11-785 15/05/2020

Lb Review

(*) Note the key point → learning rate/step-size is same for every component of the parameter update. (I)

(*) Essence of normalised update rule is to amend the error/loss function, which is a function defined over a multi-variate parameter space, by effectively rescaling/normalising parameters.

(*) For the multivariate case, we have :-

convex loss

(11) $\underline{w}^{(k+1)} = \underline{w}^{(k)} - \eta H_E(\underline{w}^{(k)})^{-1} \nabla_{\underline{w}} E(\underline{w}^{(k)})^T$

(*) Normalisation constant is the Hessian; and we set $\eta = 1$

(*) Note similarity with setting of η_{opt} in scalar case

(*) In this case, we alter the parameters/gradient of error and keep learning rate $\eta = 1$ to resolve issue (I)

(*) Derivative-information inspired algorithms - Rprop and Quickprop

(*) Both make use of information associated with the derivative/2nd deriv. (i.e. some kind of compression/approx.)

(*) Both make component-wise updates to the parameters to improve convergence of gradient descent

---

Rprop

(12), (13)

- Note Rprop and quickprop are modifications to how the parameters are updated. (indep.)

(*) Resilient propagation

(*) Steps in different dir. not coupled

- Accounts for overshooting a minimum

Rprop (at each time-step)

- At each time:

→ If derivative at current location recommends continuing in same dir as before (i.e. no change of sign)

- Increase the step size; continue in same dir.

→ If the derivative has changed sign (i.e. minimum overshot)
- Reduce the step and reverse direction

- Rprop → see slides (non hee) (*) functionality
- The former and latter are governed by parameters α and β respectively

- select initial $\hat{w}$ and compute derivative
- Step Δw against the derivative (in direction that reduces)

→ $\Delta w = \text{sign}\left(\frac{dE(\hat{w})}{dw}\right)\Delta w$

→ $\hat{w} = \hat{w} - \Delta w$

- compute deriv in new location (governed by α)
  - no change in sign from previous location; increase step size and take longer step

* $\alpha > 1$

→ $\Delta w = \alpha \Delta w$

→ $\hat{w} = \hat{w} - \Delta w$

- Compute deriv in new location
  - If derivative has changed sign; return to previous location
  - shrink step size (governed by β)
  - Take a smaller step forward

$\beta < 1$

→ $\hat{w} = \hat{w} + \Delta w$

→ $\Delta w = \beta \Delta w$

→ $\hat{w} = \hat{w} - \Delta w$

(*) Note that Rprop is still used with backpropogation, which is used to compute derivatives

Rprop (pseudocode)

- parameters $w_{l,i,j}$ updated independently

- set $\alpha = 1.2$ $\beta = 0.5$
- For each layer l, and each i,j :-
  - init $w_{l,i,j}$, $\Delta w_{l,i,j} > 0$
  - prev $D(l,i,j) = \frac{dErr(w_{l,i,j})}{dw_{l,i,j}}$ (*) obtained via backprop
  - set $\Delta w_{l,i,j} = sign(prev\ D(l,i,j))\, \Delta w_{l,i,j}$
  - WHILE not converged :-
    - $w_{l,i,j} = w_{l,i,j} - \Delta w_{l,i,j}$
    - $D(l,i,j) = \frac{dErr(w_{l,i,j})}{dw_{l,i,j}}$ (*) as above
    - IF $sign(prev\ D(l,i,j)) = sign(D(l,i,j))$:-
      - $\Delta w_{l,i,j} = \alpha \Delta w_{l,i,j}$
      - prev $D(l,i,j) = D(l,i,j)$
    - ELSE:
      - $w_{l,i,j} = w_{l,i,j} + \Delta w_{l,i,j}$
      - $\Delta w_{l,i,j} = \beta \Delta w_{l,i,j}$

(*) Quickprop (Scott Fahlman)

- similar to Rprop, updates parameters independently, component-wise
- Quickprop can be viewed as employing Newton updates (i.e. 2nd order method involving Hessian) except
  - i) dimensions componentwise treated indep.
  - ii) uses approximation of 2nd deriv via finite differences

Quickprop - key equations

(For each dimension of the weight :-)

- For $i = 1 : N$

$$w_i^{(k+1)} = w_i^{(k)} - E''\left(w_i^{(k)} \mid w_j^{(k)}\ j \neq i\right)^{-1} E'\left(w_i^{(k)} \mid w_j^{(k)}, j \neq i\right)$$

$$w_i^{(k+1)} = w_i^{(k)} - \underbrace{D\left(w_i^{(k)}, w_i^{(k-1)}\right)^{-1}}_{\text{finite diff. approx to } 2^{nd} \text{ derivative assuming quad } E()} E'\left(w_i^{(k)} \mid w_j^{(k)}, j \neq i\right)$$

Yielding: $$w_i^{(k+1)} = w_i^{(k)} - \left(\frac{E'(w^{(k)}) - E'(w^{(k-1)})}{\Delta w^{(k-1)}}\right)^{-1} E'(w^{(k)})$$

- For every layer l, every connection from node i in the $(l-1)^{th}$ layer to node j in the l^{th} layer :-

$$\Delta w_{l,i,j}^{(k)} = \frac{\Delta w_{l,i,j}^{(k-1)}}{E'\left(w_{l,i,j}^{(k)}\right) - E'\left(w_{l,i,j}^{(k-1)}\right)} \underbrace{E'\left(w_{l,i,j}^{(k)}\right)}_{\text{- compute using backprop}}$$

$$w_{l,i,j}^{(k+1)} = w_{l,i,j}^{(k)} - \Delta w_{l,i,j}^{(k)}$$

(*) Issues with treating dimensions independently / making component-wise updates to the parameters

- solution converges smoothly in some dir; many oscillate/diverge in others

---

(*) Motivation for momentum / Nesterov's accel. gradient methods

- keep track of oscillations / div / convergence as an indicator
- For smooth convergence, increase step size
- oscillation / divergence → shrink.

Momentum methods → (Q) distill.pub (Polyak 1964)

- key equations already covered in notes
- β is set at around 0.9

(!!) Note the order of operation that the equations specify

$$\Delta \underline{w}^{(k)} = \beta \Delta \underline{w}^{(k-1)} - \eta \nabla_{\underline{w}} E(\underline{w}^{(k-1)})$$

(omitting layer subscripts)

(!!): $\nabla_{\underline{w}_k} E(\underline{w}^{(k-1)})$

- gradient of error eval at $\underline{w}^{(k-1)}$ (previous it.)

Nesterov's accel. grad.

(*) Similar to momentum, except change in order of operations

$$\Delta \underline{w}^{(k)} = \beta \Delta \underline{w}^{(k-1)} - \eta \nabla_{\underline{w}} E(\underline{w}^{(k-1)} + \beta \Delta \underline{w}^{(k-1)})$$

(*) At each iteration:-

i) Extend previous step

ii) Compute gradient step at resultant position

iii) Add the two

(*) Principle behind momentum (and Nesterov's accel. gradients?)

- Ideally - a component specific step size
  → Too many meta param (maintain sep step for every weight)

(!!): Adaptive solutions: (in learning rate/step-size)

- Start with common learning rate
- Shrink step size when weight oscillates
- Expand step size when weight moves directionally consistently

---

(**) Recap: Improving convergence

i) 2nd order methods (normalise variance across dim.)

ii) Adaptive/decaying learning rates

iii) Decouple dimensions (Rprop, Quickprop)

iv) Momentum methods → emphasise steadily improving; deemphasise unstable dir.

(*) Incremental updates → a ch8

- Full batch parameter updates → computationally costly (memory, time wise also)

(*) Note that in the 'gradient descent algorithm' previously pres;

Recall: (BR notation): (I)

$$E(\underline{W}) = \frac{1}{T}\sum_{t=1}^{T} Div(f(\underline{X}_t; \underline{W}), \underline{d}_t)$$

- overall error as an additive sum of per instance divergences

$$\nabla_{\underline{W}_R} E = \frac{1}{T}\sum_{t=1}^{T} \nabla_{\underline{W}_R} Div(\underline{Y}_t, \underline{d}_t)$$

- gradient distributes over sums linearly

a 5.9

(**) There are 2 key design choices here:-

i) The size of the training set we decide to use to compute the gradient of the error over.

ii) The frequency ~~of~~ with which we update parameters.

(*) At this stage of the lecture, a distinction is being made between computing the per-instance divergence and the overall error over the entire training set (batch), and making parameter updates accordingly.

(*) incremental updates occur ~~ad~~ refer to instance by instance computation of per instance divergence, additive error, ~~and~~ followed by parameter update after every instance.

(*) I have accidentally included other issues raised here, which will come up later: so (I) is useful to have for later exposition

---

(*) Pathological behaviour with incremental updating.

(*) The key point is that a cyclic behaviour as a pathology of algorithm behaviour can occur if we decide to do incremental updates without injecting stochasticity

(*) We can get stuck in cycles and algorithm fails to converge.

(*) Incremental update + SGD

- In context of incremental updates (param update after one training instance) we have to randomly permute training data between epochs

- definitions: (I) - Iterations can make multiple passes over training data
(II) - one pass through entire training set - epoch
(III) - Epoch over a training set of T samples results in T updates of the parameters (under incremental updates)

(*) Issues with incremental updates

- Heuristic example to understand when, and what conditions mean. updates work.
- Extreme case (data clotting):-

$$\frac{dE}{dw_{i,j}^{(k)}} = \frac{1}{T}\sum_{t=1}^{T} \frac{d\,Div(\underline{Y}(\underline{X}_t), \underline{d}_t)}{dw_{i,j}^{(k)}} = \frac{d\,Div(\underline{Y}(\underline{X}_i), \underline{d}_i)}{dw_{i,j}^{(k)}}$$

- When all training instances are same, average of derivatives over all per-instance divergences is same as derivative of single instance.

(*) Batch vs incremental updates (SGD) (AI2)

(SII) - Batch → operates over T training instances to get a single parameter update

(SIII): Incremental/online SGD:- Same computation yields T parameter updates

---

(W): Incremental/online SGD only reveals one sense of 'stochastic'; in the sense of random permutation of training instances for which parameter updates are conducted.

(W): There is another (related) sense in which stochastic gradient descent has statistical connections (in that we are computing an estimator of a comp. infeasible quantity)

(*) Contents → learning rate η with mean updates / SGD (A14)

- Intuitive explanation offered by BR is ~~contained in~~ is captured in formal results.
- The decreasing learning rate η is part of a package of assumptions that ensure that SGD converges:-
  - i) SGD converges almost surely to a global minimum for a convex loss
  - ii) SGD converges otherwise almost surely to a local minimum.
- A consequence of the Robbins-Siegmund theorem

(*) Reference material for SGD:- - also cited in DL book.

- i) Papers by Bottou → placed in folder.
  - → in particular 2003 SS lectures → (ch8)
- ii) more info in DL book (will supplement with formalism)

(*) There are further sufficient conditions on learning rate η (other than the decay) mentioned above to guarantee SGD convergence.

- Already noted
- Also corroborates with DL book concerning heuristic decay.

(*) Convergence results are a little unclear to me

(Q): Can you link this to your work on 36-705; that is conventional convergence / boundedness

- Is the convergence pointwise or uniform?

(*) Following results are placed here, but require further scrutiny:-

- Convergence: No. iterations taken to get within ϵ of optimal SOL.

$$|f(w^{(k)}) - f(w^*)| < \epsilon$$

(*) SGD convergence results

- Strongly convex fn; optimal learning rate:-

$|\underline{w}^{(k)} - \underline{w}^*| < \frac{1}{k} |\underline{w}^{(0)} - \underline{w}^*|$ "iterations to ϵ convergence $O(\frac{1}{\epsilon})$" (?)

- Generically convex
  - "ϵ-convergence" of $\frac{1}{\sqrt{k}}$ using learning rate of $\frac{1}{\sqrt{k}}$

DL Book Ch 8:

SGD for convex, excess error $J(\underline{\theta}) - \min_{\underline{\theta}} J(\underline{\theta}) = O(\frac{1}{\sqrt{k}})$ after k iter.

- SGD for strongly convex ——— " ——— $= O(\frac{1}{k})$

(*) Batch gradient convergence

- Strongly convex functions

$|\underline{w}^{(k)} - \underline{w}^{(*)}| < c^k |w^{(0)} - w^*|$ "ite to ϵ convergence $O(\log(\frac{1}{\epsilon}))$"

- Generically convex

"iterations to ϵ-convergence $O(\log(\frac{1}{\epsilon}))$" (?)

(*) Batch gradient descent is faster (corroborates DL) in theory

- SGD performs T updates for every batch update

DL (Goodfellow) ⟶ some discussion on Cramer-Rao bound, boundedness of generalisation error, practical adv. of SGD.

(*) See 10-725 notes; ~~+ 3 more~~ SGD not fully worked out yet from a formal classical / convex opt. perspective.

- Relation of empirical and expected error, pe training instance error

- going back to DL equations:-

$$J^*(\underline{\theta}) = \mathbb{E}_{(\underline{x},y)\sim p_{data}} L(f(\underline{x};\underline{\theta}),y)$$

$$J(\underline{\theta}) = \mathbb{E}_{(\underline{x},y)\sim \hat{p}_{data}} L(f(\underline{x};\underline{\theta}),y)$$

L - pe example loss
$f(\underline{x};\underline{\theta})$ - predicted output when input $\underline{x}$
$\hat{p}_{data}$ - empirical.
p_{data} - true distri.

ERM:

$$\min_{\underline{\theta}} J(\underline{\theta}) = \min_{\underline{\theta}} \mathbb{E}_{(\underline{x},y)\sim \hat{p}_{data}(\underline{x},y)} [L(f(\underline{x};\underline{\theta});y)] = \frac{1}{m}\sum_{i=1}^{m} L(f(\underline{x}^{(i)};\underline{\theta}),y^{(i)})$$

(empirical risk)

(Q): Having difficulty squaring BR's presentations symbolically with DL book.

- list claims verbally → re-review → (O/S)

- very confused by:- (BR's presentation)

i) we minimise empirical risk ✓

ii) Empirical risk is an unbiased estimate of expected loss

iii) NO guarantee minimising empirical risk will minimise expected loss

iv) variance of empirical risk $\text{var(loss)} = \frac{1}{N}\text{var(oliv)}$

(derivation given)

OR

v) large variance i.e. the $\underline{w}$ that minimises the empirical risk will differ significantly from $\underline{w}$ that minimises the expected loss.

(W): It will help if this is placed in a formal construct vs uniform convergence setting → (O/S) using std. notation

- BR very unclear here

(*) Heuristic point:-

- see graph; variance of 'true fn - estimated' is higher over incremental update SGD than over entire batch → (O/S) (2)

(*) mini-batch updates

- again, some (similar arguments which are very unclear

DL ch8: next piece clarifies what is going on.

MLE: $\underline{\theta}_m = \underset{\underline{\theta}}{\operatorname{argmax}} \sum_{i=1}^{m} \log p_{model}(\underline{x}^{(i)}, y^{(i)}; \underline{\theta})$

- ML estimation: max Σ (sum) $\Leftrightarrow$ max. exp. our empirical distri defin by training set.

i.e. $J(\underline{\theta}) = \mathbb{E}_{(\underline{x},y)\sim \hat{p}_{data}} \log p_{model}(\underline{x}, y; \underline{\theta})$

- properties of obj fn J can be viewed as expectations our training set

- (*) $\nabla_{\underline{\theta}} J(\underline{\theta}) = \nabla_{\theta} \mathbb{E}_{(\underline{x},y)\sim \hat{p}_{data}} \log p_{model}(\underline{x}, y; \underline{\theta})$

$= \mathbb{E}_{(\underline{x},y)\sim \hat{p}_{data}} [\nabla_{\underline{\theta}} \log p_{model}(\underline{x}, y; \underline{\theta})]$ tr.

- computing $\mathbb{E}(\cdot)$ very expensive; eval model on every example in dataset
- compute these expectations using a random sample of training data points and taking an average.

→ mini-batch

(*) motivation for mini-batch gradient descent :-

- follows gradient of generalisation error (expected) $J^*(\underline{\theta}) = \mathbb{E}_{(\underline{x},y)\sim p_{data}} L(f(\underline{x};\underline{\theta}), y)$

IF no examples are repeated

(*) discrete $\underline{x}, y$; abstract gen error (as a sum of arbitrary L) gen. error

$J^*(\underline{\theta}) = \sum_{\underline{x}} \sum_{y} p_{data}(\underline{x}, y) L(f(\underline{x};\underline{\theta}), y)$

exact grad. of gen. error

$g = \nabla_{\underline{\theta}} J^*(\underline{\theta}) = \sum_{\underline{x}} \sum_{y} p_{data}(\underline{x}, y) \nabla_{\underline{\theta}} L(f(\underline{x};\underline{\theta}), y)$

(*)(SM): Obtain an unbiased estimator of exact gradient of gen error, g by sampling a minibatch $\{x^{(1)}, \ldots, x^{(m)}\}$ of training examples with corres. targets $y^{(i)}$ from data gen. distri p_{data}.

- compute grad of loss wrt param on mini-batch

$$\hat{g} = \frac{1}{m} \nabla_\theta \sum_i L(f(x^{(i)}, \theta), y^{(i)})$$

est. of grad of loss wrt param over minibatch

- updating θ in direction of $\hat{g}$ → SGD or gen. error

(*) IF examples not re-used

(*) Minibatch conv.

- convex fn; conv. for SGD $O\left(\frac{1}{\sqrt{k}}\right)$
- minibatch updates with size b batches:- $O\left(\frac{1}{\sqrt{bk}} + \frac{1}{k}\right)$
  - improvement of $\sqrt{b}$ over SGD
  - But batch size b → b times as many comp/it. as SGD
  - degradation of $\sqrt{b}$.

In practice

- Non conv. obj.
- mini batches more effective with right η.
- vector processing