0032797686, …
2  [[0.00016869778, 0.00028577558, 0.90405023, 0…
3  [[0.0009995383, 0.0007337396, 0.0064695496, 0…
4  [[0.00013425016, 0.00026139984, 0.040037967, 0…

                                    attributes_labels  \
0  [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,…
1  [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,…
2  [[0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,…
3  [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,…
4  [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,…

          accuracy_classification             accuracy_multilabel
0  {'matched': 10, 'counts': 12}  {'matched': 11, 'counts': 12}
1   {'matched': 9, 'counts': 12}  {'matched': 11, 'counts': 12}
2   {'matched': 8, 'counts': 12}  {'matched': 11, 'counts': 12}
3   {'matched': 9, 'counts': 12}  {'matched': 11, 'counts': 12}
4   {'matched': 9, 'counts': 12}  {'matched': 11, 'counts': 12}
```

[52]:

```python
import matplotlib.pyplot as plt
#% matplotlib inline

import seaborn as sns

# Use plot styling from seaborn.
sns.set(style='darkgrid')

# Increase the plot size and font size.
sns.set(font_scale=1.5)
plt.rcParams["figure.figsize"] = (12,6)

# Plot the learning curve.
plt.plot(df_stats['Training Loss'], 'b-o', label="Training")
plt.plot(df_stats['Valid. Loss'], 'g-o', label="Validation")
```

```python
# Label the plot.
plt.title("Training & Validation Loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.xticks([1, 2, 3, 4])

plt.show()
```



[ ]:

```python

```