Neural networks

Training CRFs - pseudolikelihood

GENERAL CRF

Topics: CRFs in general

- Gradients in general CRFs always take the form:

$$\frac{\partial -\log p(\mathbf{y}^{(t)}|\mathbf{X}^{(t)})}{\partial \theta} = -\Bigg(\overbrace{\textstyle\sum_f \frac{\partial}{\partial \theta} \log \Psi_f(\mathbf{y}^{(t)}, \mathbf{X}^{(t)})}^{\text{make } y^{(t)} \text{ more likely}}$$

$$\underbrace{- \mathrm{E}_{\mathbf{y}}\left[\textstyle\sum_f \frac{\partial}{\partial \theta} \log \Psi_f(\mathbf{y}, \mathbf{X}^{(t)}) \,\middle|\, \mathbf{X}^{(t)}\right]}_{\text{make everything less likely}}\Bigg)$$

- The expectation over $\mathbf{y}$ will often need to be approximated, using loopy belief propagation
    - it will often involve only a few of the y_k variables

GENERAL CRF

Topics: pseudolikelihood

- Why not just change the loss function to a tractable one

$$-\sum_{k=1}^{K} \log p(y_k | y_1, \ldots, y_{k-1}, y_{k+1}, \ldots, y_K, \mathbf{X})$$

  - predict, in turn, each y_k not just from $\mathbf{X}$, but also all the other elements of $\mathbf{y}$
  - can compute the exact gradients
    - the probabilities only require normalizing y_k individually, like in a regular softmax
    - each conditional often only depend on few variables (local Markov property)
  - however, often tends to work less well
  - we still need to compute $p(y_k|\mathbf{X})$ to do predictions anyways