Faculty of Mathematics, Physics and Informatics
Comenius University in Bratislava

Neural Networks

Lecture 5

Gradient-based learning and optimization

Igor Farkaš

2021

Optimization vs NN learning

- Although optimization provides a way to minimize the loss function for NN learning, the goals are fundamentally different:
- goal of optimization = to reduce the training error
- goal of NN learning (statistical inference) = to reduce expected generalization error (risk) => ML acts indirectly
- We have only access to a finite training data sample (not the whole data distribution)
- Empirical risk minimization: $E_{(\mathbf{x},d)\sim p(\text{data})}[Loss(f(\mathbf{x};\mathbf{w}),d)]$
- … is based on a finite training sample $\{\mathbf{x},d\}$, rather than known data distribution, hence is prone to overfitting.

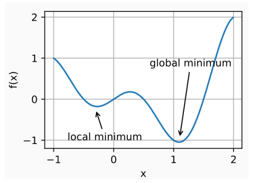
(Goodfellow et al, 2015)

Surrogate loss function

- Sometimes, the loss function we actually care about (e.g. classification error) is not one that can be optimized efficiently.
- Exactly minimizing expected 0-1 loss is typically intractable (exponential in the input dimension)
- In such situations, one typically optimizes a surrogate loss function instead, which acts as a proxy, but has advantages:
- e.g. the negative log-likelihood of the correct class is used $(-\log P(y_i))$
- test set 0-1 loss often continues to decrease for a long time after the training set 0-1 loss has reached zero, which improves the robustness of the classifier by further pushing the classes apart from each other
- This leads to extracting more information from the training data (than would have been possible by simply minimizing the average 0-1 loss on training set).

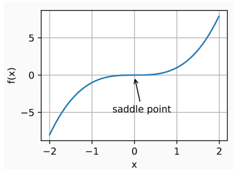
Problems in gradient-based NN learning

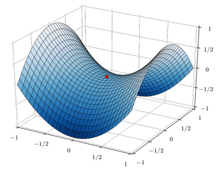
Local minima



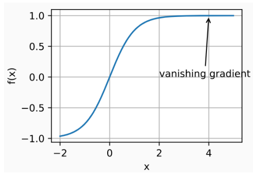
x = weights,
F = loss f.

Saddle points





Vanishing gradients



Ill-conditioning of Hessian matrix $\mathbf{H}$, i.e. rate of its change for small $\Delta\mathbf{w}$

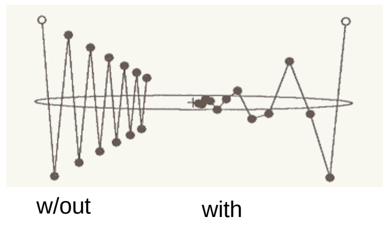
- given by condition number (CN) = ratio of its max/min eigenvalues
- for large CN, $\mathbf{H}^{-1}$ is particularly sensitive to error in the input

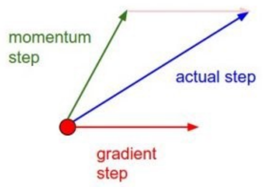
These problems slow down or hinder convergence.

Early modifications of gradient descent learning

Adding a momentum: $\Delta\mathbf{w}(t) = -\alpha\nabla E(\mathbf{w}(t)) + \gamma\Delta\mathbf{w}(t-1)$ $\quad 0 \le |\gamma| < 1$

- helps speed up SGD and dampen oscillations

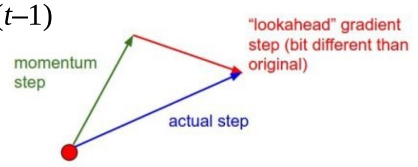




Nesterov accelerated gradient (Nesterov, 1983)

$\Delta\mathbf{w}(t) = -\alpha\nabla E[\mathbf{w}(t) + \gamma\Delta\mathbf{w}(t-1)] + \gamma\Delta\mathbf{w}(t-1)$

- helps adjust speed by looking into the near future



Role of the Hessian matrix

- $\mathbf{H}$ plays an important role in supervised training of neural networks:

$$\mathbf{H} = \left[\frac{\partial^2 E(\mathbf{w})}{\partial w_i \partial w_j} / \mathbf{w}_0\right]_{ij}$$

- Eigenvalues of $\mathbf{H}$ have a profound influence on the dynamics of back-propagation learning (condition number = ratio of max/min eigenvalues)
- The inverse of $\mathbf{H}$ provides a basis for pruning (i.e., deleting) insignificant synaptic weights from a multilayer perceptron.
- $\mathbf{H}$ is basic to the formulation of second-order optimization methods as an alternative to BP learning.
- Typical profile of $\mathbf{H}$ in BP learning (LeCun et al., 1998): a few small eigenvalues, many medium-sized eigenvalues, and a few large eigenvalues => a wide spread in the eigenvalues of the Hessian.

Towards second-order optimization methods

$E(\mathbf{w}) = E(\mathbf{w}_0) + \mathbf{g}^{T}(\mathbf{w}_0)\Delta\mathbf{w} + 1/2\ \Delta\mathbf{w}^{T}\,\mathbf{H}(\mathbf{w}_0)\Delta\mathbf{w} + O^{3+}(\Delta\mathbf{w})$

$\Delta\mathbf{w} = \mathbf{w} - \mathbf{w}_0$

Taylor expansion:

$$1D: f(x) = \sum_{i=0}^{\infty} \frac{f^{(i)}(x_0)}{i!}(x-x_0)^i$$

Gradient vector:

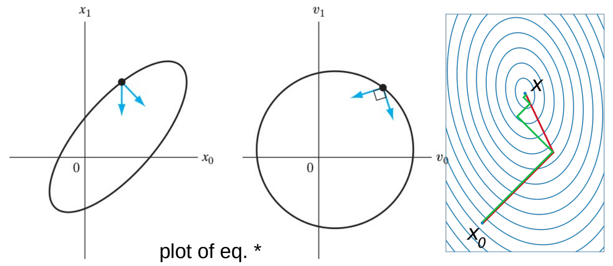
$$\mathbf{g}(\mathbf{w}_0) = \nabla E(\mathbf{w}_0) = \left[\frac{\partial E}{\partial w_1}/\mathbf{w}_0, \ldots, \frac{\partial E}{\partial w_{|W|}}/\mathbf{w}_0\right]^T$$

- Error back-propagation is a linear approximation of E: $\Delta\mathbf{w}(t) = -\alpha\,\mathbf{g}(t)$
- Quadratic approx. of $E(\mathbf{w})$ → Newton's method: $\Delta\mathbf{w} = -\mathbf{H}^{-1}(t)\,\mathbf{g}(t)$
- Quasi-Newton method approximates $\mathbf{H}^{-1}(t)$ with a positive definite matrix
- Conjugate-gradients methods are intermediate between the steepest descent and the Newton's method, by achieving faster convergence (than the former) and lower computational complexity (than the latter).

Conjugate-gradient methods

- 2nd order optimization methods
- minimize the quadratic function $f(\mathbf{x}) = ½\,\mathbf{x}^{T}\mathbf{A}\mathbf{x} + \mathbf{b}^{T}\mathbf{x} + c$ $\quad (*)$
- -> set of linear eqations: $\mathbf{A}\mathbf{x} = \mathbf{b}$ ($\mathbf{A}$ = positive definite and symmetric)
- Solution: $\mathbf{x}^* = \mathbf{A}^{-1}\mathbf{b}$
- Given the matrix $\mathbf{A}$, a set of nonzero vectors $\mathbf{s}(0), \mathbf{s}(1), \ldots$, (up to dim($\mathbf{A}$)) is $\mathbf{A}$-conjugate (i.e., non-interfering with each other in the context of $\mathbf{A}$) if: $\mathbf{s}(i)^{T}\mathbf{A}\mathbf{s}(j) = 0$. (for $\mathbf{A} = \mathbf{Id}$, conjugacy = orthogonality).

- Example: $\mathbf{x} = [x_0, x_1]$
- Let $\mathbf{v} = \mathbf{A}^{1/2}\mathbf{x}$
- Iterative CG method: $\Delta\mathbf{x} = \eta(t)\,.\,\mathbf{s}(t)$



plot of eq. *

(Hestenes & Stiefel, 1952)

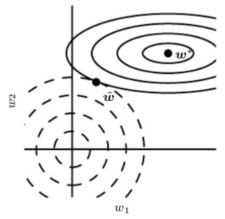
Regularization

Risk function: $R(\boldsymbol{w}) = E(\boldsymbol{w}) + \lambda\, C(\boldsymbol{w})$ [performance + complexity]

- Explicit:

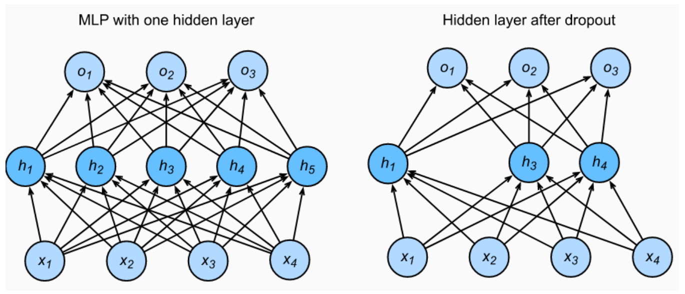
  $L_1(\boldsymbol{w}) = \epsilon \sum_i |w_i|$ $\quad L_2(\boldsymbol{w}) = \frac{\epsilon}{2}\|\boldsymbol{w}\|^2$

- Implicit:
  - weight decay $w_l^{\text{new}}(t) = \epsilon . w_l^{\text{new}}(t),\ 0 \ll \epsilon < 1,$
  - ...leads to L_2-regul.
  - dropout (Hinton, 2012): random turning off neurons during training
- data augmentation – increasing the size of the training set, e.g. by elastic distortions



Dropout

- Applied only during training
- Helps to avoid overfitting
- Free parameter = number of (randomly) dropped units



MLP with one hidden layer
Hidden layer after dropout
o_1 o_2 o_3
h_1 h_2 h_3 h_4 h_5
x_1 x_2 x_3 x_4

(Zhang et al, 2019)

AdaGrad algorithm

- Introduces the variable that accumulates gradient variance (vector)

  $\boldsymbol{g}(t) = \nabla E(\boldsymbol{w}(t)) \qquad \boldsymbol{s}(t) = \boldsymbol{s}(t-1) + \boldsymbol{g}^2(t) \qquad \Delta\boldsymbol{w}(t) = -\frac{\alpha}{\sqrt{\boldsymbol{s}(t)+\epsilon}} . \boldsymbol{g}(t)$
- decreases the learning rate dynamically on per-coordinate basis
- uses the magnitude of the gradient as a means of adjusting how quickly progress is achieved – coordinates with large gradients are compensated with a smaller learning rate.
- First-order method (the gradient can be a useful proxy)
- On deep learning problems Adagrad can sometimes be too aggressive in reducing learning rates. Mitigating strategies exist.

(Duchi et al, 2011)

RMSprop

- decouples rate scheduling from coordinate-adaptive learning rates

  $\boldsymbol{s}(t) = \gamma\, \boldsymbol{s}(t-1) + (1-\gamma)\, \boldsymbol{g}^2(t) \qquad \Delta\boldsymbol{w}(t) = -\frac{\alpha}{\sqrt{\boldsymbol{s}(t)+\epsilon}} . \boldsymbol{g}(t) \qquad \epsilon = 10^{-6}$
- coefficient γ determines how long the history is when adjusting the per-coordinate scale.
- RMSprop shares with momentum the leaky averaging. However, RMSProp uses the technique to adjust the coefficient-wise preconditioner (for reducing the condition number).

(Tieleman & Hinton, 2012)

AdaDelta

- Yet another variant of AdaGrad: it decreases the amount by which the learning rate is adaptive to coordinates
- It does not literally have a learning rate since it uses the amount of change itself as calibration for future change:

$$\boldsymbol{s}(t)=\rho\,\boldsymbol{s}(t-1)+(1-\rho)\boldsymbol{g}^2(t) \qquad \boldsymbol{g}'(t)=\sqrt{\frac{\Delta\boldsymbol{w}(t-1)+\epsilon}{\boldsymbol{s}(t)+\epsilon}}\,.\,\boldsymbol{g}(t)$$

$$\boldsymbol{w}(t)=\boldsymbol{w}(t-1)-\boldsymbol{g}'(t)$$

$$\Delta\boldsymbol{w}(t)=\rho\,\Delta\boldsymbol{w}(t-1)+(1-\rho)\boldsymbol{w}^2(t)$$

(Zeiler, 2012)

Adam algorithm

- Combines 3 preceeding techniques into one efficient algorithm
- uses leaky averaging to obtain an estimate of both the momentum and also the second moment of the gradient

$$\boldsymbol{v}(t) = \beta_1\, \boldsymbol{v}(t\!-\!1) + (1-\beta_1)\, \boldsymbol{g}(t) \qquad \boldsymbol{v}'(t) = \boldsymbol{v}(t) / (1-\beta_1^{\,t}) \qquad \beta_1 = 0.9$$

$$\boldsymbol{s}(t) = \beta_2\, \boldsymbol{s}(t\!-\!1) + (1-\beta_2)\, \boldsymbol{g}^2(t) \qquad \boldsymbol{s}'(t) = \boldsymbol{s}(t) / (1-\beta_2^{\,t}) \qquad \beta_2 = 0.999$$

$$\Delta\boldsymbol{w}(t)=-\frac{\alpha}{\sqrt{\boldsymbol{s}'(t)+\epsilon}}\,.\,\boldsymbol{v}'(t)$$

(Kingma & Ba, 2014)

- Still, gradients with significant variance may hinder convergence ($\boldsymbol{s}(t)$ can blow up)
- Yogi algorithm addresses this: $\boldsymbol{s}(t) = \boldsymbol{s}(t\!-\!1) + (1-\beta_2)\,(\boldsymbol{g}^2(t) - \boldsymbol{s}(t\!-\!1))$

(Zaheer et al, 2018)

$$\boldsymbol{s}(t) = \boldsymbol{s}(t\!-\!1) + (1-\beta_2)\, \boldsymbol{g}^2(t)\ .\ \mathrm{sgn}\,(\boldsymbol{g}^2(t) - \boldsymbol{s}(t\!-\!1))$$

Natural gradient learning

- use Fisher information: a positive semidefinite matrix ($|\boldsymbol{w}|\times|\boldsymbol{w}|$), defines a Riemannian metric (-> information geometry) (Amari, 1998)
- look at p.d.f. via $KL(f(\boldsymbol{x};\boldsymbol{w})\,\|\,f(\boldsymbol{x};\boldsymbol{w}+\Delta\boldsymbol{w})) = \ldots \approx \tfrac{1}{2}\,(\Delta\boldsymbol{w})^{\mathrm{T}}\,\mathbf{F}\,\Delta\boldsymbol{w}$
- matrix $\mathbf{F}$ is the negative expected Hessian of $\log f(\boldsymbol{x};\boldsymbol{w})$
- $\Delta\boldsymbol{w}^* = \arg\min_{\Delta\boldsymbol{w}} \{L(\boldsymbol{w}+\Delta\boldsymbol{w}) + \lambda.\mathrm{KL}(f(\boldsymbol{x};\boldsymbol{w})\,\|\,f(\boldsymbol{x};\boldsymbol{w}+\Delta\boldsymbol{w})) - c)\}$
- $\Delta\boldsymbol{w}(t) = -\,\alpha\,\mathbf{F}^{-1}(\boldsymbol{w}(t))\,\boldsymbol{g}(t)$, i.e. natural gradient $\boldsymbol{g}_{nat}(t) = \mathbf{F}^{-1}(\boldsymbol{w})\,\boldsymbol{g}(t)$
- can be interpreted as curvature of the log likelihood function f
- in NG descent, we control movement in prediction space (rather than parameter space)
- Approximations of $\mathbf{F}^{-1}$ possible (Amari et al, 2019)

Summary

- NN learning and classical optimization have different objectives
- NN goal = minimize generalization error (sometimes using surrogate loss functions)
- Various known problems hinder first-order gradient methods
- Second-order methods provide more informaton but are much more costly
- Earlier methods focused on approximating the Hessian
- Recent methods foces only on gradients and its adaptive versions
- Natural gradient learning uses Riemannian metric
- Further improvements possible (found useful in deep learning), to be mentioned later