ASY Lab

Jannik Gut Luca Blum

April 2020

1 Introduction

We assume the amount of hidden states fit into the cache.
We assume the amount of hidden states, distinct signals, different measures (time) are behave well. (We have to say a bit more later.) Also we assume the variables are bigger than 1.

2 Some Derivations

2.1 $P(Y|\Theta)$

According to Wikipedia, the goal of the Baum-Welch algorithm is to find $\theta^* = argmax_\theta[P(Y|\theta)]$.
We see that $P(Y|\theta) = \Sigma_{i=1}^N[P(Y, X_T = i|\theta)]$, where i is all the hidden states. This is exactly the definition of $\Sigma_{i=1}^N[\alpha_i(T)]$.
Further, we show that (for a t somewhere in the middle):

$$\Sigma_{i=1}^N[\alpha_i(t) * \beta_i(t)] =^? \Sigma_{j=1}^N[\alpha_j(t+1) * \beta_j(t+1)]$$

Apply the definitions of α on the right and β on the left.

$$\Sigma_{i=1}^N[\alpha_i(t)*(\Sigma_{j=1}^N[\beta_j(t+1)*a_{ij}*b_j(y_{t+1})])] =^? \Sigma_{j=1}^N[(b_j(y_{t+1})*\Sigma_{i=1}^N[\alpha_i(t)*a_{ij}])*\beta_j(t+1)]$$

$$\Sigma_{i=1}^N\Sigma_{j=1}^N[\alpha_i(t)*([\beta_j(t+1)*a_{ij}*b_j(y_{t+1})])] =^? \Sigma_{j=1}^N\Sigma_{i=1}^N[(b_j(y_{t+1})*\alpha_i(t)*a_{ij})*\beta_j(t+1)]$$

Keep this representation in mind, we can now reorder the terms and proof our point:

$$\Sigma_{i=1}^N\Sigma_{j=1}^N[\alpha_i(t)*([\beta_j(t+1)*a_{ij}*b_j(y_{t+1})])] =^? \Sigma_{i=1}^N\Sigma_{j=1}^N[\alpha_i(t)*([\beta_j(t+1)*a_{ij}*b_j(y_{t+1})])]$$

This means we can use the same $P(Y|\Theta)$ for all γ. If we look at the second to last line of the derivation we see it has the same format as the $P(Y|\Theta)$ for all ξ.

2.2 divide by constant factor

The only place where we need $\xi or \gamma$ outside of a division is for π and there we only need $\gamma_i(1)$. They both are divided by $P(Y|\Theta)$, a constant factor.

$$\frac{\Sigma_{t=1}^{T-1}\xi_{ij}(t)}{\Sigma_{t=1}^{T-1}\gamma(t)} = \frac{\Sigma_{t=1}^{T-1}\frac{P(X_t=i,X_{t+1}=j,Y|\Theta)}{P(Y|\Theta)}}{\Sigma_{t=1}^{T-1}\frac{P(X_t=i,Y|\Theta)}{P(Y|\Theta)}}$$

Since $P(Y|\Theta)$ is a constant factor, we can take it out of the sum.

$$\frac{\Sigma_{t=1}^{T-1}\frac{P(X_t=i,X_{t+1}=j,Y|\Theta)}{P(Y|\Theta)}}{\Sigma_{t=1}^{T-1}\frac{P(X_t=i,Y|\Theta)}{P(Y|\Theta)}} = \frac{\frac{1}{P(Y|\Theta)}\Sigma_{t=1}^{T-1}P(X_t=i,X_{t+1}=j,Y|\Theta)}{\frac{1}{P(Y|\Theta)}\Sigma_{t=1}^{T-1}P(X_t=i,Y|\Theta)}$$

We now see that we can take away $\frac{1}{P(Y|\Theta)}$ and do not need to divide everything. This unfortunately does not work for $\alpha_i(t)$, that also appears in both terms, but is not constant. :(

3 Other Optimizations

3.1 numerical stability for $P(Y|\Theta)$

Since we have to compute $\gamma_i(t)$, for every t, also the middle, we also need to compute $\alpha_i(T/2) * \beta_i(T/2)$, adding it over all hidden states i, we get a more numerically stable $P(Y|\Theta)$, for free, since we do not need it before computing $\gamma_i(T/2)$:)

3.2 Σ^T makes storage easier for ξ

There is no need to store $\xi_{ij}(t)$ for all t's, since we (mostly) only need the sum of all, that is why we can safe space and stack everything in (Ξ_{ij}). The other exception is b_i^*, that gets discussed in the subsection after the next.

3.3 ξ is a waste of space

The only time ξ is used is for $a_{ij}*$, we might as well create a new a array with $a_{ij}^* = sum\Xi_{ij}$ and never store Ξ_{ij} anywhere else. In the end we have to divide by γ_i.

3.4 γ is a waste of space

With a couple exceptions, the same that counts for ξ can be said for γ. One exception to that is $\pi, \gamma_i(1)$, but that is also exactly π_i^*, which we have to store anyway. Another exception is $b_i^*(y_t)$, where we use a similar trick as for Ξ with the αs to accommodate the 1-operator. We have to keep a copy of $sum(\Gamma_i)$ over all t in local mem (or somewhere similar). The last time we need an explicit γ

is $\gamma_i(T)$ (for all i), so we can subtract it for the sum for a^*_{ij}. Meaning, there is only space for 3*N +b space needed for all γs, one of those N we have to share explicitly (π).

3.5 β is a waste of space

We do not explicitly need α or β, but especially β can be skipped for storage, by directly storing it in $\xi or \gamma$, which are also directly forwarded.
We see that the different parts used in the sum of β are the exact parts, which are needed for ξ and the whole, big β has to be used for γ after one another time step we do not need β anymore; we only need one array to write $\beta_i(t)$ and one array to read $\beta_j(t+1)$. With this tactic, we have to do the forward step at the same time as the update step.

3.6 $\beta_j(t+1) * a_{ji} * b_j(y_{t+1})$ can be reused

This term is needed in the backward step to compute $\beta_i(t)$ and in the update step to compute $\xi_{ij}(t)$. We can store it such that we do not need to calculate it twice.

4 What does not work

4.1 We need either α or β

For γ we need both $\alpha(t)$ and $\beta(t+1)$, we can keep one of those in local memory and iterate in that direction. The other component has to be fully computed to get to time t.

4.2 a, b and α need to be separate

A simple first idea is to check their dimensions and try to argue from there.
a has size [N,N]
b has size [N,K]
α has size [N,T]

5 Accesses

5.1 α

To compute $\alpha_i(t+1)$ we always have to fetch $\alpha_j(t)$, so j given t.
In the compute step, we have to fetch α once, preferably in state-major order, given t. State major is the way.

$$\begin{pmatrix} \alpha_0(0) & \alpha_1(0) & ... & \alpha_N(0) \\ ... & ... & ... & \\ \alpha_0(N) & \alpha_1(N) & ... & \alpha_N(N) \end{pmatrix}$$

5.2 a

In the forward step, a is fetched in first-state order, given the second state.
In the compute step, a is fetched in second-state order, given the first state, but saved in the second-state order, given the first state.
Second state major is the way to go.

$$\begin{pmatrix} \mathrm{a}_{00} & \mathrm{a01} & ... & \mathrm{a}_{0N} \\ ... & ... & ... & \\ \mathrm{a}_{N0} & \mathrm{a}_{N1} & ... & \mathrm{a}_{NN} \end{pmatrix}$$

5.3 b

In the forward step, b is fetched in state order, given the result.
In the backward step, a is fetched in state order.
In the update step, b can be created created in state order.(???)
State major is the way to go.

$$\begin{pmatrix} \mathrm{b}_0(0) & \mathrm{b}_1(0) & ... & \mathrm{b}_N(0) \\ ... & ... & ... & \\ \mathrm{b}_0(dO) & \mathrm{b}_1(dO) & ... & \mathrm{b}_N(dO) \end{pmatrix}$$

5.4 others

All the others are at most an array, which trivially only has one form.

5.5 Working Set

Our bottleneck is at the combined backward and optimize step.
We need all states of $\beta_j(t+1)$, $b_j(y_{t+1})$, $\alpha_i(t)$ and unfortunately all of a for a combined $0(N^2+3N)$.
We then store that in Γ_j, π_j, $b_j^*(y_t)$ and unfortunately all of a^* to get $0(N^2+3N)$.
Together they add up to $0(2N^2+6N)$.

6 the correct ordering

For the first loop, set gamma[i]=0, gammas[i]=1.
Start with the forward step. For t=1, you have to divide by gamma[i] for $b_i(y_1)$.
For t=2, compute α as usual, but save the a's divided by their gammas and the gammas of b; $a_ij = (gammas[i] - gamma[T]) * gammas[j]$. Following, compute all αs as usual, where we store them fully.
Next, we start to compute the other variables:
$\beta(T)$ gets initialized to 1.
We first compute $\eta_{ij}(t)$ and iterate over the second states first, then the first states. While iterating, we store the ηs at the correct place in a* and we add the η_{ij}s up, so that after all j's, we have γ_i, which we then can also add in the correct part of b*. Then we compute $\beta_i(t) = \gamma_i/\alpha_i(t)$. In the end we add $\gamma_i(t)$ to gammas[i] and we have to keep $\gamma_i(T)$ and $\gamma_i(1)$ for π_i.

At time T/2, we have to sum all of our $\gamma_i(T/2)$ over the N to get $P(Y|\Theta)$.
We have to change a with a^* and b with b^*.
Last, we have to test if we have converged with $P(Y|\Theta)$, if not we start at the top. If we did, we have to do the normalization of α, β with the gammas after the loop

7 Blocking

7.1 forward step

The forward step could benefit from blocking. In that step we multiply the row vector $\alpha(t)$ with the entire transition matrix. Thereby the transition matrix is accessed in column major order. This is a waste of memory accesses. In contrast to blocking in row major order, blocking in column major order would reduce the accesses to $\alpha(t)$.

8 possible optimizations

Blocking(?) No, since t is a barrier.
Loop enrolling state direction.
Vectorization in state direction, not in time direction.

9 Questions for TA

Initizalization good enough?
How to check implementation is correct?
Stable or general version?
Finishing criteria part of the Timing?