Prof. Ryan Cotterell

Assignment 2: Conditional Random Fields

04/11/2022 - 09:37h

The first half of the assignment is a series of theoretical questions about the material. When you have answered the questions to your satisfaction, you should upload a pdf file with your solutions (preferably written in LaTeX) to Moodle. The second question is a coding task that is to be solved in the released Google Colab notebook. You have to copy the notebook to your own drive in order to edit it. When submitting your solution, please execute all cells in your copied notebook, then save it and include a link to your notebook in your PDF file submission. We will run software on your submission to *test for plagiarism.*

Important: Please ensure, that all cells in the notebook are already **executed** and that the notebook is accessible via the shared link in **Editor mode**! The notebook must also be executable from top to bottom without throwing errors.

1 Theory

Question 1: Entropy of a Conditional Random Field (25 pts)

Entropy is a fundamental mathematical quantity in various scientific and engineering disciplines. In this question, we discuss its computation in the context of an NLP model—a conditional random field (CRF) for part-of-speech tagging. Consider a discrete random variable X with values taken from $\mathcal{X}$. We define **entropy** as

$$\mathrm{H}(\mathrm{X}) \overset{\text{def}}{=} -\sum_{x\in\mathcal{X}} p(x)\log p(x) \tag{1}$$

We can easily devise a brute-force $\mathcal{O}\left(|\mathcal{X}|\right)$ algorithm to directly compute Eq. (1). However, in the case of a conditional random field (CRF), applying this naïve algorithm to the computation of entropy leads to an intractable algorithm. Recall from lecture that a CRF can most easily be understood as a log-linear model over a lattice structure.

Definition 1.1 (Conditional Random Fields)**.** Let $\mathcal{T}$ be an alphabet of tags, e.g., in the case of part-of-speech tags we have $\mathcal{T} = \{\textsc{noun}, \textsc{verb}, \ldots\}$. Further, let $\mathcal{W}$ be a vocabulary of possible words, e.g., $\mathcal{W} = \{\textsc{i}, \textsc{like}, ...\}$. Suppose we consider a sentence of length $|\boldsymbol{w}| = N$, then $\mathcal{T}^N$ is the set of all $|\mathcal{T}|^N$ part-of-speech taggings and $\mathcal{W}^N$ is the set of all $|\mathcal{W}|^N$ word sequences, i.e., $\boldsymbol{w} \in \mathcal{W}^N$ and $\boldsymbol{t} \in \mathcal{T}^N$. Now, given a score function $\mathrm{score}_{\boldsymbol{\theta}}$, we define a CRF:

$$p(\boldsymbol{t} \mid \boldsymbol{w}) = \frac{\exp \mathrm{score}_{\boldsymbol{\theta}}(\boldsymbol{t}, \boldsymbol{w})}{\sum_{\boldsymbol{t}'\in\mathcal{T}^N} \exp \mathrm{score}(\boldsymbol{t}', \boldsymbol{w})} = \frac{\exp \mathrm{score}_{\boldsymbol{\theta}}(\boldsymbol{t}, \boldsymbol{w})}{Z(\boldsymbol{w})} \tag{2}$$

where we define the **partition function** as

$$Z(\boldsymbol{w}) \overset{\text{def}}{=} \sum_{\boldsymbol{t}' \in \mathcal{T}^N} \exp \mathrm{score}_{\boldsymbol{\theta}}(\boldsymbol{t}', \boldsymbol{w}) \tag{3}$$

In order to be able to compute $Z(\boldsymbol{w})$ efficiently, we define a score function that decomposes additively over tag bigrams, i.e., we define

$$\mathrm{score}_{\boldsymbol{\theta}}(\boldsymbol{t}, \boldsymbol{w}) \overset{\text{def}}{=} \sum_{n=1}^{N} \mathrm{score}_{\boldsymbol{\theta}}(t_{n-1}, t_n, \boldsymbol{w}) \tag{4}$$

where t_0 is a distinguished beginning-of-tagging symbol.

We will also use the notation $\mathrm{T}_{\boldsymbol{w}}$ to refer to the discrete $\mathcal{T}^N$-valued random variable distributed according to $p(\boldsymbol{t} \mid \boldsymbol{w})$. We view $\mathrm{score}_{\boldsymbol{\theta}} : \mathcal{T} \times \mathcal{T} \times \mathcal{W}^N \to \mathbb{R}$ as a user-defined function parametrized by $\boldsymbol{\theta}$. The function $\mathrm{score}_{\boldsymbol{\theta}}$ tells us how good the combination of $\langle t_{n-1}, t_n, \boldsymbol{w} \rangle$ is with higher values indicating better combinations. In modern NLP, $\mathrm{score}_{\boldsymbol{\theta}}$ is nearly always a neural network, and this will be the case in the second half of this assignment.

The goal of this question is to for you to develop a dynamic program for the efficient calculation of the entropy of a CRF. In the practical section, you will also be asked to implement the algorithm and use it to train an entropy-regularized CRF for part-of-speech tagging. To develop our efficient algorithm, we start with the introduction of a semiring that you did not see in class.

Definition 1.2 (Expectation semiring)**.** Let x, y, x', y' be real numbers. Define the following operations over the set of all pairs of real numbers $\mathbb{R} \times \mathbb{R}$:

$$\langle x, y \rangle \oplus \langle x', y' \rangle \overset{\text{def}}{=} \langle x + x', y + y' \rangle \tag{5}$$

$$\langle x, y \rangle \otimes \langle x', y' \rangle \overset{\text{def}}{=} \langle x \cdot x', x \cdot y' + y \cdot x' \rangle \tag{6}$$

$$\mathbf{0} \overset{\text{def}}{=} \langle 0, 0 \rangle \tag{7}$$

$$\mathbf{1} \overset{\text{def}}{=} \langle 1, 0 \rangle \tag{8}$$

The semiring $\langle \mathbb{R} \times \mathbb{R}, \oplus, \otimes, \mathbf{0}, \mathbf{1} \rangle$ is called the **expectation semiring**.

a) **(5 pts)** Prove that the expectation semiring (Definition 1.2) satisfies the semiring axioms.

b) **(5 pts)** The **unnormalized entropy** of a CRF is defined as

$$\mathrm{H}_{\mathrm{U}}(\mathrm{T}_{\boldsymbol{w}}) = - \sum_{\boldsymbol{t} \in \mathcal{T}^N} \exp(\mathrm{score}_{\boldsymbol{\theta}}(\boldsymbol{t}, \boldsymbol{w}))\ \mathrm{score}_{\boldsymbol{\theta}}(\boldsymbol{t}, \boldsymbol{w}) \tag{9}$$

Now, suppose we lift a CRF into the expectation semiring where each arc gets the weight $\omega \mapsto \langle \omega, -\omega \log \omega \rangle$ where ω was the original arc weight. Prove that running the forward or backward algorithm in the expectation semiring with the above lifting strategy computes the unnormalized entropy given in Eq. (9).

c) **(10 pts)** Prove the following identity

$$\mathrm{H}(\mathrm{T}_{\boldsymbol{w}}) = Z(\boldsymbol{w})^{-1}\,\mathrm{H}_{\mathrm{U}}(\mathrm{T}_{\boldsymbol{w}}) + \log Z(\boldsymbol{w}) \tag{10}$$

In words, the above identity says that, given the log-normalizer and the unnormalized entropy of CRF, we can compute its entropy by plugging those values into Eq. (10).

d) **(5 pts)** Complete the argument by proving that $\mathrm{H}(\mathrm{T}_{\boldsymbol{w}})$ can be computed in $\mathcal{O}\left(N \cdot |\mathcal{T}|^2\right)$, i.e., in the same amount of time it takes to compute $\log Z(\boldsymbol{w})$. How quickly can we compute the gradient of $\mathrm{H}(\mathrm{T}_{\boldsymbol{w}})$ with respect to the CRF's parameters? A big-O bound suffices.

Question 2: Decoding a CRF with Dijkstra's Algorithm (25 pts)

In this question, you will use Dijkstra's algorithm to compute the best part-of-speech tagging under a CRF for a sentence $\boldsymbol{w}$.

Definition 1.3 (Arctic semiring)**.** The semiring $\langle \mathbb{R} \cup \{-\infty\}, \max, +, -\infty, 0\rangle$ is called the **Arctic semiring**.

In class, we discuss how to compute the best tagging using the Viterbi algorithm, i.e., the forward or backward algorithm over the arctic semiring (Definition 1.3).

This question explores when it may be advantageous to use Dijkstra's algorithm. Consider the pseudocode given below in the semiring notation for Dijkstra's.

Algorithm 1 Dijkstra's algorithm for finding the highest-scoring part-of-speech tagging under a CRF.

```
1.  def Dijkstra($\mathcal{T}, \boldsymbol{w}, R = \langle A, \oplus, \otimes, \mathbf{0}, \mathbf{1}\rangle$, stop_early):   ▷$R$ is a semiring that binds $\oplus$ and $\otimes$ throughout; stop_early is a boolean indicating whether we want to stop early
2.      popped ← {}
3.      queue ← new PriorityQueue()   ▷PriorityQueue returns 'best' scores first, according to the partial ordering of $R$.
4.      $\boldsymbol{\gamma} \leftarrow \mathbf{0}$   ▷Initialize forward values; $\gamma \in A^{(|w|+1)\times|\mathcal{T}|}$
5.      push $\langle\langle 0, \text{BOT}\rangle, \mathbf{1}\rangle$ to queue   ▷Initialize the queue to have a beginning-of-tagging symbol BOT
6.      while |queue| > 0 :
7.          pop$\langle\langle n, t\rangle, \text{score}\rangle$ from queue
8.          add $\langle n, t\rangle$ to popped
9.          $\gamma[n, t] \leftarrow$ score
10.         if $n = |\boldsymbol{w}|$ and stop_early :   ▷If our partial tagging is of length $|\boldsymbol{w}|$, stop early
11.             return score
12.         if $n < |\boldsymbol{w}|$ :
13.             for $t' \in \mathcal{T}$ :
14.                 if $\langle n+1, t'\rangle$ not in popped :   ▷Only push new tuples
15.                     push $\left\langle \langle n+1, t'\rangle, \text{score}(t, t', \boldsymbol{w}) \otimes \gamma[n, t] \right\rangle$ to queue
16.     return $\boldsymbol{\gamma}$   ▷Return forward values
```

Note that on line ten to eleven, we have included an early stopping condition that allows the algorithm to stop before all of $\boldsymbol{\gamma}$ has been computed.

Note that when we say **push**, for example in line five and 15, we would like to add a new mapping to our PriorityQueue from a key $\langle n, t\rangle$ to some value x. If the key is already present, we update it using $\oplus$ in our semiring, i.e., PriorityQueue$[\langle n, t\rangle]$ = PriorityQueue$[\langle n, t\rangle] \oplus x$. Otherwise (i.e., if the key was not previously present), we simply add it to PriorityQueue (that is, we create a new mapping from the new key to the new value). **push** runs in log(|PriorityQueue|) time, where |PriorityQueue| is equal to the number of mappings it contains. You may also refer to this implementation of a priority queue in Python to get a better understanding of its semantics:

`https://github.com/rycolab/aflt-f2022/blob/main/rayuela/base/datastructures.py#L234`

To get an idea of what running Algorithm 1 may look in Python, please refer to:

`https://github.com/rycolab/aflt-f2022/blob/main/rayuela/fsa/pathsum.py#L213`

For subquestions a), b) and c), you may assume that our CRF has been lifted into a variant of the arctic semiring R suitable for running Dijkstra to compute the best tagging for a given word sequence $\boldsymbol{w}$: $R = \langle \mathbb{R}_{\leq 0} \cup \{-\infty\}, \max, +, -\infty, 0\rangle$. **Hint:** Think carefully about the difference between $\mathbb{R}_{\leq 0}$ and $\mathbb{R}$ in this case; it will be relevant for question e). Throughout this exercise, you may assume that score is such that its result conforms to the domain of R's set.

a) **(5 pts)** Prove the following fact about Dijkstra's algorithm: The score for the first complete tagging, popped from the priority queue is the score for the best part-of-speech tagging under the CRF.

b) **(5 pts)** Show that if you keep running Dijkstra's until the queue is empty, then Dijkstra's has computed all the same values that Viterbi would have, i.e., the array $\boldsymbol{\gamma}$ both algorithms compute are the same.

   Hint: You may ignore the fact forward Viterbi is usually instantiated with the initial scores score$(\textsc{bot}, t', \boldsymbol{w})\,\forall t' \in \mathcal{T}$ directly, and will thus have one less row than γ produced by Dijkstra's here (i.e., effectively, you can just ignore the first row of γ produced by Dijkstra's).

c) **(5 pts)** Compute a runtime bound for Dijkstra's. If not already familiar with priority queues, you may wish to read about them on `https://en.wikipedia.org/wiki/Priority_queue`. Note that this question is *not* asking how fast Dijkstra's can be made to run, but rather, given a specific implementation of a priority queue, which is up to you, how fast *does* it run. Compare this runtime to that of Viterbi in order to answer when you would want to run Dijkstra's instead of Viterbi. Is there any advantage of Dijkstra's that is not given by the runtime bound?

d) **(5 pts)** Now let us suppose that we change the semiring. First, we define:

$$x \oplus_{\log} y \overset{\text{def}}{=} -\log(\exp(-x) + \exp(-y)) \tag{11}$$

   Does Dijkstra's correctly calculate the best part-of-speech tagging in the $R = \langle \mathbb{R} \cup \{-\infty, +\infty\}, \oplus_{\log}, +, +\infty, 0\rangle$ semiring? A qualitative answer suffices.

e) **(5 pts)** Come up with a class of semirings such that when we Dijkstra's over that semiring and we stop early, the algorithm provably returns the best path.

Hint: Consider a version of of the tropical semiring $\langle \mathbb{R} \cup \{\infty\}, \min, +, \infty, 0\rangle$. What goes wrong? Also compare the semirings $\langle \mathbb{R}_{\leq 0} \cup \{-\infty\}, \max, +, -\infty, 0\rangle$ (note that this is the variant of the arctic semiring we previously used) and $\langle \mathbb{R} \cup \{-\infty\}, \max, +, -\infty, 0\rangle$.

2 Practice

Question 3: Coding a Neural CRF for Tagging (50 pts)

In this question, we ask you to code a Neural CRF in Python for part of speech tagging in a Google Colab notebook:

`https://colab.research.google.com/drive/1p8Q-WIemWLcxAXjjco4QkDWXFbOnL--V?usp=sharing`

In principle, you only need to implement the methods that currently raise `NotImplementedError`; for the bonus question, you may also change hyperparameters and other aspects of the `NeuralCRF` class.

Note: You will be working with the English language version of Universal Dependencies dataset [Nivre et al., 2017]. Do not modify the `torchtext` pipeline and write only code as necessary for the notebook to work based on the skeleton.

a) **(10 pts)** Implement the backward algorithm as discussed in class in log-space in the `backward_log_Z` method. Ensure that it gives you identical results to the naïve implementation for the normalizer by running the first skeleton cell of Q3a).

b) **(5 pts)** Implement the forward algorithm in log-space in the `forward_log_Z` method. Ensure that it gives you identical results to the backward algorithm for the normalizer by running the first skeleton cell of Q3b).

c) **(5 pts)** Implement the Viterbi algorithm as discussed in class for calculating the score of the best tagging sequence $\boldsymbol{t} \in \mathcal{T}^N$ for a given CRF and word sequence $\boldsymbol{w}$ in log-space in the
`backward_viterbi_log` method. Ensure that it gives you identical results to the naïve implementation of finding the score of the best tagging by running the first skeleton cell of Q3c).

d) **(10 pts)** Implement Dijkstra's for calculating the best tagging $\boldsymbol{t}$ for a given CRF and word sequence $\boldsymbol{w}$ (Algorithm 1) in log-space in the `dijkstra_viterbi_log` method. Ensure that it gives you identical results to the naive implementation of finding the best tagging by running the first skeleton cell of Q3d).

   NB: You can implement Dijkstra's in the variant of the arctic semiring defined in Question 2 by setting $\omega(t_K, t', w) = \text{score}(\langle t_K, t'\rangle, w) - \log Z$, where Z is the normaliser of the full sequence (which you can compute by, for example, using backward). The transformation back to the unnormalised score we compute using Viterbi is done for you in the skeleton (since you will compute $\log Z$ within Dijkstra's, you can just return it to do the transformation afterward).

   NB 2: To make the above work with 'regular' priority queue implementations, you may simply define a new class for our tuples that are pushed into the queue and override its `__lt__` method. Alternative solutions (e.g., negating the semiring above and negating the score in the end), are also perfectly acceptable.

NB 3: The above works because $\mathbf{p}(\boldsymbol{t} \mid \boldsymbol{w}) = \frac{\exp\{\text{score}(\boldsymbol{t},\boldsymbol{w})\}}{Z} \in [0, 1]$, thus $\text{score}(\langle t_K, t' \rangle, \boldsymbol{w}) - \log Z = \log(\mathbf{p}(\boldsymbol{t} \mid \boldsymbol{w})) \in \mathbb{R}_{\leq 0} \cup \{-\infty\}$. Technically, using the normaliser Z for the full sequence for a subsequence will not be 'tight'. in that the probabilities will not sum to one, but for our purpose here, this does not matter.

e) **(5 pts)** Compare the wallclock time of Dijkstra's and Viterbi given a short word sequence $\boldsymbol{w}$ by running the cells in Q3e) with your newly implemented algorithms. Do the results match your theoretical analysis in Q2? If not, discuss possible reasons for the discrepancy.

f) **(5 pts)** Train your CRF using the `train_model_report_accuracy` function provided in the first cell of Q3f). Use Viterbi to decode and analyse the performance of your model on the development set in terms of per-tag accuracy, whose formula is given below:

$$\frac{1}{I}\sum_{i=1}^{I}\frac{1}{N_i}\sum_{n_i=1}^{N_i}\mathbb{1}\{\widehat{t_{in_i}} = t_{in_i}\} \tag{12}$$

Where $\widehat{t_{in_i}}$ and t_{in_i} denote the predicted and true part of speech tag for the n_i-th tag of the i-th subsequence of our held-out dataset.

Note: Your accuracy should be higher than 90% on the held-out data set.

g) **(10 pts)** Lift your CRF into the expectation semiring (Definition 1.2). Recall that an ordinary CRF is trained by maximizing its log-likelihood. To add an entropy regularizer to this objective, we simply add the entropy of the CRF, weighted by a hyperparameter β, to the log-likelihood, and we train the model under this augmented objective (Eq. (13)):

$$\ell(\boldsymbol{\theta}) = \frac{1}{I}\sum_{i=1}^{I}\left(\text{score}_{\boldsymbol{\theta}}(\boldsymbol{t}^{(i)}, \boldsymbol{w}^{(i)}) - \log\sum_{\boldsymbol{t}'\in\mathcal{T}^N}\exp\text{score}_{\boldsymbol{\theta}}(\boldsymbol{t}', \boldsymbol{w}^{(i)})\right) + \frac{\beta}{I}\cdot\sum_{i=1}^{I}\text{H}(\text{T}_{\boldsymbol{w}^{(i)}}) \tag{13}$$

where $\boldsymbol{\theta}$ are the model's parameters. Train three Conditional Random Field (CRF)s, each with an entropy regularizer with a strength of $\beta \in \{0.1, 1.0, 10.0\}$. Once again, analyze the accuracy of your CRFs on the development set and compare their performance to the unregularised CRF by running the `train_model_report_accuracy` function provided in the cells in Q3g.

Note: If you are having numerical issues, it may help to make sure all of your tensors are of `dtype torch.float64`.

h) **(15 pts) Bonus Question**: Modify the CRF code to achieve > 96% accuracy on the held-out set. You are allowed to use whatever innovations you wish from the NLP literature. Write two paragraphs describing your approach.

References

Joakim Nivre, Željko Agić, Lars Ahrenberg, Maria Jesus Aranzabe, Masayuki Asahara, Aitziber Atutxa, Miguel Ballesteros, John Bauer, Kepa Bengoetxea, Riyaz Ahmad Bhat, Eckhard Bick, Cristina Bosco, Gosse Bouma, Sam Bowman, Marie Candito, Gülşen Cebiroğlu Eryiğit, Giuseppe G. A. Celano, Fabricio Chalub, Jinho Choi, Çağrı Çöltekin, Miriam Connor, Elizabeth Davidson, Marie-Catherine de Marneffe, Valeria de Paiva, Arantza Diaz de Ilarraza, Kaja Dobrovoljc, Timothy Dozat, Kira Droganova, Puneet Dwivedi, Marhaba Eli, Tomaž Erjavec, Richárd Farkas, Jennifer Foster, Cláudia Freitas, Katarína Gajdošová, Daniel Galbraith, Marcos Garcia, Filip Ginter, Iakes Goenaga, Koldo Gojenola, Memduh Gökırmak, Yoav Goldberg, Xavier Gómez Guinovart, Berta Gonzáles Saavedra, Matias Grioni, Normunds Grūzītis, Bruno Guillaume, Nizar Habash, Jan Hajič, Linh Hà Mỹ, Dag Haug, Barbora Hladká, Petter Hohle, Radu Ion, Elena Irimia, Anders Johannsen, Fredrik Jørgensen, Hüner Kaşıkara, Hiroshi Kanayama, Jenna Kanerva, Natalia Kotsyba, Simon Krek, Veronika Laippala, PhÆ°Æ¡ng Lê H`ông, Alessandro Lenci, Nikola Ljubešić, Olga Lyashevskaya, Teresa Lynn, Aibek Makazhanov, Christopher Manning, Cătălina Mărănduc, David Mareček, Héctor Martínez Alonso, André Martins, Jan Mašek, Yuji Matsumoto, Ryan McDonald, Anna Missilä, Verginica Mititelu, Yusuke Miyao, Simonetta Montemagni, Amir More, Shunsuke Mori, Bohdan Moskalevskyi, Kadri Muischnek, Nina Mustafina, Kaili Müürisep, LÆ°Æ¡ng Nguy˜ên Thị, Huy`ên Nguy˜ên Thị Minh, Vitaly Nikolaev, Hanna Nurmi, Stina Ojala, Petya Osenova, Lilja Øvrelid, Elena Pascual, Marco Passarotti, Cenel-Augusto Perez, Guy Perrier, Slav Petrov, Jussi Piitulainen, Barbara Plank, Martin Popel, Lauma Pretkalniņa, Prokopis Prokopidis, Tiina Puolakainen, Sampo Pyysalo, Alexandre Rademaker, Loganathan Ramasamy, Livy Real, Laura Rituma, Rudolf Rosa, Shadi Saleh, Manuela Sanguinetti, Baiba Saulīte, Sebastian Schuster, Djamé Seddah, Wolfgang Seeker, Mojgan Seraji, Lena Shakurova, Mo Shen, Dmitry Sichinava, Natalia Silveira, Maria Simi, Radu Simionescu, Katalin Simkó, Mária Šimková, Kiril Simov, Aaron Smith, Alane Suhr, Umut Sulubacak, Zsolt Szántó, Dima Taji, Takaaki Tanaka, Reut Tsarfaty, Francis Tyers, Sumire Uematsu, Larraitz Uria, Gertjan van Noord, Viktor Varga, Veronika Vincze, Jonathan North Washington, Zdeněk Žabokrtský, Amir Zeldes, Daniel Zeman, and Hanzhi Zhu. Universal dependencies 2.0, 2017. URL `http://hdl.handle.net/11234/1-1983`. LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.