Children often ask their caregivers to find episodes of their favorite TV shows based only on a very short (and loosely relevant!) description of it ("the one where Arthur has a wiggly tooth") but video services like Netflix and Amazon don't currently provide such content-based search. Given summaries of each episode, can we use sequence embeddings to solve this retrieval problem?

Before beginning this homework, install the following libraries:

```
conda install -c huggingface transformers
pip install -U sentence-transformers
conda install -c conda-forge ipywidgets
```

First, let's read in our data for the TV show "Wild Kratts" (from Wikipedia), which has the following (tab-separated) form:

Episode	Title	Summary
1	Mom of a Croc	At the Nile River, zoologists Chris and Martin Kratt (voiced by their real-world selves) are on a mission to show one of their fellow Wild Kratts team members—brilliant young inventor Aviva Corcovado (Athena Karkanis)—that there's more to crocodiles than just violence and snapping jaws. After shrinking themselves down to a few inches tall by using Aviva's Miniaturizer invention, the Kratt Brothers disguise themselves as crocodile eggs and sneak into a mother crocodile's new nest. In the Wild Kratts team's turtle-shaped aircraft and headquarters—the Tortuga, one of Aviva's greatest inventions—the Wild Kratts tech team, consisting of Aviva, communications expert and mechanic Koki (Heather Bambrick), and skilled pilot Jimmy Z (Jonathan Malen) monitor Chris and Martin and watch as the mother crocodile faithfully guards her nest against predators for months without even eating anything. Eventually, as the crocodile eggs hatch and the crocodile mom uses her mouth to carry several of her newly hatched babies to the river, Aviva changes her mind about crocodiles and decides that these reptiles are in fact caring and dedicated mothers. But when the mother crocodile leaves the river to go get more hatchlings from her nest, predators threaten the first batch of baby crocodiles. The Kratt Brothers must use the incredible Creature Power Suits—two of Aviva's inventions—to gain the abilities of crocodiles and protect the vulnerable crocodile hatchlings.
2	Whale of a Squid	The Kratt Brothers use Aviva's amphipod-inspired submersible, the Amphisub, to dive into the deep waters of the Southern Ocean. There, they witness a never-before-seen wildlife moment: a battle between a sperm whale and a giant squid. However, the water pressure at the extreme depths where the battle is taking place badly damages and partially crushes the Amphisub, forcing Aviva to use her new ExtendoArm invention to pull the submersible back to the Tortuga. To allow Chris and Martin to return to the site of the whale-versus-squid battle, Aviva programs two new Creature Power Suits—Sperm Whale Power for Chris, and Squid Power for Martin. The Kratt Brothers use their new Creature Powers to dive back into the deep sea, where the sperm whale and the giant squid are still locked in combat. Suddenly, the sperm whale becomes entangled in a discarded fishing net and begins sinking toward an area full of underwater volcanoes. To make matters worse, a colossal squid attacks the sperm whale's calf. Chris and Martin must put their Creature Powers of both sperm whale and squid to good use to rescue the mother sperm whale and her calf.

In [87]:
```
def read_data(filename):
    data=[]
    with open(filename, encoding="utf-8") as file:
        for line in file:
            cols=line.rstrip().split("\t")
            episode=cols[0]
            title=cols[1]
            summary=cols[2]
            data.append((episode, title, summary))
    return data
```

In [88]:
```
data=read_data("../data/wild_kratts_episodes.txt")
```

In [89]:
```
def get_document_reps_for_data(data, sequence_embedding_function, model):

    # This function applies the sequence_embedding_function argument (a fu
    # element in the input data list, and returns a copy of that list with

    data_with_reps=[]

    for episode, title, summary in data:
        data_with_reps.append((episode, title, summary, sequence_embedding_

    return data_with_reps
```

In [90]:
```
def cosine_similarity(a, b):
    return np.dot(a, b)/(np.linalg.norm(a)*np.linalg.norm(b))
```

First, we may be tempted to use the [CLS] token for BERT to represent an entire input string (as is often done in *supervised* document classification models). How well does this work as an out-of-the-box document representation not optimized for our particular task?

In [91]:
```
from transformers import BertModel, BertTokenizer
import numpy as np
from sentence_transformers import SentenceTransformer
```

In [92]:
```
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertModel.from_pretrained('bert-base-uncased')
```

```
Some weights of the model checkpoint at bert-base-uncased were not used whe
n initializing BertModel: ['cls.predictions.transform.LayerNorm.bias', 'cls
.predictions.transform.LayerNorm.weight', 'cls.seq_relationship.bias', 'cls
.predictions.transform.dense.weight', 'cls.predictions.decoder.weight', 'cl
s.seq_relationship.weight', 'cls.predictions.transform.dense.bias', 'cls.pr
edictions.bias']
- This IS expected if you are initializing BertModel from the checkpoint of
a model trained on another task or with another architecture (e.g. initiali
zing a BertForSequenceClassification model from a BertForPreTraining model)
.
- This IS NOT expected if you are initializing BertModel from the checkpoin
t of a model that you expect to be exactly identical (initializing a BertFo
rSequenceClassification model from a BertForSequenceClassification model).
```

Q1: Fill out the `get_cls_token_for_doc` function to return the [CLS] embedding for the input string. The output should be a single 768-dimensional numpy vector (see `4.embeddings/BERT.ipynb` for converting between a pytorch tensor and a numpy object).

In [93]:
```
def get_cls_token_for_doc(model, string):
    inputs = tokenizer(string, return_tensors="pt")
    # your code goes here
    tokens=tokenizer.convert_ids_to_tokens(inputs["input_ids"][0])
    cls_index = 0
    outputs = model(**inputs)
    return outputs.last_hidden_state[0][cls_index].detach().numpy()
```

In [94]:
```
bert_cls_data=get_document_reps_for_data(data, get_cls_token_for_doc, model
```

Q2: Use these representations to find the episode that is most similar to the description "The one where they bounce back in time" by having the highest cosine similarity between representations. A sample function shell `run_query` is provided below, along with the only arguments you need, but feel free to adapt it as you see fit.

In [97]:
```
def run_query(query, data_with_reps, sequence_embedding_function, model):
    # your code goes here
    max_cos_sim = 0
    match_title=""
    query_rep = sequence_embedding_function(model, query)
    for ep, title, summ, rep in data_with_reps:
        if(cosine_similarity(query_rep,rep)>=max_cos_sim):
            max_cos_sim = cosine_similarity(query_rep,rep)
            match_title=title
    print("Title: ",match_title," Cosine Similarity: ",max_cos_sim)
```

In [98]:
```
query="The one where they bounce back in time"
run_query(query, bert_cls_data, get_cls_token_for_doc, model)
```

```
Title:  Little Howler  Cosine Similarity:  0.75525886
```

Now let's try a sentence embedding model that was optimized for generating sentence representations: Sentence-BERT (Reimers and Gurevych 2019). Example usage (in the context of the Huggingface transformers library) can be found here.

In [99]:
```
sentence_model = SentenceTransformer('sentence-transformers/all-distilrobe
```

Q3: Fill out the `get_sentence_embedding` function below to return the sentence embedding for the input string, and use it again to find the episode that is most similar to the description "The one where they bounce back in time" by having the highest cosine similarity between representations. Which method for generating sentence embeddings appears better for this task?

In [100…
```
def get_sentence_embedding(model, string):
    # your code goes here
    embedding = model.encode(string)
    return embedding
```

In [101…
```
sentence_transformer_data=get_document_reps_for_data(data, get_sentence_emb
```

In [102…
```
query="The one where they bounce back in time"
run_query(query, sentence_transformer_data, get_sentence_embedding, sentenc
```

```
Title:  Back in Creature Time: Day of the Dodo  Cosine Similarity:  0.34919
497
```

The second method (i.e. using Sentence-BERT) for generating sentence embeddings appears to be better for this task.

In [ ]:
```
```