

Rețele neuronale și deep learning

Cursul 2: Antrenarea rețelelor: propagarea înapoi a erorii și optimizarea

Prof. dr. Daniel Traian PELE

Academia de Studii Economice din București (ASE), Departamentul de Statistică și Econometrie
Academia Română, Institutul de Prognoză Economică (IPE)
IDA Institute Digital Assets ▪ AI4EFin ▪ MSCA Digital Finance

Obiectivele cursului

- Să scrieți coborîrea pe gradient (GD) ca metodă de rezolvare a minimizării riscului empiric și să explicați de ce condiționarea îi stabilește viteza.
- Să derivați propagarea înapoi a erorii pentru un perceptron multistrat (MLP), să o calculați manual pe date reale și să o verificați cu diferențierea automată.
- Să explicați dispariția și explozia gradienților și să alegeți o inițializare care păstrează semnalul.
- Să comparați coborîrea stohastică pe gradient (SGD), metoda momentului, propagarea cu medie pătratică (RMSProp), Adam și programele ratei de învățare pe date reale, cu mai multe inițializări.
- Să antrenați o rețea cuantilică pentru valoarea la risc (VaR) pe o zi a indicelui S&P 500 și să o testați retroactiv.

Abrevieri

AdamW Adam cu penalizarea ponderilor decuplată
CPU procesor central
ES pierderea așteptată în coada distribuției (expected shortfall)
EWMA medie mobilă ponderată exponențial
GD coborîrea pe gradient
GPU procesor grafic
LLM model lingvistic mare
MLP perceptron multistrat
MPS Metal Performance Shaders (interfața PyTorch pentru procesorul grafic Apple)
MSE eroarea pătratică medie
OpenML platforma Open Machine Learning
PCA analiza componentelor principale
PR-AUC aria de sub curba precizie-sensibilitate
ReLU unitate liniară rectificată
RMSProp propagarea cu medie pătratică (root mean square propagation)
SGD coborîrea stohastică pe gradient
ULB Université libre de Bruxelles
VaR valoarea la risc

Cuprins

- De la pierdere la gradient
- Propagarea înapoi a erorii
- Optimizarea stohastică
- Studiu de caz: valoarea la risc cu o rețea cuantilică
- Sinteză

Antrenarea unei rețele este o minimizare nonconvexă, în dimensiune mare

- **Pe scurt:** nu putem afla ponderile printr-o formulă, ca la cele mai mici pătrate; coborîm pe suprafața pierderii, pas cu pas.

$$\hat{\boldsymbol{\theta}} = \operatorname*{arg\,min}_{\boldsymbol{\theta}\in\mathbb{R}^P} \mathcal{L}(\boldsymbol{\theta}), \qquad \mathcal{L}(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} \ell\big(y_i, f(\mathbf{x}_i;\boldsymbol{\theta})\big), \qquad \boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta\,\nabla\mathcal{L}(\boldsymbol{\theta}_t)$$

- **unde**
  - P numărul de parametri, $\eta > 0$ rata de învățare, $\nabla\mathcal{L}$ gradientul, calculat prin propagare înapoi (secțiunea 2)
- De ce funcționează: $\mathcal{L}(\boldsymbol{\theta} - \eta\mathbf{g}) = \mathcal{L}(\boldsymbol{\theta}) - \eta\|\mathbf{g}\|^2 + O(\eta^2)$, deci un pas mic contra gradientului scade pierderea.
- MLP 64-64 din acest curs are 4 801 de parametri pe 12 384 de grupuri de antrenare
  - Modelele din producție au milioane.
- Două întrebări: cum obținem ieftin $\nabla\mathcal{L}$ și cum facem pași buni.

- **În practică:** fiecare model din acest curs este antrenat cu o variantă a acestei actualizări de un rînd.

(Bishop & Bishop, 2024, ch. 7)

Datele acestui curs: locuințe, plăți cu cardul și indicele S&P 500

Setul de date	Rînduri	Sarcina	Împărțirea
California housing	20 640	regresie: valoarea locuinței	aleatoare 60/20/20: 12 384 / 4 128 / 4 128
Fraude cu carduri (eșantion)	28 924	clasificare: fraudă	cronologic: primele 60 % antrenare, următoarele 20 % validare, ultimele 20 % test
S&P 500	1 716	cuantila de 1 % a randamentului de a doua zi	cronologic: antrenare pînă în 2016, validare 2017–2019, test din 2020

- Frauda: datele Université libre de Bruxelles (ULB) și Worldline, pe platforma Open Machine Learning (OpenML); toate cele 492 de fraude și un eșantion de 10 % din plățile legitime; intrările V1–V28 sînt scoruri de analiză a componentelor principale (PCA) publicate de bancă.
- Ponderile refac rata fraudei din datele complete, 0,173 %, în orice metrică de tip precizie; S&P 500: randamente logaritmice zilnice în %, rîndurile de test = zilele prognozate.

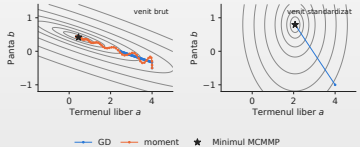
- **În practică:** seriile de timp și frauda sînt împărțite cronologic: modelul nu vede niciodată viitorul.

(Pace & Barry, 1997; Dal Pozzolo et al., 2015)

Exercițiu practic: notebook §1

Intrările nescalate transformă pierderea într-o vale îngustă

- **Pe scurt:** pierderea reală a lui $\hat{y} = a + b \cdot \text{venit}$ pe California: coborîrea pe gradient face zigzaguri pe venitul brut și merge direct pe venitul standardizat.



Sursa: StatLib, locuințe din California, recensământul din 1990 (Pace și Barry, 1997). Date: 20 640 de grupuri de locuințe recensământul din 1990.

- Numărul de condiționare $\kappa = \lambda_{\max}/\lambda_{\min}$ al hessianei: 104,3 brut, 1 standardizat
  - După 60 de pași: eroarea pătratică medie (MSE) 1,486 (GD, brut), 0,704 (moment, brut), 0,701 (GD, standardizat)
  - Minimul 0,701.

- **În practică:** standardizați fiecare intrare pe setul de antrenare înainte de a antrena orice rețea.

(calcul propriu)

NN_L2_gd_paths

Pe o funcție pătratică, pasul trebuie să rămînă sub $2/\lambda_{\max}$

- **Pe scurt:** pe fiecare direcție proprie a hessianei, coborîrea pe gradient înmulțește eroarea cu $(1-\eta\lambda_k)$.

$$\mathcal{L}(\theta) = \mathcal{L}^* + \tfrac{1}{2}(\theta-\theta^*)^\top \mathbf{H}(\theta-\theta^*) \;\Rightarrow\; \theta_t - \theta^* = (\mathbf{I}-\eta\mathbf{H})^t(\theta_0-\theta^*), \qquad |1-\eta\lambda_k|<1 \iff 0<\eta<\frac{2}{\lambda_{\max}}$$

- **unde**
  - $\mathbf{H}$ hessiana, cu valorile proprii $\lambda_{\min} \le \cdots \le \lambda_{\max}$, θ^* minimul
- Cu cel mai bun pas fix, eroarea scade cu factorul $(\kappa-1)/(\kappa+1)$ la fiecare pas: lent cînd κ este mare.
- Venitul brut: $\kappa = 104{,}3$, deci pasul $1/\lambda_{\max} = 0{,}0258$ abia mișcă direcția plată.
- Rețelele nu sînt pătratice, dar lîngă un minim se comportă ca una: aceeași limită explică divergența la η mare.

- **În practică:** dacă pierderea explodează în primele epoci, împărțiți rata de învățare la 3 înainte de a schimba altceva.

(Bishop & Bishop, 2024, ch. 7)

Cuprins

- De la pierdere la gradient
- Propagarea înapoi a erorii
- Optimizarea stohastică
- Studiu de caz: valoarea la risc cu o rețea cuantilică
- Sinteză

Propagarea înapoi este regula de derivare a funcțiilor compuse, aplicată de la pierdere înapoi

- **Pe scurt:** fiecare nod al grafului de calcul înmulțește gradientul primit cu propria derivată locală și îl transmite mai departe.

$$\mathbf{z}^{(l)} = \mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}, \quad \mathbf{h}^{(l)} = a(\mathbf{z}^{(l)}); \qquad \boldsymbol{\delta}^{(L)} = \frac{\partial \ell}{\partial \mathbf{z}^{(L)}}, \quad \boldsymbol{\delta}^{(l)} = \left(\mathbf{W}^{(l+1)\top}\boldsymbol{\delta}^{(l+1)}\right) \odot a'(\mathbf{z}^{(l)})$$

$$\frac{\partial \ell}{\partial \mathbf{W}^{(l)}} = \boldsymbol{\delta}^{(l)}\,\mathbf{h}^{(l-1)\top}, \qquad \frac{\partial \ell}{\partial \mathbf{b}^{(l)}} = \boldsymbol{\delta}^{(l)}$$

- **unde**
  - $\boldsymbol{\delta}^{(l)}$ semnalul de eroare al stratului l, $\odot$ produsul element cu element, a' derivata funcției de activare
- Eroarea de ieșire: $\hat{y} - y$ (pierdere pătratică), $p - y$ (sigmoidă cu entropie încrucișată, cursul 1).
- Gradientul unei ponderi = eroarea stratului înmulțită cu intrarea lui, ca la gradientul logistic.
- **În practică:** propagarea înainte trebuie să rețină $\mathbf{z}^{(l)}$ și $\mathbf{h}^{(l)}$: memoria, nu calculul, limitează mărimea lotului pe un procesor grafic (GPU).

(Rumelhart, Hinton & Williams, 1986)

Propagarea înapoi, de mînă, pentru un grup real de locuințe

- **Pe scurt:** două intrări standardizate, două unități liniare rectificate (ReLU), ieșire liniară, $\ell = \frac{1}{2}(\hat{y} - y)^2$.

Înainte	neuronul 1	neuronul 2
intrarea x (venit, latitudine)	$-0{,}21$	$-0{,}67$
ponderile $\mathbf{W}^{(1)}$, rîndul 1 / 2	$-0{,}28$, $-0{,}56$	$0{,}50$, $-0{,}02$
$z = \mathbf{W}^{(1)}x$	$0{,}433$	$-0{,}093$
$h = \max(0, z)$	$0{,}433$	$0{,}000$
ponderile de ieșire $w^{(2)}$	$-0{,}87$	$-0{,}19$
$\hat{y} = w^{(2)\top} h$; y	$-0{,}377$	$2{,}067$

Înapoi	neuronul 1	neuronul 2
$\delta^{(2)} = \hat{y} - y$	$-2{,}444$	
$\partial \ell / \partial w^{(2)} = \delta^{(2)} h$	$-1{,}059$	$-0{,}000$
$\delta^{(1)} = \delta^{(2)} w^{(2)} \odot \mathbb{1}\{z > 0\}$	$2{,}126$	$0{,}000$
$\partial \ell / \partial W^{(1)}_{k1} = \delta^{(1)}_k x_1$	$-0{,}453$	$-0{,}000$
$\partial \ell / \partial W^{(1)}_{k2} = \delta^{(1)}_k x_2$	$-1{,}419$	$-0{,}000$

- Neuronul 2 este oprit ($z_2 < 0$): nu transmite gradient, deci ponderile lui nu se modifică.
- Diferențierea automată din PyTorch dă aceiași gradienți (diferența maximă 0,0).

- **În practică:** un neuron ReLU oprit pentru toți clienții nu mai învață niciodată: urmăriți ponderea neuronilor „morți”.

(calcul propriu)

Exercițiu practic: notebook §2

Modul invers costă cam încă o propagare înainte, oricîte ponderi ar fi

- **Pe scurt:** o singură trecere înapoi dă toate cele P derivate parțiale ale unei pierderi scalare; diferențele finite ar cere P treceri în plus.

$$\text{cost}\big(\nabla_{\boldsymbol{\theta}}\mathcal{L}\big) \;\leq\; c \cdot \text{cost}\big(\mathcal{L}\big), \quad c \approx 2\text{–}3; \qquad \frac{\partial \mathcal{L}}{\partial \theta_j} \approx \frac{\mathcal{L}(\boldsymbol{\theta} + \epsilon \mathbf{e}_j) - \mathcal{L}(\boldsymbol{\theta} - \epsilon \mathbf{e}_j)}{2\epsilon}$$

- Modul direct (numere duale) este ieftin pentru puține intrări și multe ieșiri
  - Modul invers pentru multe intrări (ponderi) și o ieșire (pierderea).
- Verificarea gradientului pe MLP 64-64: 30 de ponderi alese aleator, diferențe centrale cu $\epsilon = 10^{-6}$ în dublă precizie, eroarea relativă maximă $7{,}0 \times 10^{-7}$.
- PyTorch înregistrează graful în timpul propagării înainte
  - `loss.backward()` îl parcurge invers.

- **În practică:** cînd scrieți un strat sau o pierdere proprie, rulați o dată verificarea gradientului înainte de a vă baza pe ele.

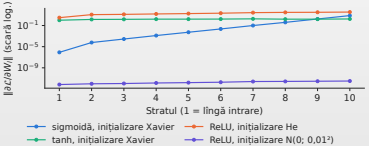
(Baydin et al., 2018; Paszke et al., 2019)

Exercițiu practic: notebook §2

Dispariția gradientului în rețelele adînci cu activare sigmoidă

- **Pe scurt:** semnalul înapoi este un produs de matrice jacobiene; cu $\sigma' \leq 1/4$, el scade la fiecare strat.

$$\frac{\partial \ell}{\partial \mathbf{h}^{(l)}} = \left(\prod_{k=l+1}^{L} \mathbf{W}^{(k)\top} \operatorname{diag}\left(a'(\mathbf{z}^{(k)})\right) \right) \frac{\partial \ell}{\partial \mathbf{h}^{(L)}}$$



Sursa: StatLib, locuințe din California, recensămîntul din 1990 (Pace și Barry, 1997). Date: 10 straturi ascunse de 64 de neuroni, un lot de 512 grupuri de antrenare, 5 inițializări.

- 10 straturi ascunse, un lot real: norma gradientului în primul/ultimul strat $8{,}5 \times 10^{-7}$ față de $7{,}5 \times 10^{0}$ cu sigmoide
  - $3{,}1 \times 10^{0}$ față de $3{,}5 \times 10^{1}$ cu ReLU și inițializare He.

- **În practică:** rețelele adînci au nevoie de activări ReLU, de o inițializare scalată și de normalizare (cursul 3).

(calcul propriu)

NN_L2_vanishing

Inițializarea păstrează constantă varianța semnalului între straturi

- **Pe scurt:** prea mică și semnalul dispare, prea mare și explodează; scara depinde de numărul de intrări și de activare.

$$\mathrm{Var}(z_j^{(l)}) = n_{l-1}\,\mathrm{Var}(W)\,\mathbb{E}\left[(h^{(l-1)})^2\right] \;\Rightarrow\; \underbrace{\mathrm{Var}(W) = \frac{2}{n_{l-1}+n_l}}_{\text{Xavier (tanh, sigmoid)}}, \qquad \underbrace{\mathrm{Var}(W) = \frac{2}{n_{l-1}}}_{\text{He (ReLU)}}$$

- **unde**
  - n_{l-1} numărul de intrări, n_l numărul de ieșiri ale stratului l, pentru ReLU, $\mathbb{E}[h^2] = \frac{1}{2}\,\mathrm{Var}(z)$, de unde factorul 2
- Ponderi $\mathcal{N}(0; 0{,}01^2)$, rețea ReLU cu 10 straturi: norma gradientului în primul strat $6{,}9 \times 10^{-13}$, deci nicio învățare.
- Nu inițializați toate ponderile la fel: toți neuronii ar învăța același lucru.

- **În practică:** valorile implicite din PyTorch sînt apropiate de aceste reguli
  - Pentru rețele ReLU adînci, setați explicit inițializarea He.

(Glorot & Bengio, 2010; He et al., 2015)

Exercițiu practic: notebook §3

Cuprins

- De la pierdere la gradient
- Propagarea înapoi a erorii
- Optimizarea stohastică
- Studiu de caz: valoarea la risc cu o rețea cuantilică
- Sinteză

Un mini-lot dă un gradient nedeplasat și zgomotos, la o fracțiune din cost

- **Pe scurt:** media gradientului pe B exemple aleatoare este suficientă pentru a ști aproximativ încotro este coborîrea.

$$\mathbf{g}_t = \frac{1}{B}\sum_{i\in\mathcal{B}_t}\nabla\ell_i(\boldsymbol{\theta}_t), \qquad \mathbb{E}[\mathbf{g}_t] = \nabla\mathcal{L}(\boldsymbol{\theta}_t), \qquad \mathrm{Var}(\mathbf{g}_t) \approx \frac{1}{B}\left(1-\frac{B}{n}\right)\Sigma(\boldsymbol{\theta}_t)$$

- **unde**
  - $\mathcal{B}_t$ un lot aleator de mărime B, Σ covarianța gradienților pe exemple, o epocă $= \lceil n/B \rceil$ pași
- Robbins–Monro: cu $\sum_t \eta_t = \infty$ și $\sum_t \eta_t^2 < \infty$, SGD converge către un punct staționar.
- Zgomotul nu este doar un cost: ajută la ieșirea din minimele abrupte, care tind să generalizeze mai slab (Keskar et al., 2017).

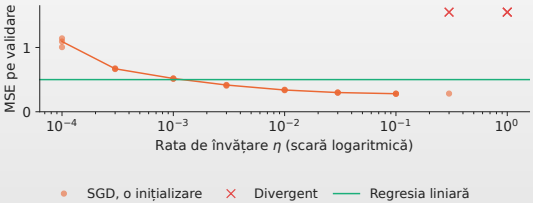
- **În practică:** loturile de 32–256 de exemple sînt compromisul obișnuit între zgomot, viteză și memorie.

(Robbins & Monro, 1951)

Exercițiu practic: notebook §4

Efectul ratei de învățare asupra antrenării

- **Pe scurt:** SGD simplu pe un MLP 64-64, 30 de epoci, loturi de 128, pentru nouă rate de învățare și trei inițializări.



Sursa: StatLib, locuințe din California, recensământul din 1990 (Pace și Barry, 1997). Date: 12 384 de grupuri de antrenare și 4 128 de validare; 30 de epoci.

- MSE pe validare 1,093 la $\eta = 10^{-4}$ (mai slab decît regresia liniară, 0,500), 0,281 la $\eta = 0{,}1$
  - De la $\eta = 0{,}3$, cel puțin o inițializare diverge.

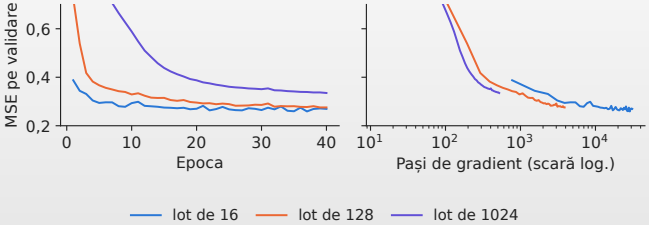
- **În practică:** ajustați mai întîi rata de învățare, pe o grilă logaritmică, înaintea oricărui alt hiperparametru.

(calcul propriu)

NN_L2_lr_sweep

Efectul dimensiunii lotului: progresul pe epocă și pe secundă

- **Pe scurt:** Adam cu rata de învățare 0,001, 40 de epoci, loturi de 16, 128 și 1 024 de exemple.



Sursa: StatLib, locuințe din California, recensământul din 1990 (Pace și Barry, 1997). Date: Adam, rata de învățare 0,001, 40 de epoci.

- După 5 epoci: MSE pe validare 0,294 (lot 16), 0,367 (128), 0,870 (1 024)
  - Durata a 40 de epoci pe un fir al procesorului: 25,8, 3,7 și 1,0 secunde.

- **În practică:** comparați bugetele de antrenare în pași de gradient sau în secunde, nu în epoci.

(calcul propriu)

NN_L2_batch_size

Metoda momentului: media mobilă a gradienților trecuți

- **Pe scurt:** componentele în zigzag se anulează în medie, iar componenta constantă spre vale se acumulează.

$$\mathbf{v}_{t+1} = \beta\,\mathbf{v}_t + \mathbf{g}_t, \qquad \boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta\,\mathbf{v}_{t+1}; \qquad \text{Nesterov: } \mathbf{g}_t = \nabla\mathcal{L}(\boldsymbol{\theta}_t - \eta\beta\mathbf{v}_t)$$

- **unde**
  - $\mathbf{v}$ viteza, $\beta \in [0,1)$ coeficientul de moment, de obicei 0,9, pe o pantă constantă, pasul efectiv este $\eta/(1-\beta)$, de zece ori η
- Pe o funcție pătratică, momentul optim îmbunătățește contracția de la $\frac{\kappa-1}{\kappa+1}$ la $\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}$ (Polyak, 1964).
- Regresia pe venitul brut după 60 de pași: MSE 0,704 cu moment, față de 1,486 pentru GD simplu.

- **În practică:** SGD cu moment 0,9 rămîne alegerea implicită pentru modelele foarte mari de vedere artificială.

(Polyak, 1964)

Exercițiu practic: notebook §5

Metodele adaptive dau fiecărei ponderi propriul pas

- **Pe scurt:** împărțiți pasul la o măsură curentă a mărimii gradienților acelei ponderi.

$$\text{AdaGrad: } s_t = s_{t-1} + g_t^2; \qquad \text{RMSProp: } s_t = \rho\, s_{t-1} + (1-\rho)\, g_t^2; \qquad \theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{s_t} + \epsilon}\, g_t$$

- **unde**
  - Toate operațiile element cu element, $\rho \approx 0{,}99$, $\epsilon \approx 10^{-8}$ evită împărțirea la zero
- Suma din AdaGrad doar crește, deci pașii lui scad spre zero: bun pentru variabile rare, prea lent pentru rețele adînci (Duchi, Hazan & Singer, 2011).
- Propagarea cu medie pătratică (RMSProp) uită exponențial gradienții vechi, deci pasul se adaptează regiunii curente (Tieleman & Hinton, 2012).
- Rescalarea pe fiecare pondere compensează parțial condiționarea proastă: direcțiile plate sau rar actualizate primesc pași mai mari.

- **În practică:** metodele adaptive sînt mai puțin sensibile la rata de învățare, ceea ce le face o primă alegere sigură.

Exercițiu practic: notebook §5

Adam: moment, RMSProp și corecția deplasării

- **Pe scurt:** păstrăm medii mobile ale gradientului și ale pătratului lui, le corectăm pentru pornirea de la zero și facem un pas egal cu raportul lor.

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t, \quad v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2, \quad \hat{m}_t = \frac{m_t}{1-\beta_1^t}, \; \hat{v}_t = \frac{v_t}{1-\beta_2^t}, \quad \theta_{t+1} = \theta_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

- **unde**
  - Valori implicite $\beta_1 = 0{,}9$, $\beta_2 = 0{,}999$, $\epsilon = 10^{-8}$, $\eta = 10^{-3}$, $m_0 = v_0 = 0$, de unde corecția (anexa)
- Pasul este aproximativ limitat la η pentru fiecare pondere, oricare ar fi scara pierderii: ușor de ajustat.
- Adam cu penalizarea ponderilor decuplată (AdamW) micșorează direct ponderile, $\theta \leftarrow \theta - \eta\lambda\theta$, în loc să adauge $\lambda\theta$ la g_t (Loshchilov & Hutter, 2019; cursul 3).

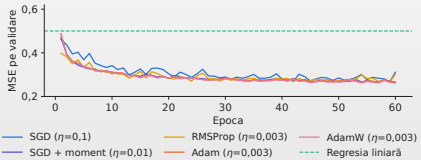
- **În practică:** porniți de la AdamW cu $\eta = 10^{-3}$
  - Treceți la SGD cu moment, ajustat, doar dacă cîștigați în afara eșantionului.

(Kingma & Ba, 2015)

Exercițiu practic: notebook §5

Pe datele California, optimizatorii ajustați ating erori apropiate, iar metodele adaptive converg mai sigur

- **Pe scurt:** cea mai bună dintre trei rate de învățare (pe validare), media a 5 inițializări noi.



Sursa: StatLib, locuințe din California, recensământul din 1990 (Pace și Barry, 1997). Date: 12 384 de grupuri de antrenare și 4 128 de validare.

- MSE pe validare după 10 epoci: SGD 0,342, moment 0,310, RMSProp 0,313, Adam 0,305, AdamW 0,304
  - Regresia liniară 0,500.

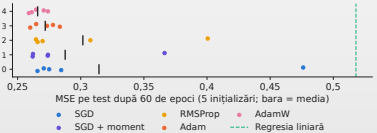
- **În practică:** rețeaua înjumătățește eroarea modelului liniar de referință, oricare ar fi optimizatorul.

(calcul propriu)

NN_L2_optimizers

De la o inițializare la alta, Adam și AdamW sînt mult mai stabile decît SGD simplu

- **Pe scurt:** MSE pe test pentru 5 inițializări la fiecare optimizator; o singură rulare le-ar fi ordonat la întîmplare.



Sursa: StatLib, locuințe din California, recensămîntul din 1990 (Pace și Barry, 1997). Date: 4 128 de grupuri de testare.

- MSE pe test, media (abaterea standard): SGD 0,314 (0,091), Adam 0,272 (0,010), AdamW 0,266 (0,006)
  - Testul t pe perechi SGD față de Adam: $t = 1{,}01$, $p = 0{,}371$.

- **În practică:** cîștigul mediu nu este semnificativ cu 5 inițializări, dar riscul unei rulări proaste este mult mai mic cu Adam.

(calcul propriu)

NN_L2_optimizer_seeds

Programele ratei de învățare: în trepte, cosinus și one-cycle

- **Pe scurt:** explorăm cu un pas mare, apoi îl reducem, ca zgomotul SGD să se așeze într-un minim.

$$\text{trepte: } \eta_t = \eta_0 \gamma^{\lfloor t/s \rfloor}; \qquad \text{cosinus: } \eta_t = \frac{\eta_0}{2}\left(1 + \cos\frac{\pi t}{T}\right)$$

$$\text{un ciclu: creștere liniară pînă la } \eta_0, \text{ apoi cosinus spre } \approx 0$$

- **unde**
  - η_0 rata maximă, γ factorul de scădere la fiecare s pași, T numărul total de pași
- Descreșterea cosinus (Loshchilov & Hutter, 2017) și programul cu un singur ciclu (Smith & Topin, 2019) au nevoie doar de rata maximă și de buget.
- Încălzirea protejează primii pași, cînd estimările momentului al doilea din Adam și statisticile loturilor nu sînt încă fiabile.

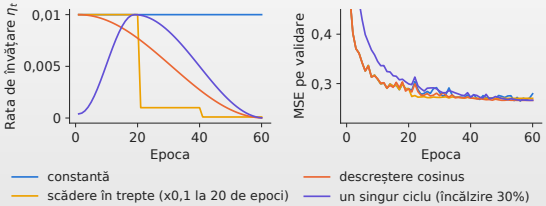
- **În practică:** un program cosinus pe numărul planificat de epoci este o alegere implicită sigură.

(Bishop & Bishop, 2024, ch. 7)

Exercițiu practic: notebook §6

Scăderea ratei de învățare reduce eroarea finală

- **Pe scurt:** SGD cu moment 0,9 și rata maximă 0,01; stînga: programele, dreapta: MSE pe validare (media a 3 inițializări).



Sursa: StatLib, locuințe din California, recensămîntul din 1990 (Pace și Barry, 1997). Date: SGD cu moment 0,9, rata maximă 0,01, 3 inițializări.

- MSE final pe validare: constantă 0,279, trepte 0,270, cosinus 0,266, un ciclu 0,266
  - Pe test 0,286 față de 0,265 (constantă față de cosinus).

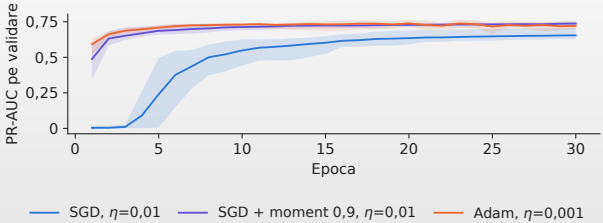
- **În practică:** programul ratei este un cîștig ieftin: același model, același buget, eroare mai mică.

(calcul propriu)

NN_L2_schedules

La fraudele cu carduri, optimizatorul determină viteza învățării

- **Pe scurt:** MLP 64-64 antrenat pe primele 60 % din cele două zile, evaluat pe următoarele 20 %.



Sursa: ULB Machine Learning Group și Worldline, tranzacții cu carduri (OpenML 1597). Date: septembrie 2013, 2 zile; antrenare = primele 33,4 ore, validare pînă la ora 40,3; 5 inițializări.

- Aria de sub curba precizie-sensibilitate (PR-AUC) după o epocă: SGD 0,004, moment 0,488, Adam 0,592 (orele de test: 0,786, 0,823, 0,803).

- **În practică:** cu o clasă rară, SGD simplu are nevoie de multe epoci, momentul sau Adam de una singură.

(calcul propriu)

NN_L2_fraud_optimizers

Compararea optimizatorilor

Metoda	Actualizarea	Memorie suplimentară	Cînd se folosește
SGD	$\theta - \eta g$	nimic	modele mari de imagini, program ajustat
Moment	$v \leftarrow \beta v + g$, $\theta - \eta v$	1 vector	pierderi prost condiționate
RMSProp	$\theta - \eta g/\sqrt{s}$, $s = \text{medie}(g^2)$	1 vector	rețele recurente, pierderi nestaționare
Adam	$\theta - \eta \hat{m}/(\sqrt{\hat{v}} + \epsilon)$	2 vectori	prima alegere implicită
AdamW	Adam + micșorare $(1 - \eta\lambda)\theta$	2 vectori	ori de cîte ori se folosește penalizarea

- Memoria este pe pondere: Adam reține două numere suplimentare pentru fiecare parametru, ceea ce contează la modelele foarte mari.

- **În practică:** alegeți după buget și stabilitate
  - Pe datele noastre, optimizatorii ajustați ajung aproape, iar Adam și AdamW sînt cei mai stabili.

(Bishop & Bishop, 2024, ch. 7)

Cuprins

- De la pierdere la gradient
- Propagarea înapoi a erorii
- Optimizarea stohastică
- Studiu de caz: valoarea la risc cu o rețea cuantilică
- Sinteză

Pierderea pinball transformă orice rețea într-un model de prognoză a cuantilelor

- **Pe scurt:** penalizăm randamentele de sub prognoză de $(1-\tau)/\tau$ ori mai mult decît pe cele de deasupra, iar minimul este cuantila τ.

$$\mathrm{VaR}^{\tau}_{t+1} = -q_\tau(r_{t+1} \mid \mathcal{F}_t), \qquad \rho_\tau(u) = u\big(\tau - \mathbb{1}\{u<0\}\big), \qquad \hat{q} = \arg\min_{q} \ \mathbb{E}\,\rho_\tau(r-q)$$

- **unde**
  - r_{t+1} randamentul logaritmic de a doua zi, în %, $\mathcal{F}_t$ informația pînă în ziua t, $\tau = 1\,\%$ pentru un VaR de 99 %
- Rețeaua: intrări r_t, $|r_t|$, media pe 5 zile, volatilitatea pe 5, 22 și 66 de zile
  - MLP 32-32 antrenat cu Adam pe $\frac{1}{n}\sum \rho_\tau(r_{t+1} - \hat{q}_t)$.
- Antrenare de la 8 aprilie 2005 pînă în 2016, validare 2017–2019 pentru oprirea timpurie, test 3 ianuarie 2020 – 30 octombrie 2026
  - Media a 5 inițializări.
- Modele de referință: simularea istorică pe 250 de zile și volatilitatea calculată cu media mobilă ponderată exponențial (EWMA), $\lambda = 0{,}94$, cu cuantila normală.

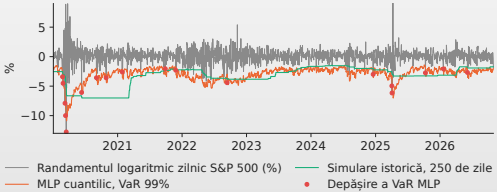
- **În practică:** aceeași rețea cu altă pierdere dă altă măsură de risc: pierderea definește ce se învață.

(Koenker & Bassett, 1978)

Exercițiu practic: notebook §8

Depășirile VaR-ului MLP cuantilic în pandemia din 2020 și după aceasta sînt apropiate de rata nominală

- **Pe scurt:** doar perioada de test; punctele roșii sînt zilele în care S&P 500 a pierdut mai mult decît VaR-ul de 99 % al MLP.



Sursa: Yahoo Finance, cotații zilnice. Date: 3 ianuarie 2020 - 30 octombrie 2026, zile de tranzacționare (perioada de test).

- Depășiri pe 1 716 de zile (așteptat 17,2): MLP 18, simularea istorică 31, EWMA 41
  - VaR mediu 2,96 %, 3,28 %, 2,52 %.

- **În practică:** un VaR care reacționează în cîteva zile la volatilitate evită grupările de depășiri ale simulării istorice.

(calcul propriu) NN_L2_var_backtest

Testul Kupiec acceptă MLP și respinge ambele modele de referință

- **Pe scurt:** dacă VaR-ul este corect, numărul de depășiri este binomial, cu probabilitatea τ; testăm rata observată față de ea.

$$\mathrm{LR}_{\mathrm{POF}} = -2 \ln \frac{(1-\tau)^{n-x}\tau^{x}}{(1-\hat{\pi})^{n-x}\hat{\pi}^{x}} \sim \chi^2_1, \quad \hat{\pi} = \frac{x}{n},\ n \text{ zile},\ x \text{ depășiri}$$

Model	Depășiri	Rata (%)	$\mathrm{LR}_{\mathrm{POF}}$	p	Pierderea pinball
MLP cuantilic (5 inițializări)	18	1,05	0,04	0,840	0,0420
Simulare istorică, 250 de zile	31	1,81	9,10	0,003	0,0577
EWMA, $\lambda = 0{,}94$	41	2,39	24,08	0,000	0,0473

- Inițializările individuale dau rate de depășire între 1,05 % și 1,22 %; pierderea pinball este scorul potrivit pentru compararea prognozelor de cuantile.

- **În practică:** o autoritate de supraveghere se uită întîi la numărul de depășiri
  - Un manager de risc ar trebui să compare și pierderile pinball.

(Kupiec, 1995)

Exercițiu practic: notebook §8

Echipamentul de calcul: procesorul grafic al unui laptop este de zece ori mai rapid decît procesorul central pe un MLP mare

- **Pe scurt:** antrenarea înseamnă înmulțiri de matrice; un procesor grafic (GPU) le execută în paralel, cu miile.
- Măsurat pe laptopul cursului (arm64, Darwin): un pas Adam pentru un MLP 2048-2048 (4 216 833 de parametri), pe un lot de 2 048 de rînduri reale California, durează 333,6 ms pe procesorul central (CPU, 4 fire) și 32,5 ms pe GPU prin Metal Performance Shaders (MPS): de 10,3 ori mai rapid.
- Rețelele tabelare mici din cursurile 1–3 se antrenează mai repede pe un singur fir al procesorului: copierea loturilor mici pe GPU costă mai mult decît economisește.
- Pe Google Colab, același cod rulează pe un GPU NVIDIA cu `device='cuda'`
  - Setați-l o dată și mutați modelul și datele cu `.to(device)`.
- **În practică:** măsurați înainte să cumpărați: pentru modele tabelare ajunge un CPU, pentru imagini și text un GPU este indispensabil.

(calcul propriu)

Exercițiu practic: notebook §9

O procedură standard de antrenare

Înainte de antrenare

- Standardizați intrările pe setul de antrenare
  - Împărțiți cronologic cînd timpul contează.
- Straturi ReLU cu inițializare He
  - Ieșirea și pierderea după țintă (MSE, entropie încrucișată, pinball).
- Întîi supra-ajustați un lot mic: dacă pierderea nu ajunge la zero, există o eroare în cod.

În timpul antrenării

- AdamW, η pe o grilă logaritmică în jurul lui 10^{-3}, loturi de 64–256, program cosinus.
- Urmăriți pierderea de antrenare și de validare la fiecare epocă
  - Păstrați ponderile cele mai bune pe validare (cursul 3).
- Repetați cu cel puțin 3–5 inițializări înainte de a anunța un cîștig.

În practică: scrieți rețeta în documentația modelului: ea face rezultatul reproductibil.

Cuprins

- De la pierdere la gradient
- Propagarea înapoi a erorii
- Optimizarea stohastică
- Studiu de caz: valoarea la risc cu o rețea cuantilică
- Sinteză

Idei principale

- Antrenarea este coborîre pe gradient pe riscul empiric
  - Condiționarea și rata de învățare îi stabilesc viteza și stabilitatea.
- Propagarea înapoi este regula de derivare a funcțiilor compuse aplicată invers: toți gradienții pentru costul a încă o propagare înainte
  - Verificați numeric codul propriu.
- Activările care se saturează și scările inițiale greșite fac gradienții să dispară
  - ReLU cu inițializare He îi păstrează.
- Momentul și pașii adaptivi (Adam, AdamW) ajung la soluții bune mai repede și mai sigur între inițializări
  - Programele ratei reduc eroarea finală.
- Pierderea definește produsul: cu pierderea pinball, un MLP dă un VaR de 99 % care trece testul Kupiec (18 depășiri, așteptat 17,2).

Ecuațiile principale

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta\, \mathbf{g}_t, \quad 0 < \eta < 2/\lambda_{\max}$$

$$\boldsymbol{\delta}^{(l)} = (\mathbf{W}^{(l+1)\top} \boldsymbol{\delta}^{(l+1)}) \odot a'(\mathbf{z}^{(l)})$$

$$\partial \ell / \partial \mathbf{W}^{(l)} = \boldsymbol{\delta}^{(l)} \mathbf{h}^{(l-1)\top}$$

$$\mathrm{Var}(W) = 2/n_{\text{in}} \quad \text{(He)}$$

$$\mathbf{v}_{t+1} = \beta \mathbf{v}_t + \mathbf{g}_t$$

$$\theta_{t+1} = \theta_t - \eta\, \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$$

$$\eta_t = \tfrac{\eta_0}{2}(1 + \cos \pi t / T)$$

$$\rho_\tau(u) = u(\tau - \mathbb{1}\{u < 0\})$$

Cercetările grupului nostru: prognoze de risc fără antrenare prin gradient

- **Riscul estimat de modele lingvistice** (Pele et al., 2026): valoarea la risc și pierderea așteptată în coada distribuției (ES) sînt obținute prin interogarea unor modele lingvistice mari (LLM) generale cu istoricul randamentelor (LLM-VaR, LLM-ES), fără antrenare prin gradient specifică sarcinii — cursul 12.
- **Modele fundație pentru riscul extrem** (Jheng, Tak, Găman, Mazurencu-Marinescu-Pele & Pele, 2026): modele fundație preantrenate pentru serii de timp prognozează VaR și ES fără antrenare suplimentară (zero-shot), iar calibrarea conformală le corectează acoperirea, pe piețele din Europa Centrală și de Est — cursul 13.
- **Legătura cu acest curs**: aceleași teste retroactive (numărul de depășiri, testul Kupiec, pierderea pinball) compară MLP-ul nostru cuantilic, antrenat prin gradient, cu aceste modele preantrenate.
- Rezultatele sînt cele declarate în lucrări; cifrele de pe celelalte slide-uri provin din calculele noastre.

Explorați vizual

- **TensorFlow Playground**: apăsați butonul de pas pentru a vedea cîte o actualizare a gradientului, apoi comparați ratele de învățare 0,03, 3 și 0,0001: pierderea se stabilizează, oscilează sau abia se mișcă.
- **3Blue1Brown, capitolele 2–4**: coborîrea pe gradient, intuiția propagării înapoi și calculul ei, cu animații ale regulii de derivare a funcțiilor compuse.

Lecturi și resurse suplimentare

- **Bishop & Bishop (2024)**, capitolele 7–8: coborîrea pe gradient și propagarea înapoi: bishopbook.com.
- **Dive into Deep Learning**, capitolul 12 (algoritmi de optimizare): d2l.ai.
- **A. Karpathy, micrograd**: propagarea înapoi în 100 de rînduri: github.com/karpathy/micrograd.
- **Tutorialul PyTorch despre autograd**: pytorch.org/tutorials.
- **Acest curs**: cod, date, notebook-uri, Quantlets — github.com/danpele/NEURAL_BIZ.
- **Lucrările originale**: Kingma & Ba (2015), Loshchilov & Hutter (2019).

Test 1: pasul stabil

- Regresia pe venitul brut are valorile proprii ale hessianei 0,372 și 38,81.
  - Pînă la ce rată de învățare converge coborîrea pe gradient?

Răspuns

- $\eta < 2/\lambda_{\max} = 0{,}0515$
  - Cu $\kappa = 104{,}3$, direcția plată se contractă foarte lent: standardizați sau folosiți momentul.

Test 2: un neuron ReLU mort

- În exemplul calculat de mînă, de ce $\partial \ell / \partial W_{2j}^{(1)} = 0$ pentru ambele intrări?

Răspuns

- Neuronul 2 are $z_2 = -0{,}093 < 0$, deci $\mathbb{1}\{z_2 > 0\} = 0$ și $\delta_2^{(1)} = 0$: un ReLU oprit nu transmite gradient.

Test 3: dispariția gradienților

- De ce învață atît de lent primele straturi ale unei rețele cu sigmoide și 10 straturi?

Răspuns

- Semnalul înapoi este înmulțit cu $\sigma' \leq 0{,}25$ la fiecare strat
  - Pe lotul nostru, norma gradientului în primul strat este $8{,}5 \times 10^{-7}$.

Test 4: corecția deplasării în Adam

- La pasul $t = 1$, cu $m_0 = 0$ și $\beta_1 = 0{,}9$, cît sînt m_1 și $\hat{m}_1$?

Răspuns

- $m_1 = 0{,}1\, g_1$, deplasat spre zero
  - $\hat{m}_1 = m_1/(1 - 0{,}9) = g_1$, corecția elimină deplasarea.

Test 5: testarea retroactivă a unui VaR

- Pe 1 716 de zile de test, cîte depășiri ale unui VaR corect de 99 % ne așteptăm și ce model trece testul Kupiec?

Răspuns

- Circa 17,2
  - MLP cuantilic are 18 ($p = 0{,}840$), simularea istorică 31 ($p = 0{,}003$): doar MLP trece.

Bibliografie (1/2)

- Baydin, A. G., Pearlmutter, B. A., Radul, A. A. & Siskind, J. M. (2018). Automatic differentiation in machine learning: a survey. *Journal of Machine Learning Research*, 18(153), 1–43. arXiv:1502.05767
- Bengio, Y., Simard, P. & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2), 157–166. doi:10.1109/72.279181
- Bishop, C. M. & Bishop, H. (2024). *Deep Learning: Foundations and Concepts*. Springer. doi:10.1007/978-3-031-45468-4
- Dal Pozzolo, A., Caelen, O., Johnson, R. A. & Bontempi, G. (2015). Calibrating probability with undersampling for unbalanced classification. *IEEE Symposium Series on Computational Intelligence*, 159–166. doi:10.1109/SSCI.2015.33
- Duchi, J., Hazan, E. & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12, 2121–2159.
- Glorot, X. & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. *Proceedings of AISTATS*, PMLR 9, 249–256.
- He, K., Zhang, X., Ren, S. & Sun, J. (2015). Delving deep into rectifiers. *IEEE International Conference on Computer Vision*, 1026–1034. doi:10.1109/ICCV.2015.123
- Jheng, S.-L., Tak, R., Găman, Ș., Mazurencu-Marinescu-Pele, M. & Pele, D. T. (2026). Can foundation models manage risk? Zero-shot VaR and ES forecasting with conformal calibration in CEE markets. *Economic Computation and Economic Cybernetics Studies and Research*, 60(2), 267–290. doi:10.24818/18423264/60.2.26.14
- Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M. & Tang, P. T. P. (2017). On large-batch training for deep learning: Generalization gap and sharp minima. *International Conference on Learning Representations*. arXiv:1609.04836
- Kingma, D. P. & Ba, J. (2015). Adam: A method for stochastic optimization. *International Conference on Learning Representations*. arXiv:1412.6980
- Koenker, R. & Bassett, G. (1978). Regression quantiles. *Econometrica*, 46(1), 33–50. doi:10.2307/1913643

Bibliografie (2/2)

- Kupiec, P. H. (1995). Techniques for verifying the accuracy of risk measurement models. *The Journal of Derivatives*, 3(2), 73–84. doi:10.3905/jod.1995.407942
- Loshchilov, I. & Hutter, F. (2017). SGDR: Stochastic gradient descent with warm restarts. *International Conference on Learning Representations*. arXiv:1608.03983
- Loshchilov, I. & Hutter, F. (2019). Decoupled weight decay regularization. *International Conference on Learning Representations*. arXiv:1711.05101
- Pace, R. K. & Barry, R. (1997). Sparse spatial autoregressions. *Statistics & Probability Letters*, 33(3), 291–297. doi:10.1016/S0167-7152(96)00140-X
- Paszke, A. et al. (2019). PyTorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, 32. arXiv:1912.01703
- Pele, D. T., Bolovăneanu, V., Lin, M.-B., Ren, R., Ginavar, A. T., Spilak, B., Andrei, A.-V., Toma, F.-M., Lessmann, S. & Härdle, W. K. (2026). In the beginning was the Word: LLM-VaR and LLM-ES. *Expert Systems with Applications*, 295, 128676. doi:10.1016/j.eswa.2025.128676
- Polyak, B. T. (1964). Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5), 1–17. doi:10.1016/0041-5553(64)90137-5
- Robbins, H. & Monro, S. (1951). A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3), 400–407. doi:10.1214/aoms/1177729586
- Rumelhart, D. E., Hinton, G. E. & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323, 533–536. doi:10.1038/323533a0
- Smith, L. N. & Topin, N. (2019). Super-convergence: Very fast training of neural networks using large learning rates. *Proceedings of SPIE*, 11006. arXiv:1708.07120
- Tieleman, T. & Hinton, G. (2012). Lecture 6.5 — RMSProp: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*.

Anexă: propagarea înapoi în formă matriceală pentru un mini-lot

$$\mathbf{Z}^{(l)} = \mathbf{H}^{(l-1)}\mathbf{W}^{(l)\top} + \mathbf{1}\mathbf{b}^{(l)\top}, \quad \mathbf{H}^{(l)} = a(\mathbf{Z}^{(l)}), \qquad \mathbf{\Delta}^{(l)} = \left(\mathbf{\Delta}^{(l+1)}\mathbf{W}^{(l+1)}\right) \odot a'(\mathbf{Z}^{(l)})$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(l)}} = \frac{1}{B}\mathbf{\Delta}^{(l)\top}\mathbf{H}^{(l-1)}, \qquad \frac{\partial \mathcal{L}}{\partial \mathbf{b}^{(l)}} = \frac{1}{B}\mathbf{\Delta}^{(l)\top}\mathbf{1}$$

- Rîndurile lui $\mathbf{H}$, $\mathbf{Z}$, $\mathbf{\Delta}$ sînt cele B exemple ale lotului
  - Fiecare pas este un produs de matrice, de aceea ajută procesoarele grafice.
- Costul: înainte $\sum_l B\, n_{l-1} n_l$ înmulțiri și adunări
  - Înapoi, cam de două ori mai mult.

Anexă: de ce are Adam nevoie de corecția deplasării

$$m_t = (1-\beta_1)\sum_{k=1}^{t}\beta_1^{t-k}g_k \;\Rightarrow\; \mathbb{E}[m_t] = \mathbb{E}[g]\,(1-\beta_1)\sum_{k=1}^{t}\beta_1^{t-k} = \mathbb{E}[g]\,(1-\beta_1^t)$$

- Dacă distribuția gradientului este staționară, $m_t/(1-\beta_1^t)$ este nedeplasat pentru $\mathbb{E}[g]$
  - La fel v_t pentru $\mathbb{E}[g^2]$.
- Cu $\beta_2 = 0{,}999$, v_t fără corecție este de 1000 de ori prea mic la $t = 1$: primii pași ar fi uriași.

(Kingma & Ba, 2015)

Anexă: testul raportului de verosimilitate al lui Kupiec

- Depășirile $I_t = \mathbb{1}\{r_t < -\mathrm{VaR}_t\}$
  - Dacă modelul este corect, $I_t \overset{\text{i.i.d.}}{\sim} \text{Bernoulli}(\tau)$ și $x = \sum I_t \sim \text{Bin}(n, \tau)$.
- Verosimilitatea sub H_0: $(1-\tau)^{n-x}\tau^x$
  - Sub alternativă, maximizată în $\hat{\pi} = x/n$.
- $\mathrm{LR} = -2\ln(L_0/L_1) \to \chi^2_1$
  - Respingem la 5 % cînd $\mathrm{LR} > 3{,}84$.
- Testul verifică doar numărul
  - Independența depășirilor cere testul Christoffersen (cursul 13).

(Kupiec, 1995)