Deep Learning

Lecture 6

Nathanaël Perraudin
based on Thomas Hofmann lectures

Swiss Data Science Center
ETH Zurich and EPFL – datascience.ch

October 29, 2018

Overview

1. Optimization for Deep Networks

2. Stochastic Gradient Descent

Section 1

Optimization for Deep Networks

Learning as Optimization

Machine Learning: uses optimization, but is not equal to optimization

First: empirical risk is only a proxy for the expected risk

- practically: monitoring on validation set, early stopping
- we should not overfit the training data

Second: loss function may only be a surrogate

- for instance: logistic loss instead of (0/1)-classification error
- we should not overfit the loss function
- (finally: we should not overfit to the task)

Objectives as Expectations

Typical structure of learning objective: large finite sums

- many relevant quantities: sums over all training instances
- example: gradient

$$\nabla_\theta \mathcal{R}(\mathcal{S}_N) = \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta \mathcal{R}(\theta; \mathbf{x}[i], \mathbf{y}[i])$$

- accuracy-complexity trade-off: subsample terms in sum
- in practice: use of mini-batches of data

Large Scale Learning

Some (not all!) data sets have grown “faster” than compute power

Or let us say: computation, not data, is the bottleneck.

Need to trade-off statistical power (more data) with computational power (memory, compute cycles):

- “super-trooper” algorithm: process few data points in an expensive manner
- “cheap & easy” algorithm: process many data points in a cheap manner
- practically: favor cheap over expensive

Bousquet & Bottou. "The tradeoffs of large scale learning." NIPS 2008.

Gradient Descent

Compute full gradient (across all parameters) and descent

$$\theta(t+1) = \theta(t) - \eta \nabla_\theta \mathcal{R}$$

- $\eta > 0$: step size or learning rate – alternatively: use of line search
- continuous time dynamics: ordinary differential equation $=$ gradient flow (Euler's method)

$$\dot{\theta} = -\nabla_\theta \mathcal{R}$$

Gradient Descent: Classic analysis

Convex objective $\mathcal{R}$

- $\mathcal{R}^*$: minimum, θ^*: minimizer, i.e. $\mathcal{R}^* = \mathcal{R}(\theta^*) \leq \mathcal{R}(\theta)$
- $\mathcal{R}$ has L-Lipschitz-continuous gradients $\Longrightarrow$

$$\mathcal{R}(\theta(t)) - \mathcal{R}^* \leq \frac{2L}{t+1} \|\theta(0) - \theta^*\|^2 \in \mathbf{O}\left(t^{-1}\right)$$

- $\mathcal{R}$ is μ-strongly convex in θ $\Longrightarrow$

$$\mathcal{R}(\theta(t)) - \mathcal{R}^* \leq \left(1 - \frac{\mu}{L}\right)^t (\mathcal{R}(\theta(0)) - \mathcal{R}^*)$$

  - exponential convergence ("linear rate")
  - rate depends adversely on condition number $\frac{L}{\mu}$
- Lower bound (general case): $\mathbf{O}\left(t^{-2}\right)$ – achieved by Neterov acceleration

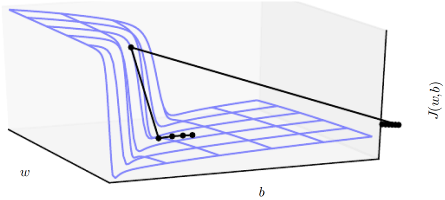
Proofs link: `https://perso.telecom-paristech.fr/rgower/pdf/M2_statistique_optimisation/grad_conv.pdf`

Challenge : curvature

[cf. DL, Section 8.2.3]

Models with multiplication of many weights (depth, recurrence): sharp non-linearities

- Very large Lipschitz constant
- Would theoretically require very small gradient steps → very slow optimization



Motivates gradient clipping heuristics and learning rate decay.

Challenges: Curvature

[cf. DL, Section 8.2.1]

Curvature may require to use small step sizes:

$$\mathcal{R}(\theta - \eta\nabla\mathcal{R}) \overset{\text{Taylor}}{\approx} \mathcal{R}(\theta) - \eta\|\nabla\mathcal{R}\|^2 + \frac{\eta^2}{2}\underbrace{\nabla\mathcal{R}^\top\mathbf{H}\nabla\mathcal{R}}_{=\|\nabla\mathcal{R}\|^2_{\mathbf{H}}}$$

- Hessian matrix: $\mathbf{H} := \left[\nabla^2\mathcal{R}\right]$
- problematic ill-conditioning:

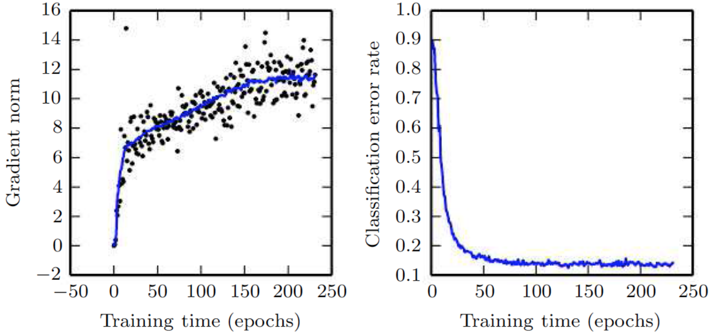
$$\frac{\eta}{2}\|\nabla\mathcal{R}\|^2_{\mathbf{H}} \;\gtrsim\; \|\nabla\mathcal{R}\|^2$$

- remedy for first order methods: small step sizes η

Challenges: Curvature

[cf. DL, Section 8.2.1]

Gradient descent may not arrive at a critical point of any kind



Can be checked empirically!

Challenges: Local Minima

[cf. DL, Section 8.2.3]

Neural network cost functions can have many local minima and/or saddle points – and this is typical. Gradient descent can get stuck.

Questions

- Are local minima a practical issue? Sometimes not: Gori & Tesi, 1992
- Do local minima even exist? Sometimes not (auto-encoder): Baldi & Hornik, 1989
- Are local minima typically worse? Often not (large networks): e.g. Choromanska et al, 2015
- Can we understand the learning dynamics? Deep linear case has similarities with non-linear case, e.g. Saxe et al., 2013

Least Squares: Preliminaries

Assumptions, Notation, Identities

- inputs whitened $\mathbf{E}[\mathbf{x}\mathbf{x}^\top] = \mathbf{I}$
- trace identities:

$$\mathbf{v}^\top \mathbf{w} = \sum_i v_i w_i = \mathrm{Tr}(\mathbf{v}\mathbf{w}^\top), \quad \mathrm{Tr}(\mathbf{A}+\mathbf{B}) = \mathrm{Tr}(\mathbf{A}) + \mathrm{Tr}(\mathbf{B})$$

- $\mathbf{E}\mathrm{Tr}(\mathbf{X}) = \mathrm{Tr}\mathbf{E}[\mathbf{X}]$ (b/c linearity of trace)

Objective (with $\mathbf{A} \in \mathbb{R}^{m \times n}$, such that $F(\mathbf{x}) = \mathbf{A}\mathbf{x}$)

$$\mathcal{R}(\mathbf{A}) = \mathbf{E}\|\mathbf{y} - \mathbf{A}\mathbf{x}\|^2$$

- expectation with regard to empirical distribution
- average over $(\mathbf{x}, \mathbf{y})$-training pairs

Least Squares: Single Layer Linear Network

Rewrite objective

$$\begin{aligned}
\mathcal{R}(\mathbf{A}) =& \operatorname{Tr} \mathbf{E}\left[(\mathbf{y}-\mathbf{A}\mathbf{x})(\mathbf{y}-\mathbf{A}\mathbf{x})^\top\right] \\
=& \underbrace{\operatorname{Tr} \mathbf{E}\left[\mathbf{y}\mathbf{y}^\top\right]}_{\text{indep. of } \mathbf{A}} \\
&+ \operatorname{Tr}\left(\mathbf{A} \underbrace{\mathbf{E}\left[\mathbf{x}\mathbf{x}^\top\right]}_{= \mathbf{I} \text{ by assumpt.}} \mathbf{A}^\top\right) \\
&- 2\operatorname{Tr}\left(\mathbf{A} \underbrace{\mathbf{E}\left[\mathbf{x}\mathbf{y}^\top\right]}_{:= \mathbf{\Gamma}^\top,\, \mathbf{\Gamma} \in \mathbb{R}^{m\times n}}\right)
\end{aligned}$$

Least Squares: Single Layer Linear Network

Gradient (in denominator matrix layout)

$$\nabla_{\mathbf{A}} \mathcal{R} = \nabla_{\mathbf{A}} \mathsf{Tr}\left(\mathbf{A}\mathbf{A}^\top\right) - 2\nabla_{\mathbf{A}} \mathsf{Tr}\left(\mathbf{A}\mathbf{\Gamma}^\top\right) = 2\left(\mathbf{A} - \mathbf{\Gamma}\right)$$

- trace differentiation rules (cf. wikipedia, The Matrix Cookbook)

$$\nabla_{\mathbf{A}} \mathsf{Tr}(\mathbf{A}\mathbf{A}^\top) = 2\mathbf{A}, \quad \nabla_{\mathbf{A}} \mathsf{Tr}(\mathbf{A}\mathbf{B}) = \mathbf{B}^\top$$

- gradient descent: $\mathbf{A} \to \mathbf{\Gamma}$ (simple)

Least Squares: Two-Layer Linear Network

[Saxe, McClelland & Ganguli, 2013]

Simple two-layer linear network with squared error $\mathbf{A} = \mathbf{Q}\mathbf{W}$ (with $\mathbf{Q} \in \mathbb{R}^{m \times k}$, $\mathbf{W} \in \mathbb{R}^{k \times n}$, k : width of hidden layer)

$$\mathcal{R}(\mathbf{Q}, \mathbf{W}) = \text{const.} + \text{Tr}\left(\mathbf{Q}\mathbf{W} \cdot (\mathbf{Q}\mathbf{W})^\top\right) - 2\text{Tr}\left(\mathbf{Q}\mathbf{W} \cdot \mathbf{\Gamma}^\top\right)$$

Taking derivatives

$$\tfrac{1}{2}\nabla_{\mathbf{Q}}\mathcal{R} = (\mathbf{Q}\mathbf{W})\mathbf{W}^\top - \mathbf{\Gamma}\mathbf{W}^\top = (\mathbf{A} - \mathbf{\Gamma})\,\mathbf{W}^\top \in \mathbb{R}^{m \times k}$$

$$\tfrac{1}{2}\nabla_{\mathbf{W}}\mathcal{R} = \mathbf{Q}^\top (\mathbf{A} - \mathbf{\Gamma}) \in \mathbb{R}^{k \times n}$$

Least Squares: Two-Layer Linear Network

Perform SVD of $\mathbf{\Gamma}$ (only data dependence) $\mathbf{\Gamma} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$.

Linearly transform variables: $\tilde{\mathbf{Q}} = \mathbf{U}^\top \mathbf{Q}$ and $\tilde{\mathbf{W}} = \mathbf{W}\mathbf{V}$

$$\mathbf{A} - \mathbf{\Gamma} = \mathbf{Q}\mathbf{W} - \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top = \mathbf{U}\left(\tilde{\mathbf{Q}}\tilde{\mathbf{W}} - \mathbf{\Sigma}\right)\mathbf{V}^\top$$

Gradients in new parametrization

$$\frac{1}{2}\nabla_{\tilde{\mathbf{Q}}}\mathcal{R} = \mathbf{U}^\top \nabla_{\mathbf{Q}}\mathcal{R} = \underbrace{\mathbf{U}^\top\mathbf{U}}_{=\mathbf{I}}(\tilde{\mathbf{Q}}\tilde{\mathbf{W}} - \mathbf{\Sigma})\underbrace{\mathbf{V}^\top\mathbf{V}}_{=\mathbf{I}}\tilde{\mathbf{W}}^\top = (\tilde{\mathbf{Q}}\tilde{\mathbf{W}} - \mathbf{\Sigma})\tilde{\mathbf{W}}^\top$$

$$\frac{1}{2}\nabla_{\tilde{\mathbf{W}}}\mathcal{R} = \tilde{\mathbf{Q}}^\top\left(\tilde{\mathbf{Q}}\tilde{\mathbf{W}} - \mathbf{\Sigma}\right)$$

- note that: $(\tilde{\mathbf{Q}}\tilde{\mathbf{W}} - \mathbf{\Sigma}) = \left(\mathbf{U}^\top\mathbf{A}\mathbf{V} - \mathbf{\Sigma}\right)$.

Least Squares: Two-Layer Linear Network

Define $\mathbf{w}_r :=$ r-th column of $\tilde{\mathbf{W}}$, $\mathbf{q}_r :=$ r-th row of $\tilde{\mathbf{Q}}$.

We can write the gradients as (*left as a simple excercise*)

$$\tfrac{1}{2}\nabla_{\mathbf{q}_r}\mathcal{R} = (\mathbf{q}_r^\top \mathbf{w}_r - \sigma_r)\mathbf{w}_r + \sum_{s\neq r}(\mathbf{q}_r^\top \mathbf{w}_s)\mathbf{w}_s$$

$$\tfrac{1}{2}\nabla_{\mathbf{w}_r}\mathcal{R} = (\mathbf{q}_r^\top \mathbf{w}_r - \sigma_r)\mathbf{q}_r + \sum_{s\neq r}(\mathbf{q}_s^\top \mathbf{w}_r)\mathbf{q}_s$$

Equivalent energy function

$$\tilde{\mathcal{R}}(\tilde{\mathbf{Q}}, \tilde{\mathbf{W}}) = \sum_r (\mathbf{q}_r^\top \mathbf{w}_r - \sigma_r)^2 + \sum_{s\neq r}(\mathbf{q}_s^\top \mathbf{w}_r)^2$$

- cooperation: same input-output mode weight vectors align
- competition: different mode weight vectors are decoupled

Least Squares: Two-Layer Linear Network

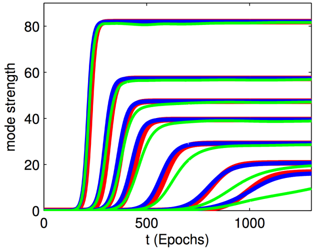
As learning advances: modes decouple $=$ independent learning dynamics for each mode

For matched weight a, b the dynamics is governed by a loss of the form

$$\ell(a, b) = (\sigma - ab)^2$$

which can be fully analysed (in the continuous time limit of infinitesimal step sizes).

Simulation experiments:



- red: analytic
- blue $=$ linear
- green $=$ tanh

Challenges: Conclusion

- despite the deficit in theoretical analyses (compared to the convex case)
- ... gradient descent may work in practice, but:
- ... certain modifications are required
- ... computational as well as conceptual
- ... more complex learning dynamics
  (even in deep linear networks)

Section 2

Stochastic Gradient Descent

Stochastic Gradient Descent

Stochastic gradient descent: chose update direction $\mathbf{v}$ at random such that $\mathbf{E}[\mathbf{v}] = -\nabla \mathcal{R}$.

- randomization scheme is unbiased

SGD via subsampling

- pick random subset $\mathcal{S}_K \subseteq \mathcal{S}_N$, $K \leq N$
- note that $\mathbf{E}\mathcal{R}(\mathcal{S}_K) = \mathcal{R}(\mathcal{S}_N) \Longrightarrow \mathbf{E}\nabla\mathcal{R}(\mathcal{S}_K) = \nabla\mathcal{R}(\mathcal{S}_N)$
- SGD update step (randomization at each t)

$$\theta(t+1) = \theta(t) - \eta(t)\nabla\mathcal{R}(t), \quad \mathcal{R}(t) := \mathcal{R}(\mathcal{S}_K(t))$$

Stochastic Gradient Descent

In practice: permute instances and break-up into mini-batches

Epoch = one sweep through the data

- harder to analyse theoretically
- typically works better in practice
- no permutation $\Longrightarrow$ danger of "unlearning"

Mini-batch size

- "standard SGD": $k = 1$, often most efficient in terms of #backprop steps
- but: larger k better for utilizing concurrency in GPUs or multicore CPUs

Stochastic Gradient Descent: Rates

Under certain conditions SGD converges to the optimum:

- convex or strongly convex objective
- Lipschitz gradients
- decaying learning rate: $\sum_{t=1}^{\infty} \eta^2(t) < \infty$, $\sum_{t=1}^{\infty} \eta(t) = \infty$, typically: $\eta(t) = Ct^{-\alpha}$, $\frac{1}{2} < \alpha \leq 1$ (cf. hyperharmonic series)
- iterate (Polyak) averaging

Convergence rates

- strongly-convex case: can achieve $O(1/t)$ suboptimality rate
- non-strongly convex case: $O(1/\sqrt{t})$ suboptimality rate

Stochastic Gradient Descent: Practicalities

Almost none of the analysis applies to the non-convex case.

Choosing a learning rate schedule can be a nuisance.

Fast decay schedules may lead to super-slow convergence

In practice: tend to use larger step sizes and level out at a minimal step size.

- justification: SGD with fixed steps size is known to converge to a ball around the optimum (strongly convex case)

Common belief: stochasticity of SGD is a “feature”

- escape from regions with small gradients via perturbations

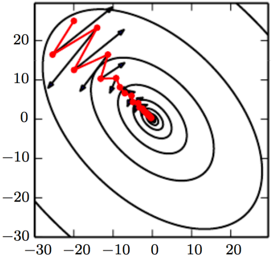
Momentum

Accumulate the gradient over several updates, as a ball would accumulates speed

- Update the momentum:

$$m(t) = \alpha m(t-1) - (1-\alpha)\nabla_\theta \mathcal{R}(\theta(t-1)), \quad \alpha < 1$$

- Bias correction: $\hat{m}(t) = \frac{m(t)}{(1-\alpha^t)}$
- Update θ: $\theta(t) = \theta(t-1) + \eta \hat{m}(t)$



Nesterov Momentum

Compute the gradient where momentum push you and make a correction

- The gradient is evaluated at $\theta(t-1) + \eta\alpha\hat{m}(t-1)$

$$v(t) = \alpha v(t-1) - \epsilon \nabla_\theta \mathcal{R}(\theta(t-1) + \eta\alpha\hat{m}(t-1)), \quad \alpha < 1$$

Better in practice

AdaGrad

Consider the entire history of gradients – gradient matrix

$$\theta \in \mathbb{R}^d, \quad \mathbf{G} \in \mathbb{R}^{d \times t_{\mathsf{max}}}, \quad g_{it} = \left.\frac{\partial \mathcal{R}(t)}{\partial \theta_i}\right|_{\theta=\theta(t)}$$

Compute (partial) row sums of $\mathbf{G}$ (note: not(!) gradient norms)

$$\gamma_i^2(t) := \sum_{s=1}^{t} g_{is}^2$$

Adapt learning rate per parameter

$$\theta_i(t+1) = \theta_i(t) - \frac{\eta}{\delta + \gamma_i(t)} \nabla \mathcal{R}(t), \quad \delta > 0 \text{ (small)}$$

AdaGrad (cont'd) and RMSprop

Intuitively: learning rate decays faster for weights that have seen significant updates

Theoretical justification: regret bounds for convex objectives ([Duchi, Hazan, Singer, 2011]; beyond the scope of this lecture)

Non-convex variant of AdaGrad: RMSprop [Tieleman & Hinton, 2012]

$$\gamma_i^2(t) := \sum_{s=1}^{t} \rho^{t-s} g_{is}^2, \quad \rho < 1$$

- moving average, exponentially weighted

ADAM

ADAM is derived from adaptive moment estimation [Kingma & Ba, 2014]

AdaGrad + Momentum for SGD

- $g(t) = \nabla_\theta \mathcal{R}(\theta(t-1))$ [Get the gradient]
- $m(t) = \beta_1 m(t-1) + (1-\beta_1) g(t)$ [Update the biased first moment estimate]
- $v(t) = \beta_2 v(t-1) + (1-\beta_2) g(t)^2$ [Update the biased second raw moment estimate]
- $\hat{m}(t) = m(t)/(1-\beta_1^t)$ [Bias correction first moment estimate]
- $\hat{v}(t) = v(t)/(1-\beta_2^t)$ [Bias correction second raw moment estimate]
- $\theta(t) = \theta(t-1) + \eta \hat{m}(t)/(\sqrt{\hat{v}(t)} + \epsilon)$ [Update parameters]

Typical values: $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$,