Programming Hidden Markov Models (60 P)

In this exercise, you will experiment with hidden Markov models, in particular, applying them to modeling character sequences, and analyzing the learned solution. As a starting point, you are provided in the file `hmm.py` with a basic implementation of an HMM and of the Baum-Welch training algorithm. The names of variables used in the code and the references to equations are taken from the HMM paper by Rabiner et al. downloadable from ISIS. In addition to the variables described in this paper, we use two additional variables: Z for the emission probabilities of observations O, and ψ (i.e. psi) for collecting the statistics of Equation (40c).

Question 1: Analysis of a small HMM (30 P)

We first look at a toy example of an HMM trained on a binary sequence. The training procedure below consists of 100 iterations of the Baum-Welch procedure. It runs the HMM learning algorithm for some toy binary data and prints the parameters learned by the HMM (i.e. matrices A and B).

Question 1a: Qualitative Analysis (15 P)

- *Run* the code several times to check that the behavior is consistent.
- *Describe* qualitatively the solution A, B learned by the model.
- *Explain* how the solution $\lambda = (A, B)$ relates to the sequence of observations O that has been modeled.

In [1]:

```
import numpy,hmm

O = numpy.array([1,0,1,0,1,1,0,0,1,0,0,0,1,1,1,0,1,0,0,0,1,1,0,1,1,0,0,1,
                 0,0,0,1,0,0,0,1,1,0,0,1,0,0,1,1,0,0,0,1,0,1,0,1,0,0,0,1,
                 0,0,1,0,1,0,1,0,0,0,1,1,1,0,1,0,0,0,1,0,0,0,1,0,1,0,1,0,
                 0,1,1,1,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,
                 1,0,0,0,1,1,0,0,1,0,1,1,1,0,0,1,1,0,0,0,1,1,0,0,1,1,0,0,
                 0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,1,0,1,0,0,0,1,0,0,0,1,
                 0,0,1,0,0,0,1,1,0,0,1,1,0,0,1,1,0,0,1,0,0,0,1,1,0,0])

hmmtoy = hmm.HMM(4,2)

for k in range(100):
    hmmtoy.loaddata(O)
    hmmtoy.forward()
    hmmtoy.backward()
    hmmtoy.learn()

print('A')
```

```
print("\n".join([" ".join(['%.3f'%a for a in aa]) for aa in hmmtoy.A]))
print(' ')
print('B')
print("\n".join([" ".join(['%.3f'%b for b in bb]) for bb in hmmtoy.B]))
print(' ')
print('Pi')
print("\n".join(['%.3f'%b for b in hmmtoy.Pi]))
```

```
A
0.000 0.000 0.000 1.000
1.000 0.000 0.000 0.000
0.000 1.000 0.000 0.000
0.000 0.000 1.000 0.000

B
0.800 0.200
0.720 0.280
0.000 1.000
0.880 0.120

Pi
0.000
0.000
1.000
0.000
```

Question 1b: Finding the best number N of hidden states (15 P)

For the same sequence of observations as in Question 1a, we would like to determine automatically what is a good number of hidden states $N = \mathrm{card}(S)$ for the model.

- *Split* the sequence of observations into a training and test set (you can assume stationarity).
- *Train* the model on the training set for several iteration (e.g. 100 iterations) and for multiple parameter N.
- *Show* for each choice of parameter N the log-probability $\log p(O|\lambda)$ for the test set. (If the results are unstable, perform several trials of the same experiment for each parameter N.)
- *Explain* in the light of this experiment what is the best parameter N.

In [2]:
```
import solutions
solutions.question1b(O,hmm.HMM)
```

```
N=2
trial 0 logptrain= -64.971 logptest= -66.283
trial 1 logptrain= -66.876 logptest= -66.302
trial 2 logptrain= -56.241 logptest= -61.575
trial 3 logptrain= -56.241 logptest= -61.575

N=4
trial 0 logptrain= -37.774 logptest= -36.301
trial 1 logptrain= -37.774 logptest= -36.301
trial 2 logptrain= -61.325 logptest= -61.215
trial 3 logptrain= -37.774 logptest= -36.301

N=8
trial 0 logptrain= -34.938 logptest= -60.113
trial 1 logptrain= -36.588 logptest= -71.286
trial 2 logptrain= -37.493 logptest= -35.913
trial 3 logptrain= -36.887 logptest= -38.785

N=16
trial 0 logptrain= -31.757 logptest= -96.748
trial 1 logptrain= -31.616 logptest= -72.682
trial 2 logptrain= -28.200 logptest=-552.458
trial 3 logptrain= -30.121 logptest=-133.546
```

Question 2: Text modeling and generation (30 P)

We would like to train an HMM on character sequences taken from English text. We use the 20 newsgroups dataset that is accessible via scikits-learn http://scikit-learn.org/stable/datasets/twenty_newsgroups.html. (For this, you need to install scikits-learn if not done already.) Documentation is available on the website. The code below allows you to (1) read the dataset, (2) sample HMM-readable sequences from it, and (3) convert them back into string of characters.

In [3]:

```
from sklearn.datasets import fetch_20newsgroups

# Download a subset of the newsgroup dataset
newsgroups_train = fetch_20newsgroups(subset='train',categories=['sci.med
newsgroups_test  = fetch_20newsgroups(subset='test' ,categories=['sci.med

# Sample a sequence of T characters from the dataset
# that the HMM can read (0=whitespace 1-26=A-Z).
#
# Example of execution:
# O = sample(newsgroups_train.data)
# O = sample(newsgroups_test.data)
#
def sample(data,T=50):
    i = numpy.random.randint(len(data))
    O = data[i].upper().replace('\n',' ')
    O = numpy.array([ord(s) for s in O])
    O = numpy.maximum(O[(O>=65)*(O<90)+(O==32)]-64,0)
    j = numpy.random.randint(len(O)-T)
    return O[j:j+T]

# Takes a sequence of integers between 0 and 26 (HMM representation)
# and converts it back to a string of characters
```

```python
def tochar(O):
    return "".join(["%s"%chr(o) for o in (O+32*(O==0)+64*(O>0.5))])
```

Question 2a (15 P)

In order to train the HMM, we use a stochastic optimization algorithm where the Baum-Welch procedure is applied to randomly drawn sequences of $T = 50$ characters at each iteration. The HMM has 27 visible states (A-Z + whitespace) and 200 hidden states. Because the Baum-Welch procedure optimizes for the sequence taken as input, and no necessarily the full text, it can take fairly large steps in the parameter space, which is inadequate for stochastic optimization. We consider instead for the parameters $\lambda = (A, B, \Pi)$ the update rule $\lambda^{new} = (1 - \gamma)\lambda + \gamma\bar{\lambda}$, where $\bar{\lambda}$ contains the candidate parameters obtained from Equations 40a-c. A reasonable value for γ is 0.1.

- *Create* a new class `HMMChar` that extends the class `HMM` provided in `hmm.py`.
- *Implement* for this class a new method `HMMchar.learn(self)` that overrides the original methods, and implements the proposed update rule instead.
- *Implement* the stochastic training procedure and run it.
- *Monitor* $\log p(O|\lambda)$ on the test set at multiple iterations for sequences of same length as the one used for training. (Hint: for less noisy log-probability estimates, use several sequences or a moving average.)

In [4]:
```python
import solutions

class HMMChar(hmm.HMM):

    def learn(self):

        # Compute gamma
        self.gamma = self.alpha*self.beta / self.pobs

        # Compute xi and psi
        self.xi = self.alpha[:-1,:,na]*self.A[na,:,:]*self.beta[1:,na,:]*
        self.psi = self.gamma[:,:,na]*(self.O[:,na,na] == numpy.arange(se

        # Update HMM parameters
        self.A  = 0.9*self.A + 0.1 * (self.xi.sum(axis=0)  / self.gamma[:
        self.B  = 0.9*self.B + 0.1 * (self.psi.sum(axis=0) / self.gamma.s
        self.Pi = 0.9*self.Pi + 0.1 * (self.gamma[0])

    def generate(self,l):

        N,M = self.B.shape

        s = numpy.random.choice(N,p=self.Pi)

        O = []

        for i in range(l):

            O += [ numpy.random.choice(M,p=self.B[s]) ]
```

```
            s = numpy.random.choice(N,p=self.A[s])

        return numpy.array(O)


hmmchar = solutions.HMMChar(200,27)
trainsample = lambda: sample(newsgroups_train.data)
testsample  = lambda: sample(newsgroups_test.data)
```

Question 2b (15 P)

In order to visualize what the HMM has learned, we would like to generate random text from it. A well-trained HMM should generate character sequences that have some similarity with the text it has been trained on.

- *Implement* a method `generate(self,T)` of the class `HMMChar` that takes as argument the length of the character sequence that has to be generated.
- *Test* your method by generating a sequence of 250 characters and comparing it with original text and a purely random sequence.
- *Discuss* how the generated sequences compare with written English and what are the advantages and limitations of the HMM for this problem.

In [5]:

```
for k in range(1000):

    Otrain = trainsample()
    Otest = testsample()
    hmmchar.loaddata(Otrain)
    hmmchar.forward(); pobstrain = hmmchar.pobs
    hmmchar.backward()
    hmmchar.learn()
    hmmchar.loaddata(Otrain)
    hmmchar.forward(); pobstest = hmmchar.pobs

    if k%100 == 0: print(k,numpy.log(pobstrain),numpy.log(pobstest))
```

```
0 -164.8678560469606 -160.07325838775895
100 -136.0878402764852 -132.9031613543354
200 -116.95102219042211 -108.13146789437837
300 -129.43592804485758 -118.34661470625417
400 -113.31431042440062 -100.00063633710337
500 -127.64094914677273 -116.06520443359337
600 -122.6619584970173 -108.95633934540777
700 -126.64280724729173 -111.6415449061965
800 -119.39968752429826 -109.58618131008961
900 -134.96061089179096 -120.33262301732977
```

In [6]:

```
print("original:\n"+tochar(sample(newsgroups_test.data,T=250)))
print("\nlearned:\n"+tochar(hmmchar.generate(250)))
print("\nrandom:\n" +tochar(solutions.HMMChar(200,27).generate(250)))
```

```
original:
HE DENVER COMMUNITY  THE UNIVERSITY HAS NEITHER CONTROL OVER NOR RESPONSIB
ILITY FOR THE OPINIONS OF USERS ORGANIATION NYX PUBLIC ACCESS UNIX AT U OF
DENVER MATHCS DEPT DISTRIBUTION USA LINES   COULD SOMEONE TELL ME WHAT THE
DENSITY OF SKULL BONE IS O

learned:
 OCIRACEELPARTIVY ADHEY I FORMENT TURLINENTE THERDWHIVIROSALT FINENT  MINT
IS POFELSDLAR FOS  U PHE WEPAS URUMS GHE DY ODIOCT CORBUSESATT THIST HUONT
TUMECU HUSINIME PORS LOC TORBBS ISCHEUSANELE BOBLUM I CECY  E WOCES THE AI
TUN OLD C ES CON WHODYNM

random:
TXRMLRXZJJZIQYGIRJBURANDXUJG X WNSNOIDNUAYWRHHZTGCGKZJOZDRJI PNOZYBQJBDNYO
YASISFNO MPJOGAJMTTC GMADNMFT NDBRHYCMIKUKMOYFNAGGJ BXPAULLXKKSORXQF BMYXU
DHQZZK NOXFONFFCTEKUE XNEYBJWMTLXXWOSH YDRJYJMESMWQSWU YYNHQJVNUTGONKNJIOS
XYJMKJESJA VXLJR YSGOBIIZLOU
```

```
In [ ]:
```