Multiple EM for Motif Elicitation

In [1]:
```
import numpy as np
from sklearn.preprocessing import normalize
from Bio import SeqIO
```

In [2]:
```
# Constant used in this exercise
# Fill in all of the ...s/TODOs
width = 8

# Helper dict for indexing
let_dict = {"A":0, "C":1, "G":2, "T":3}
```

1. Read in Fasta Sequences

In [3]:
```
fastas = open("motif-regions.fa")
sequences = SeqIO.parse(fastas, 'fasta')
seqs = {}
for i, sequence in enumerate(sequences):
    seqs[i] = sequence
seqs = list(seqs.values())
```

2. Create p_0

In [4]:
```
def init_p(l, w, seqs, let):
    p = np.zeros((4, w+1))

    # Uniform background for each A C G T
    for i in range(len(p)):
        p[i][0] = 0.25

    # set motif positions
    for i in range(l-w+1):
        for sequence in seqs:
            for j in range(w):
                nuc = str(sequence.seq)[i+j]
                p[let[nuc]][j+1] += 1

    p = normalize(p, axis=0, norm = 'l1')
    return p
```

3. Fill in EM iteration

In [5]:
```
def run_EM(w, seqs, let, init_p, up_prob, up_motif, epsilon = 2**-64):

    l = len(str(seqs[0].seq))
    no_change = False
    p_t_1 = init_p(l, w, seqs, let)

    while not no_change:
        z_t = up_motif(l, w, p_t_1, seqs, let) # E step
        p_t = up_prob(l, w, z_t, seqs, let) # M step

        diff = np.subtract(p_t, p_t_1)
        check = np.absolute(diff)

        if np.amax(check) <= epsilon:
            no_change = True
        else:
            p_t_1 = p_t

    return p_t, z_t
```

4. Fill in function to update z_t

In [6]:
```
def up_motif(l, w, p_t_1, seqs, let):
    z_t = np.zeros((len(seqs), l-w+1))
    other = (0.25)**(l-w)

    for i, sequence in enumerate(seqs):
        total = 0
        for j in range(l-w+1):
            p = 1
            for k in range(w):
                nuc = str(sequence.seq)[j+k]
                p *= p_t_1[let[nuc]][k+1]

            p *= other
            z_t[i][j] = p
            total += p

        for j in range(l-w+1):
            z_t[i][j] /= total

    z_t = normalize(z_t, axis=0, norm='l1')
    return z_t
```

5. Fill in function to update p_t

In [7]:
```
def up_prob(l, w, z_t, seqs, let):
    p_t = np.zeros((4, w+1))
    n = np.zeros((4, w+1))

    # k > 0
    for k in range(1, w+1):
        for base in let.keys():
            sum_z = 0
            for i, sequence in enumerate(seqs):
                current = str(sequence.seq)
                j_vals = [j for j in range(w-k+1) if current[j+k-1] ==
                for j in j_vals:
                    sum_z += z_t[i][j]

            n[let[base]][k] = sum_z

    # k = 0
    joined_seq = "".join([str(sequence.seq) for sequence in seqs])
    counts = [0, 0, 0, 0]
    sum_n_j = [0, 0, 0, 0]
    for base in let.keys():
        counts[let[base]] = joined_seq.count(base) # total number of b
        for sequence in seqs:
            current = str(sequence.seq)
            sum_n_j[let[base]] += current[:w].count(base)
        n[let[base]][0] = counts[let[base]] - sum_n_j[let[base]]

    totals = np.sum(n, axis=0) + 4
    totals = np.full((4, len(totals)), totals)
    p_t = np.divide(n+1, totals)

    return p_t
```

6. Run the EM to find the final p and z

In [8]:
```
# Use the variables set at the start and
# TODO:
p_end, z_end = run_EM(width, seqs, let_dict, init_p, up_prob, up_motif
```

7. Determine Motifs

In [9]:
```python
# Find the indices of the max element for each row in z_end
motif_indices = [np.argmax(z_end[i]) for i in range(np.shape(z_end)[0]

# Get the 'width' characters long motifs using seqs
motifs = [str(sequence.seq)[motif_indices[i]:motif_indices[i] + width]
print(motifs)
```

```
['TTTTTCTT', 'TTTATTCT', 'TTATTTCC', 'TTTTTTTT', 'TTTTCTAA', 'TTCTTTC
T', 'TTTTTCTC', 'TTTTTTTT', 'TTTATAGT', 'TTTTTTCG', 'TTTTCTAG', 'TTTT
TTCA', 'TTTATTTA', 'TTTTTTGG', 'TTAAAAGT', 'TTTTATCA', 'TTTTTCCA', 'T
TTTTCAT', 'TTTTCTGG', 'TATTTGAC', 'TTATTCAG', 'TTTCTAGA', 'TTTTAATA',
'TTTTGTAA', 'TTTCCTAC', 'TTGTTAGG', 'TATATGTA', 'TTTTTATC', 'TTTTTTGC
', 'TTCTTTTT', 'TTATGTTC', 'TTTCTTCT', 'TTTTCTGG', 'TTTTTTCT', 'TTTTT
CTA', 'ATTACCAG', 'TATTTTAA', 'TTTTTTTT', 'TTTTTTCA', 'TTTTTGGC', 'TT
TTTCA', 'TTTTTGCT', 'TTTTTTGA', 'TTTTTTCC', 'TTTCTAAG', 'TATATGAA',
'TTTTTCTT', 'TTTTTTTC', 'AATTTGAA', 'TTTCTTTA']
```

In [ ]:
```

```