Entiol Liko: Assignment 2

enliko@student.ethz.ch

1 Theory

1.1 Question 1

1.1.1 Point a

$(\mathbb{R} \times \mathbb{R}, +)$ is a commutative monoid with identity element $< 0, 0 >$:

1. $< a, b > + < c, d > = < a + c, b + d > = < c, d > + < a, b >$
2. $< a, b > + < 0, 0 > = < a, b > = < 0, 0 > + < a, b >$
3. $(< a, b > + < c, d >) + < e, f > = < a + c, b + d > + < e, f > = < a + c + e, b + d + f >$
   $< a, b > + (< c, d > + < e, f >) = < a, b > + < c + e, d + f > = < a + c + e, b + d + f >$

$(\mathbb{R} \times \mathbb{R}, \times)$ is a monoid with identity element $< 1, 0 >$:

1. $< a, b > \times < 1, 0 > = < a \times 1, a \times 0 + b \times 1 > = < a, b > = < 1, 0 > + < a, b >$
2. $(< a, b > \times < c, d >) \times < e, f > = < ac, ad + bc > \times < e, f > = < ace, acf + ade + bce >$
   $< a, b > \times (< c, d > \times < e, f >) = < a, b > \times < ce, cf + de > = < ace, acf + ade + bce >$

Multiplication left and right distributes over addition:

1. $< a, b > \times (< c, d > + < e, f >) = < a, b > \times < c + e, d + f > = < ac + ae, ad + af + bc + be >$
   $(< a, b > < c, d >) + (< a, b > < e, f >) = < ac, ad + bc > + < ae, af + be > = << ac + ae, ad + af + bc + be >>$
2. $(< a, b > + < c, d >) \times < e, f > = < a + c, b + d > \times < e, f > = < ae + ce, af + fc + be + de >$
   $(< a, b > < e, f >) + (< c, d > < e, f >) = < ae, af + be > + < ce, cf + de > = < ae + ce, af + fc + be + de >$

Multiplication by 0 annihilates $\mathbb{R} \times \mathbb{R}$:

1. $< a, b > \times < 0, 0 > = < a0, a0 + b0 > = < a0, a0 + b0 > = < 0, 0 > \times < a, b >$

1.1.2 Point b

We will start by defining some symbols so we don't have to write the whole notation. Let $\omega(t_n^i, t_m^j) := exp(score_\theta(< t_n^i, t_m^j >, \omega))$ and $\alpha(t_n^i, t_m^j) := score_\theta(< t_n^i, t_m^j >, \omega)$

We can think of having a work of length N and of a set of tags of length T. For the last but one layer we have:

$\beta(t_{N-1}^i, N-1) = < \omega(t_{N-1}^i, t_N^0), -\omega(t_{N-1}^i, t_N^0)\alpha(t_{N-1}^i, t_N^0) >$

We can calculate the same thing for the last to second layer.

$\beta(t^i_{N-2}, n) = \sum_{j=0}^{T} < \omega(t^i_{N-2}, t^j_{N-1}), -\omega(t^i_{N-2}, t^j_{N-1})\alpha(t^i_{N-2}, t^j_{N-1}) > \times$
$< \omega(t^j_{N-1}, t^0_N), -\omega(t^j_{N-1}, t^0_N)\alpha(t^j_{N-1}, t^0_N) >= ... =$
$< \sum_{j=0}^{T} exp(\alpha(t^i_{N-2}, t^j_{N-1}) > + \alpha(t^j_{N-1}, t^0_N)), -exp(\alpha(t^i_{N-2}, t^j_{N-1}) + \alpha(t^j_{N-1}, t^0_N))(\alpha(t^i_{N-2}, t^j_{N-1}) +$
$\alpha(t^j_{N-1}, t^0_N)) >$

It can be easily seen that if we continue the procedure the second component of the result is $H_U(T_\omega)$, what we are looking for.

1.1.3 Point c

$H(T_\omega) = -\sum_{t \in T} p(t|\omega) log(p(t|\omega)) = -\sum_{t \in T} \frac{exp(score_\theta(t,\omega))}{Z(\omega)} (score_\theta(t, \omega) - log(Z(\omega))) =$
$\frac{\sum_{t \in T} exp(score_\theta(t,\omega) score_\theta(t,\omega)}{Z(\omega)} + log(Z(\omega)) = \frac{H_U(T_\omega)}{Z(\omega)} + log(Z(\omega))$

1.1.4 Point d

Since we can compute $H_U(T_\omega)$ and Z_ω in $\mathcal{O}(N|T|^2)$ we can calculate $H(T_\omega)$ in $\mathcal{O}(N|T|^2)$. We can compute the gradient of $H_U(T_\omega)$ in $\mathcal{O}(N|T|^2)$ if we assume that we can calculate the gradient of the score function with respect to the parameters θ. In fact we could reuse the backlog algorithm but with a different semiring. For example to calculate the gradient of Z we can use the expectation semiring where the elements are vectors where each component is a tuple ¡w, derivate of w with respect to the i-th parameter¿.

1.2 Question 2

1.2.1 Point a

We could do a proof by induction. We can clearly see that the algorithm is true when we have a single node, the BOT node. We assume that the algorithm is true and correct up the last popped node and we have to prove that it holds for the next popped node. We can do a proof by contradiction and assume that there exists another path that passes by a node which is not popped that connects BOT with the current popped node with a better score that the current path. If we prove that this cannot be true than we have finished the proof. The proof of this contradiction is pretty straightforward since it is very similar to the proof that the Dijkstra algorithm for shortest path is correct. In essence we show that since we have negative weights each node that is added to the path lowers the score and so if there exist a more optimal path then in that path there is a node with a higher score than the score of the node we just popped (because of negative weights which are specified by the ring) but that is a contradiction so there can't exists a more optimal path.

1.2.2 Point b

I think that we can easily see that both algorithms return the same result by noting that the + and max operations are commutative. This means that it doesn't matter if we start

from the left or from the right. In the end we calculate, for each node, the score of the best part-of-speech tagging for that node and that score is unique so we get the same result. From the result of point a we can say that the algorithm works for every one, in the sense that it calculates the best tagging from the BOT to that node and since the path is unique is returns the same result as Viterbi.

1.2.3 Point c

We assume that the insert and pop operations for the PriorityQueue have a time complexity equal to $\mathcal{O}(logV)$, where V is the number of elements in the queue. The time complexity of Dijkstra is $\mathcal{O}(N|T|^2logV)$, where N is the number of words in the sentence, T the number of possible tags and V the number of elements in the priority queue. We have to do a push for every "connection" between nodes . There are N*T nodes and T connections for each node. That gives us the this time complexity. In the case of the Viterbi algorithm runs in $\mathcal{O}(N|T|^2)$. The benefit of using Dijkstra over Viterbi is that Viterbi visits every single edge but Dijkstra may reach the end, so best tagging sequence, without necessarily having visited every edge so even if the time complexity is greater in reality it may finish sooner.

1.2.4 Point d

I think that this operation is not adequate for what we are trying to do. We are sertching for the best tagging but the problem is that the using this operation gives us a value which is closest to the smallest value not the biggest.

1.2.5 Point e

In our setting the best path represents the path with the highest score. What we need is a semiring for which the proof of point a is still true. We could write that we want that if $a = a \oplus b => a = a \oplus (b \otimes c)$, where $\oplus$ is max or min. What we are basically saying is that if one node is has a larger value then another node the path that passes from the second node and end with the first node will have a lower score than the optimal path from BOT to the first node. This way we guarantee that the proof of point a is correct. This is the reason the other semirings specified in the question do not work. Note: The semiring $< R_{\geq 0}, min, +, +\infty, 0 >$ used with Dikjstra returns the path with the lowest score.

2 Practice

For the practical part of the algorithms were vectorized using a for loop and the implementation was done using the pseudo code found in the course notes.