Johanna Ribas: Assignment 2

jribas@student.ethz.ch

12/12/2022 - 09:37h

Question 1: Entropy of a Conditional Random Field

a) For $< RxR, \oplus, \otimes, \mathbf{0}, \mathbf{1} >$ to be a semiring, we need to proof the following 4 propoerties:

1. $< R \times R, \otimes, 0 >$ is a commutative monoid.

Distributivity:

$(A \oplus B) \oplus C = \langle A_1 + B_1, A_2 + B_2 \rangle \oplus \langle C_1, C_2 \rangle = \langle A_1 + B_1 + C_1, A_2 + B_2 + C_2 \rangle = \langle A_1, A_2 \rangle \oplus \langle B_1 + C_1, B_2 + C_2 \rangle = A \oplus (B \oplus C)$

Anihilator: $O \oplus A = \langle 0 + A_1, 0 + A_2 \rangle = \langle A_1, A_2 \rangle = A$

Commutativity: $A \oplus B = \langle A_1 + B_1, A_2 + B_2 \rangle = \langle B_1 + A_1, B_2 + A_2 \rangle = B \oplus A$

2. $< R \times R, \otimes, 1 >$ is a monoid

$(A \otimes B) \otimes C = \langle A_1 \cdot B_1, A_1 \cdot B_2 + A_2 \cdot B_1 \rangle \otimes \langle C_1, C_2 \rangle =$

$\langle (A_1 \cdot B_1) C_1, (A_1 \cdot B_1) C_2 + (A_1 \cdot B) C_1 + (A_2 \cdot B_1) \cdot C_1 \rangle =$

$\langle A_1 \cdot (B_1 \cdot C_1), A_1 \cdot (B_1 \cdot C_2 + B_1 \cdot C_1) + A_2 \cdot (B_1 \cdot C_1) \rangle =$

$\langle A_1, A_2 \rangle \otimes (\langle B_1, B_2 \rangle \otimes \langle C_1, C_2 \rangle)$

$1 \otimes A = \langle 1, 0 \rangle \otimes \langle A_1, A_2 \rangle = \langle 1 \cdot A_1, 1A_2 + 0 \cdot A_1 \rangle = \langle A_1, A_2 \rangle = A$

3. Distributivity of the product over the sum.

$A \otimes (B \oplus C) = A \otimes (\langle B_1 + C_1, B_2 + C_2 \rangle) =$
$= \langle A_1 \cdot (B_1 + C_1), A_1 \cdot (B_2 + C_2) + A_2 (B_1 + C_1) \rangle$
$= \langle A_1 B_1 + A_1 \cdot C_1, A_1 \cdot B_2 + A_2 B_1 + A_1 C_2 + A_2 \cdot C_1) =$
$= (A \otimes B) \oplus (A \otimes C)$

4. 0 is the anihilator

$0 \otimes A = \langle O \cdot A_1, O A_2 + O \cdot A_1 \rangle = A \otimes 0 = 0$

b) We will use the deffinition of the entropy and the probability distribution of a CRF.

$H(T_w) = -\sum_{t_{1:N}\in\mathcal{T}^{\mathcal{N}}} p(t)\log(p(t)) = -\sum_{t_{1:N}\in\mathcal{T}^{\mathcal{N}}} \frac{\exp(score)}{z}\log\frac{\exp(score)}{z} =$

$= \frac{1}{z}(-\sum_{t_{1:N}\in\mathcal{T}^{\mathcal{N}}} \exp(score)(\log(\exp(score)) - \log z)) =$

$= \frac{1}{z}(H_u + \sum_{t_{1:N}\in\mathcal{T}^{\mathcal{N}}} \exp(score)\log(z)) = \frac{1}{z}H_u + \log z(\omega)$

c) From the deffinition of $\oplus$, we can easily see that:
$\bigoplus_{i=1}^{N} \langle X_i, X_i' \rangle = \left\langle \sum_{i=1}^{N} X_i, \sum_{i=1}^{N} {X_i}' \right\rangle$

From the deffinition of $\otimes$, we derive:

$\bigotimes_{i=1}^{N} \langle x_i, x_i' \rangle = \langle x_1, x_2, x_1x_2 + x_1x_2 \rangle \otimes \left(\bigotimes_{i=3}^{N} \langle x_i, x_i' \rangle \right)$

$= \langle x_1x_2x_3, x_1x_2x_3' + (x_1x_2' + {x_1}'x_2)\cdot x_3 \rangle \otimes \left(\bigotimes_{i=4}^{N} \langle x_1, x_1' \rangle \right)$

$= \cdots = \left\langle \prod_{i=1}^{N} x_i, \prod_{i=1}^{N} x_i \prod_{\substack{j=1 \\ j\neq i}}^{N} x_j \right\rangle$

Now, if we combine what we derived above with what the backward algorithm computes, we obtain:

$$\bigoplus_{\mathbf{t}_{1:N}\in\mathcal{T}^N} \bigotimes_{i=1}^{N} \langle \omega_{\mathbf{t},i}, -\omega_{\mathbf{t},i}\log(\omega_{\mathbf{t},i}) \rangle =$$

$$\left\langle \sum_{\mathbf{t}_{1:N}\in\mathcal{T}^N} \prod_{i=1}^{N} \omega_{\mathbf{t},i}, \sum_{\mathbf{t}_{1:N}\in\mathcal{T}^N} \sum_{i=1}^{n} -\omega_{\mathbf{t},i}\log(\omega_{\mathbf{t},i}) \prod_{j=1, j\neq i}^{n} \omega_{\mathbf{t},j} \right\rangle$$

If we then introduce the deffinition of $\omega_{\mathbf{t},i} = \exp\{\text{score}(t_{i-1}, t_i, \mathbf{w})\}$. Then the second element of the tuple becomes:

$$\sum_{\mathbf{t}_{1:N}\in\mathcal{T}^N}\sum_{i=1}^{n}-\omega_{\mathbf{t},i}\log\left(\omega_{\mathbf{t},i}\right)\prod_{j=1,j\neq i}^{n}\omega_{\mathbf{t},j}$$

$$=-\sum_{\mathbf{t}_{1:N}\in\mathcal{T}^N}\sum_{i=1}^{n}\exp\left\{score_{\mathbf{t},i}\right\}score_{\mathbf{t},i}\prod_{j=1,j\neq i}^{n}\exp\left\{score_{\mathbf{t},j}\right\}$$

$$=-\sum_{\mathbf{t}_{1:N}\in\mathcal{T}^N}\sum_{i=1}^{n}score_{\mathbf{t},i}\prod_{j=1}^{n}\exp\left\{score_{\mathbf{t},j}\right\}$$

$$=-\sum_{\mathbf{t}_{1:N}\in\mathcal{T}^N}\prod_{j=1}^{n}\exp\left\{score_{\mathbf{t},j}\right\}\sum_{i=1}^{n}score_{\mathbf{t},i}$$

$$=-\sum_{\mathbf{t}_{1:N}\in\mathcal{T}^N}\exp\left\{\sum_{j=1}^{n}score_{\mathbf{t},j}\right\}\sum_{i=1}^{n}score_{\mathbf{t},i}$$

$$=-\sum_{\mathbf{t}_{1:N}\in\mathcal{T}^N}\exp\left\{\mathrm{score}_\theta(\mathbf{t},\mathbf{w})\right\}\mathrm{score}_\theta(\mathbf{t},\mathbf{w})=\mathrm{H_U}\left(\mathrm{T_w}\right)$$

d) We have proved that with the above lifting strategy in the expectation semiring, we can compute the unnormalized entropy of the CRF with the generalized viterbi algorithm. To compute the $H(T_w)$ we will have to run the algoirthm twice, once to compute the unnormalized entropy and one to compute the log-normalizer. Each of these can be computed in $O(N|T|^2)$ and thus te total runtime will be $O(2N|T|^2) = O(N|T|^2)$.

The gradient of $H(T_w)$ can be calculated with backpropagation. The bigo of the runtime of the gradient of a function is equal to the bigo of the runtime of the function. For this, the gradient of the entropy can also be run in $O(N|T|^2)$.

Question 2: Decoding a CRF with Dijkstras Algorithm

a) Because any score s_i must be 0 or negative, the sum of scores will decrease when adding new terms. Thus, any subtagging will have a higher score than a complete tagging.

Let $<< n, t^a >, s^a >$ be the tuple of the first complete tagging T_a. We assume that another complete tagging exists T_b with tuple $<< n, t^b >, s^b >$, where $s^b > s^a$.

If one of the tupples representing a subtagging of T_b was popped, then the following tupples of this tagging would be recursively pushed and popped.

For any subtagging with $n' < n$ and score $s^{b\prime}$, the tuple of T_b would be popped before the tupple of T^a because $s^{b\prime} > s^b > s^a$. Consequently, T^a would never be popped from the queue. This proofs that the score of the first complete tagging will be the highest and no subsequently popped scores will be higher.

b) In the viterbi algorithm, for every word in the sequence, starting from the BOS and going backwards, $n \in (N-1, ..., 1)$ and for every tag in the tagset $t \in \mathcal{T}$, we compute the Viterbi variable:

$\gamma_v[n,t] = \max_{t' \in \mathcal{T}} \text{score}(t, t', w) \otimes \gamma_v[n+1, t']$ Where t corresponds to the tag at the current layer n and t' corresponds to the tag at layer n+1.

In Dijkstra's algorithm, for all tuple $< n, t >$ we push the tuple:
$<< n+1, t' >, \text{score}(t, t', w) \otimes \gamma_d[n+1, t'] >$. If the tuple is already present in the queue, when we push it to the same tuple we are updating it with the max score. If we compute until the queue is empty, for every tag and at each layer we are computing:

$\gamma_v[n,t] = \max_{t' \in \mathcal{T}} \text{score}(t, t', w) \otimes \gamma_d[n+1, t']$.

Both algorithms will have computed the same values, $\gamma_d[n,t] = \gamma_v[n,t]$ for all $t \in \mathcal{T}$

c) Viterbi runs in O(NT^2), because at each node it calculates the score for T tags. There are T nodes in each layer (i.e., T^2) and N layers (NT^2).

In Dijkstra's algorithm, we set the Priority Queue that returns best scores first. The slowest case would be the one in which all the scores at each transition are the same except for the last tag. Then there would be no early stop and we would loop (line 13) for all $n \in (N-1, ..., 1)$ and $t \in \mathcal{T}$. This would be $O(NT)$. For each pushed tuple $< n, t >$, $O(T)$ elements are added to the queue. Because the runtime of one push to the queue takes $O(log(T))$, this would result in $O(Tlog(T))$ for each pushed tuple. The runtime is then $O(NT^2log(T))$. However, this is the worst case scenario for Dijsktras but in cases where the highest scoring path is more obvious, the running time can be decreased. On the contrary, the runtime for the Viterbi algorithm would stay the same even if there was one clear winning path.

d) It will not calculate the correct tagging. For Dijkstra's to work, it has to hold that the sum of scores decreases monotonically. Then the tuple is only popped when the popped score is equal to the $\oplus$ sum over all the tags. In the proposed semiring, the scores will be updated with the $\oplus_{log}$ sum and the result is also in R,i.e., it can be negative or positive. Thus, when pushing a new tuple, it will not always hold that the sum of scores will decrease and thus Dijkstra's will not work.

e) We assume that the PriorityQueue returns the best scores first according to the partial ordering of the semiring set, and thus returns a maximum.

If this is the case, then the semiring should have an $\otimes$ operator such that:
$\bigotimes_{i=0}^{n} s_i \leq \bigotimes_{i=0}^{n-1} s_i$
Where s_i is the score calculated in line 9 of Dijkstra's pseudocode ($\gamma[n,t]$). This ensures that every time we push a new tuple, the acumulated score will only decrease.

Question 3: Coding a Neural CRF for tagging

Link to notebook:

`https://colab.research.google.com/drive/1Qs0AYQbUKcVVPVxHEMPp2tcG7PQOjyhE`

*Note that I only completed sections a), b) and c) because, after having worked on it for weeks, I ran out of time to complete it all.