Baum-Welch

Team 035

Luca Blum Jannik Gut Jan Schilliger

May 23, 2020

The Baum-Welch Algorithm

- (local) Expectation Maximation of Hidden Markov Models
  - N Hidden states
  - K different signals
  - T time instances

The Baum-Welch Algorithm

- (local) Expectation Maximation of Hidden Markov Models

  N Hidden states
  K different signals
  T time instances

- Important matrices:

  a transition matrix [NxN]
  b emission matrix [NxK]
  π probabilities of first state [N]
  y observations [T]

The Baum-Welch Algorithm

- (local) Expectation Maximation of Hidden Markov Models
  - N Hidden states
  - K different signals
  - T time instances
- Important matrices:
  - a transition matrix [NxN]
  - b emission matrix [NxK]
  - π probabilities of first state [N]
  - y observations [T]
- Theoretical asymptotic run time:
  - Flops: $3KNT + 10N^2T$
  - Memory: $2KNT + 14N^2T$
  - Working set: $TN + 4T + N + K$

The Forward Step

- From t=1 to t=T
- Formulas:

$$\alpha_i(t) = \Pr[Y_1 = y_1, \ldots, Y_t = y_t, X_t = i|\theta]$$
$$\alpha_i(1) = \pi_i b_i(y_1)$$
$$\alpha_i(t+1) = b_i(y_{t+1}) \sum_{j=1}^{N} \alpha_j(t) a_{ji}$$

The Backward Step

- From t=T-1 to t=1
- Formulas:

$$\beta_i(t) = \Pr[Y_{t+1} = y_{t+1}, \ldots, Y_T = y_T | X_t = i, \theta]$$
$$\beta_i(T) = 1$$
$$\beta_i(t) = \sum_{j=1}^{N} \beta_j(t+1) a_{ij} b_j(y_{t+1})$$

The Update Step

$$\gamma_i(t) = \Pr[X_t = i | Y, \theta] = \frac{\alpha_i(t)\beta_i(t)}{\Pr[Y|\theta]}$$

$$\xi_{ij}(t) = \Pr[X_t = i, X_{t+1} = j | Y, \theta] = \frac{\alpha_i(t) a_{ij} \beta_j(t+1) b_j(y_{t+1})}{\Pr[Y|\theta]}$$

$$\pi_i^* = \gamma_i(1)$$

$$a_{ij}^* = \frac{\sum_{t=1}^{T-1} \xi_{ij}(t)}{\sum_{t=1}^{T-1} \gamma_i(t)}$$

$$b_i^*(v_k) = \frac{\sum_{t=1}^{T-1} \mathbb{1}_{y_t = v_k} \gamma_i(t)}{\sum_{t=1}^{T-1} \gamma_i(t)}$$

$\Pr[Y|\theta]$ for stopping criteria

If not stopping, repeat with a*, b* and π^*

Stability improvements

- α

$$\alpha'_t(i) = \Sigma_{j=1}^{N} \tilde{\alpha}_{t-1}(j) a_{ji} b_i(y_t)$$
$$c_t = \frac{1}{\Sigma_{i=1}^{N} \alpha'_t(i)}$$
$$\tilde{\alpha}_t(i) = c_t \alpha'_t(i)$$

- β

$$\beta'_t(i) = \Sigma_{j=1}^{N} \tilde{\beta}_{t+1}(j) a_{ij} b_j(y_{t+1})$$
$$\tilde{\beta}_i(i) = c_t \beta'_t(i)$$

- log for finishing criteria
- Maximal amount of steps

Stability improvements

- α

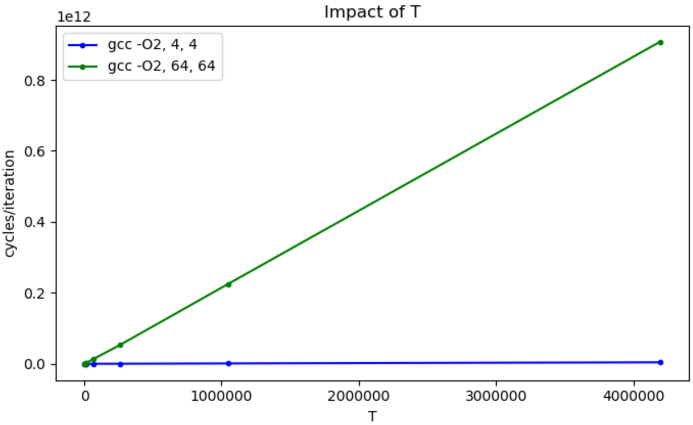
$$\alpha'_t(i) = \Sigma_{j=1}^{N} \tilde{\alpha}_{t-1}(j) a_{ji} b_i(y_t)$$
$$c_t = \frac{1}{\Sigma_{i=1}^{N} \alpha'_t(i)}$$
$$\tilde{\alpha}_t(i) = c_t \alpha'_t(i)$$

- β

$$\beta'_t(i) = \Sigma_{j=1}^{N} \tilde{\beta}_{t+1}(j) a_{ij} b_j(y_{t+1})$$
$$\tilde{\beta}_i(i) = c_t \beta'_t(i)$$

- log for finishing criteria
- Maximal amount of steps
- Tested version, that we measure everything against

Stable implementation

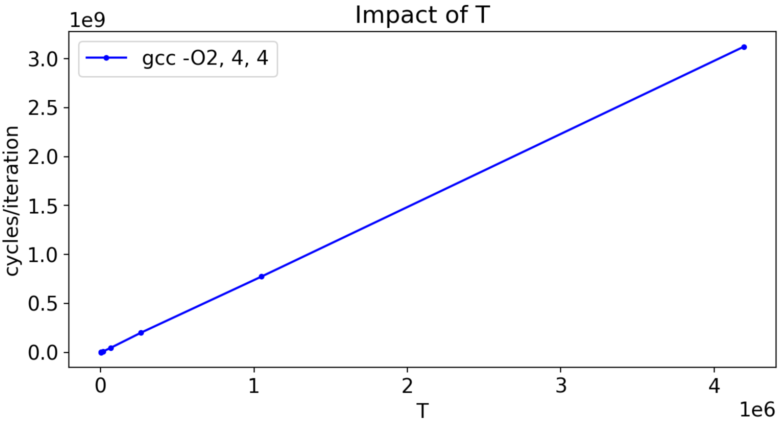


Warm cache, Intel Core i5-6200U, 2.3 GHz

Simple optimizations

- Scalar replacement
- Instruction weakening
- Inlining
- Pragma-ivdep (experiments)

C optimizations



Warm cache, Intel Core i5-3210M, 2.5 GHz

Code motion I

- Cancel constant denominator

$$\gamma_i(t) = \frac{\alpha_i(t)\beta_i(t)}{\boxed{\Pr[Y|\theta]}}$$

$$\xi_{ij}(t) = \frac{\alpha_i(t)a_{ij}\beta_j(t+1)b_j(y_{t+1})}{\boxed{\Pr[Y|\theta]}}$$

$$a^*_{ij} = \frac{\sum_{t=1}^{T-1} \boxed{\xi_{ij}(t)}}{\sum_{t=1}^{T-1} \boxed{\gamma_i(t)}}$$

$$b^*_i(v_k) = \frac{\sum_{t=1}^{T-1} \mathbb{1}_{y_t=v_k} \boxed{\gamma_i(t)}}{\sum_{t=1}^{T-1} \boxed{\gamma_i(t)}}$$

Code motion I

- Cancel constant denominator
- Only take sums, not every result

$$a_{ij}^* = \frac{\boxed{\sum_{t=1}^{T-1}} \xi_{ij}(t)}{\boxed{\sum_{t=1}^{T-1}} \gamma_i(t)}$$

$$b_i^*(v_k) = \frac{\boxed{\sum_{t=1}^{T-1}} \mathbb{1}_{y_t = v_k} \gamma_i(t)}{\boxed{\sum_{t=1}^{T-1}} \gamma_i(t)}$$

Code motion I

- Cancel constant denominator
- Only take sums, not every result
- Indicator function is only true for one of K values at t

$$b_i^*(v_k) = \frac{\sum_{t=1}^{T-1} \mathbb{1}_{y_t=v_k} \gamma_i(t)}{\sum_{t=1}^{T-1} \gamma_i(t)}$$

Code motion II

- re-use results

$$\beta_i(t) = \textstyle\sum_{j=1}^{N} \boxed{\beta_j(t+1) a_{ij} b_j(y_{t+1})}$$
$$\gamma_i(t) = \alpha_i(t) \boxed{\beta_i(t)}$$
$$\xi_{ij}(t) = \alpha_i(t) \boxed{a_{ij}\beta_j(t+1) b_j(y_{t+1})}$$
$$\gamma_i(t) = \frac{\sum_{j=1}^{T} \xi_{ij}}{\alpha_i(t)}$$

Code motion II

- re-use results
- save sums of γ and ξ directly at the right place
- "out of order" scaling of a in first real forward step
  - when a is first needed
- pre-computing of matrices (ab for backward & update)

Code motion effects

- Flops
  - Standard: $3KNT + 10N^2T$
  - Reordered: $KN^2 + 6N^2T$

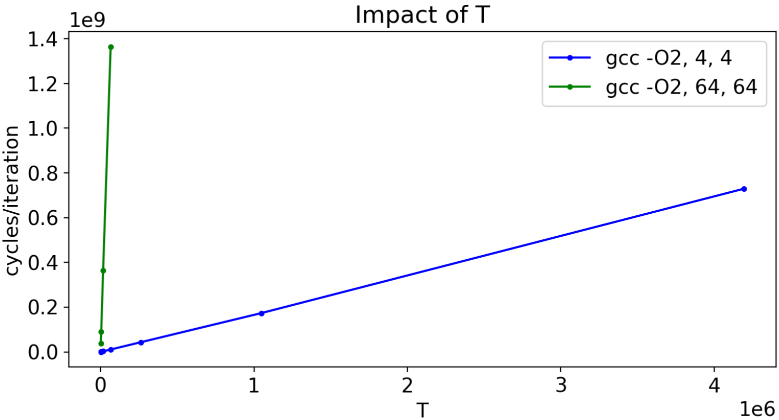
Code motion effects

- Flops
  - Standard: $3KNT + 10N^2T$
  - Reordered: $KN^2 + 6N^2T$
- Memory
  - Standard: $2KNT + 14N^2T$
  - Reordered: $3KN^2 + 6N^2T$

Code motion effects

- Flops
  - Standard: $3KNT + 10N^2T$
  - Reordered: $KN^2 + 6N^2T$
- Memory
  - Standard: $2KNT + 14N^2T$
  - Reordered: $3KN^2 + 6N^2T$
- Working set
  - Standard: $TN + 4T + N + K$
  - Reordered: $2N^2 + 6N$

Reordered graph

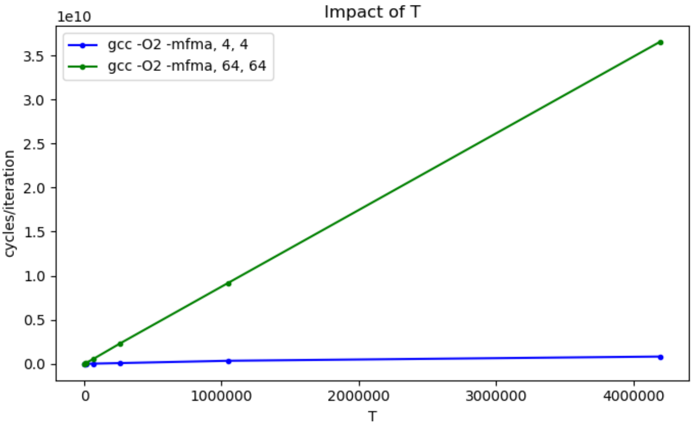


Warm cache, Intel Core i5-3210M, 2.5 GHz

Vectorization

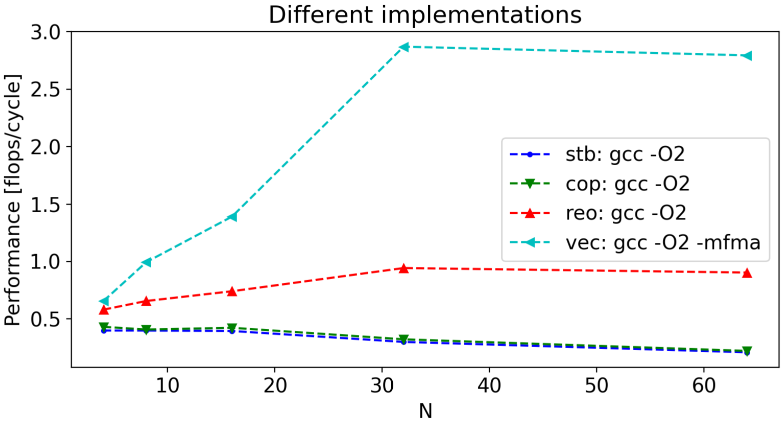
1. Unrolling
2. Blocking of transpose
3. Vectorization

Vectorization



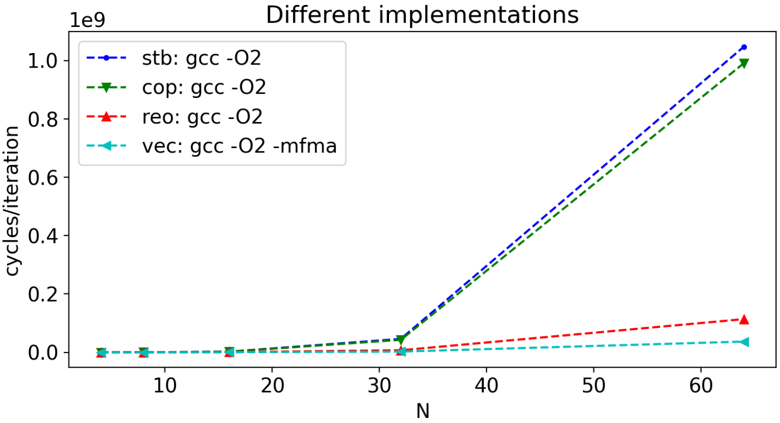
Warm cache, Intel Core i7 4790K @ 4.00 GHz

Comparison Performance



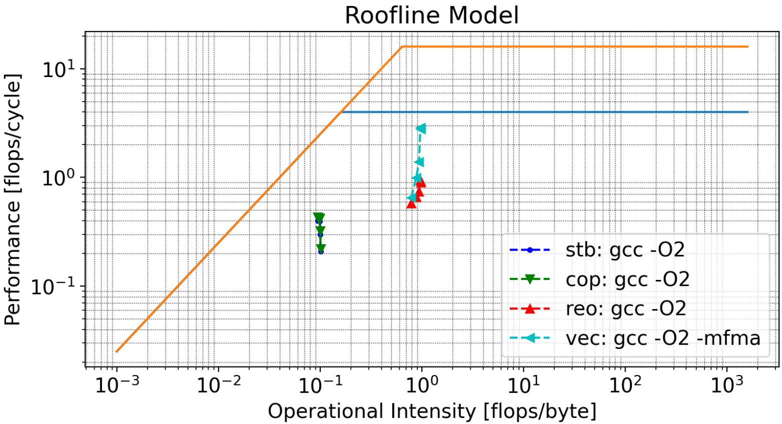
Warm cache, Intel Core i7 4790K @ 4.00 GHz

Comparison Cycles/Iteration



Warm cache, Intel Core i7 4790K @ 4.00 GHz

Comparison Roofline Plot



Warm cache, Intel Core i7 4790K @ 4.00 GHz

Other achievements

- Correct, stable
- No memory leaks
- Instruction count lowered by a factor of 10
- Perfect spatial locality (except for 2 transposes)
- Simple interface for testing with different parameters
- Three out of four people are still around

What can you anticipate for final report

- ICC comparison
- Effect of different parameters
- Comparison with third party library
- Comparison with other flags
- BLAS use in reordered version