11-785 13/05/2020

L7: SGD, Overfitting, regular., tricks

Recap

- Rprop, quickprop → dependency lost → efficiency issues
- momentum smooth fluctuations via using running averages
  - Polyak → read
- Nesterov's method
  - Nesterov pape (in Russian)
- incremental methods: size of training set pertited.
  - potentially leads to high variance of estimates
- (*) convergence for incremental updates depends on learning rate (schedule)
  - sufficient conditions in learning rate

(*) Still require learning rate schedule

(*) trend algos (topic)

- Every time you update, doing it on different loss (in the sense that it is over a completely different set of samples)
- variance in this case refers to differences of loss functions and recommended updates across sample draws.

(*) momentum helps (in this case?) - using running averages
  - dampen/smooth oscillations (of what?); updates over error surface?
  (*) incremental SGD, mini-batch → momentum may perform variance cleanup

---

(*) minibatch with momentum pseudocode

(*) Nesterov's accelerated gradient
  - Superior to standard momentum
  - provably faster (in Russian pape)

(*)
- Also can be combined with incremental methods

(BR): momentum style methods using running averages shrink variance of estimated gradients

(W): need to clarify variance of what exactly

(BR): what is being smoothed using running average

BR: only using 1st moment of derivatives (there may be other moments using / leveraging other statistical info)

momentum

(W)(A2): compute expected value of derivative at the next step; take step along expected value → sufficient?

- expectation may be 0; but variance may be high (~~do not~~ test 1st mom) or variance near 0 (1st mom ok)

---

smoothing trajectory:

(*) Horizontal direction: - steps always positive

vertical —11— :- steps swinging sign

(*) This is after momentum is applied (?)

- make use of 2nd moments of corrections

(*) momentum → running average; but still have learning rate

BR: Account for 2nd moment of deriv by adjusting learning rate (?)

BR: dampen step size in directions of high variance

increase —11— in —11— low variance.

(*) Normalise steps with variance (high-level)

- BR: compare net movement vs total movement (arbitrary) direct.)

i) if in past, total movement in g-comp high } shrink step size η_j

net movement —11— small }

ii) —11— total movement large } increase step size η_j

net movement large }

(*) modify gradient based update with relative scaling in axis dir; according to variation.

RMS prop (Hinton)

- Notation:- $\partial_w D$ is same as $\frac{\partial D}{\partial w}$ (derivative

$\partial_w^2 D$ is not $\frac{\partial^2 D}{\partial w^2}$ but $\left[\frac{\partial D}{\partial w}\right]^2$ (i.e. squared deriv; not 2nd

$\mathbb{E}[\partial_w^2 D] = \mathbb{E}\left[\left(\frac{\partial D}{\partial w}\right)^2\right]$ (mean squared deriv.)

RMS Prop

(*) Focus is mean squared derivative - use

(*). 2nd moment of derivative

$$(*)\quad \mathbb{E}[\partial_w^2 D]_k = \gamma\, \mathbb{E}[\partial_w^2 D]_{k-1} + (1-\gamma)(\partial_w^2 D)_k$$

$$(*)\quad w_{k+1} = w_k - \frac{\eta}{\sqrt{\mathbb{E}[\partial_w^2 D]_k + \epsilon}}\, \partial_w D$$

- standard gradient descent (*)
- normalising learning rate by RMS value (*)

(*) Similarity to Rprop.

RMSprop pseudocode

(*) maintain running average of 2nd moment of the derivative $\mathbb{E}[\partial_w^2 D]_k$

(*) Normalise by square root of 2nd moment.

Q: where is the 1st moment?

- momentum / Nesterov - 1st moment; RMSprop - 2nd moment

- BR: Try both?

ADAM: RMSprop with momentum

- ADAM utilises smoothed version of momentum-augmented gradient with both 1st and 2nd moments

(*) - maintain running estimate of mean derivative for each param.
- maintain —— " —— of mean squared value of derivatives —" —
- scale update of param by inverse of root mean squared deriv.

(01): $m_R = \delta m_{R-1} + (1-\delta)(\partial_W D)_R$

$v_R = \gamma v_{R-1} + (1-\gamma)(\partial_W^2 D)_R$

(?)(?) - surely divide by 0?

$$\hat{m}_R = \frac{m_R}{1-\delta^R} \qquad \hat{v}_R = \frac{v_R}{1-\gamma^R}$$

$$W_{R+1} = W_R - \frac{\eta}{\sqrt{\hat{v}_R + \epsilon}} \hat{m}_R$$

(*) $(1-\delta^R)$ - typ. delta values close to 1 ⇒ initialisation of m_{R-1} and v_{R-1} at 0
⇒ output $(1-\delta)$ close to 0 as δ close to 1
- slow update at early it of means/variances; 1st, 2nd moments

AD HOC ext/correctn

⇒ Early iterations → don't prioritise $(1-\delta)$
As $R \to \infty$, δ^R and γ^R will become large; (0)

(W)(02): A little cont. here.
- Review this

- Through this lens; can generate more ideas/tricks
- Alex Radford animations (W)(A2) - (Q) - animations

Long valley

- RMSprop - steps become fixed when slope is constant

$$\frac{\text{derivative}}{\text{RMS value of deriv}} = 1 \Rightarrow \text{fixed step size}$$

(*) Recall

$$\text{Loss} = \frac{1}{T}\sum_t \text{Div}(\underline{Y}_t, \underline{d}_t; \underline{W}_1, \underline{W}_2, \ldots, \underline{W}_K)$$

(error)

BR: Ability to minimise loss; and hence divergence depends on nature of divergence.

- Desiderata of good divergence
- (*) - Hill climbing is a really good analogy for this

(*) more 'bowly' than quadratic → flat near min.

(*) Divergence depends on activation (pre-req.)

- L_2 divergence and KL divergence
  - quad
  - ?

(*) Think that L_2 is bowl-shaped X wrong?

- KL divergence same as cross-entropy, with extra term.

L_2 or KL single (logistic fn)

- consider ~~perceptron~~ with 2 inputs, no bias
- compute output over training samples
- compute output of logistic function
- compute L_2 dir / KL div on logistic output and target outputs.
- Parameters are the 2 weights of logistic function (?)
  - 2 dimensions
- (*) (!!) :- It's not the shape wrt output of classifier
  - Rather the shape of loss wrt to parameters of the classifier

↓

KL divergence → convex bowl; L_2 - non convex (although global minimum)

(?) - We don't know what loss fn looks like
- Examine in case near end of network → pedagogical heuristic for intuition

- How do methods leave saddle points

(*) Local behaviour depends on nature of error surface & where gradients point

(*) (FYI): Difficult to say much is optimal → we don't know what error surface in general looks like at any point

- But with loss function of a particular kind, we can recommend techniq, as we know how they behave.

(*) Story so far
- see slides.

(*) Convergence can be improved with smoothed updates
e.g. RMSprop, ADAM etc.

(*) Distinction between variance from randomly sampling data differently
AND natural variance over entire batch

(FYI) Need assurance of convergence to 'right spot' our entire training data.

(Q): using normalisation of step size via 2nd moments (learning rate)
- do we reduce learning rate according to a schedule as before?

BR: learning rate adjusts

- 2nd moment techniques - do not exogenously adjust learning rate η as 2nd moment adjusts η automatically; excludes need for tuning out optimal learning rate / schedule

BUT: we have other hyperparams to tune.
(related to initial step size, & how running avs. maintained)
- no such thing as a free lunch

(#) Generalisation / tricks of the trade:

- dropout
- batch norm
- gradient clip, data aug, hacks

(*) See papers

---

backprop → error backprop

$$\nabla_{\underline{z}} \frac{1}{2}\|\underline{y}-\underline{d}\|_2^2 = (\underline{y}-\underline{d})\underline{J}_{\underline{y}}(\underline{z})$$

- Propagate $(\underline{y}-\underline{d})$ backwards

(*) Choice of divergence affects both learned network and results

---

(*) Covariate shifts

- Assumption using mini-batch :-

(*) 'Each mini-batch has more or less same statistics as every other mini-batch so that minimising for one mini-batch ⇒ close to minimum for other mini-batches; and that overall mini-batches; correct soln.'

(*) When minibatches are far apart, minimisation for one does not give info about others.

(W) (AY): uncomfortable → presentation too informal review

- Is this to do with uniform convergence?
- Is there a better way of presenting using this?
- What is 'space' of the training samples shown

(?) - don't understand his heuristic presentation of moving mini-batches → standard mean 0, unit variance; then move pile to where they think they lived if considered all of them at once

---

Batch normalisation

(*) Covariate adjustment after the weighted addition of inputs (affine trans.) and before activation

Training: Adjustment over indiv mini-bats.

- done independently for each unit

(*) Purpose of batch norm → cancel out covariate shifts

Batch norm:

- standardise mini-batch; 'move it to correct pos'; to be determined via a parameterised setting

(OTT)

(AG): Review + list diagram: → (x)(x)(x)

- compute mean of mini-bat μ_B and σ_B^2 (estimators?)
- Normalise instances of mini-batch (every)
- Then conduct shift to a unit-specific location (per neuron)
  - as seen for each neuron live in a specific location param space

- (*) comprised of 2 operations (key)

Derivatives: exploits additivity

- Batch norm complicates additivity.

$$Div(MB) = \frac{1}{B}\sum_t Div(Y_t(X_t, \mu_B, \sigma_B^2), d_t(X_t))$$

- Divergence for all Y_t depends on all X_t within mini-batch.

Batch norm - vector fn. over minibatch

(AT): Review

- (*) shows not dimensions of input, but B instances of mini-batch
- (*) each line rep one training instance of minibatch at one node in the network
- (*) Note dependence / influence diag style arg.
- Derivative computed globally over entire minibatch (must consider all $\hat{z}_j$s wrt z_i
- not locally for every instance in mini-batch
- One to interactions.

Batchnorm dependency diagram (48) (57)

(48): Review

(*) This should be sufficient to write derivatives. (next slides)

-Batchnorm - backprop

(*) Complexity in backprop happens after $\frac{dDiv}{d\hat{z}} = f'(\hat{z}) \frac{dDiv}{dy}$ step.

(*) For each neuron have γ and β parameters

-Have to compute deriv wrt γ and β also.

- μ and σ_B are multi-batch specific during forward pass

- Running mean - separate

BR: $\hat{z} = \gamma u + \beta$ u - computed during forward pass

$$\frac{dDiv}{d\hat{z}}$$

$$\frac{dDiv}{d\beta} = \frac{dDiv}{d\hat{z}} \cdot \underbrace{\frac{d\hat{z}}{d\beta}}_{=1}$$

$$= \frac{dDiv}{d\hat{z}}$$

$$\frac{dDiv}{d\gamma} = \frac{dDiv}{d\hat{z}} \underbrace{\frac{d\hat{z}}{d\gamma}}_{=u} = \frac{dDiv}{d\hat{z}} u$$

· Have to keep backpropagating

$$\frac{dDiv}{du} = \frac{dDiv}{d\hat{z}} \frac{d\hat{z}}{du} = \frac{dDiv}{d\hat{z}} \gamma$$

BR: Issue is taking $\frac{\partial Div}{\partial u}$ back to unnormalised $\underline{z}$

Final step:- $\frac{\partial Div}{\partial z_i}$

$Div = function(u_i, \mu_B, \sigma_B^2)$ $\quad u_i = f(z_i, \mu_B, \sigma_B^2)$

$$\frac{\partial Div}{\partial z_i} = \frac{\partial Div}{\partial u_i}\frac{\partial u_i}{\partial z_i} + \frac{\partial Div}{\partial \sigma_B^2}\cdot\frac{\partial \sigma_B^2}{\partial z_i} + \frac{\partial Div}{\partial \mu_B}\cdot\frac{\partial \mu_B}{\partial z_i}$$

(A8):
(*) derivation of $\frac{\partial Div}{\partial \sigma_B^2}$, $\frac{\partial \sigma_B^2}{\partial z_i}$

(*) derivation of $\frac{\partial Div}{\partial \mu_B}$, $\frac{\partial \mu_B}{\partial z_i}$

(*) derivation of $\frac{\partial Div}{\partial u_i}$, $\frac{\partial u_i}{\partial z_i}$ (C11)

BR: $\frac{\partial Div}{\partial z_i}$ - simple formula → work though it

---

Batch normalis: inference

(A9): A lot to review → (*)

Batch normalis.

(*) no need to apply to all layers
(*) anecdotal evidence → no need for dropout
(*) Batch norm: within each batch for each neuron
- for sigmoids, moves data to where 'it's most linear'
i.e. 0 position (?) (?)
- for RELUS: using 0 as standard mean, not good idea
- due to ½ data ending up in '0 area' where there is no derivative

(A7): Review
(C11)
- use influence diagram (to account for all paths)
- remember route of influence topologically through network

Batchnorm - empirical result

- Ioffe & Szegedy (2015) - greatly reduce training time

---

data underspecification

- 2D plot; use training points (see divergence diagrams)
- overfitting
- BR(*): get a sense of how little data we have (esp. in high dimensions)

  e.g 100 dim, binary inputs

  - inputs on corners of 100-dim hypercube
  - How many corners? 10^{30} locations where data can exist
- E.g. 1 quadrillion training instances - 10^{15}

(**) High dimensionality

$$\frac{10^{15}}{10^{30}} = \frac{1}{10^{15}}$$

(*) Require additional constraints

---

weight manipulation

- (*) indiv neurons capable of arbitrary changes in steepness
  - (**) (A8): How does sigmoid activation change in response to inputs / affine input weights changing?

(*) → Heaviside step rather than smooth fn.

(*) ⇒ steep changes → overfitted responses → perceptrons with large w

(*) constrant weights to be low → ↓ smooth output

(regularisation)

- Place constraints on weights
- Add regularisation via modifying loss via weight regulariser

---

Regularising weights

- lambda - regulaser

Network structure

(*) to impose smoothness

(*) For a given no. param deeper nets impose more smoothness than shallow ones.

- Each layer places constraints on what can be 'represented'
- Next layer treates what it already has

(*) Depth and training data for deeper nets.

(A11) - Review insight

- L1 regularisation
- Noise added reg.
- Dropout

} → (A12) go through review