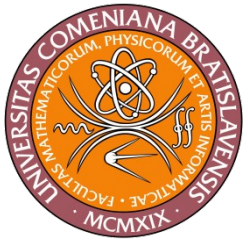
Faculty of Mathematics, Physics and Informatics
Comenius University Bratislava



Neural Networks

Lecture 5

Gradient-based learning and optimization

Igor Farkaš

2024

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 error on given dataset
- 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.
- Typically we apply early stopping criterion using the validation set.

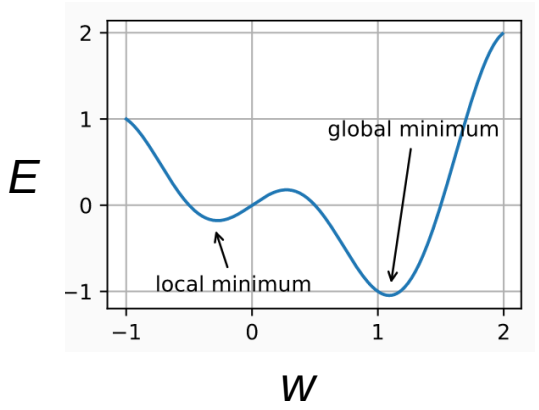
(Goodfellow et al, 2015)

Surrogate loss function

- Sometimes, the loss function we actually care about (e.g. classification error) is not the 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 $(-\log P(y_i))$
- Decrease of error, after the training set 0-1 loss has reached zero, 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).

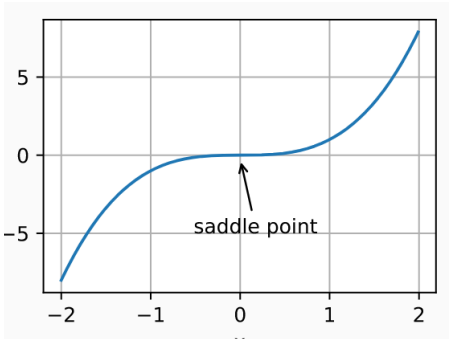
Problems in gradient-based NN learning

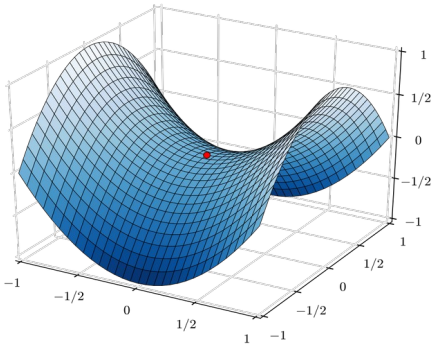
Local minima



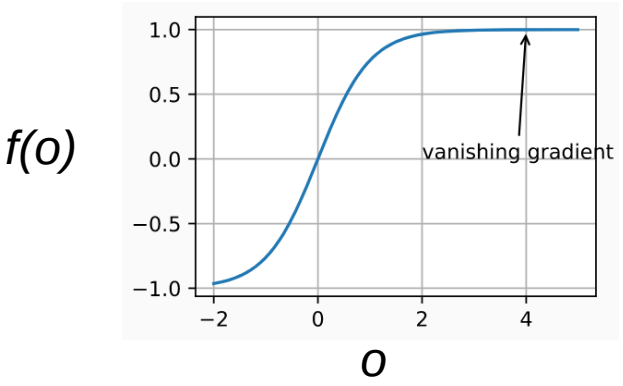
w = weight,
E = loss f.

Saddle points





Vanishing gradients



Ill-conditioning of Hessian matrix $\boldsymbol{H}$,

i.e. rate of its change for small $\Delta \boldsymbol{w}$:

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

These problems slow down or hinder convergence.

Role of the Hessian matrix

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

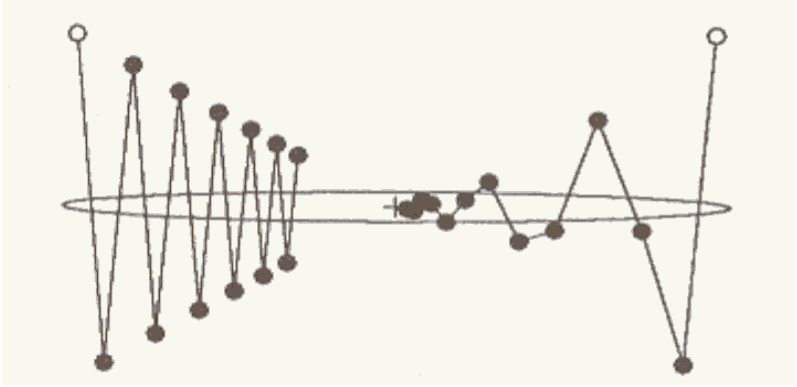
$$\boldsymbol{H} = \left[\frac{\partial^2 E(\boldsymbol{w})}{\partial w_i \partial w_j} / \boldsymbol{w}_0 \right]_{ij}$$

- Spread of eigenvalues of $\boldsymbol{H}$ has a profound influence on the dynamics of back-propagation learning (condition number).
- The inverse of $\boldsymbol{H}$ provides a basis for pruning (i.e., deleting) insignificant synaptic weights from a multilayer perceptron.
- $\boldsymbol{H}$ is basic to the formulation of second-order optimization methods as an alternative to BP learning.
- Typical profile of $\boldsymbol{H}$ 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.

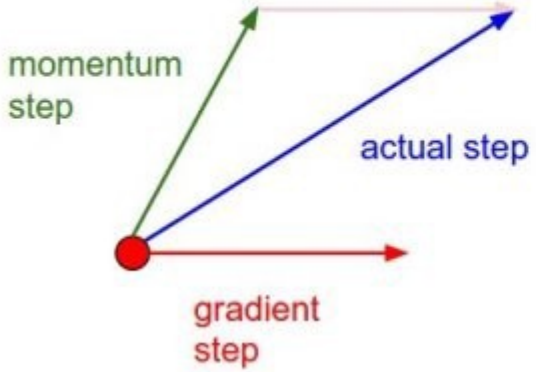
Early modifications of gradient descent learning

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

- helps speed up SGD and dampen oscillations



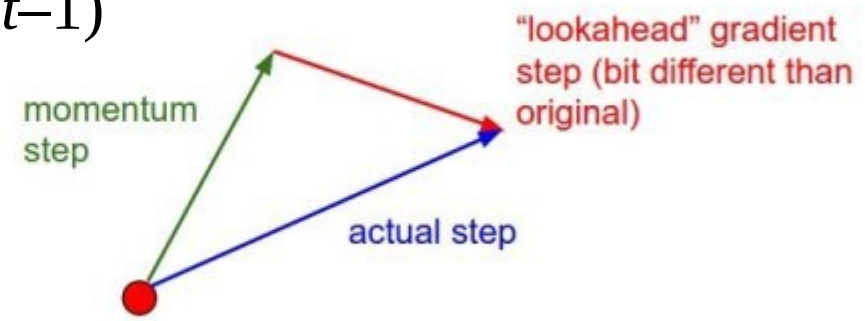
w/out with



Nesterov accelerated gradient (Nesterov, 1983)

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

- helps adjust learning speed by looking into the near future



Towards second-order optimization methods

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

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

Gradient vector:

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

Taylor expansion:

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

- Error back-propagation is a linear approximation of E: $\Delta\boldsymbol{w}(t) = -\alpha\,\boldsymbol{g}(t)$
- Quadratic approx. of $E(\boldsymbol{w})$ → Newton's method: $\Delta\boldsymbol{w} = -\boldsymbol{H}^{-1}(t)\,\boldsymbol{g}(t)$
- Quasi-Newton method approximates $\boldsymbol{H}^{-1}(t)$ 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

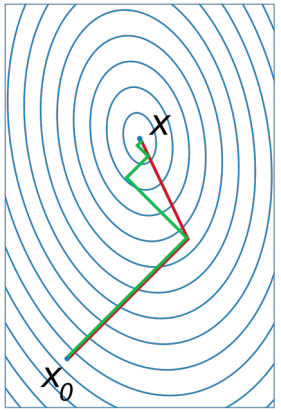
- 2nd order optimization methods
- minimize the quadratic function $f(\boldsymbol{x}) = \frac{1}{2}\,\boldsymbol{x}^{\mathrm{T}}\mathbf{A}\boldsymbol{x} + \boldsymbol{b}^{T}\boldsymbol{x} + c \quad (*)$
- -> set of linear eqations: $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ ($\boldsymbol{A}$ = positive definite and symmetric)
- Solution: $\boldsymbol{x}^{*} = \boldsymbol{A}^{-1}\boldsymbol{b} \quad (\boldsymbol{A} \equiv \boldsymbol{H})$
- Given the matrix $\boldsymbol{A}$, a set of nonzero vectors $\boldsymbol{s}(0), \boldsymbol{s}(1), \ldots,$ (up to dim($\boldsymbol{A}$)) is $\boldsymbol{A}$-conjugate (i.e., non-interfering with each other in the context of $\boldsymbol{A}$) if: $\boldsymbol{s}(i)^{\mathrm{T}}\boldsymbol{A}\boldsymbol{s}(j) = 0$. (for $\boldsymbol{A} = \boldsymbol{I}$, conjugacy = orthogonality).
- Find conjugate directions (for minibatch) without computing $\boldsymbol{H}^{-1}(t)$, e.g. via Polak-Ribière method:

$$\beta(t) = \frac{(\boldsymbol{g}(t) - \boldsymbol{g}(t-1))^{T}\,\boldsymbol{g}(t)}{\boldsymbol{g}(t-1)^{T}.\,\boldsymbol{g}(t-1)}$$

$$\rho(t) = -\boldsymbol{g}(t) + \beta(t).\rho(t-1)$$

$$\boldsymbol{w}(t+1) = \boldsymbol{w}(t) + \alpha^{*}(t).\rho(t)$$

Computed analytically



(Haykin, 2009)

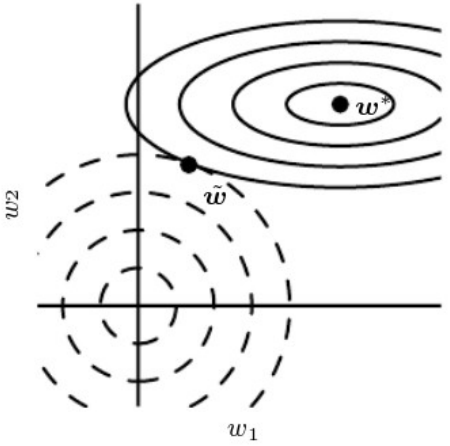
Regularization

Goal – minimize overfitting

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

- Explicit:

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

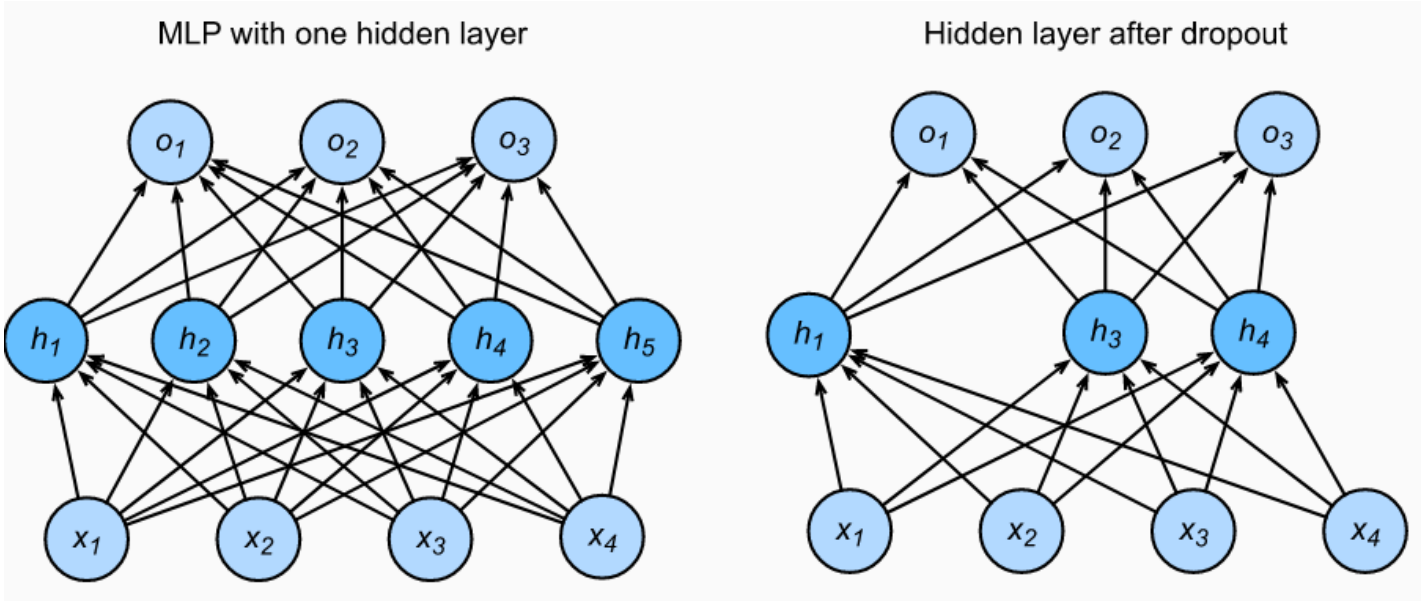


(Goodfellow, 2015)

- 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



(Zhang et al, 2019)

AdaGrad algorithm

- Back to first-order methods
- 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:

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

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

$$\Delta \boldsymbol{w}(t)=\rho\,\Delta \boldsymbol{w}(t-1)+(1-\rho)\boldsymbol{g}'^2(t)$$

(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)

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

Natural gradient learning

- uses 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 $KL(p(\boldsymbol{x};\boldsymbol{w}) \,\|\, p(\boldsymbol{x};\boldsymbol{w}+\Delta\boldsymbol{w})) = \ldots \approx \tfrac{1}{2}\,(\Delta\boldsymbol{w})^{\mathrm{T}}\,\boldsymbol{F}\,\Delta\boldsymbol{w}$
- matrix $\boldsymbol{F}$ is the negative expected Hessian of $\log p(\boldsymbol{x};\boldsymbol{w})$
- $\Delta\boldsymbol{w}^* = \arg\min_{\Delta\boldsymbol{w}} \{L(\boldsymbol{w}+\Delta\boldsymbol{w}) + \lambda.\mathrm{KL}(p(\boldsymbol{x};\boldsymbol{w}) \,\|\, p(\boldsymbol{x};\boldsymbol{w}+\Delta\boldsymbol{w})) - c)\}$
- $\Delta\boldsymbol{w}(t) = -\,\alpha\,\boldsymbol{F}^{-1}(\boldsymbol{w}(t)).\boldsymbol{g}(t)$, i.e. natural gradient $\boldsymbol{g}_{nat}(t) = \boldsymbol{F}^{-1}(\boldsymbol{w})\,\boldsymbol{g}(t)$
- can be interpreted as curvature of the log-likelihood function p
- in NG descent, we control movement in prediction space (rather than parameter space)
- efficient approximations of $\boldsymbol{F}^{-1}$ 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 (2nd oder) uses Riemannian metric
- Further improvements possible (found useful in deep learning), to be mentioned later