Baysian Learning with Cuda

GPU Architecture

Marc Haubenstock (1525175)

June 26, 2016

1 HIdden Markov Model(HMM)

1.1 Overview

$S = s_1 s_2 \cdots s_N$	a set of N states.
$V = v_1 v_2 \cdots v_{\|V\|}$	a set of distinct observations symbols. $\|V\|$ denotes the number of distinct observations.
$O = o_1 o_2 \cdots o_T$	a sequence of T observations; each drawn from the observation symbol set V.
$Q = q_1 q_2 \cdots q_T$	a sequence of state; q_t denotes the state at time t.
$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1N} \\ a_{21} & a_{22} & \cdots & a_{2N} \\ \vdots & & & \\ a_{N1} & a_{N2} & \cdots & a_{NN} \end{bmatrix}$	a transition probability matrix A; each a_{ij} representing the probability of moving from state s_i to state s_j, i.e. $a_{ij} = P(q_{t+1} = s_j \mid q_t = s_i)$
$B = \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1\|V\|} \\ b_{21} & b_{22} & \cdots & b_{2\|V\|} \\ \vdots & & & \\ b_{N1} & b_{N2} & \cdots & b_{N\|V\|} \end{bmatrix}$	an emission probability matrix B, each b_{ij} representing the probability of the observation v_j being generated from a state s_i, i.e. $b_{ij} = P(o_t = v_j \mid q_t = s_i)$
$\pi = [\pi_1, \pi_2, \cdots, \pi_N]$	an initial state distribution: $\pi_i = P(q_1 = S_i)$

Figure 1: The notation of a HMM [1]

1.2 Forward

1.3 Viterbi

1.4 Baum-Welch

2 Implementation

2.1 Overview

As seen in the previous section it is not possible to fully parallize the individual algorithms due to their dependence on a value of a previous time t. However, hmms are usually computed with many observation sequences **M**. Each algorithm acts on one such sequence. The main idea presented in [1] exploits this by parallizing over **M** instead of T.

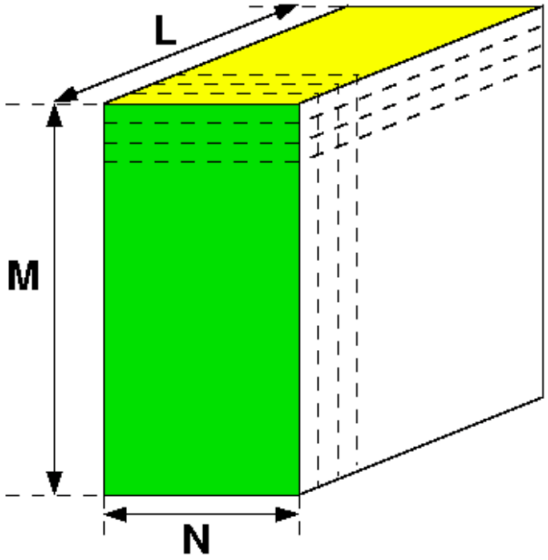

Figure 2: Graphical representation of the data structure presented in *A Cuda Implementation of Hidden Markov Models*[1]

Thus, the for-loop of the T domain remains in the code while the CUDA kernel computes all M slices at time t in parallel. Graphically this can be seen as moving through the 3D data structure either front-to-back, in the forward case, or back-to-front in the backward case. The current element is being computed by applying the following formulae.

$$D_{ij} = B_{O_i} . * C_i \times A_j$$

Where B_{O_i} is the row of the emission matrix at observation O_i, C_i is the previous slice and A_j is the j^{th} column of the transition. matrix A.

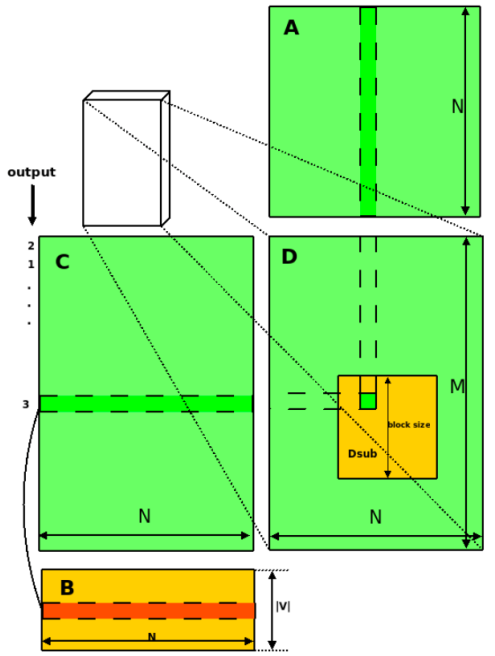

Figure 3: The computation of a slice[1]

2.2 Forward

```
initTrellis<<<M,N>>>(..);

for(int t = 0; t < T; t++){

...

computeBRow<<<M,N>>>(..); // copute B_o

pointwiseMul<<<M,N>>>(..); // W = B_o .* C_i

cublasDeviceMultiply(..); // wrapper function for cublas multiply; D = W x A

...

}
```

2.3 Viterbi

2.4 Baum-Welch

3 Performance Evaluation

References

[1] Chuan Liu. A cuda implementation of hidden markov models. 2009.