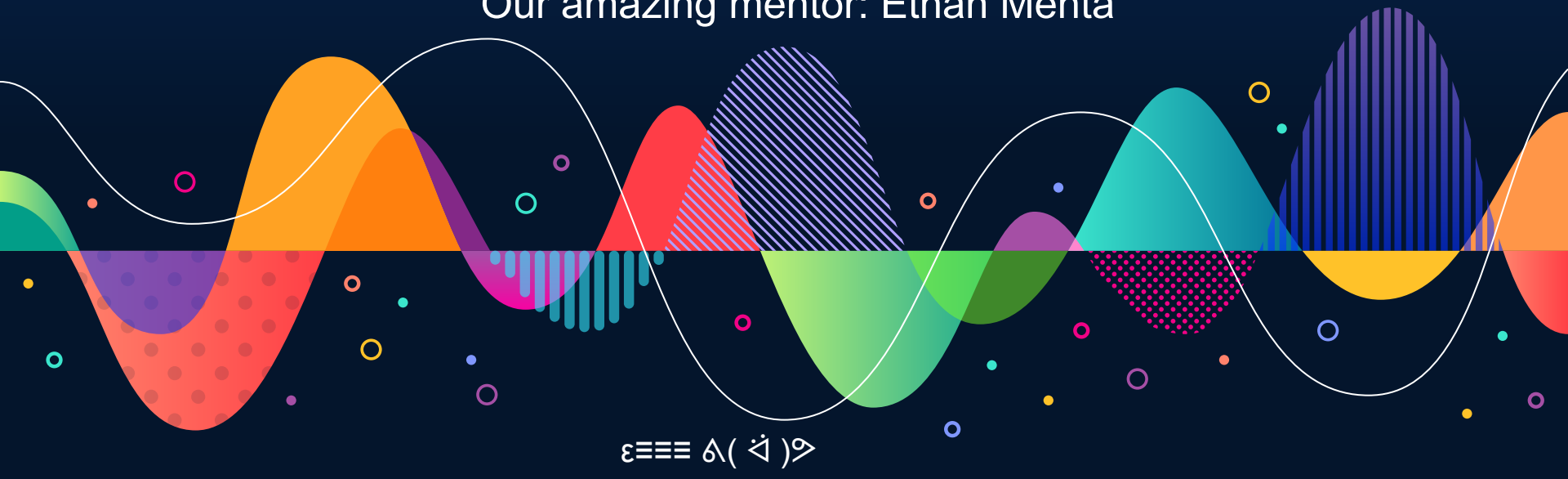
Understanding Siri

By: Jessica Li, Sai A. Dhanasiri, Isaac Serrato
Our amazing mentor: Ethan Mehta



What is natural language processing?

How is it used?

What is NLP?

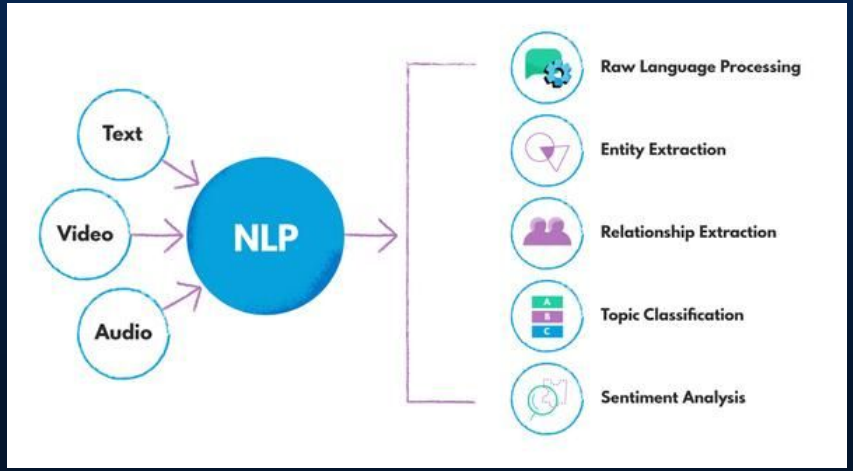
Gives the computer the ability to understand language

Subfield of AI

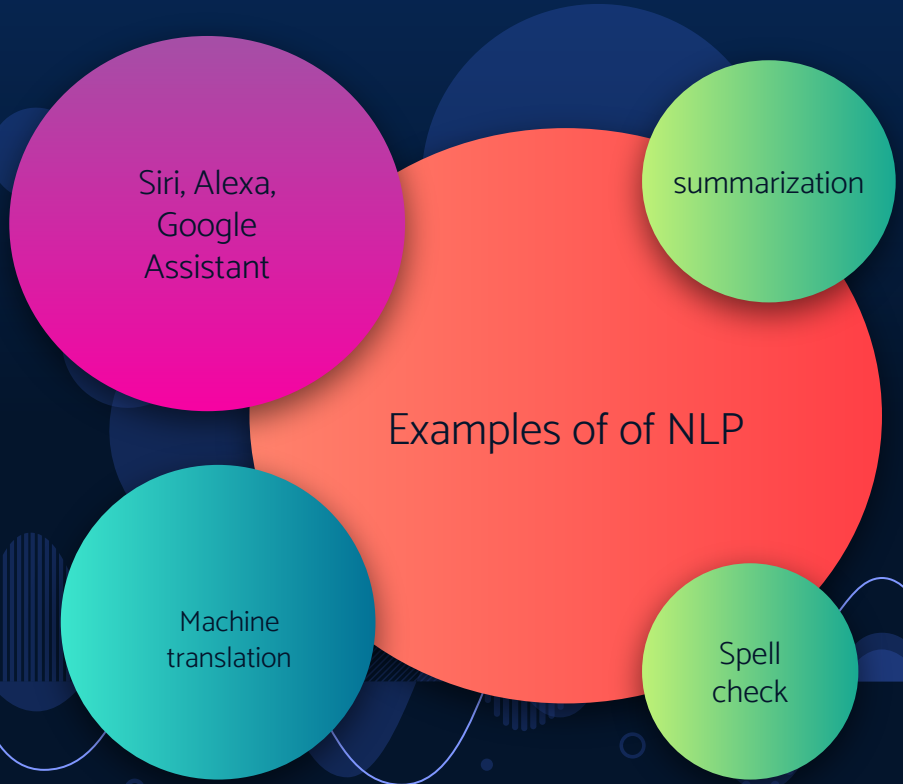
What is Natural Language Processing?

Combination of computational linguistics and ML models

Processing of language



How is it used? (examples of NLP)



Named Entity Recognition

In the 19th century, there was something called the "cult of domesticity" for many American women. This meant that most married women were expected to stay in the home and raise children. As in other countries, American wives were very much under the control of their husband, and had almost no rights. Women who were not married had only a few jobs open to them, such as working in clothing factories and serving as maids. By the 19th century, women such as Lucretia Mott and Elizabeth Cady Stanton thought that women should have more rights. In 1848, many of these women met and agreed to fight for more rights for women, including voting. Many of the women involved in the movement for women's rights were also involved in the movement to end slavery.

WIKIPEDIA

Tag colors:

LOCATION PERSON TERM DATE CONDITION PROCESS PEOPLE

mobidev

Introduction: How has NLP made an impact?

Impact 1 : Online customer service

Impact 2: Removing hate speech

Impact 3: Therapy

BIG CONCEPT

Bring the attention of your audience over a key concept using icons or illustrations

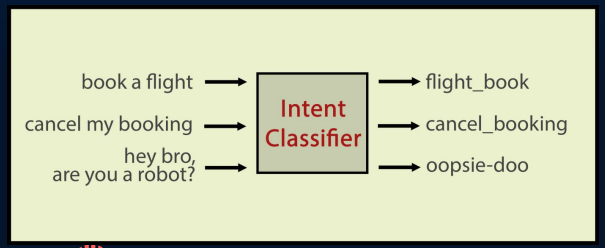
Introduction: Our project

A) Predicting the intent of the speaker

Intent Classification

“What the person wants”

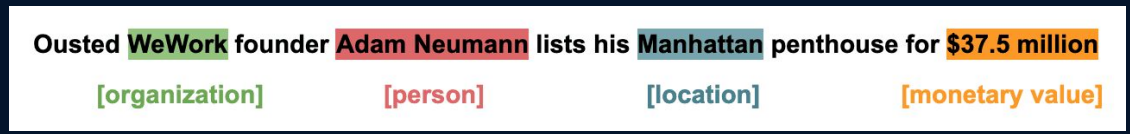
“Requests”



B) Extracting interesting named entities within the command

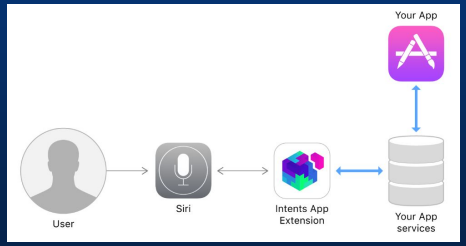
Named Entity Recognition (NER)

“Identifies key elements”



Introduction: Our project

How is this used in Siri?



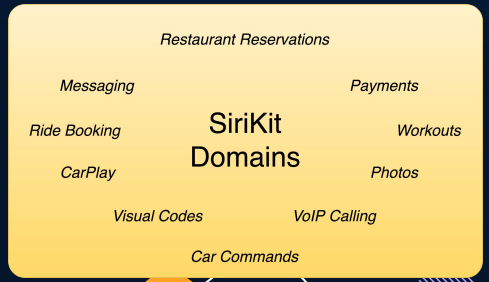
Use of NER:

- Used to process data
- Helps machine understand the subject of the given task

> *Query: I'm looking for a pack of horizon chocolate milk*
>
> *Intent:* Search
>
> *Entities:* pack <*unit*>, horizon <*brand*>, chocolate milk <product>

Use of Intent classification:

- Used to identify the users request



Our Data: The datasets

Out intent categories:

```
['AddToPlaylist',
 'BookRestaurant',
 'GetWeather',
 'PlayMusic',
 'RateBook',
 'SearchCreativeWork',
 'SearchScreeningEvent']
```

Explains what our model will be able to understand

Used for intent classification

Our Entities:

```
{'B-album': 1,
 'B-artist': 2,
 'B-best_rating': 3,
 'B-city': 4,
 'B-condition_description': 5,
 'B-condition_temperature': 6,
 'B-country': 7,
 'B-cuisine': 8,
 'B-current_location': 9,
 'B-entity_name': 10,
 'B-facility': 11,
 'B-genre': 12,
 'B-geographic_poi': 13,
 'B-location_name': 14,
 'B-movie_name': 15,
 'B-movie_type': 16,
 'B-music_item': 17,
 'B-object_location_type': 18,
 'B-object_name': 19,
 'B-object_part_of_series_type': 20,
 'B-object_select': 21,
 'B-object_type': 22,
 'B-party_size_description': 23,
 'B-party_size_number': 24,
 'B-playlist': 25,
 'B-playlist_owner': 26,
 'B-poi': 27,
 'B-rating_unit': 28,
 'B-rating_value': 29,
 'B-restaurant_name': 30,
 'B-restaurant_type': 31,
 'B-served_dish': 32,
 'B-service': 33,
 'B-sort': 34,
 'B-spatial_relation': 35,
 'B-state': 36,
 'B-timeRange': 37,
 'B-track': 38,
 'B-year': 39,
```

Resembles the key elements in the query

Used for NER

Examples of the data in action:

```
[27]  1 nlu("I would like to add taylor swift to my playlist  ", intent_names, slot_names)

{'intent': 'AddToPlaylist',
 'slots': {'artist': 'taylor swift', 'playlist_owner': 'my'}}

[26]  1 nlu(" I would like to eat at Le Ritz at 7 ", intent_names, slot_names)

{'intent': 'BookRestaurant',
 'slots': {'restaurant_name': 'Le Ritz', 'timeRange': '7'}}

[25]  1 nlu("Is it going to be hot today, I am cold  ", intent_names, slot_names)

{'intent': 'GetWeather',
 'slots': {'condition_temperature': 'cold', 'timeRange': 'today,'}}

[24]  1 nlu("Hey do you think you could play this awesome song by Drake ", intent_names, slot_names)

{'intent': 'PlayMusic',
 'slots': {'artist': 'Drake', 'music_item': 'song', 'sort': 'awesome'}}

  1 nlu("Honestly, I would definetly rate this book at 10. ", intent_names, slot_names)

{'intent': 'RateBook',
 'slots': {'object_select': 'this',
  'object_type': 'book',
  'rating_value': '10.'}}

[30]  1 nlu("Search a poem ", intent_names, slot_names)

{'intent': 'SearchCreativeWork', 'slots': {'object_type': 'poem'}}
```

```
1 nlu("What time is Spiderman? ", intent_names, slot_names)

{'intent': 'SearchScreeningEvent', 'slots': {'object_name': 'Spiderman?'}}
```

Methods Used (BERT) – Sai

- Introduce BERT
- How it learns from context, etc? (Reference the BERT slides in NB 2)
- Input to BERT (sequence or a sentence tokenized)
- Output to BERT (pooled output that we can use for sentence level classification (ie the intent classification) and tokens output (token/BERT embeddings) that we can use for Named Entity Recognition)

What is bert?

Bidirectional Encoder Representations from Transformers

A technique for NLP pre-training developed by Google and Published in 2018

Novel model architecture that uses both the left and right context words in a sequence to generate good embeddings for words.

Applications of Bert

Text classification and insights from customer reviews

Google Search Queries

Sentiment Analysis

Q&A Chatbot

Named Entity Recognition

Named Entity Recognition

You: Siri book a restaurant for 3 people tomorrow at Buffalo Wild Wings.

Without Sentence Level Classification (BERT):

chirp *chirp*

Siri: I do not understand

With Sentence Level Classification (BERT):

Intent: Book a restaurant

Number of people: 3

Time range: Tomorrow:

Location: Buffalo wild wings

Siri: Okay Done!

3 steps in the Machine Learning pipeline

Data Pre Processing

Build and Train Model

Prediction & Testing

Step 1: Data Pre-Processing

Tokenizing our Data

Padding Tokenized Sequences

Encoding our token

Tokenizing our Data

BERT expects input sequences to be broken down into individual *tokens*; a [CLS] token marks the beginning and a [SEP] marks the end of a given sequence (as seen in the red input sentences in the image above).

The [CLS] token is used by BERT for the pre-training task for sequence classification.

The [SEP] token is a separator for the pre-training task that classifies if a pair of sentences are consecutive in a corpus or not, i.e. next sentence prediction (NSP).

```
# Import the BERT tokenizer and utilize the BERT base pretrained model
from transformers import BertTokenizer

model_name = "bert-base-cased"
tokenizer = BertTokenizer.from_pretrained(model_name)
```

```
Downloading vocab.txt: 100% 208k/208k [00:00<00:00, 1.55MB/s]
Downloading tokenizer_config.json: 100% 29.0/29.0 [00:00<00:00, 876B/s]
Downloading config.json: 100% 570/570 [00:00<00:00, 14.7kB/s]
```

Wow - there's roughly 30k tokens that BERT knows!

Let's see exactly how the BERT tokenizer works on one training example.

```
# Print the first training data sequence
first_sentence = df_train.iloc[0]["words"]
print(first_sentence)
print("This sentence contains {} words.".format(len(first_sentence.split(" "))))
```

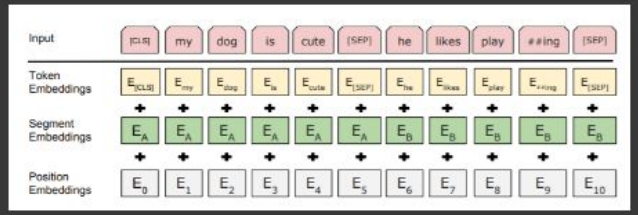
```
Add Don and Sherri to my Meditate to Sounds of Nature playlist
This sentence contains 12 words.
```

```
tokens = tokenizer.tokenize(first_sentence)
for t in tokens:
  print(t)
print("\nThis sentence turns into {} tokens.".format(len(tokens)))
```

```
Ad
##d
Don
and
She
##rri
to
my
Me
##dit
##ate
to
Sounds
of
Nature
play
##list

This sentence turns into 17 tokens.
```

Encoding Our token




```
[ ]  1 # See the first 10 (token,id) pairs within BERT.
     2 bert_vocab_items = list(tokenizer.vocab.items())
     3 bert_vocab_items[:10]
```

```
[('[PAD]', 0),
 ('[unused1]', 1),
 ('[unused2]', 2),
 ('[unused3]', 3),
 ('[unused4]', 4),
 ('[unused5]', 5),
 ('[unused6]', 6),
 ('[unused7]', 7),
 ('[unused8]', 8),
 ('[unused9]', 9)]
```

```
[ ]  1 # See more (token,id) examples within BERT's vocab.
     2 bert_vocab_items[1100:1110]
```

```
[('-', 1100),
 ('/', 1101),
 (':', 1102),
 ('the', 1103),
 ('of', 1104),
 ('and', 1105),
 ('to', 1106),
 ('in', 1107),
 ('was', 1108),
 ('The', 1109)]
```

This is done because… each token is mapped to a unique integer id that makes it fast to lookup the correct column in the input layer token embedding.

So if our first sentence is along the lines of:
Add don and Sherri to my Meditate to Sounds of Nature Playlist

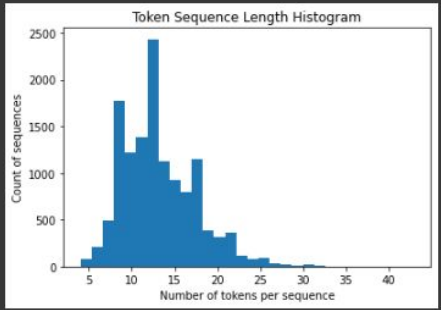
Then the encoded version would be:

```
[101, 24930, 1181, 1790, 1105,
1153, 14791, 1106, 1139, 2508,
17903, 2193, 1106, 10560,
1104, 7009, 1505, 7276, 102]
```

Padding Tokenized Sequences

In the Training Data Analysis section of the 1st notebook, we created a histogram according to the lengths of training sequences. Let's take a look at the histogram of the **tokenized** sequences now.

```
[ ]  1 # Print histogram of training tokenized sequence lengths.
     2 # Hint: use tokenizer.encode on every sequence within df_train
     3 # & get the lengths; then use plt.hist()
     4 train_sequence_lengths = [len(tokenizer.encode(text))
     5                           for text in df_train["words"]]
     6 plt.hist(train_sequence_lengths, bins = 30)
     7 plt.xlabel("Number of tokens per sequence")
     8 plt.ylabel("Count of sequences")
     9 plt.title("Token Sequence Length Histogram")
    10 plt.show()
```



Using *pre trained* BERT requires that the sequences are **padded**, which means they all have the same length.

The histogram shows that after tokenization, 43 tokens is long enough to represent all the voice commands in the training set.

- Hi, my name is Andrew: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 43, 3, 56, 6]
- Hi, I'm an analyst and I use Tensorflow for my deep learning projects: [43, 11, 9, 34, 2, 22, 15, 4, 5, 8, 19, 10, 26, 27]

```
[ ]  1 max_token_len = max(train_sequence_lengths)
     2 print("Maximum tokenized sequence length: {} tokens per sequence".format(max_token_len))
```

```
Maximum tokenized sequence length: 43 tokens per sequence
```

Encoding the Sequence Classification Targets

Encoding the Sequence Classification Targets

Aside from encodings of the tokenized word sequences, we also need to get the encodings of the target intent classification labels.

```
[ ]  1 # Build a map from target intent label to a unique id.
     2 intent_names = Path("vocab.intent").read_text().split()
     3 intent_map = dict((label, idx) for idx, label in enumerate(intent_names))
     4 intent_map
```

```
{'AddToPlaylist': 0,
 'BookRestaurant': 1,
 'GetWeather': 2,
 'PlayMusic': 3,
 'RateBook': 4,
 'SearchCreativeWork': 5,
 'SearchScreeningEvent': 6}
```

```
[ ]  1 # Convert list of target labels into their corresponding unique id.
     2 intent_train = df_train["intent_label"].map(intent_map).values
     3 intent_train
```

```
array([0, 0, 0, ..., 6, 6, 6])
```

Step 2: Building and Training

Using a Pretrained BERT model

Yay - we have all the pre-processing steps done! Now it's time to explore pretrained BERT.

```
from transformers import TFBertModel

# recall that we earlier defined model_name to be "bert-base-cased"
base_bert_model = TFBertModel.from_pretrained(model_name)
base_bert_model.summary()
```

```
Downloading tf_model.h5: 100% 502M/502M [00:09<00:00, 59.1MB/s]
Some layers from the model checkpoint at bert-base-cased were not used when initializing TFBertModel: ['mlm___cls', 'nsp___cls']
- This IS expected if you are initializing TFBertModel 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 TFBertModel from the checkpoint of a model that you expect to be exactly identical (initializing a BertForSequenceClassification model from a BertForSequenceClassification model).
All the layers of TFBertModel were initialized from the model checkpoint at bert-base-cased.
If your task is similar to the task the model of the checkpoint was trained on, you can already use TFBertModel for predictions without further training.
Model: "tf_bert_model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 bert (TFBertMainLayer)      multiple                  108310272

=================================================================
Total params: 108,310,272
Trainable params: 108,310,272
Non-trainable params: 0
_________________________________________________________________
```

Outputs of BERT

We can see that there are two outputs from the BERT model.

1. The **first** output of the BERT model is a tensor with shape: `(batch_size, seq_len, output_dim)` which computes **features for each token in the input sequence**

```
[35]  1 token_features = outputs[0]
      2 token_features.shape
```

```
TensorShape([700, 43, 768])
```

Note that `seq_len` is `max_token_len` (ie. 43).

2. The **second** output of the BERT model is a tensor with shape: `(batch_size, output_dim)` which is the feature vector of `[CLS]`. This vector is typically used as a **pooled representation for the sequence as a whole**. We can use this as the features of our intent classifier

```
[36]  1 sentence_representation = outputs[1]
      2 sentence_representation.shape
```

```
TensorShape([700, 768])
```

Remember that our goal is to use BERT to compute some **representation** of a single voice command at a time.

We have two options to obtain this sentence-level representation, which is used as the input for the final sequence classification layer:

1. We can reuse the representation of the `[CLS]` token.

or

2. We can pool the representations (encodings) of all the tokens within the sequence (i.e. global average).

Remember that our goal is to use BERT to compute some **representation** of a single voice command at a time.

We have two options to obtain this sentence-level representation, which is used as the input for the final sequence classification layer:

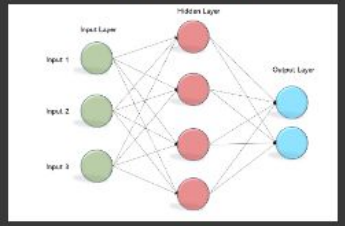
1. We can reuse the representation of the `[CLS]` token.

or

2. We can pool the representations (encodings) of all the tokens within the sequence (i.e. global average).

Pooled Representation

The pooled output comes from the application of a small multi-layer-perception (MLP) layer called the "Pooler" which is applied to the output representation of the special `[CLS]` token.




```
[37]  1 base_bert_model.bert.pooler.dense
```

```
<keras.layers.core.dense.Dense at 0x7ff35016e490>
```

```
[38]  1 # Extract the features across all batches for the 0th ([CLS]) token
      2 first_token_states = token_features[:, 0]
      3 pooled_outputs = base_bert_model.bert.pooler.dense(first_token_states)
      4 pooled_outputs.shape
```

```
TensorShape([700, 768])
```

Now let's double-check that taking this route of extracting all of the token features for `[CLS]` and passing it through the "Pooler" results in close to the same output as using simply the `sentence_representation`.

Take a look at the documentation for the `allclose` function used below if you're interested.

```
[39]  1 np.allclose(pooled_outputs, sentence_representation)
```

```
True
```

Training the model & Testing

```
[ ]
```

```python
import tensorflow as tf
from transformers import TFBertModel
from tensorflow.keras.layers import Dropout, Dense, GlobalAveragePooling1D
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import SparseCategoricalCrossentropy
from tensorflow.keras.metrics import SparseCategoricalAccuracy


class IntentClassificationModel(tf.keras.Model):

    def __init__(self, intent_num_labels=None,
                 dropout_prob=0.1):
        super().__init__(name="intent_classifier")

        # Load the pretrained BERT model in the constructor
        self.bert = base_bert_model

        # TODO: Specify the dropout
        self.dropout = Dropout(dropout_prob) ### YOUR CODE HERE ###

        # TODO: define a Dense classification layer which will compute
        # the intent for each sequence in a batch. The number of
        # output classes is given by the intent_num_labels parameter.
        # Use the default linear activation (no softmax) to compute logits.
        # The softmax normalization will be computed in the loss function
        # instead of the model.

        self.intent_classifier = Dense(intent_num_labels)  ### YOUR CODE HERE ###

    def call(self, inputs, **kwargs):
        # Use the pretrained model to extract features from our encoded inputs:
        tokens_output, pooled_output = self.bert(inputs, **kwargs, return_dict=False)

        # The second output of the main BERT layer has shape
        # (batch_size, output_dim) and gives the pooled representation
        # for the full sequence (from the hidden state corresponding to [CLS]).
        pooled_output = self.dropout(pooled_output, \
                                     training=kwargs.get("training", False))

        # TODO: use classifier layer to compute logits from pooled features.
        intent_logits = self.intent_classifier(pooled_output) ### YOUR CODE HERE ###

        return intent_logits

# TODO: create an instantiation of this class and pass in the correct
# parameter for intent_num_labels.
### YOUR CODE HERE ###
intent_model = IntentClassificationModel(intent_num_labels=len(intent_map))

intent_model.compile(optimizer=Adam(learning_rate=3e-5, epsilon=1e-08),
                     loss=SparseCategoricalCrossentropy(from_logits=True),
                     metrics=[SparseCategoricalAccuracy('accuracy')], run_eagerly=True)
```

```python
# Train the model.
# Note: this cell will take a bit of time to execute.
history = intent_model.fit(encoded_train["input_ids"], intent_train, epochs=1, batch_size=32, \
                           validation_data=(encoded_valid["input_ids"], intent_valid))
```

```
409/409 [==============================] - 219s 473ms/step - loss: 0.1768 - accuracy: 0.9439 - val_loss: 0.0426 - val_accuracy: 0.9914
```

Classification

The last step to making predictions is writing a `classify` function that will use the `tokenizer` (from Step 1: Tokenize our Data) and the `intent_model` (from Exercise 4).

The main point is choosing which of the 7 classes has the highest probability after the softmax.

```
[36]
```

```python
# Classification takes a voice command as input, as well as
# the 7 possible classes (intent_names), and uses the BERT
# tokenizer and the intent_model.
def classify(text, intent_names):
    inputs = tf.constant(tokenizer.encode(text))[None, :]  # batch_size = 1
    class_id = intent_model(inputs).numpy().argmax(axis=1)[0]
    return intent_names[class_id]
```

Let's see how our classify function works on some examples!

```
[37]
```

```python
classify("Book a table for two at La Tour d'Argent for Friday night.", intent_names)
```

```
'BookRestaurant'
```

```
[38]
```

```python
classify("I would like to listen to Anima by Thom Yorke.", intent_names)
```

```
'PlayMusic'
```

```
[39]
```

```python
classify("Will it snow tomorrow in Saclay?", intent_names)
```

```
'GetWeather'
```

```
[40]
```

```python
classify("Where can I see to the last Star Wars near Odéon tonight?", intent_names)
```

```
'SearchScreeningEvent'
```

```python
# Define the class for the model that will create predictions
# for the overall intent of a sequence, as well as the NER token labels.
class JointIntentAndSlotFillingModel(tf.keras.Model):

    def __init__(self, intent_num_labels=None, slot_num_labels=None, ## This defines the model structure
                 dropout_prob=0.1):
        super().__init__(name="joint_intent_slot")

        self.bert = base_bert_model

        # TODO: define the dropout, intent & slot classifier layers
        self.dropout = Dropout(dropout_prob)
        self.intent_classifier = Dense(intent_num_labels, name = "intent_classifier")
        self.slot_classifier = Dense(slot_num_labels, name = "slot_classifier")

    def call(self, inputs, **kwargs):
        # Extract features from the inputs using pre-trained BERT.
        # TODO: what does the bert model return?
        tokens_output,pooled_output = self.bert(inputs, **kwargs, return_dict=False)

        # TODO: use the new layers to predict slot class (logits) for each
        # token position in input sequence (size: (batch_size, seq_len, slot_num_labels)).
        tokens_output = self.dropout(tokens_output, training = kwargs.get("training",False)) ## dropout
        slot_logits = self.slot_classifier(tokens_output) ## Slot

        # TODO: define a second classification head for the sequence-wise
        # predictions (size: (batch_size, intent_num_labels)).
        # (Hint: create pooled_output to get the intent_logits).
        # Remember that the 2nd output of the main BERT layer is size
        # (batch_size, output_dim) & gives a "pooled" representation for
        # full sequence from hidden state corresponding to [CLS]).
        pooled_output = self.dropout(pooled_output, training = kwargs.get("training",False)) ## dropout
        intent_logits = self.intent_classifier(pooled_output) ##intent

        return slot_logits, intent_logits

# TODO: create an instantiation of this model
joint_model = JointIntentAndSlotFillingModel(intent_num_labels=len(intent_map), slot_num_labels=len(slot_map))
```

This model not only find the intent of the end user, but it also finds the other important information (labelled as Slots), and classifies them.

[10]
```python
# Define one classification loss for each output (intent & NER):
losses = [SparseCategoricalCrossentropy(from_logits=True),
          SparseCategoricalCrossentropy(from_logits=True)]

joint_model.compile(optimizer=Adam(learning_rate=3e-5, epsilon=1e-08),
                    loss=losses,
                    metrics=[SparseCategoricalAccuracy('accuracy')], run_eagerly=True)
```

[11]
```python
# Train the model.
history = joint_model.fit(encoded_train["input_ids"], (slot_train, intent_train), \
    validation_data=(encoded_valid["input_ids"], (slot_valid, intent_valid)), \
    epochs=1, batch_size=32)
```

```
409/409 [==============================] - 195s 473ms/step - loss: 0.4859 - output_1_loss: 0.2718 - output_2_loss: 0.2142 - output_1_accuracy: 0.9383 - output_2_accuracy: 0.9294 - val_loss: 0.0905 - val_output_1_loss: 0.0447 - val_ou
```

```
[12]  1 # Use the model we trained to get the intent & slot logits
      2 # and print the actual string of the class corresponding to
      3 # highest logit score for each token, and the sentence overall.
      4 def show_predictions(text, intent_names, slot_names):
      5     inputs = tf.constant(tokenizer.encode(text))[None, :]  # batch_size = 1
      6     outputs = joint_model(inputs)
      7     slot_logits, intent_logits = outputs  ### YOUR CODE HERE ###
      8     slot_ids = slot_logits.numpy().argmax(axis=-1)[0, 1:-1]
      9     intent_id = intent_logits.numpy().argmax(axis=-1)[0]
     10     print("## Intent:", intent_names[intent_id])  ### YOUR CODE HERE ###
     11     print("## Slots:")
     12     for token, slot_id in zip(tokenizer.tokenize(text), slot_ids):
     13         print(f"{token:>10} : {slot_names[slot_id]}")
```

Let's see how our classification function works on some examples!

```
[13]  1 show_predictions("Book a table for two at Le Ritz for Friday night!", intent_names, slot_names)
```

```
## Intent: BookRestaurant
## Slots:
      Book : O
         a : O
     table : O
       for : O
       two : B-party_size_number
        at : O
        Le : B-restaurant_name
         R : I-restaurant_name
     ##itz : I-restaurant_name
       for : O
    Friday : B-timeRange
     night : O
         ! : O
```

```
[ ]  1 show_predictions("Will it snow tomorrow in Saclay?", intent_names, slot_names)
```

```
## Intent: GetWeather
## Slots:
      Will : O
        it : O
      snow : B-condition_description
  tomorrow : B-timeRange
        in : O
        Sa : B-city
       ##c : I-city
     ##lay : I-city
         ? : O
```

```
  1 show_predictions("I would like to listen to Anima by Thom Yorke.", intent_names, slot_names)
```

```
## Intent: PlayMusic
## Slots:
         I : O
     would : O
      like : O
        to : O
    listen : O
        to : O
        An : B-artist
     ##ima : I-album
        by : O
      Thom : B-artist
      York : I-artist
       ##e : I-artist
         . : O
```

[16]
```python
def decode_predictions(text, intent_names, slot_names,
                       intent_id, slot_ids):
    info = {"intent": intent_names[intent_id]}
    collected_slots = {}
    active_slot_words = []
    active_slot_name = None
    for word in text.split():
        tokens = tokenizer.tokenize(word)
        current_word_slot_ids = slot_ids[:len(tokens)]
        slot_ids = slot_ids[len(tokens):]
        current_word_slot_name = slot_names[current_word_slot_ids[0]]
        if current_word_slot_name == "O":
            if active_slot_name:
                collected_slots[active_slot_name] = " ".join(active_slot_words)
                active_slot_words = []
                active_slot_name = None
        else:
            # Naive BIO: handling: treat B- and I- the same...
            new_slot_name = current_word_slot_name[2:]
            if active_slot_name is None:
                active_slot_words.append(word)
                active_slot_name = new_slot_name
            elif new_slot_name == active_slot_name:
                active_slot_words.append(word)
            else:
                collected_slots[active_slot_name] = " ".join(active_slot_words)
                active_slot_words = [word]
                active_slot_name = new_slot_name
    if active_slot_name:
        collected_slots[active_slot_name] = " ".join(active_slot_words)
    info["slots"] = collected_slots
    return info
```

[17]
```python
def nlu(text, intent_names, slot_names):
    inputs = tf.constant(tokenizer.encode(text))[None, :]  # batch_size = 1
    outputs = joint_model(inputs)
    slot_logits, intent_logits = outputs
    slot_ids = slot_logits.numpy().argmax(axis=-1)[0, 1:-1]
    intent_id = intent_logits.numpy().argmax(axis=-1)[0]

    return decode_predictions(text, intent_names, slot_names, intent_id, slot_ids)
```

[18]
```python
nlu("Will it snow tomorrow in Saclay", intent_names, slot_names)
```

```
{'intent': 'GetWeather',
 'slots': {'city': 'Saclay',
  'condition_description': 'snow',
  'timeRange': 'tomorrow'}}
```

[76]
```python
nlu("Rate the Maze Runner by James Dashner", intent_names, slot_names)
```

```
{'intent': 'RateBook',
 'slots': {'artist': 'James Dashner', 'object_name': 'Maze Runner'}}
```

[77]
```python
nlu("Search for Michael Jackson's songs", intent_names, slot_names)
```

```
{'intent': 'SearchCreativeWork',
 'slots': {'artist': "Michael Jackson's", 'music_item': 'songs'}}
```

```
[18] nlu("Will it snow tomorrow in Saclay", intent_names, slot_names)
```

```
[24]
nlu("Hey Cortana, can you add the song 'insert name here' by 'insert other name here' to a random playlist? Thanks!", intent_names, slot_names)
```

```
{'intent': 'AddToPlaylist',
 'slots': {'artist': 'Cortana,',
  'entity_name': "other name here'",
  'music_item': 'song'}}
```

```
{'intent': 'PlayMusic', 'slots': {'artist': 'Thom Yorke'}}
```

```
[76] nlu("Rate the Maze Runner by James Dashner", intent_names, slot_names)
```

```
[82] nlu("Add Sarah by Karen and Melody by Pran to Red playlist", intent_names, slot_names)
```

```
{'intent': 'AddToPlaylist',
 'slots': {'artist': 'Pran',
  'entity_name': 'and',
  'music_item': 'Melody',
  'playlist': 'Red'}}
```

```
'timeRange': 'Friday'}}
```

```
[77] nlu("Search for Michael Jackson's songs", intent_names, slot_names)
```

```
nlu("Add the song World by Seventeen to my playlist Red", intent_names, slot_names)
```

```
{'intent': 'AddToPlaylist',
 'slots': {'entity_name': 'World by Seventeen',
  'music_item': 'song',
  'playlist': 'Red',
  'playlist_owner': 'my'}}
```

```
{'intent': 'PlayMusic',
 'slots': {'artist': 'RyanHiga?',
  'object_type': 'video',
  'service': 'Youtuber'}}
```

Only 7 Intent Classes

```
[78] nlu("What's the latest video by Youtuber RyanHiga?", intent_names, slot_names)

{'intent': 'PlayMusic',
 'slots': {'artist': 'RyanHiga?',
  'object_type': 'video',
  'service': 'Youtuber'}}
```

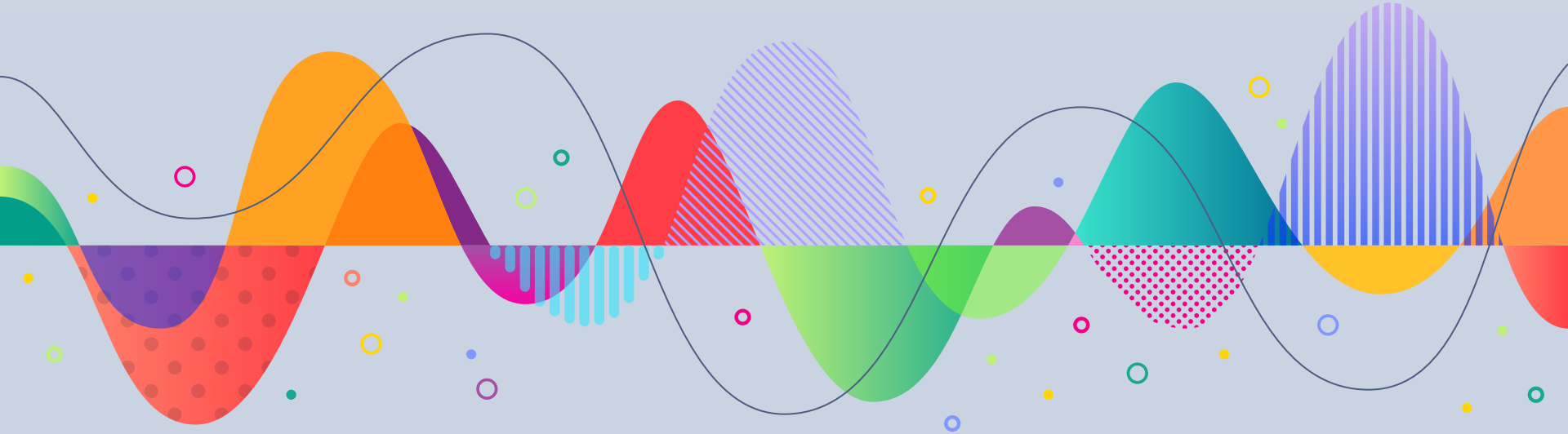
PlayMusic? For a video?

```
[72] intent_names

['AddToPlaylist',
 'BookRestaurant',
 'GetWeather',
 'PlayMusic',
 'RateBook',
 'SearchCreativeWork',
 'SearchScreeningEvent']
```

No option of playing a video; PlayMusic is the closest

Project:
Limitations and Extensions



Limitation - Longer Commands

Semi-working example

```
nlu("Add the song World by Seventeen to my playlist Red", intent_names, slot_names)
```

```
{'intent': 'AddToPlaylist',
 'slots': {'entity_name': 'World by Seventeen',
  'music_item': 'song',
  'playlist': 'Red',
  'playlist_owner': 'my'}}
```

The only thing that our model didn't recognize was the artist and the music item(Seventeen and World)

Pretty far off example

```
[84]
nlu("Hey Cortana, can you add the song 'insert name here' by 'insert other name here' to a random playlist? Thanks!", intent_names, slot_names)
```

```
{'intent': 'AddToPlaylist',
 'slots': {'artist': 'Cortana,', 'music_item': 'song'}}
```

The song name and the artist were processed incorrectly.

Limitation - Compound Commands pt.1

```
[82] nlu("Add Sarah by Karen and Melody by Pran to Red playlist", intent_names, slot_names)

     {'intent': 'AddToPlaylist',
      'slots': {'artist': 'Pran',
       'entity_name': 'and',
       'music_item': 'Melody',
       'playlist': 'Red'}}
```

The correct output would be:

Intent : AddToPlaylist

Slots : {artist: {Karen, Pran}, music_item : {Sarah, Melody}, playlist : Red }

Our model can't process and output correctly for two or more tasks under the same intent.

Limitation - Compound Commands

This command works:

```
[113] nlu("Please put on Hello by Adele Adkins", intent_names, slot_names)

{'intent': 'PlayMusic',
 'slots': {'artist': 'Adele Adkins', 'music_item': 'Hello'}}
```

And so does this one

```
[109] nlu("Search up Adele's next concert" , intent_names, slot_names)

{'intent': 'SearchCreativeWork', 'slots': {'object_type': 'concert'}}
```

But together?

```
nlu("Please put on Hello by Adele Adkins and search up Adele's next concert date", intent_names, slot_names)

{'intent': 'PlayMusic',
 'slots': {'artist': 'Adele Adkins',
  'entity_name': 'Hello',
  'music_item': 'concert'}}
```

Limitation - Symbols

An earlier example:

```
nlu("Add the song World by Seventeen to my playlist Red", intent_names, slot_names)
```

```
{'intent': 'AddToPlaylist',
 'slots': {'entity_name': 'World by Seventeen',
  'music_item': 'song',
  'playlist': 'Red',
  'playlist_owner': 'my'}}
```

```
[87] nlu("Add the song _World by Seventeen to my playlist Red", intent_names, slot_names)
```

```
{'intent': 'AddToPlaylist',
 'slots': {'entity_name': 'Seventeen',
  'music_item': 'song',
  'playlist': 'Red',
  'playlist_owner': 'my'}}
```

Limitation - Languages

This limitation applies to any BERT model because they are all built and pretrained on the English language.

Beyond different symbols, the other major difference is sentence structures.

Simple example:

English is a subject-prominent language.

“I walked to the park yesterday.”

Chinese is topic-prominent language.

The same sentence roughly translates to “I yesterday walked to the park.”

But with no spaces.

Ethics

Problems with Training Data

Our model, and BERT in general, is trained on unfiltered data from the internet. In other words, there could be hate speech, gender bias, or any other form of bias in that data.

This isn't going to impact speech commands too much since BERT handles processing and not the output, but there will be a negative effect on applications that use BERT *and* handle output.

Ex. Translators and Search Queries

Problems with Acquiring Data

BERT's training data is acquired from the internet. But voice command data is acquired from the usage of Siri/Alexa/Cortana in people's everyday lives. If opted in, those services only record sound it receives after being turned on...still a problem.

Right now, if one of you were to say "Hey Cortana", you'd activate my computer's Cortana. In fact, even saying something like "Poor taunting skills" or "Core-taught studies" slow enough will activate it.

There's always the possibility of accidentally activating the device. While all 3 usually activate w/ a sound effect, there's always those points in time when you have everything muted. Who knows when, and how many times, you accidentally activated the device then?

Generic BERT Ethics

Influence; Lightly touched on the Training Data part. Generally, any influence a device like Siri would have on a person would be minimal since Siri is built to respond to us through an action be it playing music or reading through the headlines from our favorite news source. Influence will be a key ethical point once Siri/Alexa/Cortana are capable of holding long conversations with us.

Falsified Info; The development of NLPs like BERT is bound to come with bots that can also write sentences. This leads into its own set of ethics with the most recognizeable one being misinformation.

Humorous Goodbye

Here's some grass to touch since we've all been focusing on doing this project and learning as much as we can throughout this program.

Thank you for all the presentations so far! Thank you Inspirit AI for allowing all of us the chance to learn so much about AI.

Thank you Ethan for being an amazing project leader and guiding us throughout the project. We literally couldn't have done any of this without you.

And to all the people here;

Have a great rest-of-year!

Cheers

-The Understanding Siri Team