Multiple EM for Motif Elicitation

In [8]:
```
import numpy as np
from sklearn.preprocessing import normalize
```

In [9]:
```
# Constants 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 [10]:
```
# Read in the files using BioPython
# TODO:
from Bio import SeqIO
fastas = [i for i in SeqIO.parse("motif_regions.fa", "fasta")]
```

In [11]:
```
# Extract strings of sequences from the above files
# TODO:

sequences = [str(i.seq) for i in fastas]
```

2. Create p_0

In [12]:
```
# Initialize p with a uniform background

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

    # set a uniform background
    for i in let.keys():
        # TODO:
        p[let[i]][0] = 0.25

    # set motif positions
    for i in range(l-w+1):
        for sequence in seqs:
            for j in range(w):

                # Fill in p_0
                # TODO:
                p[ let[ sequence[i + j] ] ][j + 1] += 1 #ANSWER

    # normalize columns to sum to 1
    p = normalize(p, axis = 0, norm = 'l1')

    return p
```

3. Fill in EM iteration

In [13]:
```
# Define a general function to run EM

def run_EM(w, seqs, let, init_p, up_prob, up_motif, epsilon = 0.0001):
    l = len(seqs[0])

    no_change = False

    # set an initial p_t_1
    # TODO:
    p_t_1 = init_p(l, w, seqs, let) #ANSWER

    while not no_change:

        # Label the following steps as E step or M step in the comment preceding

        # TODO:
        # E step #ANSWER

        z_t = up_motif(l, w, p_t_1, seqs, let)

        # TODO:
        # M step #ANSWER

        p_t = up_prob(l, w, z_t, seqs, let)

        diff = np.subtract(p_t, p_t_1)

        # Write a condition to stop the EM iterations (use epsilon and diff)
        # TODO:
        if 4*(w+1) == np.sum(diff<epsilon): #ANSWER
            no_change = True
        else:
            # Update p_t_1
            # TODO:s
            p_t_1 = p_t #ANSWER

    return p_t, z_t
```

4. Fill in function to update z_t

In [16]:
```
# Define a function to update z

def up_motif(l, w, p_t_1, seqs, let):
    z_t = np.zeros((len(seqs), l-w+1))

    for i, sequence in enumerate(seqs):
        for j in range(l-w+1):

            # Fill in z_t using p_t_1

            p_vals = []
            for position, letter in enumerate(sequence):

                # TODO:
                if position>=j and position<j+w:
                    p_vals.append(p_t_1[let[letter]][position-j+1])
                else:
                    p_vals.append(p_t_1[let[letter]][0])

            # TODO:
            z_t[i][j] = np.prod(muls)

            ### One-line solution
            #
            # z_t[i][j] = np.prod([p_t_1[let[letter]][int(position>=j)*int(posit
ion<j+w)*(position-j+1)] for position, letter in enumerate(sequence)])
            #
            ###

    # Normalize z_t so each row sums to 1
    # TODO:
    z_t = normalize(z_t, axis = 1, norm = 'l1') #ANSWER

    return z_t
```

5. Fill in function to update p_t

In [17]:
```
from collections import Counter

# Define a function to update p

def up_prob(l, w, z_t, seqs, let):
    p_t = np.zeros((4, w+1))

    n = np.zeros((4, w+1))

    # Fill in n for k > 0
    for k in range(1, w+1):
        for letter in let.keys():
            sum_z = 0
            for i, sequence in enumerate(seqs):

                # Write j_vals according to the condition seen in lecture
                # TODO:
                j_vals = [i for i in range(l-w+1) if sequence[i+k-1] == letter]
#ANSWER

                # Fill in the sum using z_t
                # TODO:
                sum_z += np.sum([z_t[i][j] for j in j_vals]) #ANSWER

            # Fill in the correct indices
            # TODO:
            n[let[letter]][k] = sum_z #ANSWER

    # Fill in n for k == 0

    # May help to make the next step easier
    joined_seq = "".join(seqs)

    # Create a dict with total counts of A,C,G,T
    # TODO:
    counts = Counter(joined_seq) #ANSWER

    # Sum across the rows of n
    # TODO:
    sum_n_j = np.sum(n, axis = 1) #ANSWER


    for letter in let.keys():

        # Fill in the correct indices and its value
        # TODO:
        n[let[letter]][0] = counts[letter] - sum_n_j[let[letter]] #ANSWER


    # Use n to fill in p_t
    # Pseudo-count = 1
    # TODO:
    p_t = np.divide(n + 1, np.sum(n, axis = 0) + 4) #ANSWER

    return p_t
```

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

In [18]:
```
%%time
# TODO:
p_end, z_end = run_EM(width, sequences, let_dict, init_p, up_prob, up_motif, 0.0
001) #ANSWER
```

```
CPU times: user 18.6 s, sys: 1.22 ms, total: 18.6 s
Wall time: 18.6 s
```

7. Determine Motifs

In [19]:
```
# Find the indices of the max element for each row in z_end
# TODO:
motif_indices = np.argmax(z_end, axis=1) #ANSWER

# Get the 'width' characters long motifs using seqs
# TODO:
motifs = [sequences[i][pos:pos+width+1] for i,pos in enumerate(motif_indices)]
print(motifs)
```

```
['AGAAAAATT', 'GGACGGTTC', 'GGTAACAT', 'GGAAAATTG', 'AGAAGATTC', 'AGAAAAAAC', 'A
GAACAATT', 'AGAAAAAAA', 'AGAATATTC', 'GAAAAAATT', 'AGAAAAAAC', 'AGAAAAAA', 'GGAA
AAAAA', 'AGAATATTG', 'GGAAGGTTC', 'GGAAAATAA', 'AGAAAATGA', 'AGAAAAAAG', 'AGCTAA
ATT', 'AGAAAAAAA', 'GGAAAAAGA', 'AGAACCATC', 'GGAAATTTC', 'CGAAGGTTC', 'GGAAAAAT
A', 'AGAACAAAA', 'AGAAAAATT', 'AGAAAAATA', 'AGAAGAAAT', 'AGAAATTTC', 'GGAAAAATA
', 'CGAAAATTT', 'GGAAAATTC', 'AAAAGATTA', 'AGAAAAAAG', 'GGAAGAAAT', 'AGAAAAAGA',
'AAAAGATTG', 'AGAACCTTC', 'GGAAAAATT', 'AGAAAAAAA', 'AGAATAATA', 'AGAACAATA', 'A
AAAGCTGA', 'AGAAAAATA', 'AGAAAAATA', 'GGAAAATCT', 'GGAAAATTT', 'AGAAAAAAA', 'GTA
AAGAGC']
```

In [20]:
```
np.argmax(p_end, axis=0)
```

Out[20]:
```
array([0, 0, 2, 0, 0, 0, 0, 0, 3])
```