Lekce 9 [60]

> Considering a linear-chain CRF, write down how a score of a label sequence y is defined, and how can a log probability be computed using the label sequence scores. [5]

Linear-chain CRF je lineární graf, ve kterém hrany definují závislosti mezi prvky výstupní sekvence.

Skóre nějaké výstupní sekvence y v závislosti na vstupu X se počítá jako součet pravděpodobnosti jednotlivých labelů $f(y_i|X)$ a přechodů mezi nimi $A_{y_{i-1}y_i}$.

$$s(X, y; \theta, A) = \sum_{i=1}^{N} \left(A_{y_{i-1},y_i} + f_\theta\left(y_i \mid X\right) \right)$$

Jakmile spočteme skóre, můžeme vypočítat pravděpodobnost celé “věty” y pomocí softmaxu. Cross entropii této vzniklé distribuce poté spočítáme zlogaritmováním této pravděpodobnosti:

$$\log p(y \mid X) = s(X, y) - \operatorname{logsumexp}_{z \in Y^N}(s(X, z))$$

> Write down the dynamic programming algorithm for computing log probability of a linear-chain CRF, including its asymptotic complexity. [10]

Když do sebe zanořuji logsumexpy, logy a expy se vyruší, takže z toho nakonec vznikne jeden velký logsumexp se sumami uvnitř. Proto je také

$$\operatorname{logsumexp}_{k=1}^{Y}\left(\alpha_N(k)\right) = \operatorname{logsumexp}_{z \in Y^N}(s(z)),$$

kde $\alpha_t(k)$ označuje log-pravděpodobnost sekvence dlouhé t a končící na $k

$$\alpha_t(k) = f_\theta\left(y_t = k \mid X\right) + \operatorname{logsumexp}_{j \in Y}\left(\alpha_{t-1}(j) + A_{j,k}\right)$$

Inputs: Network computing $f_{\boldsymbol{\theta}}(y_t = k|\boldsymbol{X})$, an unnormalized probability of output sequence element probability being k at time t.
Inputs: Transition matrix $\boldsymbol{A} \in \mathbb{R}^{Y \times Y}$.
Inputs: Input sequence $\boldsymbol{X}$ of length N, gold labeling $\boldsymbol{g} \in Y^N$.
Outputs: Value of $\log p(\boldsymbol{g}|\boldsymbol{X})$.
Time Complexity: $\mathcal{O}(N \cdot Y^2)$.

- For $t = 1, \ldots, N$:
  - For $k = 1, \ldots, Y$:
    - $\alpha_t(k) \leftarrow f_{\boldsymbol{\theta}}(y_t = k|\boldsymbol{X})$
    - If $t > 1$:
      - $\alpha_t(k) \leftarrow \alpha_t(k) + \operatorname{logsumexp}\left(\alpha_{t-1}(j) + \boldsymbol{A}_{j,k} \,\middle|\, j = 1, \ldots, Y\right)$
- Return $\sum_{t=1}^{N} f_{\boldsymbol{\theta}}(y_t = g_t|\boldsymbol{X}) + \sum_{t=2}^{N} \boldsymbol{A}_{g_{t-1},g_t} - \operatorname{logsumexp}_{k=1}^{Y}(\alpha_N(k))$

> Write down the dynamic programming algorithm for linear-chain CRF decoding, i.e., an algorithm computing the most probable label sequence y. [10]

Algoritmus je stejný jako výše, pouze místo logsumexpů se použije max. Také musíme sledovat, *kde* bylo maxima dosaženo.

Inputs: Network computing $f_{\boldsymbol{\theta}}(y_t = k|\boldsymbol{X})$, an unnormalized probability of output sequence element probability being k at time t.
Inputs: Transition matrix $\boldsymbol{A} \in \mathbb{R}^{Y \times Y}$.
Inputs: Input sequence $\boldsymbol{X}$ of length N~~, gold labeling $\boldsymbol{g} \in Y^N$~~.
Outputs: ~~Value of $\log p(\boldsymbol{g}|\boldsymbol{X})$~~. decoded seq.
Time Complexity: $\mathcal{O}(N \cdot Y^2)$.

- For $t = 1, \ldots, N$:
  - For $k = 1, \ldots, Y$: $P(\alpha_t(k)) = \operatorname{argmax}(\alpha_{t-1}(j) + A_{j,k} \mid j \ldots)$
    - $\alpha_t(k) \leftarrow f_{\boldsymbol{\theta}}(y_t = k|\boldsymbol{X})$
    - If $t > 1$:
      - $\alpha_t(k) \leftarrow \alpha_t(k) + \overset{\max}{\cancel{\operatorname{logsumexp}}}\left(\alpha_{t-1}(j) + \boldsymbol{A}_{j,k} \,\middle|\, j = 1, \ldots, Y\right)$
- Return $\cancel{\sum_{t=1}^{N} f_{\boldsymbol{\theta}}(y_t = g_t|\boldsymbol{X}) + \sum_{t=2}^{N} \boldsymbol{A}_{g_{t-1}, g_t} -} \overset{\operatorname{argmax}}{\cancel{\operatorname{logsumexp}}}{}_{k=1}^{Y}(\alpha_N(k))$ + backtrack

Figure 1: image-20210629215723224

> In the context of CTC loss, describe regular and extended labelings and write down an algorithm for computing the log probability of a gold label sequence y. [10]

Regular labeling je labeling s délkou $\leq$ délka vstupní sekvence. Síť ale generuje extended labeling, který má stejnou délku, a obsahuje speciální znak *blank*. Regulární labeling můžeme vyrobit z extended tím, že spojíme shodné sousední znaky a poté vymažeme blanky.

Pro nějakou sekvenci y definujeme $\alpha^t(s)$ jako pravděpodobnost, že prvních t kroků sítě vygenerovalo prvních s znaků sekvence y,

$$\alpha^t(s) \stackrel{\text{def}}{=} \sum_{\substack{\text{extended} \\ \text{labelings } \pi: \\ \mathcal{B}(\pi_{1:t}) = y_{1:s}}} \prod_{t'=1}^{t} p_{\pi_{t'}}^{t'}$$

Toto $\alpha^t(s)$ se dá vypočítat jako součet $\alpha_{-}^t(s)$, které označuje, že vygenerovaná sekvence π končí na blank, a $\alpha_*^t(s)$, která označuje, že π na blank nekončí. Inicializujeme

$$\alpha_{-}^1(0) \leftarrow p_{-}^1$$
$$\alpha_*^1(1) \leftarrow p_{y_1}^1$$

a provedeme indukční krok

$$
\begin{aligned}
\alpha_-^t(s) &\leftarrow p_-^t\left(\alpha_*^{t-1}(s)+\alpha_-^{t-1}(s)\right)\\
\alpha_*^t(s) &\leftarrow \begin{cases} p_{y_s}^t\left(\alpha_*^{t-1}(s)+\alpha_-^{t-1}(s-1)+\alpha_*^{t-1}(s-1)\right), & \text{if } y_s \neq y_{s-1}\\ p_{y_s}^t\left(\alpha_*^{t-1}(s)+\alpha_-^{t-1}(s-1)\right), & \text{if } y_s = y_{s-1}\end{cases}
\end{aligned}
$$

V druhém případě je nutné uvažovat, zda v mé extended π je znak π_s stejný jako π_{s-1} — pokud ano, tak α^{t-1} musí vygenerovat celých s znaků, jinak by mu stačilo vygenerovat $s-1$, protože ten s-tý jsem vygeneroval teď v čase t.

Reálně to pak celé bude zlogaritmováno, tj. místo násobení bude + a místo + budou logsumexpy.

> Describe how are CTC predictions performed using a beam-search. [5]

Obecně si v kroce t si nechám k nejlepších regulárních labelingů, které umím vygenerovat v t krocích (tj. z extended labelingu délky t). Uloženy mám i jejich $\alpha^t(y)$, tj. součet jejich pravděpodobností napříč extended labelingy.

1. Vygeneruji nové extended labelingy tak, že ka každý ze svých regulárních přidám buďto blank, nebo jiný label.
2. Všechny předělám na regulární labelingy
3. Stejné labeling seskupím a sečtu jejich pravděpodobnosti
4. Z této množiny prodloužených regulárních labelingů vyberu k nejlepších a iteruji.

> Draw the CBOW architecture from `word2vec`, including the sizes of the inputs and the sizes of the outputs and used non-linearities. Also make sure to indicate where are the embeddings being trained. [5]

Embedding je matice $W_{V\times N}$. Za output vrstvou je Softmax, který rozhoduje, které že slovo bylo v díře mezi těmi vstupními.

> Draw the SkipGram architecture from `word2vec`, including the sizes of the inputs and the sizes of the outputs and used non-linearities. Also make sure to indicate where are the embeddings being trained. [5]

Aktivace je opět softmax, embeddingem je opět matice $W_{V\times N}$. Z jednoho slova predikujeme jeho kontext.

> Describe the hierarchical softmax used in `word2vec`. [5]

Ze tříd (tj. ze slov) postavím binární strom, místo jedné klasifikace do k tříd udělám *hloubka* $\in O(\log k)$ binárních klasifikací. Pokud pak slovo w ve stromě odpovídá cestě $n_1, n_2, \ldots, n_L$, poté

$$
p_{\mathrm{HS}}\left(w \mid w_i\right) \stackrel{\text{def}}{=} \prod_{j=1}^{L-1} \sigma\left(\left[+1 \text{ if } n_{j+1} \text{ is right child else } -1\right] \cdot W_{n_j}^{\top} V_{w_i}\right)
$$

Tohle má sice špatnou accuracy, ale nám to nevadí, protože embeddingy vzniknou hezké.

> Describe the negative sampling proposed in `word2vec`, including the choice of distribution of negative samples. [5]

1. Místo velkého softmaxu udělám nad každým slovem sigmoid; hodnoty nebudou 100% správně (nenasčítá se to do jedničky), ale derivace budou zhruba fungovat.
2. Místo, abych tlačil dolů pravděpodobnosti *všech* negativních příkladů, nasampluji náhodně k z nich — jinak by mi negativní příklady úplně udusily ten jeden pozitivní.

$$l_{\mathrm{NEG}}\left(w_o, w_i\right) \stackrel{\mathrm{def}}{=} \log \sigma\left(W_{w_o}^{\top} V_{w_i}\right) + \sum_{j=1}^{k} \mathbb{E}_{w_j \sim P(w)} \log\left(1 - \sigma\left(W_{w_j}^{\top} V_{w_i}\right)\right)$$

Slova samplujeme z unigramového rozdělení $U(w)^{3/4}$, což je rozdělení slov, kterém jim přiděluje pravděpodobnost podle počtu jejich výskytů v korpusu.