Department of Computer Science
Technical University of Cluj-Napoca

Intelligent Systems

Laboratory activity 2021-2022

Machine Learning

Project title: Deep learning models for textual entailment

Name: Izabella Bartalus
Group: 30231
Email: bartalusiza08@gmail.com

Asst. Prof. Eng. Roxana Ramona Szomiu
Roxana.Szomiu@cs.utcluj.ro

Contents

Chapter 1

Project description

1.1 Introduction

Una dintre cele mai mari provocari in NLP este lipsa unor date suficiente de antrenament, chiar daca, în general, exista o cantitate enorma de date text disponibile. Dar, daca vrem sa cream seturi de date specifice sarcinii, noi trebuie sa impartim acea gramada in foarte multe domenii diverse. Pentru a rezolva aceasta problema, diferite tehnici au fost dezvoltate pentru formarea modelelor de reprezentare a limbajului de uz general folosind enormele gramezi de text neadnotat pe web - acest lucru este cunoscut sub numele de pre-training.

Modelul de reprezentare a limbajului numit BERT, care inseamna Bidirectional Encoder Representations from Transformers, este o tehnica de invatare automata pentru procesarea limbajului natural (NLP) dezvoltat de Google.

Pentru a realiza modelele, am pornit de la un model de deep learning existent, astfel am folosit modelul initial iar apoi am mai creat eu inca 2 modele.

Link catre modelul initial: `https://keras.io/examples/nlp/semantic_similarity_with_bert/`

Documentatia pentru modelul BERT: `https://huggingface.co/docs/transformers/model_doc/bert`

1.2 The main goal of the project

Scopul principal al acestui proiect este de a crea 3 modele de deep learning pentru textual entailment, adica NLP(natural language processing) cu Deep Learning.

1.3 Dataset

Dataset-ul folosit pentru cele 3 modele este: ambiguousknightsknaves.json

Chapter 2

Implementation details

2.1 Model 1

Algorithm description and steps:

1. Load the dataset

Mai intai, am citit setul de date apoi l-am impartit in 3 parti: train, validation si test.

```
[8]  PATH = '/content/drive/MyDrive/Colab Notebooks/ambiguous_knights_knaves.json'
```

```
import json
def read_jsonl_file(PATH):
  df = pd.read_json(PATH)

  with open(PATH,'r') as f:
    data = json.loads(f.read())

  df_nested_list = pd.json_normalize(data=data['puzzles'], record_path='QA', meta=['puzzle_text'])
  return df_nested_list
```

```
[10] df = pd.DataFrame(read_jsonl_file(PATH))
```

```
[11] # SE IMPARTE DATASET-UL IN TRAIN, VALIDARE SI TEST
     train_df = df[['qid','puzzle_text', 'question', 'answer']][:700]
     valid_df  = df[['qid','puzzle_text', 'question', 'answer']]
     test_df  = df[['qid','puzzle_text', 'question', 'answer']]

     # Shape of the data
     print(f"Total train samples : {train_df.shape[0]}")
     print(f"Total validation samples: {valid_df.shape[0]}")
     print(f"Total test samples: {test_df.shape[0]}")

     Total train samples : 700
     Total validation samples: 940
     Total test samples: 940
```

2. Dataset preprocessing : data preparation

Pentru aceasta parte, mai intai am dorit sa observ cum este impartita partea de ’answer’ in dataset-ul nostru, atat in partea de train, cat si in partea de validare.

```
[12] print("Train Target Distribution")
     print(train_df.answer.value_counts())

     Train Target Distribution
     NOT ENTAILMENT - Unknown          536
     Entailment                         82
     NOT ENTAILMENT - Contradiction     82
     Name: answer, dtype: int64
```

```
[13] print("Validation Target Distribution")
     print(valid_df.answer.value_counts())

     Validation Target Distribution
     NOT ENTAILMENT - Unknown          730
     Entailment                        105
     NOT ENTAILMENT - Contradiction    105
     Name: answer, dtype: int64
```

De asemenea, am verificat daca apare undeva ’-’ in partea de ’answer’ atat in partea de train cat si in cea de validare.

```
[14] train_df = (
         train_df[train_df.answer != "-"]
         .sample(frac=1.0, random_state=42)
         .reset_index(drop=True)
     )
     valid_df = (
         valid_df[valid_df.answer != "-"]
         .sample(frac=1.0, random_state=42)
         .reset_index(drop=True)
     )
```

Totodata, am facut un one-hot encoding pentru train si validare pentru partea de codificare a raspunsurilor.

```
[15] train_df["label"] = train_df["answer"].apply(
         lambda x: 0 if x == "NOT ENTAILMENT - Contradiction" else 1 if x == "Entailment" else 2
     )
     y_train = tf.keras.utils.to_categorical(train_df.label, num_classes=3)

     valid_df["label"] = valid_df["answer"].apply(
         lambda x: 0 if x == "NOT ENTAILMENT - Contradiction" else 1 if x == "Entailment" else 2
     )
     y_val = tf.keras.utils.to_categorical(valid_df.label, num_classes=3)

     test_df["label"] = test_df["answer"].apply(
         lambda x: 0 if x == "NOT ENTAILMENT - Contradiction" else 1 if x == "Entailment" else 2
     )
     y_test = tf.keras.utils.to_categorical(test_df.label, num_classes=3)
```

3. Create a custom data generator

Am creat un generator de date pentru a-l putea folosit la toate modelele.

```
class BertSemanticDataGenerator(tf.keras.utils.Sequence):
    """Generates batches of data.

    Args:
        sentence_pairs: Array of premise and hypothesis input sentences.
        labels: Array of labels.
        batch_size: Integer batch size.
        shuffle: boolean, whether to shuffle the data.
        include_targets: boolean, whether to incude the labels.

    Returns:
        Tuples `([input_ids, attention_mask, `token_type_ids], labels)`
        (or just `[input_ids, attention_mask, `token_type_ids]`
         if `include_targets=False`)
    """

    def __init__(
        self,
        sentence_pairs,
        labels,
        batch_size=batch_size,
        shuffle=True,
        include_targets=True,
    ):
        self.sentence_pairs = sentence_pairs
        self.labels = labels
        self.shuffle = shuffle
        self.batch_size = batch_size
        self.include_targets = include_targets
        # Load our BERT Tokenizer to encode the text.
        # We will use base-base-uncased pretrained model.
        self.tokenizer = transformers.BertTokenizer.from_pretrained(
            "bert-base-uncased", do_lower_case=True
        )
        self.indexes = np.arange(len(self.sentence_pairs))
        self.on_epoch_end()

    def __len__(self):
        # Denotes the number of batches per epoch.
        return len(self.sentence_pairs) // self.batch_size
```

```
    def __getitem__(self, idx):
        # Retrieves the batch of index.
        indexes = self.indexes[idx * self.batch_size : (idx + 1) * self.batch_size]
        sentence_pairs = self.sentence_pairs[indexes]

        # With BERT tokenizer's batch_encode_plus batch of both the sentences are
        # encoded together and separated by [SEP] token.
        encoded = self.tokenizer.batch_encode_plus(
            sentence_pairs.tolist(),
            add_special_tokens=True,
            max_length=max_length,
            return_attention_mask=True,
            return_token_type_ids=True,
            pad_to_max_length=True,
            return_tensors="tf",
        )

        # Convert batch of encoded features to numpy array.
        input_ids = np.array(encoded["input_ids"], dtype="int32")
        attention_masks = np.array(encoded["attention_mask"], dtype="int32")
        token_type_ids = np.array(encoded["token_type_ids"], dtype="int32")

        # Set to true if data generator is used for training/validation.
        if self.include_targets:
            labels = np.array(self.labels[indexes], dtype="int32")
            return [input_ids, attention_masks, token_type_ids], labels
        else:
            return [input_ids, attention_masks, token_type_ids]

    def on_epoch_end(self):
        # Shuffle indexes after each epoch if shuffle is set to True.
        if self.shuffle:
            np.random.RandomState(42).shuffle(self.indexes)
```

4. Build model

Pentru primul model am considerat urmatoarele:

```
[ ] max_length = 128  # Maximum length of input sentence to the model.
    batch_size = 32
    epochs = 2

    # Labels in our dataset.
    labels = ["NOT ENTAILMENT - Unknown", "Entailment", "NOT ENTAILMENT - Contradiction"]
```

Layerele folosite sunt cele din modelul sursa:

```
# Add trainable layers on top of frozen layers to adapt the pretrained features on the new data.
bi_lstm = tf.keras.layers.Bidirectional(
    tf.keras.layers.LSTM(64, return_sequences=True)
)(sequence_output)
# Applying hybrid pooling approach to bi_lstm sequence output.
avg_pool = tf.keras.layers.GlobalAveragePooling1D()(bi_lstm)
max_pool = tf.keras.layers.GlobalMaxPooling1D()(bi_lstm)
concat = tf.keras.layers.concatenate([avg_pool, max_pool])
dropout = tf.keras.layers.Dropout(0.3)(concat)
output = tf.keras.layers.Dense(3, activation="softmax")(dropout)
model = tf.keras.models.Model(
    inputs=[input_ids, attention_masks, token_type_ids], outputs=output
)
```

5. Compile the model

Am compilat modelul folosind optimizer-ul Adam, loss function-ul categorical crossentropy si metrica de acuratete.

```
model.compile(
    optimizer=tf.keras.optimizers.Adam(),
    loss="categorical_crossentropy",
    metrics=["acc"],
)
```

6. Train the model

Antrenarea se face doar pentru layerele superioare pentru a efectua ”feature extraction”, ceea ce va permite modelului sa folosească reprezentarile modelului preantrenat.

```
history = model.fit(
    train_data,
    validation_data=valid_data,
    epochs=epochs,
    use_multiprocessing=True,
    workers=-1,
)
```

7. Fine-tuning

Acest pas trebuie efectuat numai dupa ce modelul de extractie a caracteristicilor a fost antrenat pentru a converge cu noile date.

Acesta este un ultim pas optional in care bert model este dezghetat si reantrenat cu o rata de invatare foarte scazuta. Acest lucru poate oferi imbunatatiri semnificative prin adaptarea progresiva a caracteristicilor pregatite in prealabil la noile date.

```
# Unfreeze the bert_model.
bert_model.trainable = True
# Recompile the model to make the change effective.
model.compile(
    optimizer=tf.keras.optimizers.Adam(1e-5),
    loss="categorical_crossentropy",
    metrics=["accuracy"],
)
model.summary()
```

8. Train the entire model end-to-end

```
history = model.fit(
    train_data,
    validation_data=valid_data,
    epochs=epochs,
    use_multiprocessing=True,
    workers=-1,
)
```

9. Obtained results

```
Epoch 1/2
/usr/local/lib/python3.7/dist-packages/transformers/tokenization_utils_base.py:2291: FutureWarning: The `pad_to_max_length` argument is deprecated and will be removed in a f
  FutureWarning,
WARNING:tensorflow:Gradients do not exist for variables ['tf_bert_model/bert/pooler/dense/kernel:0', 'tf_bert_model/bert/pooler/dense/bias:0'] when minimizing the loss. If y
WARNING:tensorflow:Gradients do not exist for variables ['tf_bert_model/bert/pooler/dense/kernel:0', 'tf_bert_model/bert/pooler/dense/bias:0'] when minimizing the loss. If y
21/21 [==============================] - 49s 1s/step - loss: 0.6925 - accuracy: 0.7664 - val_loss: 0.6660 - val_accuracy: 0.7748
Epoch 2/2
21/21 [==============================] - 27s 1s/step - loss: 0.6853 - accuracy: 0.7664 - val_loss: 0.6481 - val_accuracy: 0.7748
```

2.2 Model 2

Pentru cel de-al doilea model, pasii sunt aceeasi ca si la primul model, iar in continuare voi arata urmatorii pasi unde apar diferente:

4. Build model

Pentru cel de-al doilea model am considerat:

```
max_length = 128  # Maximum length of input sentence to the model.
batch_size = 32
epochs = 5

# Labels in our dataset.
labels = ["NOT ENTAILMENT - Unknown", "Entailment", "NOT ENTAILMENT - Contradiction"]
```

Layerele folosite:

```
# Add trainable layers on top of frozen layers to adapt the pretrained features on the new data.
bi_lstm = tf.keras.layers.Bidirectional(
    tf.keras.layers.LSTM(64, return_sequences=True)
)(sequence_output)
# Applying hybrid pooling approach to bi_lstm sequence output.
avg_pool = tf.keras.layers.GlobalAveragePooling1D()(bi_lstm)
max_pool = tf.keras.layers.GlobalMaxPooling1D()(bi_lstm)
concat = tf.keras.layers.concatenate([avg_pool, max_pool])
dropout = tf.keras.layers.Dropout(0.3)(concat)
avg_pool_2 = tf.keras.layers.GlobalAveragePooling1D()(bi_lstm)
max_pool_2 = tf.keras.layers.GlobalMaxPooling1D()(bi_lstm)
concat2 = tf.keras.layers.concatenate([avg_pool_2, max_pool_2])
dropout2 = tf.keras.layers.Dropout(0.3)(concat2)
dense1 = tf.keras.layers.Dense(1, activation='relu')(dropout2)
dense2 = tf.keras.layers.Dense(10, activation='sigmoid')(dense1)
dense3 = tf.keras.layers.Dense(20, activation='sigmoid')(dense2)
flatten = tf.keras.layers.Flatten()(dense3)
output = tf.keras.layers.Dense(3, activation="softmax")(flatten)
model2 = tf.keras.models.Model(
    inputs=[input_ids, attention_masks, token_type_ids], outputs=output
)
```

5. Compile the model

Am compilat modelul folosind optimizer-ul Adam, loss function-ul binary crossentropy si metrica de acuratete.

```
# Unfreeze the bert_model.
bert_model.trainable = True
# Recompile the model to make the change effective.
model2.compile(
    optimizer=tf.keras.optimizers.Adam(1e-5),
    loss="binary_crossentropy",
    metrics=["accuracy"],
)
model2.summary()
```

9. Obtained results

```
Epoch 1/5
/usr/local/lib/python3.7/dist-packages/transformers/tokenization_utils_base.py:2291: FutureWarning: The `pad_to_max_length` argument is deprecated and will
  FutureWarning,
WARNING:tensorflow:Gradients do not exist for variables ['tf_bert_model/bert/pooler/dense/kernel:0', 'tf_bert_model/bert/pooler/dense/bias:0'] when minimizi
WARNING:tensorflow:Gradients do not exist for variables ['tf_bert_model/bert/pooler/dense/kernel:0', 'tf_bert_model/bert/pooler/dense/bias:0'] when minimizi
21/21 [==============================] - 48s 1s/step - loss: 0.4957 - accuracy: 0.7634 - val_loss: 0.4895 - val_accuracy: 0.7748
Epoch 2/5
21/21 [==============================] - 25s 1s/step - loss: 0.4934 - accuracy: 0.7664 - val_loss: 0.4890 - val_accuracy: 0.7748
Epoch 3/5
21/21 [==============================] - 25s 1s/step - loss: 0.4923 - accuracy: 0.7679 - val_loss: 0.4886 - val_accuracy: 0.7748
Epoch 4/5
21/21 [==============================] - 25s 1s/step - loss: 0.4950 - accuracy: 0.7619 - val_loss: 0.4881 - val_accuracy: 0.7748
Epoch 5/5
21/21 [==============================] - 25s 1s/step - loss: 0.4907 - accuracy: 0.7679 - val_loss: 0.4877 - val_accuracy: 0.7748
```

2.3 Model 3

Pentru cel de-al treilea model, pasii sunt aceeasi ca si la primul model, iar in continuare voi arata urmatorii pasi unde apar diferente:

4. Build model

Pentru cel de-al doilea model am considerat:

```
max_length = 128  # Maximum length of input sentence to the model.
batch_size = 32
epochs = 10

# Labels in our dataset.
labels = ["NOT ENTAILMENT - Unknown", "Entailment", "NOT ENTAILMENT - Contradiction"]
```

Layerele folosite:

```python
# Add trainable layers on top of frozen layers to adapt the pretrained features on the new data.
bi_lstm = tf.keras.layers.Bidirectional(
    tf.keras.layers.LSTM(64, return_sequences=True)
)(sequence_output)
# Applying hybrid pooling approach to bi_lstm sequence output.
max_pooling = tf.keras.layers.MaxPooling1D(2)(bi_lstm)
dense1 = tf.keras.layers.Dense(1, activation='relu')(max_pooling)
conv1 = tf.keras.layers.Conv1D(filters=32, kernel_size=3, padding='same', activation='relu')(dense1)
dense2 = tf.keras.layers.Dense(10, activation='sigmoid')(conv1)
conv2 = tf.keras.layers.Conv1D(filters=64, kernel_size=5, padding='same', activation='relu')(dense2)
conv3 = tf.keras.layers.Conv1D(filters=32, kernel_size=5, padding='same', activation='relu')(conv2)
conv4 = tf.keras.layers.Conv1D(filters=16, kernel_size=4, padding='same', activation='relu')(conv3)
dense3 = tf.keras.layers.Dense(20, activation='sigmoid')(conv4)
conv5 = tf.keras.layers.Conv1D(filters=8, kernel_size=3, padding='same', activation='relu')(conv4)
conv6 = tf.keras.layers.Conv1D(filters=4, kernel_size=3, padding='same', activation='relu')(conv5)
dense4 = tf.keras.layers.Dense(20, activation='sigmoid')(conv6)
flatten = tf.keras.layers.Flatten()(dense4)
output = tf.keras.layers.Dense(3, activation="softmax")(flatten)
model2 = tf.keras.models.Model(
    inputs=[input_ids, attention_masks, token_type_ids], outputs=output
)
```

5. Compile the model

Am compilat modelul folosind optimizer-ul Adam, loss function-ul binary crossentropy si metrica de acuratete.

```python
model2.compile(
    optimizer=tf.keras.optimizers.Adam(),
    loss="binary_crossentropy",        # am schimbat loss function
    metrics=["acc"],
)
```

9. Obtained results

```
Epoch 1/10
WARNING:tensorflow:Gradients do not exist for variables ['tf_bert_model_10/bert/pooler/dense/kernel:0', 'tf_bert_model_10/bert/pooler/dense/bias:0'] whe
WARNING:tensorflow:Gradients do not exist for variables ['tf_bert_model_10/bert/pooler/dense/kernel:0', 'tf_bert_model_10/bert/pooler/dense/bias:0'] whe
21/21 [==============================] - 51s 2s/step - loss: 0.4297 - accuracy: 0.7693 - val_loss: 0.4223 - val_accuracy: 0.7748
Epoch 2/10
21/21 [==============================] - 25s 1s/step - loss: 0.4280 - accuracy: 0.7664 - val_loss: 0.4192 - val_accuracy: 0.7748
Epoch 3/10
21/21 [==============================] - 25s 1s/step - loss: 0.4312 - accuracy: 0.7604 - val_loss: 0.4171 - val_accuracy: 0.7748
Epoch 4/10
21/21 [==============================] - 25s 1s/step - loss: 0.4259 - accuracy: 0.7649 - val_loss: 0.4154 - val_accuracy: 0.7748
Epoch 5/10
21/21 [==============================] - 25s 1s/step - loss: 0.4204 - accuracy: 0.7693 - val_loss: 0.4146 - val_accuracy: 0.7748
Epoch 6/10
21/21 [==============================] - 25s 1s/step - loss: 0.4227 - accuracy: 0.7664 - val_loss: 0.4139 - val_accuracy: 0.7748
Epoch 7/10
21/21 [==============================] - 25s 1s/step - loss: 0.4222 - accuracy: 0.7664 - val_loss: 0.4135 - val_accuracy: 0.7748
Epoch 8/10
21/21 [==============================] - 25s 1s/step - loss: 0.4235 - accuracy: 0.7649 - val_loss: 0.4134 - val_accuracy: 0.7748
Epoch 9/10
21/21 [==============================] - 25s 1s/step - loss: 0.4173 - accuracy: 0.7708 - val_loss: 0.4129 - val_accuracy: 0.7748
Epoch 10/10
21/21 [==============================] - 25s 1s/step - loss: 0.4187 - accuracy: 0.7693 - val_loss: 0.4127 - val_accuracy: 0.7748
```

Chapter 3

Analysis of results - Comparisons

Mai intai, pentru a vedea mai multe diferente, am incercat ca la fiecare model sa schimb layerele si numarul de epoci. Ceea ce este la fel, am ales ca toate modelele sa aiba batch size = 32 si optimizer = Adam.

Pentru primul model am ales epochs = 2 si am obtinut:

```
Epoch 1/2
/usr/local/lib/python3.7/dist-packages/transformers/tokenization_utils_base.py:2291: FutureWarning: The `pad_to_max_length` argument is deprecated and will be removed in a f
  FutureWarning,
WARNING:tensorflow:Gradients do not exist for variables ['tf_bert_model/bert/pooler/dense/kernel:0', 'tf_bert_model/bert/pooler/dense/bias:0'] when minimizing the loss. If y
WARNING:tensorflow:Gradients do not exist for variables ['tf_bert_model/bert/pooler/dense/kernel:0', 'tf_bert_model/bert/pooler/dense/bias:0'] when minimizing the loss. If y
21/21 [==============================] - 49s 1s/step - loss: 0.6925 - accuracy: 0.7664 - val_loss: 0.6660 - val_accuracy: 0.7748
Epoch 2/2
21/21 [==============================] - 27s 1s/step - loss: 0.6853 - accuracy: 0.7664 - val_loss: 0.6481 - val_accuracy: 0.7748
```

Predictiile pentru primul model:

```
[19] def check_similarity(sentence1, sentence2):
         sentence_pairs = np.array([[str(sentence1), str(sentence2)]])
         test_data = BertSemanticDataGenerator(
             sentence_pairs, labels=None, batch_size=1, shuffle=False, include_targets=False,
         )

         proba = model.predict(test_data[0])[0]
         idx = np.argmax(proba)
         proba = f"{proba[idx]: .2f}%"
         pred = labels[idx]
         return pred, proba

sentence1 = "On the island where each inhabitant is either a knave or a knight , knights always tell the truth while knaves always lie . You me
sentence2 = "Is Rex the knight ?"
check_similarity(sentence1, sentence2)

Truncation was not explicitly activated but `max_length` is provided a specific value, please use `truncation=True` to explicitly truncate exam
/usr/local/lib/python3.7/dist-packages/transformers/tokenization_utils_base.py:2291: FutureWarning: The `pad_to_max_length` argument is depreca
  FutureWarning,
('NOT ENTAILMENT - Contradiction', ' 0.79%')
```

Pentru al doilea model am ales epochs = 5 si am obtinut:

```
Epoch 1/5
/usr/local/lib/python3.7/dist-packages/transformers/tokenization_utils_base.py:2291: FutureWarning: The `pad_to_max_length` argument is deprecated and will
  FutureWarning,
WARNING:tensorflow:Gradients do not exist for variables ['tf_bert_model/bert/pooler/dense/kernel:0', 'tf_bert_model/bert/pooler/dense/bias:0'] when minimizi
WARNING:tensorflow:Gradients do not exist for variables ['tf_bert_model/bert/pooler/dense/kernel:0', 'tf_bert_model/bert/pooler/dense/bias:0'] when minimizi
21/21 [==============================] - 48s 1s/step - loss: 0.4957 - accuracy: 0.7634 - val_loss: 0.4895 - val_accuracy: 0.7748
Epoch 2/5
21/21 [==============================] - 25s 1s/step - loss: 0.4934 - accuracy: 0.7664 - val_loss: 0.4890 - val_accuracy: 0.7748
Epoch 3/5
21/21 [==============================] - 25s 1s/step - loss: 0.4923 - accuracy: 0.7679 - val_loss: 0.4886 - val_accuracy: 0.7748
Epoch 4/5
21/21 [==============================] - 25s 1s/step - loss: 0.4950 - accuracy: 0.7619 - val_loss: 0.4881 - val_accuracy: 0.7748
Epoch 5/5
21/21 [==============================] - 25s 1s/step - loss: 0.4907 - accuracy: 0.7679 - val_loss: 0.4877 - val_accuracy: 0.7748
```

Predictiile pentru al doilea model:

```
def check_similarity(sentence1, sentence2):
    sentence_pairs = np.array([[str(sentence1), str(sentence2)]])
    test_data = BertSemanticDataGenerator(
        sentence_pairs, labels=None, batch_size=1, shuffle=False, include_targets=False,
    )

    proba = model2.predict(test_data[0])[0]
    idx = np.argmax(proba)
    proba = f"{proba[idx]: .2f}%"
    pred = labels[idx]
    return pred, proba

sentence1 = "On the island where each inhabitant is either a knave or a knight , knights always tell the truth while knaves always lie . You meet three inhabitants : Alice , Rex and E
sentence2 = "Is Rex the knight ?"
check_similarity(sentence1, sentence2)

Truncation was not explicitly activated but `max_length` is provided a specific value, please use `truncation=True` to explicitly truncate examples to max length. Defaulting to 'longes
/usr/local/lib/python3.7/dist-packages/transformers/tokenization_utils_base.py:2291: FutureWarning: The `pad_to_max_length` argument is deprecated and will be removed in a future versi
  FutureWarning,
('NOT ENTAILMENT - Contradiction', ' 0.70%')
```

Pentru al treilea model am ales epochs = 10 si am obtinut:

```
Epoch 1/10
WARNING:tensorflow:Gradients do not exist for variables ['tf_bert_model_10/bert/pooler/dense/kernel:0', 'tf_bert_model_10/bert/pooler/dense/bias:0'] whe
WARNING:tensorflow:Gradients do not exist for variables ['tf_bert_model_10/bert/pooler/dense/kernel:0', 'tf_bert_model_10/bert/pooler/dense/bias:0'] whe
21/21 [==============================] - 51s 2s/step - loss: 0.4297 - accuracy: 0.7693 - val_loss: 0.4223 - val_accuracy: 0.7748
Epoch 2/10
21/21 [==============================] - 25s 1s/step - loss: 0.4280 - accuracy: 0.7664 - val_loss: 0.4192 - val_accuracy: 0.7748
Epoch 3/10
21/21 [==============================] - 25s 1s/step - loss: 0.4312 - accuracy: 0.7604 - val_loss: 0.4171 - val_accuracy: 0.7748
Epoch 4/10
21/21 [==============================] - 25s 1s/step - loss: 0.4259 - accuracy: 0.7649 - val_loss: 0.4154 - val_accuracy: 0.7748
Epoch 5/10
21/21 [==============================] - 25s 1s/step - loss: 0.4204 - accuracy: 0.7693 - val_loss: 0.4146 - val_accuracy: 0.7748
Epoch 6/10
21/21 [==============================] - 25s 1s/step - loss: 0.4227 - accuracy: 0.7664 - val_loss: 0.4139 - val_accuracy: 0.7748
Epoch 7/10
21/21 [==============================] - 25s 1s/step - loss: 0.4222 - accuracy: 0.7664 - val_loss: 0.4135 - val_accuracy: 0.7748
Epoch 8/10
21/21 [==============================] - 25s 1s/step - loss: 0.4235 - accuracy: 0.7649 - val_loss: 0.4134 - val_accuracy: 0.7748
Epoch 9/10
21/21 [==============================] - 25s 1s/step - loss: 0.4173 - accuracy: 0.7708 - val_loss: 0.4129 - val_accuracy: 0.7748
Epoch 10/10
21/21 [==============================] - 25s 1s/step - loss: 0.4187 - accuracy: 0.7693 - val_loss: 0.4127 - val_accuracy: 0.7748
```

Predictiile pentru al treilea model:

```
def check_similarity(sentence1, sentence2):
    sentence_pairs = np.array([[str(sentence1), str(sentence2)]])
    test_data = BertSemanticDataGenerator(
        sentence_pairs, labels=None, batch_size=1, shuffle=False, include_targets=False,
    )

    proba = model2.predict(test_data[0])[0]
    idx = np.argmax(proba)
    proba = f"{proba[idx]: .2f}%"
    pred = labels[idx]
    return pred, proba
```

```
sentence1 = "On the island where each inhabitant is either a knave or a knight , knights always tell the truth while knaves always lie . You mee
sentence2 = "Is Rex the knight ?"
check_similarity(sentence1, sentence2)
```

```
Truncation was not explicitly activated but `max_length` is provided a specific value, please use `truncation=True` to explicitly truncate exampl
/usr/local/lib/python3.7/dist-packages/transformers/tokenization_utils_base.py:2291: FutureWarning: The `pad_to_max_length` argument is deprecate
  FutureWarning,
WARNING:tensorflow:5 out of the last 6 calls to <function Model.make_predict_function.<locals>.predict_function at 0x7f80ba8a5560> triggered tf.f
('NOT ENTAILMENT - Contradiction', ' 0.92%')
```

Ceea ce se poate observa este ca accuracy la train este la toate cele 3 modele in jur de 0.7, cu mici modificari la fiecare, dar metrica de acuratete la setul de validare este mereu 0.7748 la toate cele 3 modele.

De asemenea, se poate observa ca valorile la loss la primul model sunt in jur de 0.6, pe cand la celealte doua modele se schimba, fiind in jur de 0.4.

Totodata, pentru a putea compara mai eficient cele 3 modele, o sa atasez rezultatele obtinute la fiecare model, antrenat pe epochs = 5 si de asemenea o sa pun rezultatele facute la predictii.

Modelul 1 pe epochs = 5 si predictii:

```
Epoch 1/5
/usr/local/lib/python3.7/dist-packages/transformers/tokenization_utils_base.py:2291: FutureWarning: The `pad_to_max_length` argument is deprecate
  FutureWarning,
WARNING:tensorflow:Gradients do not exist for variables ['tf_bert_model/bert/pooler/dense/kernel:0', 'tf_bert_model/bert/pooler/dense/bias:0'] wh
WARNING:tensorflow:Gradients do not exist for variables ['tf_bert_model/bert/pooler/dense/kernel:0', 'tf_bert_model/bert/pooler/dense/bias:0'] wh
21/21 [==============================] - 51s 2s/step - loss: 0.6401 - accuracy: 0.7649 - val_loss: 0.6087 - val_accuracy: 0.7748
Epoch 2/5
21/21 [==============================] - 27s 1s/step - loss: 0.6281 - accuracy: 0.7619 - val_loss: 0.5993 - val_accuracy: 0.7748
Epoch 3/5
21/21 [==============================] - 26s 1s/step - loss: 0.6252 - accuracy: 0.7649 - val_loss: 0.5956 - val_accuracy: 0.7748
Epoch 4/5
21/21 [==============================] - 26s 1s/step - loss: 0.6215 - accuracy: 0.7545 - val_loss: 0.5794 - val_accuracy: 0.7748
Epoch 5/5
21/21 [==============================] - 27s 1s/step - loss: 0.5863 - accuracy: 0.7738 - val_loss: 0.5776 - val_accuracy: 0.7748
```

```
[20] def check_similarity(sentence1, sentence2):
         sentence_pairs = np.array([[str(sentence1), str(sentence2)]])
         test_data = BertSemanticDataGenerator(
             sentence_pairs, labels=None, batch_size=1, shuffle=False, include_targets=False,
         )

         proba = model.predict(test_data[0])[0]
         idx = np.argmax(proba)
         proba = f"{proba[idx]: .2f}%"
         pred = labels[idx]
         return pred, proba
```

```
[21] sentence1 = "On the island where each inhabitant is either a knave or a knight , knights always tell the truth while knaves always lie . You mee
     sentence2 = "Is Rex the knight ?"
     check_similarity(sentence1, sentence2)
```

```
Truncation was not explicitly activated but `max_length` is provided a specific value, please use `truncation=True` to explicitly truncate exampl
/usr/local/lib/python3.7/dist-packages/transformers/tokenization_utils_base.py:2291: FutureWarning: The `pad_to_max_length` argument is deprecate
  FutureWarning,
('NOT ENTAILMENT - Contradiction', ' 0.89%')
```

Modelul 2 pe epochs = 5 si predictii:

```
Epoch 1/5
/usr/local/lib/python3.7/dist-packages/transformers/tokenization_utils_base.py:2291: FutureWarning: The `pad_to_max_length` argument is deprecated and will
  FutureWarning,
WARNING:tensorflow:Gradients do not exist for variables ['tf_bert_model/bert/pooler/dense/kernel:0', 'tf_bert_model/bert/pooler/dense/bias:0'] when minimizi
WARNING:tensorflow:Gradients do not exist for variables ['tf_bert_model/bert/pooler/dense/kernel:0', 'tf_bert_model/bert/pooler/dense/bias:0'] when minimizi
21/21 [==============================] - 48s 1s/step - loss: 0.4957 - accuracy: 0.7634 - val_loss: 0.4895 - val_accuracy: 0.7748
Epoch 2/5
21/21 [==============================] - 25s 1s/step - loss: 0.4934 - accuracy: 0.7664 - val_loss: 0.4890 - val_accuracy: 0.7748
Epoch 3/5
21/21 [==============================] - 25s 1s/step - loss: 0.4923 - accuracy: 0.7679 - val_loss: 0.4886 - val_accuracy: 0.7748
Epoch 4/5
21/21 [==============================] - 25s 1s/step - loss: 0.4950 - accuracy: 0.7619 - val_loss: 0.4881 - val_accuracy: 0.7748
Epoch 5/5
21/21 [==============================] - 25s 1s/step - loss: 0.4907 - accuracy: 0.7679 - val_loss: 0.4877 - val_accuracy: 0.7748
```

```python
def check_similarity(sentence1, sentence2):
    sentence_pairs = np.array([[str(sentence1), str(sentence2)]])
    test_data = BertSemanticDataGenerator(
        sentence_pairs, labels=None, batch_size=1, shuffle=False, include_targets=False,
    )

    proba = model2.predict(test_data[0])[0]
    idx = np.argmax(proba)
    proba = f"{proba[idx]: .2f}%"
    pred = labels[idx]
    return pred, proba
```

```python
sentence1 = "On the island where each inhabitant is either a knave or a knight , knights always tell the truth while knaves always lie . You meet three inhabitants : Alice , Rex and E
sentence2 = "Is Rex the knight ?"
check_similarity(sentence1, sentence2)
```

```
Truncation was not explicitly activated but `max_length` is provided a specific value, please use `truncation=True` to explicitly truncate examples to max length. Defaulting to 'longes
/usr/local/lib/python3.7/dist-packages/transformers/tokenization_utils_base.py:2291: FutureWarning: The `pad_to_max_length` argument is deprecated and will be removed in a future versi
  FutureWarning,
('NOT ENTAILMENT - Contradiction', ' 0.70%')
```

Modelul 3 pe epochs = 5 si predictii:

```python
history = model2.fit(
    train_data,
    validation_data=valid_data,
    epochs=epochs,
    use_multiprocessing=True,
    workers=-1,
)
```

```
/usr/local/lib/python3.7/dist-packages/transformers/tokenization_utils_base.py:2291: FutureWarning: The `pad_to_max_length` argument is deprecat
  FutureWarning,
Epoch 1/5
WARNING:tensorflow:Gradients do not exist for variables ['tf_bert_model/bert/pooler/dense/kernel:0', 'tf_bert_model/bert/pooler/dense/bias:0'] w
WARNING:tensorflow:Gradients do not exist for variables ['tf_bert_model/bert/pooler/dense/kernel:0', 'tf_bert_model/bert/pooler/dense/bias:0'] w
21/21 [==============================] - 48s 1s/step - loss: 0.4389 - accuracy: 0.7634 - val_loss: 0.4215 - val_accuracy: 0.7748
Epoch 2/5
21/21 [==============================] - 25s 1s/step - loss: 0.4310 - accuracy: 0.7664 - val_loss: 0.4185 - val_accuracy: 0.7748
Epoch 3/5
21/21 [==============================] - 25s 1s/step - loss: 0.4261 - accuracy: 0.7679 - val_loss: 0.4165 - val_accuracy: 0.7748
Epoch 4/5
21/21 [==============================] - 25s 1s/step - loss: 0.4316 - accuracy: 0.7619 - val_loss: 0.4148 - val_accuracy: 0.7748
Epoch 5/5
21/21 [==============================] - 25s 1s/step - loss: 0.4222 - accuracy: 0.7679 - val_loss: 0.4139 - val_accuracy: 0.7748
```

```python
def check_similarity(sentence1, sentence2):
    sentence_pairs = np.array([[str(sentence1), str(sentence2)]])
    test_data = BertSemanticDataGenerator(
        sentence_pairs, labels=None, batch_size=1, shuffle=False, include_targets=False,
    )

    proba = model2.predict(test_data[0])[0]
    idx = np.argmax(proba)
    proba = f"{proba[idx]: .2f}%"
    pred = labels[idx]
    return pred, proba
```

```python
sentence1 = "On the island where each inhabitant is either a knave or a knight , knights always tell the truth while knaves always lie . You mee
sentence2 = "Is Rex the knight ?"
check_similarity(sentence1, sentence2)
```

```
Truncation was not explicitly activated but `max_length` is provided a specific value, please use `truncation=True` to explicitly truncate exampl
/usr/local/lib/python3.7/dist-packages/transformers/tokenization_utils_base.py:2291: FutureWarning: The `pad_to_max_length` argument is deprecate
  FutureWarning,
('NOT ENTAILMENT - Contradiction', ' 0.94%')
```

Chapter 4

Conclusion

In concluzie cele mai bune rezultate s-au obtinut la modelul 3, atat atunci cand celelalte au fost antrenate cu epoci mai mici si modelul 3 cu 10 epoci, cat si atunci cand toate cele 3 modele au fost antrenate cu 5 epoci.

Appendix A

Your original code

This section should contain only code developed by you, without any line re-used from other sources. This section helps me to correctly evaluate your amount of work and results obtained. Including in this section any line of code taken from someone else leads to failure of IS class this year. Failing or forgetting to add your code in this appendix leads to grade 1. Don't remove the above lines.

A.1 Model 1

```
!pip install transformers
!pip install sentencepiece

import numpy as np
import pandas as pd
import tensorflow as tf
import transformers

max_length = 128  # Maximum length of input sentence to the model.
batch_size = 32
epochs = 2

# Labels in our dataset.
labels = ["NOT_ENTAILMENT - Unknown", "Entailment", "NOT_ENTAILMENT - Contra

PATH = '/content/drive/MyDrive/Colab Notebooks/ambiguous_knights_knaves.json'

import json
def read_jsonl_file(PATH):
  df = pd.read_json(PATH)

  with open(PATH, 'r') as f:
    data = json.loads(f.read())

  df_nested_list = pd.json_normalize(data=data['puzzles'], record_path='QA',
  return df_nested_list

df = pd.DataFrame(read_jsonl_file(PATH))
```

```
# SE IMPARTE DATASET-UL IN TRAIN, VALIDARE SI TEST
train_df = df[['qid','puzzle_text', 'question', 'answer']][:700]
valid_df  = df[['qid','puzzle_text', 'question', 'answer']]
test_df  = df[['qid','puzzle_text', 'question', 'answer']]

# Shape of the data
print(f"Total train samples : {train_df.shape[0]}")
print(f"Total validation samples: {valid_df.shape[0]}")
print(f"Total test samples: {test_df.shape[0]}")

print("Train Target Distribution")
print(train_df.answer.value_counts())

print("Validation Target Distribution")
print(valid_df.answer.value_counts())

train_df = (
    train_df[train_df.answer != "-"]
    .sample(frac=1.0, random_state=42)
    .reset_index(drop=True)
)
valid_df = (
    valid_df[valid_df.answer != "-"]
    .sample(frac=1.0, random_state=42)
    .reset_index(drop=True)
)

train_df["label"] = train_df["answer"].apply(
    lambda x: 0 if x == "NOT_ENTAILMENT - Contradiction" else 1 if x == "Enta
)
y_train = tf.keras.utils.to_categorical(train_df.label, num_classes=3)

valid_df["label"] = valid_df["answer"].apply(
    lambda x: 0 if x == "NOT_ENTAILMENT - Contradiction" else 1 if x == "Enta
)
y_val = tf.keras.utils.to_categorical(valid_df.label, num_classes=3)

test_df["label"] = test_df["answer"].apply(
    lambda x: 0 if x == "NOT_ENTAILMENT - Contradiction" else 1 if x == "Enta
)
y_test = tf.keras.utils.to_categorical(test_df.label, num_classes=3)

class BertSemanticDataGenerator(tf.keras.utils.Sequence):
    """Generates batches of data.

    Args:
        sentence_pairs: Array of premise and hypothesis input sentences.
        labels: Array of labels.
        batch_size: Integer batch size.
```

```
        shuffle: boolean, whether to shuffle the data.
        include_targets: boolean, whether to incude the labels.

    Returns:
        Tuples `([input_ids, attention_mask, `token_type_ids], labels)`
        (or just `[input_ids, attention_mask, `token_type_ids]`
         if `include_targets=False`)
    """

    def __init__(
        self,
        sentence_pairs,
        labels,
        batch_size=batch_size,
        shuffle=True,
        include_targets=True,
    ):
        self.sentence_pairs = sentence_pairs
        self.labels = labels
        self.shuffle = shuffle
        self.batch_size = batch_size
        self.include_targets = include_targets
        # Load our BERT Tokenizer to encode the text.
        # We will use base-base-uncased pretrained model.
        self.tokenizer = transformers.BertTokenizer.from_pretrained(
            "bert-base-uncased", do_lower_case=True
        )
        self.indexes = np.arange(len(self.sentence_pairs))
        self.on_epoch_end()

    def __len__(self):
        # Denotes the number of batches per epoch.
        return len(self.sentence_pairs) // self.batch_size

    def __getitem__(self, idx):
        # Retrieves the batch of index.
        indexes = self.indexes[idx * self.batch_size : (idx + 1) * self.batcl
        sentence_pairs = self.sentence_pairs[indexes]

        # With BERT tokenizer's batch_encode_plus batch of both the sentences
        # encoded together and separated by [SEP] token.
        encoded = self.tokenizer.batch_encode_plus(
            sentence_pairs.tolist(),
            add_special_tokens=True,
            max_length=max_length,
            return_attention_mask=True,
            return_token_type_ids=True,
            pad_to_max_length=True,
            return_tensors="tf",
        )
```

```
        # Convert batch of encoded features to numpy array.
        input_ids = np.array(encoded["input_ids"], dtype="int32")
        attention_masks = np.array(encoded["attention_mask"], dtype="int32")
        token_type_ids = np.array(encoded["token_type_ids"], dtype="int32")

        # Set to true if data generator is used for training/validation.
        if self.include_targets:
            labels = np.array(self.labels[indexes], dtype="int32")
            return [input_ids, attention_masks, token_type_ids], labels
        else:
            return [input_ids, attention_masks, token_type_ids]

    def on_epoch_end(self):
        # Shuffle indexes after each epoch if shuffle is set to True.
        if self.shuffle:
            np.random.RandomState(42).shuffle(self.indexes)

# Create the model under a distribution strategy scope.
strategy = tf.distribute.MirroredStrategy()

with strategy.scope():
    # Encoded token ids from BERT tokenizer.
    input_ids = tf.keras.layers.Input(
        shape=(max_length,), dtype=tf.int32, name="input_ids"
    )
    # Attention masks indicates to the model which tokens should be attended
    attention_masks = tf.keras.layers.Input(
        shape=(max_length,), dtype=tf.int32, name="attention_masks"
    )
    # Token type ids are binary masks identifying different sequences in the
    token_type_ids = tf.keras.layers.Input(
        shape=(max_length,), dtype=tf.int32, name="token_type_ids"
    )
    # Loading pretrained BERT model.
    bert_model = transformers.TFBertModel.from_pretrained("bert-base-uncased"
    # Freeze the BERT model to reuse the pretrained features without modifyin
    bert_model.trainable = False

    bert_output = bert_model(
        input_ids, attention_mask=attention_masks, token_type_ids=token_type_
    )
    sequence_output = bert_output.last_hidden_state
    pooled_output = bert_output.pooler_output
    # Add trainable layers on top of frozen layers to adapt the pretrained fe
    bi_lstm = tf.keras.layers.Bidirectional(
        tf.keras.layers.LSTM(64, return_sequences=True)
    )(sequence_output)
    # Applying hybrid pooling approach to bi_lstm sequence output.
    avg_pool = tf.keras.layers.GlobalAveragePooling1D()(bi_lstm)
```

```
    max_pool = tf.keras.layers.GlobalMaxPooling1D()(bi_lstm)
    concat = tf.keras.layers.concatenate([avg_pool, max_pool])
    dropout = tf.keras.layers.Dropout(0.3)(concat)
    output = tf.keras.layers.Dense(3, activation="softmax")(dropout)
    model = tf.keras.models.Model(
        inputs=[input_ids, attention_masks, token_type_ids], outputs=output
    )

    model.compile(
        optimizer=tf.keras.optimizers.Adam(),
        loss="categorical_crossentropy",
        metrics=["acc"],
    )


print(f"Strategy: {strategy}")
model.summary()

train_data = BertSemanticDataGenerator(
    train_df[["puzzle_text", "question"]].values.astype("str"),
    y_train,
    batch_size=batch_size,
    shuffle=True,
)
valid_data = BertSemanticDataGenerator(
    valid_df[["puzzle_text", "question"]].values.astype("str"),
    y_val,
    batch_size=batch_size,
    shuffle=False,
)

history = model.fit(
    train_data,
    validation_data=valid_data,
    epochs=epochs,
    use_multiprocessing=True,
    workers=-1,
)

# Unfreeze the bert_model.
bert_model.trainable = True
# Recompile the model to make the change effective.
model.compile(
    optimizer=tf.keras.optimizers.Adam(1e-5),
    loss="categorical_crossentropy",
    metrics=["accuracy"],
)
model.summary()

history = model.fit(
```

```
    train_data,
    validation_data=valid_data,
    epochs=epochs,
    use_multiprocessing=True,
    workers=-1,
)

test_data = BertSemanticDataGenerator(
    test_df[["puzzle_text", "question"]].values.astype("str"),
    y_test,
    batch_size=batch_size,
    shuffle=False,
)
model.evaluate(test_data, verbose=1)

def check_similarity(sentence1, sentence2):
    sentence_pairs = np.array([[str(sentence1), str(sentence2)]])
    test_data = BertSemanticDataGenerator(
        sentence_pairs, labels=None, batch_size=1, shuffle=False, include_tar
    )

    proba = model.predict(test_data[0])[0]
    idx = np.argmax(proba)
    proba = f"{proba[idx]: .2f}%"
    pred = labels[idx]
    return pred, proba

sentence1 = "On the island where each inhabitant is either a knave or a knigh
sentence2 = "Is Rex the knight ?"
check_similarity(sentence1, sentence2)
```

A.2 Model 2

```
!pip install transformers
!pip install sentencepiece

import numpy as np
import pandas as pd
import tensorflow as tf
import transformers

max_length = 128  # Maximum length of input sentence to the model.
batch_size = 32
epochs = 5

# Labels in our dataset.
labels = ["NOT_ENTAILMENT - Unknown", "Entailment", "NOT_ENTAILMENT - Contra
```

```
PATH = '/content/drive/MyDrive/Colab Notebooks/ambiguous_knights_knaves.json'

import json
def read_jsonl_file(PATH):
  df = pd.read_json(PATH)

  with open(PATH, 'r') as f:
    data = json.loads(f.read())

  df_nested_list = pd.json_normalize(data=data['puzzles'], record_path='QA',
  return df_nested_list

df = pd.DataFrame(read_jsonl_file(PATH))

# SE IMPARTE DATASET-UL IN TRAIN, VALIDARE SI TEST
train_df = df[['qid','puzzle_text', 'question', 'answer']][:700]
valid_df  = df[['qid','puzzle_text', 'question', 'answer']]
test_df  = df[['qid','puzzle_text', 'question', 'answer']]

# Shape of the data
print(f"Total train samples : {train_df.shape[0]}")
print(f"Total validation samples: {valid_df.shape[0]}")
print(f"Total test samples: {test_df.shape[0]}")

print("Train Target Distribution")
print(train_df.answer.value_counts())

print("Validation Target Distribution")
print(valid_df.answer.value_counts())

train_df = (
    train_df[train_df.answer != "-"]
    .sample(frac=1.0, random_state=42)
    .reset_index(drop=True)
)
valid_df = (
    valid_df[valid_df.answer != "-"]
    .sample(frac=1.0, random_state=42)
    .reset_index(drop=True)
)

train_df["label"] = train_df["answer"].apply(
    lambda x: 0 if x == "NOT_ENTAILMENT - Contradiction" else 1 if x == "Enta
)
y_train = tf.keras.utils.to_categorical(train_df.label, num_classes=3)

valid_df["label"] = valid_df["answer"].apply(
    lambda x: 0 if x == "NOT_ENTAILMENT - Contradiction" else 1 if x == "Enta
)
y_val = tf.keras.utils.to_categorical(valid_df.label, num_classes=3)
```

```python
test_df["label"] = test_df["answer"].apply(
    lambda x: 0 if x == "NOT_ENTAILMENT - Contradiction" else 1 if x == "Enta
)
y_test = tf.keras.utils.to_categorical(test_df.label, num_classes=3)

class BertSemanticDataGenerator(tf.keras.utils.Sequence):
    """Generates batches of data.

    Args:
        sentence_pairs: Array of premise and hypothesis input sentences.
        labels: Array of labels.
        batch_size: Integer batch size.
        shuffle: boolean, whether to shuffle the data.
        include_targets: boolean, whether to incude the labels.

    Returns:
        Tuples `([input_ids, attention_mask, `token_type_ids], labels)`
        (or just `[input_ids, attention_mask, `token_type_ids]`
         if `include_targets=False`)
    """

    def __init__(
        self,
        sentence_pairs,
        labels,
        batch_size=batch_size,
        shuffle=True,
        include_targets=True,
    ):
        self.sentence_pairs = sentence_pairs
        self.labels = labels
        self.shuffle = shuffle
        self.batch_size = batch_size
        self.include_targets = include_targets
        # Load our BERT Tokenizer to encode the text.
        # We will use base-base-uncased pretrained model.
        self.tokenizer = transformers.BertTokenizer.from_pretrained(
            "bert-base-uncased", do_lower_case=True
        )
        self.indexes = np.arange(len(self.sentence_pairs))
        self.on_epoch_end()

    def __len__(self):
        # Denotes the number of batches per epoch.
        return len(self.sentence_pairs) // self.batch_size

    def __getitem__(self, idx):
        # Retrieves the batch of index.
        indexes = self.indexes[idx * self.batch_size : (idx + 1) * self.batc
```

```
        sentence_pairs = self.sentence_pairs[indexes]

        # With BERT tokenizer's batch_encode_plus batch of both the sentences
        # encoded together and separated by [SEP] token.
        encoded = self.tokenizer.batch_encode_plus(
            sentence_pairs.tolist(),
            add_special_tokens=True,
            max_length=max_length,
            return_attention_mask=True,
            return_token_type_ids=True,
            pad_to_max_length=True,
            return_tensors="tf",
        )

        # Convert batch of encoded features to numpy array.
        input_ids = np.array(encoded["input_ids"], dtype="int32")
        attention_masks = np.array(encoded["attention_mask"], dtype="int32")
        token_type_ids = np.array(encoded["token_type_ids"], dtype="int32")

        # Set to true if data generator is used for training/validation.
        if self.include_targets:
            labels = np.array(self.labels[indexes], dtype="int32")
            return [input_ids, attention_masks, token_type_ids], labels
        else:
            return [input_ids, attention_masks, token_type_ids]

    def on_epoch_end(self):
        # Shuffle indexes after each epoch if shuffle is set to True.
        if self.shuffle:
            np.random.RandomState(42).shuffle(self.indexes)

# Create the model under a distribution strategy scope.
strategy = tf.distribute.MirroredStrategy()

with strategy.scope():
    # Encoded token ids from BERT tokenizer.
    input_ids = tf.keras.layers.Input(
        shape=(max_length,), dtype=tf.int32, name="input_ids"
    )
    # Attention masks indicates to the model which tokens should be attended
    attention_masks = tf.keras.layers.Input(
        shape=(max_length,), dtype=tf.int32, name="attention_masks"
    )
    # Token type ids are binary masks identifying different sequences in the
    token_type_ids = tf.keras.layers.Input(
        shape=(max_length,), dtype=tf.int32, name="token_type_ids"
    )
    # Loading pretrained BERT model.
    bert_model = transformers.TFBertModel.from_pretrained("bert-base-uncased"
    # Freeze the BERT model to reuse the pretrained features without modifyin
```

```
    bert_model.trainable = False

    bert_output = bert_model(
        input_ids, attention_mask=attention_masks, token_type_ids=token_type_
    )
    sequence_output = bert_output.last_hidden_state
    pooled_output = bert_output.pooler_output
    # Add trainable layers on top of frozen layers to adapt the pretrained fe
    bi_lstm = tf.keras.layers.Bidirectional(
        tf.keras.layers.LSTM(64, return_sequences=True)
    )(sequence_output)
    # Applying hybrid pooling approach to bi_lstm sequence output.
    avg_pool = tf.keras.layers.GlobalAveragePooling1D()(bi_lstm)
    max_pool = tf.keras.layers.GlobalMaxPooling1D()(bi_lstm)
    concat = tf.keras.layers.concatenate([avg_pool, max_pool])
    dropout = tf.keras.layers.Dropout(0.3)(concat)
    avg_pool_2 = tf.keras.layers.GlobalAveragePooling1D()(bi_lstm)
    max_pool_2 = tf.keras.layers.GlobalMaxPooling1D()(bi_lstm)
    concat2 = tf.keras.layers.concatenate([avg_pool_2, max_pool_2])
    dropout2 = tf.keras.layers.Dropout(0.3)(concat2)
    dense1 = tf.keras.layers.Dense(1, activation='relu')(dropout2)
    dense2 = tf.keras.layers.Dense(10, activation='sigmoid')(dense1)
    dense3 = tf.keras.layers.Dense(20, activation='sigmoid')(dense2)
    flatten = tf.keras.layers.Flatten()(dense3)
    output = tf.keras.layers.Dense(3, activation="softmax")(flatten)
    model2 = tf.keras.models.Model(
        inputs=[input_ids, attention_masks, token_type_ids], outputs=output
    )

    model2.compile(
        optimizer=tf.keras.optimizers.Adam(),
        loss="binary_crossentropy",        # am schimbat loss function
        metrics=["acc"],
    )


print(f"Strategy: {strategy}")
model2.summary()

train_data = BertSemanticDataGenerator(
    train_df[["puzzle_text", "question"]].values.astype("str"),
    y_train,
    batch_size=batch_size,
    shuffle=True,
)
valid_data = BertSemanticDataGenerator(
    valid_df[["puzzle_text", "question"]].values.astype("str"),
    y_val,
    batch_size=batch_size,
    shuffle=False,
```

```
)

history = model2.fit(
    train_data,
    validation_data=valid_data,
    epochs=epochs,
    use_multiprocessing=True,
    workers=-1,
)

# Unfreeze the bert_model.
bert_model.trainable = True
# Recompile the model to make the change effective.
model2.compile(
    optimizer=tf.keras.optimizers.Adam(1e-5),
    loss="binary_crossentropy",
    metrics=["accuracy"],
)
model2.summary()

history = model2.fit(
    train_data,
    validation_data=valid_data,
    epochs=epochs,
    use_multiprocessing=True,
    workers=-1,
)

test_data = BertSemanticDataGenerator(
    test_df[["puzzle_text", "question"]].values.astype("str"),
    y_test,
    batch_size=batch_size,
    shuffle=False,
)
model2.evaluate(test_data, verbose=1)

def check_similarity(sentence1, sentence2):
    sentence_pairs = np.array([[str(sentence1), str(sentence2)]])
    test_data = BertSemanticDataGenerator(
        sentence_pairs, labels=None, batch_size=1, shuffle=False, include_tar
    )

    proba = model2.predict(test_data[0])[0]
    idx = np.argmax(proba)
    proba = f"{proba[idx]: .2f}%"
    pred = labels[idx]
    return pred, proba

sentence1 = "On the island where each inhabitant is either a knave or a knigh
sentence2 = "Is Rex the knight ?"
```

```
check_similarity(sentence1, sentence2)
```

A.3 Model 3

```
!pip install transformers
!pip install sentencepiece

import numpy as np
import pandas as pd
import tensorflow as tf
import transformers

max_length = 128  # Maximum length of input sentence to the model.
batch_size = 32
epochs = 10

# Labels in our dataset.
labels = ["NOT_ENTAILMENT - Unknown", "Entailment", "NOT_ENTAILMENT - Contra

PATH = '/content/drive/MyDrive/Colab Notebooks/ambiguous_knights_knaves.json'

import json
def read_jsonl_file(PATH):
  df = pd.read_json(PATH)

  with open(PATH, 'r') as f:
    data = json.loads(f.read())

  df_nested_list = pd.json_normalize(data=data['puzzles'], record_path='QA',
  return df_nested_list

df = pd.DataFrame(read_jsonl_file(PATH))

# SE IMPARTE DATASET-UL IN TRAIN, VALIDARE SI TEST
train_df = df[['qid','puzzle_text', 'question', 'answer']][:700]
valid_df  = df[['qid','puzzle_text', 'question', 'answer']]
test_df  = df[['qid','puzzle_text', 'question', 'answer']]

# Shape of the data
print(f"Total train samples : {train_df.shape[0]}")
print(f"Total validation samples: {valid_df.shape[0]}")
print(f"Total test samples: {test_df.shape[0]}")

print("Train Target Distribution")
print(train_df.answer.value_counts())

print("Validation Target Distribution")
print(valid_df.answer.value_counts())
```

```
train_df = (
    train_df[train_df.answer != "-"]
    .sample(frac=1.0, random_state=42)
    .reset_index(drop=True)
)
valid_df = (
    valid_df[valid_df.answer != "-"]
    .sample(frac=1.0, random_state=42)
    .reset_index(drop=True)
)

train_df["label"] = train_df["answer"].apply(
    lambda x: 0 if x == "NOT_ENTAILMENT - Contradiction" else 1 if x == "Enta
)
y_train = tf.keras.utils.to_categorical(train_df.label, num_classes=3)

valid_df["label"] = valid_df["answer"].apply(
    lambda x: 0 if x == "NOT_ENTAILMENT - Contradiction" else 1 if x == "Enta
)
y_val = tf.keras.utils.to_categorical(valid_df.label, num_classes=3)

test_df["label"] = test_df["answer"].apply(
    lambda x: 0 if x == "NOT_ENTAILMENT - Contradiction" else 1 if x == "Enta
)
y_test = tf.keras.utils.to_categorical(test_df.label, num_classes=3)

class BertSemanticDataGenerator(tf.keras.utils.Sequence):
    """Generates batches of data.

    Args:
        sentence_pairs: Array of premise and hypothesis input sentences.
        labels: Array of labels.
        batch_size: Integer batch size.
        shuffle: boolean, whether to shuffle the data.
        include_targets: boolean, whether to incude the labels.

    Returns:
        Tuples `([input_ids, attention_mask, `token_type_ids], labels)`
        (or just `[input_ids, attention_mask, `token_type_ids]`
         if `include_targets=False`)
    """

    def __init__(
        self,
        sentence_pairs,
        labels,
        batch_size=batch_size,
        shuffle=True,
        include_targets=True,
```

```
    ):
        self.sentence_pairs = sentence_pairs
        self.labels = labels
        self.shuffle = shuffle
        self.batch_size = batch_size
        self.include_targets = include_targets
        # Load our BERT Tokenizer to encode the text.
        # We will use base-base-uncased pretrained model.
        self.tokenizer = transformers.BertTokenizer.from_pretrained(
            "bert-base-uncased", do_lower_case=True
        )
        self.indexes = np.arange(len(self.sentence_pairs))
        self.on_epoch_end()

    def __len__(self):
        # Denotes the number of batches per epoch.
        return len(self.sentence_pairs) // self.batch_size

    def __getitem__(self, idx):
        # Retrieves the batch of index.
        indexes = self.indexes[idx * self.batch_size : (idx + 1) * self.batcl
        sentence_pairs = self.sentence_pairs[indexes]

        # With BERT tokenizer's batch_encode_plus batch of both the sentences
        # encoded together and separated by [SEP] token.
        encoded = self.tokenizer.batch_encode_plus(
            sentence_pairs.tolist(),
            add_special_tokens=True,
            max_length=max_length,
            return_attention_mask=True,
            return_token_type_ids=True,
            pad_to_max_length=True,
            return_tensors="tf",
        )

        # Convert batch of encoded features to numpy array.
        input_ids = np.array(encoded["input_ids"], dtype="int32")
        attention_masks = np.array(encoded["attention_mask"], dtype="int32")
        token_type_ids = np.array(encoded["token_type_ids"], dtype="int32")

        # Set to true if data generator is used for training/validation.
        if self.include_targets:
            labels = np.array(self.labels[indexes], dtype="int32")
            return [input_ids, attention_masks, token_type_ids], labels
        else:
            return [input_ids, attention_masks, token_type_ids]

    def on_epoch_end(self):
        # Shuffle indexes after each epoch if shuffle is set to True.
        if self.shuffle:
```

```
            np.random.RandomState(42).shuffle(self.indexes)

# Create the model under a distribution strategy scope.
strategy = tf.distribute.MirroredStrategy()

with strategy.scope():
    # Encoded token ids from BERT tokenizer.
    input_ids = tf.keras.layers.Input(
        shape=(max_length,), dtype=tf.int32, name="input_ids"
    )
    # Attention masks indicates to the model which tokens should be attended
    attention_masks = tf.keras.layers.Input(
        shape=(max_length,), dtype=tf.int32, name="attention_masks"
    )
    # Token type ids are binary masks identifying different sequences in the
    token_type_ids = tf.keras.layers.Input(
        shape=(max_length,), dtype=tf.int32, name="token_type_ids"
    )
    # Loading pretrained BERT model.
    bert_model = transformers.TFBertModel.from_pretrained("bert-base-uncased"
    # Freeze the BERT model to reuse the pretrained features without modifyin
    bert_model.trainable = False

    bert_output = bert_model(
        input_ids, attention_mask=attention_masks, token_type_ids=token_type_
    )
    sequence_output = bert_output.last_hidden_state
    pooled_output = bert_output.pooler_output
    # Add trainable layers on top of frozen layers to adapt the pretrained fe
    bi_lstm = tf.keras.layers.Bidirectional(
        tf.keras.layers.LSTM(64, return_sequences=True)
    )(sequence_output)
    # Applying hybrid pooling approach to bi_lstm sequence output.
    max_pooling = tf.keras.layers.MaxPooling1D(2)(bi_lstm)
    dense1 = tf.keras.layers.Dense(1, activation='relu')(max_pooling)
    conv1 = tf.keras.layers.Conv1D(filters=32, kernel_size=3, padding='same',
    dense2 = tf.keras.layers.Dense(10, activation='sigmoid')(conv1)
    conv2 = tf.keras.layers.Conv1D(filters=64, kernel_size=5, padding='same',
    conv3 = tf.keras.layers.Conv1D(filters=32, kernel_size=5, padding='same',
    conv4 = tf.keras.layers.Conv1D(filters=16, kernel_size=4, padding='same',
    dense3 = tf.keras.layers.Dense(20, activation='sigmoid')(conv4)
    conv5 = tf.keras.layers.Conv1D(filters=8, kernel_size=3, padding='same',
    conv6 = tf.keras.layers.Conv1D(filters=4, kernel_size=3, padding='same',
    dense4 = tf.keras.layers.Dense(20, activation='sigmoid')(conv6)
    flatten = tf.keras.layers.Flatten()(dense4)
    output = tf.keras.layers.Dense(3, activation="softmax")(flatten)
    model2 = tf.keras.models.Model(
        inputs=[input_ids, attention_masks, token_type_ids], outputs=output
    )
```

```python
    model2.compile(
        optimizer=tf.keras.optimizers.Adam(),
        loss="binary_crossentropy",        # am schimbat loss function
        metrics=["acc"],
    )


print(f"Strategy: {strategy}")
model2.summary()

train_data = BertSemanticDataGenerator(
    train_df[["puzzle_text", "question"]].values.astype("str"),
    y_train,
    batch_size=batch_size,
    shuffle=True,
)
valid_data = BertSemanticDataGenerator(
    valid_df[["puzzle_text", "question"]].values.astype("str"),
    y_val,
    batch_size=batch_size,
    shuffle=False,
)

history = model2.fit(
    train_data,
    validation_data=valid_data,
    epochs=epochs,
    use_multiprocessing=True,
    workers=-1,
)

# Unfreeze the bert_model.
bert_model.trainable = True
# Recompile the model to make the change effective.
model2.compile(
    optimizer=tf.keras.optimizers.Adam(1e-5),
    loss="binary_crossentropy",
    metrics=["accuracy"],
)
model2.summary()

history = model2.fit(
    train_data,
    validation_data=valid_data,
    epochs=epochs,
    use_multiprocessing=True,
    workers=-1,
)

test_data = BertSemanticDataGenerator(
```

```
    test_df[["puzzle_text", "question"]].values.astype("str"),
    y_test,
    batch_size=batch_size,
    shuffle=False,
)
model2.evaluate(test_data, verbose=1)

def check_similarity(sentence1, sentence2):
    sentence_pairs = np.array([[str(sentence1), str(sentence2)]])
    test_data = BertSemanticDataGenerator(
        sentence_pairs, labels=None, batch_size=1, shuffle=False, include_tar
    )

    proba = model2.predict(test_data[0])[0]
    idx = np.argmax(proba)
    proba = f"{proba[idx]: .2f}%"
    pred = labels[idx]
    return pred, proba

sentence1 = "On the island where each inhabitant is either a knave or a knigh
sentence2 = "Is Rex the knight ?"
check_similarity(sentence1, sentence2)
```

Bibliography

https://keras.io/api/layers/
https://huggingface.co/docs/transformers/model_doc/bert

Intelligent Systems Group