Lecture 19 - scribbles - deep learning

Tuesday, March 27, 2018 14:21

- proximal gradient method & Catalyst
- latent SVM struct + CCCP
- deep learning – RNN
- learning to search – SEARNN

proximal gradient method

↳ generalization of projected gradient method to other non-smooth functions

Composite framework : $F(w) \triangleq f(w) + \Omega(w)$ where f is convex & L-smooth
Ω is convex but not nec. smooth

constrained opt. $\Omega(w) = \delta_M(w) \triangleq \begin{cases} 0 & \text{if } w \in M \\ +\infty & \text{o.w.} \end{cases}$

(δ_M: indicator of M)

ℓ_1-regularization $\Omega(w) = \|w\|_1$

proximal gradient update:

$$w_{t+1} = \underset{w}{\operatorname{argmin}} \underbrace{f(w_t) + \langle \nabla f(w_t), w - w_t \rangle + \frac{1}{2\gamma_t} \|w - w_t\|^2}_{\triangleq B_t(w)} + \Omega(w)$$

if $\gamma_t \leq \frac{1}{L}$ then $f(w) \leq B_t(w)$ $\forall w$

we can rewrite $B_t(w) = \frac{1}{2\gamma_t} \|w - [w_t - \gamma_t \nabla f(w_t)]\|^2 + \text{cst.}$ (by completing the square)

$\Rightarrow$ if $\Omega(w) = \delta_M(w)$, we get projected gradient alg.

$\Rightarrow$ if $\Omega(w) = \partial_{\|w\|}(w)$; we get projected gradient alg.

$$w_{t+1} = \text{prox}^{\Omega}_{\gamma_t}\left(w_t - \gamma_t \nabla f(w_t)\right)$$

"proximal operator" $\text{prox}^{\Omega}_{\gamma}(z) \triangleq \underset{w}{\text{argmin}} \left\{ \Omega(w) + \frac{1}{2\gamma}\|w - z\|^2 \right\}$

like for projection, prox operator is non-expansive (ie. 1-Lipschitz)

ie. $\|\text{prox}^{\Omega}_{\gamma}(w) - \text{prox}^{\Omega}_{\gamma}(w')\|_2 \leq \|w - w'\|_2$

$\Rightarrow$ convergence rates of prox gradient method are same as unconstrained gradient descent

e.g. if f is μ-strongly convex; linear rate with $\rho = \frac{\mu}{L}$

* to be useful, need $\text{prox}^{\Omega}_{\gamma}$ to be efficiently computable

$$\text{prox}^{\|w\|_1}_{\gamma}(z) = \underset{w}{\text{argmin}} \; \|w\|_1 + \frac{1}{2\gamma}\|w - z\|_2^2$$

"soft-thresholding" $= \begin{cases} \text{sgn}(z)\left[|z_i| - \gamma\right] & \text{if } |z_i| \geq \gamma \\ 0 & \text{o.w.} \end{cases}$

FISTA $\rightarrow$ accelerated prox gradient method

Catalyst algorithm [Lin, Mairal & Harchaoui NIPS 2015]

"meta-algorithm": outer loop which uses a linearly convergent alg. in the inner loop to get overall acceleration(?)

main idea: use the accelerated proximal point algorithm with approximation inner loop of prox operator

proximal pt. alg.: is proximal gradient with $f=0$

↳ $w_{t+1} = \text{prox}_{\gamma}^{g}(w_t)$

Catalyst alg: (for μ-strongly convex F)

Let $q \triangleq \frac{\mu}{\mu + \frac{1}{\gamma}}$ (γ is an algorithmic parameter)

repeat:

$$w_{t+1} \approx \underset{w}{\text{argmin}} \underbrace{F(w) + \frac{1}{2\gamma}\|w - z_t\|^2}_{\triangleq G_t(w)} \quad \text{s.t.} \quad G_t(w_{t+1}) - \min_w G_t(w) \leq \varepsilon_t$$

ε_t: to be specified

↳ using an inner loop algorithm [e.g. SAGA or AFW]

$$z_{t+1} = w_{t+1} + \beta_{t+1}(w_{t+1} - w_t)$$

[accelerated Nesterov trick piece]

β_{t+1}: "extrapolation"

$(w_{t+1} - w_t)$: like a "momentum"

β_{t+1} is found using fancy equation so that everything works

solve for α_{t+1} in eq. $\alpha_{t+1}^2 = (1-\alpha_{t+1})\alpha_t^2 + q\alpha_{t+1}$

(pick $\alpha_{t+1} \in]0,1[$)

$\beta_{t+1} \triangleq \frac{\alpha_t(1-\alpha_t)}{\alpha_t^2 + \alpha_{t+1}}$

Catalyst trick: use γ & ε_t
s.t. overall # of inner loop calls
give an overall acceleration

$\hat{\mu} \geq \mu + \frac{1}{\gamma}$

with clever analysis of warm starting

acceleration result: if inner loop alg. has convergence rate $\exp(-\frac{\hat{\mu}}{L} t)$ (strong convexity of inner loop problem)

then with correct constants:

linear rate: $\rho = 1 - \frac{1}{\kappa} \rightsquigarrow \approx 1 - \frac{1}{\sqrt{\kappa}}$ for strongly convex case

$\frac{1}{\varepsilon} \rightsquigarrow \frac{1}{\varepsilon^2}$ for convex case

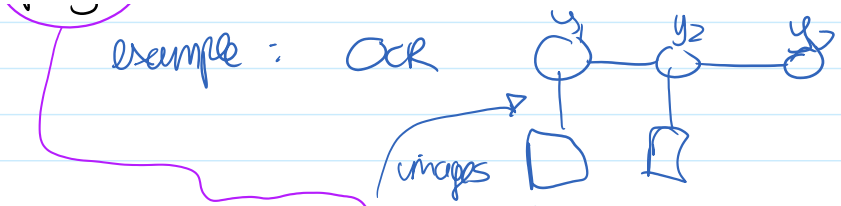
result e.g.: can get accelerated SAGA
" SVRG
" AFW
etc...

Deep learning:

go from $\langle w, \phi(x,y) \rangle$ to $\langle w, \phi(x,y;\theta) \rangle$ (can learn features)

I) plug in "deep learning" feature in a structured prediction model

example: OCR 



this class of $\ell_t(x_t, y_t) = x_t$ → learned on imagenet e.g.

instead $\ell_t(x_t, y_t) = NN_\theta(x_t)$

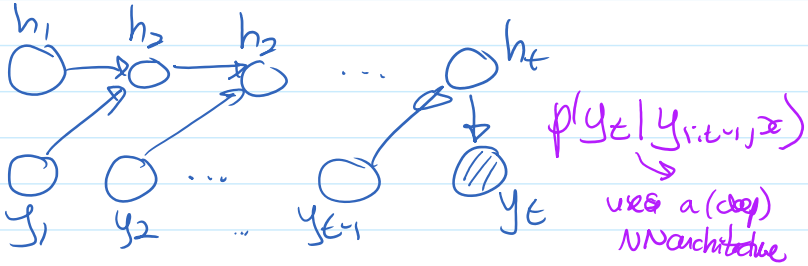
II) "end-to-end" training : structured energy prediction network (SPEN)

III) <u>recurrent neural networks (RNN)</u>

motivation: $p(y|x) = \prod_{t=1}^{T} p(y_t | y_{1:t-1}, x)$ ← chain rule

graphical model approach →
$$\prod_t p(y_t | y_{\pi_t}, x) \quad \text{[directed]}$$
$$\frac{1}{Z(x)} \prod_c \psi_c(y_c; x) \quad \text{[undirected]}$$

RNN → "structured parameterization" of $p(y_t | y_{1:t-1}, x)$



$$h_{t+1} \triangleq f(h_t, x, y_t, W)$$
$$h_t = f(f(f \cdots (y_0), \ldots), x, y_t; W)$$

define $p(y_t | y_{1:t-1}, x) \propto \exp(c(y_t)^T \tilde{W} h_t)$ e.g.

<u>learning</u>: use maximum likelihood ie. $\min_{w, \tilde{w}} -\sum_{n} \log p(y^{(i)} | x^{(i)})$ → "teacher forcing"

$\sum_t \log p(y_t^{(i)} | y_{1:t-1}^{(i)}; x^{(i)})$ exposure problem

do SGD on this
chain rule ⇝ backpropagation

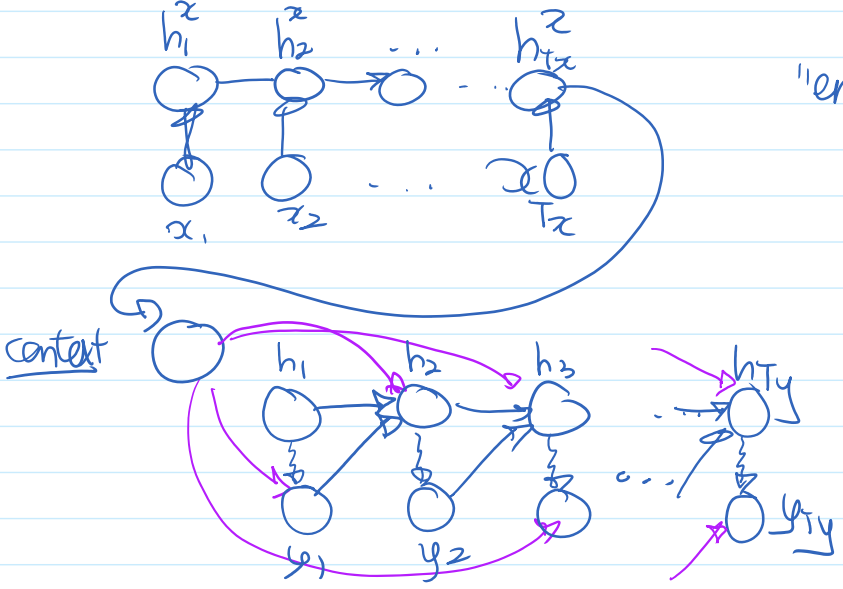
decoding: $\underset{y \in \mathcal{Y}}{\text{argmax}} \sum_t \log p(y_t | y_{1:t-1}, x)$

→ need approximation

- greedy decoding $\hat{y}_t = \underset{y \in \mathcal{Y}_t}{\text{argmax}}\ p(y | \hat{y}_{1:t-1}; x)$
- beam search "greedy with memory of size K" "beam"

Seq2seq / encoder / decoder

↳ useful way to get $p(y_t | y_{1:t-1}, x)$ for RNN when x is variable length



"encoder RNN"

"decoder RNN"

Issues:

a) variable length output?
→ end-of-sequence special character

b) long input sequence x?

problem: needs to be summarize in fixed length context vector

solution: "attention mechanism"

c) vanishing gradient?

- LSTM
- gated recurrent unit (GRU)

Learning to search (L2S)

SEARN Hal Daume's phd thesis

$h_w : \mathcal{X} \to \mathcal{Y}$

$h_w(x) = \underset{y \in \mathcal{Y}}{\operatorname{argmax}}\ s(x, y; w)$

special case: learning to do greedy search

split y in ordered # of decisions

$(y_1, \ldots, y_T)$

learn a classifier $\pi(\text{feature}(\hat{y}_1, \ldots, \hat{y}_{t-1}, x)) = \hat{y}_t$

π: classifier; "policy": decoding

L2S framework:

from $(x^{(i)}, y^{(i)})_{i=1}^n$ and $\ell(\cdot,\cdot)$

learn a good classifier/policy π_w s.t. $\hat{y}_t = \pi_w(\text{feature}(\hat{y}_1, \ldots, \hat{y}_{t-1}, x))$

$h_w(x) \triangleq (\hat{y}_1 \ldots \hat{y}_t)$ "greedy decoder"

s.t. $\ell(y^{(i)}, h_w(x^{(i)}))$ is good

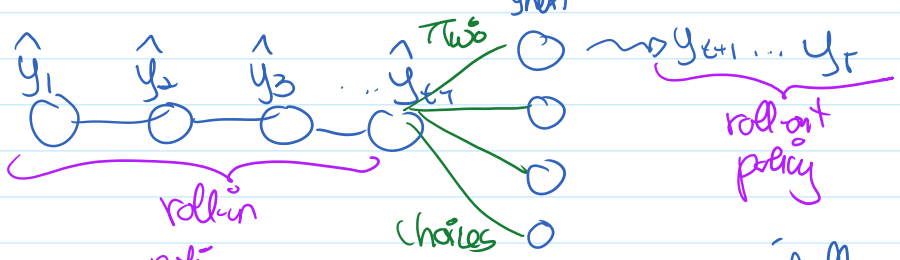
s.t. $\ell(y^{(i)}, h_w(x^{(i)}))$ is good

↳ central idea: "reduction": where reduces structured prediction learning to problem of cost-sensitive classification learning for π_w

method: generate training data for classifier π_w i.e. $\left(\hat{y}^{(j)}_{context}, x^{(j)}, cost(y_{next})\right)$ ← prefix sequence

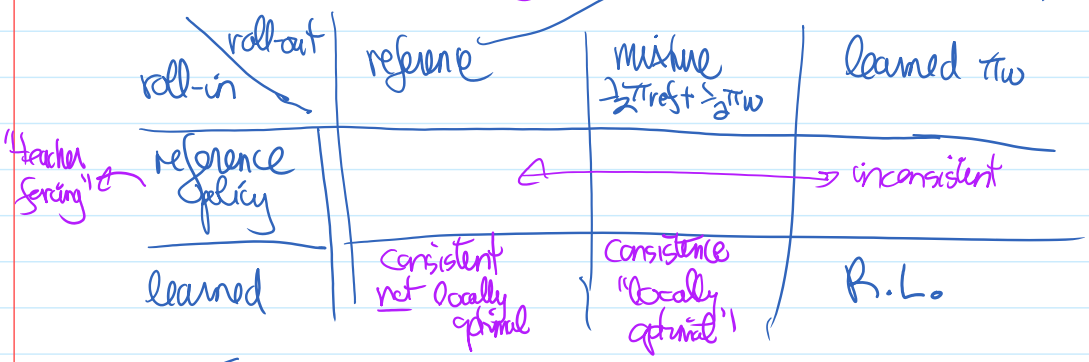
"roll-in" policy → determines how get $\hat{y}_{1:t-1}$ context

"roll-out" policy → " how get $y_{t+1}, \ldots, y_T$ "target completion"



$\ell(y^{(i)}, (\hat{y}_1 \ldots \hat{y}_{t-1}\ y_{next}\ \hat{y}_{t+1} \ldots \hat{y}_T))$

↓ cost(y_{next})



ideally: $\pi_{ref}(\hat{y}_1, \ldots, \hat{y}_{t-1}, y_{next}) = \underset{y_{completion}}{\text{argmin}}\ \ell(y^{(i)}, (\hat{y}_{1:t-1}, y_{next}, y_{completion}))$

e.g. π_{ref} for Hamming loss → copy ground truth

in practice: Heuristic approximate it

LOLS ICML 2015
" locally optimal learning to search"

↳ learning better than your teacher

ie. when heuristic is not optimal

SEARNN: apply LOLS to RNN training ie. $\pi_w(y_{1:t-1}, x) \rightarrow$ RNN cell

$p(y_{next} | y_{1:t-1}, x)$ of a RNN

if use $-\log p(\hat{y}_{target} | y_{1:t-1}, x)$ as cost "target learning"

and $\hat{y}_{target}$ = ground truth

then LOLS with ref roll-in is standard MLE

*. address exposure bias using learned roll-in
- make use of structured loss $\ell(\ldots)$ vs. MLE