Baseline output:

```
Precision = 0.243110
Recall = 0.544379
AER = 0.681684
```

Threshold -t 0.5 output:

```
Precision = 0.834483
Recall = 0.340237
AER = 0.511387
```

Best Guess output:

```
Precision = 0.574202
Recall = 0.730769
AER = 0.375826
```

In [ ]:

```python
import re

paths = {"full_a": "hansards.a",
        "full_e": "hansards.e",
        "full_f": "hansards.f",
        "dev_a": "dev.a",
        "dev_e": "dev.e",
        "dev_f": "dev.f"}

# Proof of concept with shortened (first 37) list
with open(paths["dev_a"], "r", encoding="utf-8") as dev_a:
    dev_a_sentences = dev_a.readlines()
with open(paths["dev_e"], "r", encoding="utf-8") as dev_e:
    dev_e_sentences = dev_e.readlines()
    for i, line in enumerate(dev_e_sentences):
        dev_e_sentences[i] = line.split()
with open(paths["dev_f"], "r", encoding="utf-8") as dev_f:
    dev_f_sentences = dev_f.readlines()
    for i, line in enumerate(dev_f_sentences):
        dev_f_sentences[i] = line.split()

with open(paths["full_e"], "r", encoding="utf-8") as full_e:
    full_e_sentences = full_e.readlines()
    for i, line in enumerate(full_e_sentences):
        full_e_sentences[i] = line.split()
```

```
with open(paths["full_f"], "r", encoding="utf-8") as full_f:
    full_f_sentences = full_f.readlines()
    for i, line in enumerate(full_f_sentences):
        full_f_sentences[i] = line.split()

dev_alignments = []
for sentence in dev_a_sentences:
    new_sentence = []
    for word in sentence.split():
        m = re.search('(\d+)\D(\d+)', word)
        new_sentence.append((m[1], m[2]))
    dev_alignments.append(new_sentence)
dev_a_sentences = dev_alignments
# print(dev_a_sentences[0])
# print(full_f_sentences[0])
# print(full_e_sentences[0])
```

In [ ]:

```
with open("fast_sentences.txt", "w", encoding="utf-8") as f_out:
    for (f, e) in zip(full_f_sentences, full_e_sentences):
        line = f"{' '.join(f)} ||| {' '.join(e)}\n"
        f_out.write(line)
```

Algo 1:

In [ ]:

```
from collections import defaultdict

dev_pair_count, dev_e_count = defaultdict(int), defaultdict(int) # I
for n, alignments in enumerate(dev_a_sentences): # Step through sent
    for alignment in alignments: # Step through observed alignment p
        dev_pair_count[(dev_f_sentences[n][int(alignment[0])], dev_e
        dev_e_count[dev_e_sentences[n][int(alignment[1])]] += 1 # In
p_fe = {word_pair: dev_pair_count[word_pair]/dev_e_count[word_pair[1
#print(p_fe)

# """
# Honestly not sure at all how the Algo 1 pseudocode was supposed to
# in hansards.a, not only do the alignments look terrible at a glanc
# """
# p_fe = defaultdict(float)
# pair_count = defaultdict(int) # Initialze all counts to 0
# e_count = defaultdict(int)
# for n, a in enumerate(dev_a_sentences):
#     for i, f in enumerate(dev_f_sentences[n], 1):
#         for j, e in enumerate(dev_e_sentences[n], 1):          # S
#             if int(a[i][1]) == j:
#                 pair_count[(f, e)] += 1  # Increment count of alig
#                 e_count[e] += 1            # Increment marginal co
# for word_pair in pair_count:
#     p_fe[word_pair] = pair_count[word_pair]/e_count[word_pair[1]]
```

Algo 2:

In [ ]:

```
from timeit import default_timer as timer
k = 0
theta = defaultdict(lambda:0.00000000000000000001) # A common choice
while k < 8: # Until parameters converge or some other criterion, su
    k += 1
    timer_start = timer()
    full_pair_count, full_e_count = defaultdict(int), defaultdict(in
    for n, F in enumerate(full_f_sentences): # E-Step: Compute expec
        for f in F:
            Z = 0 # Z is commonly used to denote a normalization terı
            for e in full_e_sentences[n]:
                Z += theta[(f, e)]
            for e in full_e_sentences[n]:
                c = theta[(f, e)] / Z # Compute expected count
                full_pair_count[(f, e)] += c # Increment count of al
                full_e_count[e] += c # Increment marginal count of E
    theta = {word_pair: full_pair_count[word_pair]/full_e_count[word
    #print("('très', 'very'):", theta[('très', 'very')])
    timer_stop = timer()
    time = timer_stop - timer_start
    #print("epoch", k, "took", round(time), "seconds")
```

```
theta_0 = (lambda:0.1), 10 epoch limit, sampling word
pair ('très', 'very'). Total runtime: 12min 43sec
Results nearly identical to theta_0 = (lambda:1)
('très', 'very'): 0.027585138264210947
epoch 1 took 77.6112612 seconds
('très', 'very'): 0.225722140356126
epoch 2 took 82.6098117 seconds
('très', 'very'): 0.4422172475438449
epoch 3 took 76.6076099 seconds
('très', 'very'): 0.540104671629061
epoch 4 took 75.42177760000004 seconds
('très', 'very'): 0.5835515176657871
epoch 5 took 77.22606769999993 seconds
('très', 'very'): 0.6033351981591198
epoch 6 took 74.98604260000002 seconds
('très', 'very'): 0.6119744755813445
epoch 7 took 72.43497400000001 seconds
('très', 'very'): 0.6152246452183321
```

```
epoch 8 took 76.34132110000007 seconds
('très', 'very'): 0.6159095087835013
epoch 9 took 72.46883460000004 seconds
('très', 'very'): 0.6154325496303655
epoch 10 took 77.4239576 seconds
```

theta_0 = (lambda:1), 10 epoch limit, sampling word pair ('très', 'very'). Total runtime: 13min

8 epochs seems to be the end of rapid improvement, the guess even goes down marginally in epoch 10.

```
('très', 'very'): 0.027585138264210947
epoch 1 took 73.8355653 seconds
('très', 'very'): 0.225722140356126
epoch 2 took 84.44004360000001 seconds
('très', 'very'): 0.44221724754384495
epoch 3 took 77.45663670000002 seconds
('très', 'very'): 0.540104671629061
epoch 4 took 77.74768819999997 seconds
('très', 'very'): 0.583551517665787
epoch 5 took 77.1224856 seconds
('très', 'very'): 0.6033351981591198
epoch 6 took 80.14680129999999 seconds
('très', 'very'): 0.6119744755813448
epoch 7 took 82.47481409999995 seconds
('très', 'very'): 0.615224645218332
epoch 8 took 76.8060847999999 seconds
('très', 'very'): 0.6159095087835013
epoch 9 took 74.31741080000006 seconds
('très', 'very'): 0.6154325496303653
epoch 10 took 74.46382360000007 seconds
```

theta_0 = (lambda:0.00000000000000000001), 10 epoch limit, sampling word pair ('très', 'very').

stopped early after epoch 2, since results were identical to theta_0 = (lambda:0.1) and theta_0 = (lambda:1)

Am I missing something, that my lambda doesn't matter? Maybe it'll be obvious when I start looking for AER.

```
('très', 'very'): 0.027585138264210954
epoch 1 took 76 seconds
```

```
('très', 'very'): 0.225722140356126
epoch 2 took 78 seconds
```