proximal gradient method

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

**Composite framework**: \( F(w) \triangleq f(w) + Q(w) \) where \( f \) is convex & \( \ell \)-smooth

\( Q \) is convex but not \( \ell \)-smooth

constrained opt. \( \min \{ Q(w) + \sum_{i=1}^{m} f_i(w) \} \)

\( \mathbf{w} \) - regularization \( \ell_2 \)-norm

\( \ell_2 \)-regularization \( \ell_2(w) = \| w \|_2^2 \)

proximal gradient update:

\[
\begin{align*}
\mathbf{w}_{t+1} &= \arg \min_{\mathbf{w}} \left\{ f(\mathbf{w}) + \langle \nabla f(\mathbf{w}), \mathbf{w} - \mathbf{w}_t \rangle + \frac{1}{2\xi_t} \| \mathbf{w} - \mathbf{w}_t \|_2^2 + \frac{\lambda}{\xi_t} \right\} \\
&= \mathbf{B}_t(\mathbf{w})
\end{align*}
\]

if \( \lambda t \leq 1 \) then \( f(w) \leq \mathbf{B}_t(w) \) \( \forall w \)

we can rewrite \( \mathbf{B}_t(w) = \| w - [w - x_t \nabla f(w)] \|_2^2 + \text{const.} \) (by completing the square)

\[
\Rightarrow \text{if } \mathcal{O}(w) = \mathcal{S}_m(w), \text{ we get projected gradient alg.}
\]
\[ \Rightarrow v_t = v_{t-1} - \gamma_t \nabla f(v_{t-1}) \]

\[ w_{t+1} = \text{prox}_{\frac{g}{\gamma}}(w_t - \gamma_t \nabla f(w_t)) \]

"proximal operator" \[ \text{prox}_{\frac{g}{\gamma}}(z) \triangleq \arg \min_w \left\{ g(w) + \frac{1}{2\gamma} \| w - z \|^2 \right\} \]

Like for projection, prox operator is non-expansive (i.e. 1-Lipschitz)

\[ \text{i.e.} \quad \| \text{prox}_{\frac{g}{\gamma}}(w) - \text{prox}_{\frac{g}{\gamma}}(w') \|_2 \leq \| w - w' \|_2 \]

\[ \Rightarrow \] convergence rates of prox gradient method

are same as unconstrained gradient descent

E.g., if \( \Phi \) is \( \mu \)-strongly convex; linear rate with \( \| w - w^* \|_2 \) converges

*to be useful, need \( \text{prox}_{\frac{g}{\gamma}} \) to be efficiently computable

\[ \text{prox}_{\frac{g}{\gamma}}(z) \triangleq \arg \min_w \left\{ g(w) + \frac{1}{2\gamma} \| w - z \|^2 \right\} \]

"soft-thresholding" \[ = \begin{cases} \sigma g_0(z) & \text{if } |z_t| > \gamma \\ 0 & \text{o.w.} \end{cases} \]

FISTA \( \rightarrow \) accelerated prox gradient method

\[ \text{Catalyst algorithm} \quad [\text{Lin, Mairal \\& Harchaoui, NIPS 2015}] \]

"meta-algorithm" = outer loop which uses a linearly convergent alg.

in the inner loop!

+ To get overall acceleration (\( \tilde{O} \))
Main idea: use the accelerated proximal point algorithm with approximation inner loop of prox operator

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

$L \rightarrow w_{t+1} = \text{prox}_F^\gamma(w_t)$

Catalyst alg.: (for $\mu$-strongly convex $F$)

Let $\beta = \frac{\mu}{\beta + \gamma}$ (is an algorithmic parameter)

repeat:

$w_{t+1} \approx \arg\min_w F(w) + \frac{1}{2\beta} \|w - z_t\|^2 \quad \text{s.t.} \quad G_\theta(w_{t+1}) - \min_w G_\theta(w) \leq \frac{\epsilon}{\beta}$

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

$z_{t+1} = w_{t+1} + \beta \eta \left( w_{t+1} - w_t \right)$  \hspace{1cm} [accelerated Nesterov trick piece]

'extrapolation'

\beta_t is found using fancy equation so that everything works

solve for $\alpha_{t+1}$ in eq. \( \alpha_{t+1}^2 = (1-\alpha_{t+1})\alpha_t^2 + \beta \alpha_{t+1} \)
\[ \beta_{k+1} = \frac{\alpha_k (1 - a_k)}{\alpha_k^2 + \alpha_{k+1}} \]

**Catalyst Trick:** use \( x \in \mathbb{E} \)

*st. overall # of inner loop calls* give an overall acceleration

with clever analysis of warm starting

acceleration results *if inner loop avg. has convergence exp. \( \gamma \) (w)*

then with correct constants:

**Linearize:** \( \frac{1}{x} \leadsto \frac{1}{\mathbb{E}} \) for strongly convex case

\[ \frac{1}{x} \leadsto \frac{1}{\mathbb{E}} \] for convex case

result e.g. get accelerated SAGA

**SAG**

**SFW**

etc...

Deep Learning:

go from \( \langle w, f(xy) \rangle \) to \( \langle w, f(xy; \theta) \rangle \)

I. plug in “deep learning” feature in a structured prediction model

Example: CRF

\[ \mathbb{Y}_1, \mathbb{Y}_2, \ldots \]
II) "end-to-end" training: structured energy prediction network (SEEN)

III) Recurrent neural networks (RNN)

Motivation: $P(y|x) = \prod_{t=1}^{T} P(y_t|y_{t-1}, x)$

RNN $\Rightarrow$ "structured parameterization" of $P(y|x)$

$h_{t+1} = f(h_t, x, y_t, w)$

$h_t = f(f(...(y_0), ...), x, y_t, w)$

Define $P(y_t|y_{t-1}, x) \propto \exp(C(y_t)^T \tilde{W} h_t)$

Learning: use maximum likelihood, i.e., $\min_{h_1, h_2, ..., h_T} \sum_{i} \log P(y^{(i)}_t|x^{(i)})$
do SGD on this
chain rule = backpropagation

\[ \log p(y_t | y_{1:t-1}, x) \]

decoding: argmax \[ y \in \mathcal{Y} \]
\[ \Rightarrow \text{need approximation} \]

\[ \hat{y}_t = \text{argmax}_{y \in \mathcal{Y}_t} p(y | \hat{y}_{1:t-1}, x) \]

Beam search "greedy with memory of size k"

seq2seq / encoder/decoder

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

"encoder RNN"

issues:

"decoder RNN"

a) variable length output? \( \Rightarrow \) end-of-sequence special character

b) long input sequence \( x \)？

problem: needs to be summarized in fixed length context vector

solution: "attention mechanism"
Learning to search (L2S)

\[ h_w : x \rightarrow y \]

\[ h_w(x) = \arg\max_{y \in \hat{y}} s(x,y,w) \]

**special case:** learning to do greedy search

Split \( y \) in ordered # of decisions

\[ (\hat{y}_1, \ldots, \hat{y}_t) \]

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

L2S framework:

\[ \text{from } (x^{(i)}, y^{(i)})_{i=1}^{n} \text{ and } l(\cdot, \cdot) \]

Learn a good classifier/policy \( \pi_t \) so that

\[ \hat{y}_t = \pi_t(x; \theta) \]

\[ h_w(x) \approx (\hat{y}_1, \ldots, \hat{y}_t) \] "greedy decoder"

s.t. \( l(\hat{y}^{(i)}, h_w(x^{(i)})) \) is good
Central idea: "reduction": when reduces structured prediction learning to problem of cost-sensitive classification learning for $T_{\text{wo}}$

Method: generate training data for classifier $T_{\text{wo}}$, i.e., $(y^{(i)}_{\text{context}}, x^{(i)}_{\text{context}}, \text{cost}(y^{(i)}_{\text{context}}))$

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

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

$y_1, y_2, y_3, \ldots, y_t \rightarrow \langle \hat{y}_{1:t-1} \rangle$

Ideally: $T_{\text{wo}}(\hat{y}_{1:t-1}, y_T) = \min_{y_T} \sum_{t} \text{cost}(y^{(i)}_{\text{context}}, y_t, y_{t+1}, \ldots, y_T)$

In practice: heuristic approach, e.g., Tref for Hamming loss

LOLS @ICML, 2015

"locally optimal learning to search"

"Learning better than your teacher"
i.e. when heuristic is not optimal

SEARNN: apply LS to RNN training i.e. \( T_n(Y;\tilde{t}_{i-1}, x) \rightarrow \text{RNN cell} \)

\( \hat{Y}_{t+1} / Y_{t+1} \) of a RNN

\( \text{if use} \ -\log p(y_{t+1} | x) \text{ as cost } \)

\( \text{ and } \hat{Y}_{t+1} = \text{ground truth} \)

then LS with ref roll-in is standard MLE

\( \approx \text{add exposure bias using learned roll-in} \)

make use of structural loss \( l(\cdot, \cdot) \) vs. MLE