



Neural Networks and Deep Learning

Lecture 2: Training: backpropagation and optimisation



Prof. dr. Daniel Traian PELE

Bucharest University of Economic Studies (ASE), Department of Statistics and Econometrics
Romanian Academy, Institute for Economic Forecasting (IPE)
IDA Institute Digital Assets ▪ AI4EFin ▪ MSCA Digital Finance

Learning objectives

- Write gradient descent (GD) as the solution method of empirical risk minimisation and explain why conditioning sets its speed.
- Derive backpropagation for a multilayer perceptron (MLP), compute it by hand on real data, and check it against automatic differentiation.
- Explain vanishing and exploding gradients and choose an initialisation that keeps the signal alive.
- Compare stochastic gradient descent (SGD), momentum, root mean square propagation (RMSProp), Adam and learning-rate schedules on real data, with several seeds.
- Train a quantile network for the 1-day value at risk (VaR) of the S&P 500 and backtest it.

Abbreviations

AdamW	Adam with decoupled weight decay
CPU	central processing unit
ES	expected shortfall
EWMA	exponentially weighted moving average
GD	gradient descent
GPU	graphics processing unit
LLM	large language model
MLP	multilayer perceptron
MPS	Metal Performance Shaders (Apple GPU backend of PyTorch)
MSE	mean squared error
OpenML	Open Machine Learning platform
PCA	principal component analysis
PR-AUC	area under the precision-recall curve
ReLU	rectified linear unit
RMSProp	root mean square propagation
SGD	stochastic gradient descent
ULB	Université libre de Bruxelles
VaR	value at risk

Outline

- From the loss to the gradient
- Backpropagation
- Stochastic optimisation
- Case study: value at risk with a quantile network
- Wrap-up

Training a network is a high-dimensional, non-convex minimisation

- **In one sentence:** we cannot solve for the weights as in least squares; we walk downhill on the loss, one step at a time.

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

- **where**
  - P the number of parameters, $\eta > 0$ the learning rate, $\nabla \mathcal{L}$ the gradient, computed by backpropagation (Section 2)
- Why it works: $\mathcal{L}(\boldsymbol{\theta} - \eta \mathbf{g}) = \mathcal{L}(\boldsymbol{\theta}) - \eta \|\mathbf{g}\|^2 + O(\eta^2)$, so a small step against the gradient lowers the loss.
- The MLP 64-64 of this lecture has 4 801 parameters on 12 384 training block groups
  - Models in production have millions.
- Two questions: how to get $\nabla \mathcal{L}$ cheaply and how to step well.

- **In practice:** every model of this course is trained by some variant of this one-line update.

(Bishop & Bishop, 2024, ch. 7)

The data of this lecture: houses, card payments and the S&P 500

Dataset	Rows	Task	Split
California housing	20 640	regression: house value	random 60/20/20: 12 384 / 4 128 / 4 128
Card fraud (sample)	28 924	classification: fraud	by time: first 60 % train, next 20 % validation, last 20 % test
S&P 500	1 716	1 % quantile of the next-day return	by time: train to 2016, validation 2017–2019, test from 2020

- Fraud: Université libre de Bruxelles (ULB) and Worldline data, on the Open Machine Learning platform (OpenML); all 492 frauds and a 10 % sample of legitimate payments; inputs V1–V28 are principal component analysis (PCA) scores released by the bank.
- Weights restore the full-data fraud rate of 0.173 % in every precision-type metric; S&P 500: daily log returns in %, test rows = forecast days.

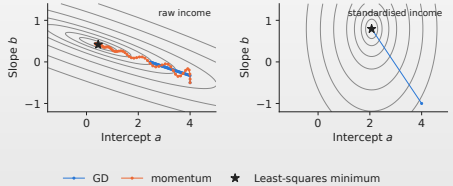
- **In practice:** time series and fraud are split in time order: the model never sees the future.

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

Experiment yourself: notebook §1

Badly scaled inputs turn the loss into a narrow valley

- **In one sentence:** the real loss of $\hat{y} = a + b \cdot \text{income}$ on California: gradient descent zig-zags on raw income and goes straight on standardised income.



Source: StatLib, California housing, 1990 census (Pace and Barry, 1997). Data: 20,640 block groups, 1990 census.

- Condition number $\kappa = \lambda_{\max}/\lambda_{\min}$ of the Hessian: 104.3 raw, 1 standardised
  - After 60 steps: mean squared error (MSE) 1.486 (GD, raw), 0.704 (momentum, raw), 0.701 (GD, standardised)
  - Minimum 0.701.

- **In practice:** standardise every input on the training set before training any network.

(own computation) NN_L2_gd_paths

On a quadratic, the step size must stay below $2/\lambda_{\max}$

- **In one sentence:** along each eigen-direction of the Hessian, gradient descent multiplies the error by $(1-\eta\lambda_k)$.

$$\mathcal{L}(\boldsymbol{\theta}) = \mathcal{L}^* + \tfrac{1}{2}(\boldsymbol{\theta}-\boldsymbol{\theta}^*)^\top \mathbf{H}(\boldsymbol{\theta}-\boldsymbol{\theta}^*) \;\Rightarrow\; \boldsymbol{\theta}_t - \boldsymbol{\theta}^* = (\mathbf{I}-\eta\mathbf{H})^t(\boldsymbol{\theta}_0-\boldsymbol{\theta}^*), \qquad |1-\eta\lambda_k|<1 \iff 0<\eta<\frac{2}{\lambda_{\max}}$$

- **where**
  - $\mathbf{H}$ the Hessian with eigenvalues $\lambda_{\min} \le \cdots \le \lambda_{\max}$, $\boldsymbol{\theta}^*$ the minimum
- With the best fixed step the error shrinks by $(\kappa-1)/(\kappa+1)$ per step: slow when κ is large.
- Raw income: $\kappa = 104.3$, so the step $1/\lambda_{\max} = 0.0258$ barely moves the flat direction.
- Networks are not quadratic, but near a minimum they behave like one: the same limit explains divergence at large η.

- **In practice:** if the loss explodes in the first epochs, divide the learning rate by 3 before changing anything else.

(Bishop & Bishop, 2024, ch. 7)

Outline

- From the loss to the gradient
- Backpropagation
- Stochastic optimisation
- Case study: value at risk with a quantile network
- Wrap-up

Backpropagation is the chain rule, applied from the loss backwards

- **In one sentence:** each node of the computational graph multiplies the incoming gradient by its own local derivative and passes it on.

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

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

- **where**
  - $\boldsymbol{\delta}^{(l)}$ the error signal of layer l, $\odot$ the element-wise product, a' the derivative of the activation
- Output error: $\hat{y} - y$ (squared loss), $p - y$ (sigmoid with cross-entropy, Lecture 1).
- Weight gradient = error of the layer times its input, as in the logistic gradient.

- **In practice:** the forward pass must store $\mathbf{z}^{(l)}$ and $\mathbf{h}^{(l)}$: this memory, not arithmetic, limits batch size on a graphics processing unit (GPU).

(Rumelhart, Hinton & Williams, 1986)

Backpropagation by hand on one real block group

- **In one sentence:** two standardised inputs, two rectified linear units (ReLU), one linear output and the loss $\ell = \frac{1}{2}(\hat{y} - y)^2$.

Forward	unit 1	unit 2
input x (income, latitude)	-0.21	-0.67
weights $\mathbf{W}^{(1)}$, row 1 / row 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
output weights $w^{(2)}$	-0.87	-0.19
$\hat{y} = w^{(2)\top} h$; y	-0.377	2.067

Backward	unit 1	unit 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

- Unit 2 is switched off ($z_2 < 0$): it passes no gradient, so its weights do not move on this example.
- PyTorch autograd gives the same gradients (largest difference 0.0).

- **In practice:** a ReLU unit that is off for every customer never learns again: watch the share of dead units.

(own computation)

Experiment yourself: notebook §2

Reverse mode costs about one extra forward pass, whatever the number of weights

- **In one sentence:** one backward sweep gives all P partial derivatives of a scalar loss; finite differences would need P extra passes.

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

- Forward mode (dual numbers) is cheap for few inputs and many outputs
  - Reverse mode for many inputs (weights) and one output (loss).
- Gradient check on our MLP 64-64: 30 random weights, central differences with $\epsilon = 10^{-6}$ in double precision, largest relative error 7.0×10^{-7}.
- PyTorch records the graph during the forward pass
  - `loss.backward()` walks it in reverse.

- **In practice:** when you write a custom layer or loss, run a gradient check once before trusting it.

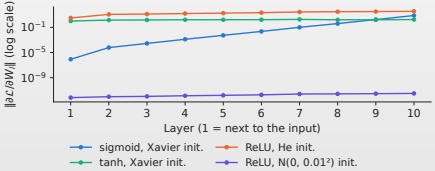
(Baydin et al., 2018; Paszke et al., 2019)

Experiment yourself: notebook §2

Vanishing gradients in deep sigmoid networks

- **In one sentence:** the backward signal is a product of Jacobians; with $\sigma' \le 1/4$ it shrinks at every layer.

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



Source: StatLib, California housing, 1990 census (Pace and Barry, 1997). Data: 10 hidden layers of 64 units, one batch of 512 training block groups, 5 seeds.

- 10 hidden layers, one real batch: first/last layer gradient norm 8.5×10^{-7} vs 7.5×10^{0} with sigmoids
  - 3.1×10^{0} vs 3.5×10^{1} with ReLU and He initialisation.

- **In practice:** deep nets need ReLU-type activations, a scaled initialisation and, later, normalisation (Lecture 3).

(own computation)

NN_L2_vanishing

Initialisation that keeps the variance of the signal constant across layers

- **In one sentence:** too small and the signal vanishes, too large and it explodes; the right scale depends on the fan-in and on the activation.

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

- **where**
  - n_{l-1} the fan-in, n_l the fan-out of layer l, for ReLU, $\mathbb{E}[h^2] = \frac{1}{2}\,\mathrm{Var}(z)$, hence the factor 2
- Weights $\mathcal{N}(0, 0.01^2)$ in our 10-layer ReLU net: first-layer gradient norm 6.9×10^{-13}, i.e. no learning at all.
- Never initialise all weights equal: every unit would learn the same thing.

- **In practice:** PyTorch's defaults are close to these rules
  - Set He initialisation explicitly for deep ReLU nets.

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

Experiment yourself: notebook §3

Outline

- From the loss to the gradient
- Backpropagation
- Stochastic optimisation
- Case study: value at risk with a quantile network
- Wrap-up

A mini-batch gives an unbiased, noisy gradient at a fraction of the cost

- **In one sentence:** averaging the gradient over B random examples is enough to know roughly which way is down.

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

- **where**
  - $\mathcal{B}_t$ a random batch of size B, Σ the covariance of the per-example gradients, one epoch $= \lceil n/B \rceil$ steps
- Robbins–Monro: with $\sum_t \eta_t = \infty$ and $\sum_t \eta_t^2 < \infty$, SGD converges to a stationary point.
- The noise is not only a cost: it helps to leave sharp minima, which tend to generalise worse (Keskar et al., 2017).

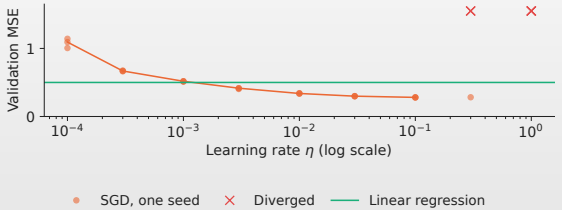
- **In practice:** batches of 32–256 are the usual compromise between noise, speed and memory.

(Robbins & Monro, 1951)

Experiment yourself: notebook §4

Effect of the learning rate on training

- **In one sentence:** plain SGD on an MLP 64-64, 30 epochs, batches of 128, for nine learning rates and three seeds.



Source: StatLib, California housing, 1990 census (Pace and Barry, 1997). Data: 12,384 training and 4,128 validation block groups; 30 epochs.

- Validation MSE 1.093 at $\eta = 10^{-4}$ (worse than linear regression, 0.500), 0.281 at $\eta = 0.1$
  - From $\eta = 0.3$ at least one seed diverges.

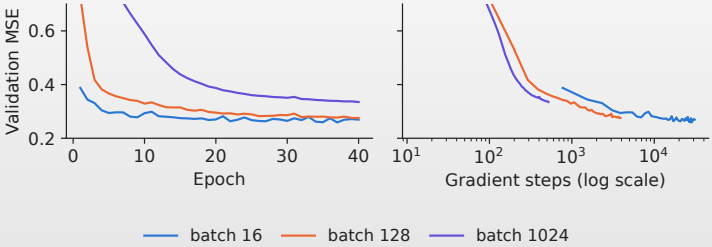
- **In practice:** tune the learning rate first, on a log grid, before any other hyperparameter.

(own computation)

NN_L2_lr_sweep

Effect of batch size: progress per epoch and per second

- **In one sentence:** Adam with learning rate 0.001, 40 epochs, batch sizes 16, 128 and 1,024.



Source: StatLib, California housing, 1990 census (Pace and Barry, 1997). Data: Adam, learning rate 0.001, 40 epochs.

- After 5 epochs: validation MSE 0.294 (batch 16), 0.367 (128), 0.870 (1,024)
  - Wall time for 40 epochs on one processor thread: 25.8, 3.7 and 1.0 seconds.

- **In practice:** compare training budgets in gradient steps or seconds, not in epochs.

(own computation) NN_L2_batch_size

Momentum: a running average of past gradients

- **In one sentence:** zig-zag components cancel in the average, the consistent downhill component accumulates.

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

- **where**
  - $\mathbf{v}$ the velocity, $\beta \in [0, 1)$ the momentum, usually 0.9, the effective step on a constant slope is $\eta/(1-\beta)$, ten times η
- On a quadratic, the best momentum improves the contraction from $\frac{\kappa-1}{\kappa+1}$ to $\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}$ (Polyak, 1964).
- Raw-income regression after 60 steps: MSE 0.704 with momentum against 1.486 for plain GD.

- **In practice:** SGD with momentum 0.9 is still the default for very large vision models.

(Polyak, 1964)

Experiment yourself: notebook §5

Adaptive methods give each weight its own step size

- **In one sentence:** divide the step by a running measure of how large that weight's gradients have been.

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

- **where**
  - All operations element by element, $\rho \approx 0.99$, $\epsilon \approx 10^{-8}$ avoids division by zero
- AdaGrad's sum only grows, so its steps shrink to zero: good for sparse features, too slow for deep nets (Duchi, Hazan & Singer, 2011).
- Root mean square propagation (RMSProp) forgets old gradients exponentially, so the step adapts to the current region (Tieleman & Hinton, 2012).
- Rescaling per weight partly undoes bad conditioning: rarely updated or flat directions get larger steps.
- **In practice:** adaptive methods are less sensitive to the learning rate, which makes them a safe first choice.

Experiment yourself: notebook §5

Adam: momentum, RMSProp and a bias correction

- **In one sentence:** keep moving averages of the gradient and of its square, correct them for starting at zero, and step by their ratio.

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

- **where**
  - Defaults $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$, $\eta = 10^{-3}$, $m_0 = v_0 = 0$, hence the correction (appendix)
- The step is roughly bounded by η per weight, whatever the scale of the loss: easy to tune.
- Adam with decoupled weight decay (AdamW) shrinks weights directly, $\theta \leftarrow \theta - \eta\lambda\theta$, instead of adding $\lambda\theta$ to g_t (Loshchilov & Hutter, 2019; Lecture 3).

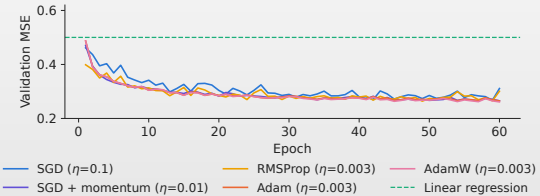
- **In practice:** start with AdamW and $\eta = 10^{-3}$
  - Move to tuned SGD with momentum only if it pays out of sample.

(Kingma & Ba, 2015)

Experiment yourself: notebook §5

On California housing, tuned optimisers reach similar errors and adaptive methods converge more reliably

- **In one sentence:** best of three learning rates per optimiser (on validation), mean of 5 new seeds.



Source: StatLib, California housing, 1990 census (Pace and Barry, 1997). Data: 12,384 training and 4,128 validation block groups.

- Validation MSE after 10 epochs: SGD 0.342, momentum 0.310, RMSProp 0.313, Adam 0.305, AdamW 0.304
  - Linear regression 0.500.

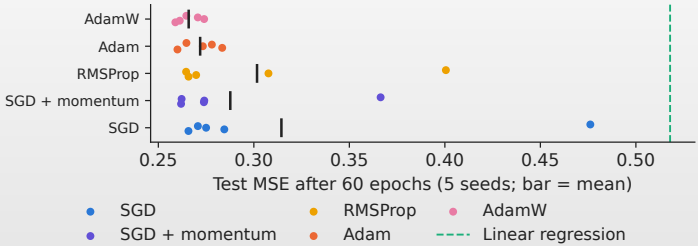
- **In practice:** the network halves the error of the linear benchmark whatever the optimiser.

(own computation)

NN_L2_optimizers

Across seeds, Adam and AdamW are far more stable than plain SGD

- **In one sentence:** test MSE of 5 seeds per optimiser; a single run would have ranked them by luck.



Source: StatLib, California housing, 1990 census (Pace and Barry, 1997). Data: 4,128 test block groups.

- Test MSE mean (standard deviation): SGD 0.314 (0.091), Adam 0.272 (0.010), AdamW 0.266 (0.006)
  - Paired t-test SGD vs Adam: $t = 1.01$, $p = 0.371$.

- **In practice:** the mean gain is not significant with 5 seeds, but the risk of a bad run is much lower with Adam.

(own computation)

NN_L2_optimizer_seeds

Learning-rate schedules: step, cosine and one-cycle

- **In one sentence:** explore with a large step, then reduce it so that the noise of SGD settles into a minimum.

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

$$\text{one-cycle: linear warm-up to } \eta_0, \text{ then cosine to } \approx 0$$

- **where**
  - η_0 the peak rate, γ the decay factor every s steps, T the total number of steps
- Cosine annealing (Loshchilov & Hutter, 2017) and one-cycle (Smith & Topin, 2019) need only the peak rate and the budget.
- Warm-up protects the first steps, when Adam's second-moment estimates and batch statistics are still unreliable.

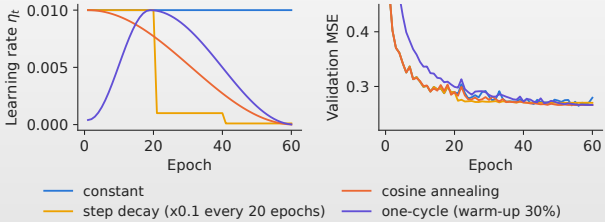
- **In practice:** a cosine schedule over the planned number of epochs is a safe default.

(Bishop & Bishop, 2024, ch. 7)

Experiment yourself: notebook §6

Decaying the learning rate lowers the final error

- **In one sentence:** SGD with momentum 0.9 and peak rate 0.01; left the schedules, right the validation MSE (mean of 3 seeds).



Source: StatLib, California housing, 1990 census (Pace and Barry, 1997). Data: SGD with momentum 0.9, peak rate 0.01, 3 seeds.

- Final validation MSE: constant 0.279, step 0.270, cosine 0.266, one-cycle 0.266
  - Test 0.286 vs 0.265 (constant vs cosine).

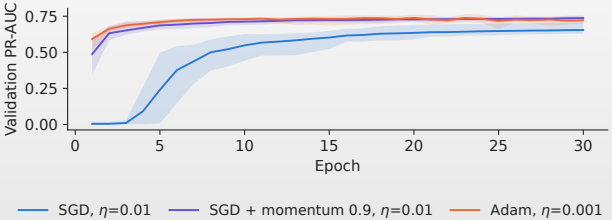
- **In practice:** the schedule is a cheap gain: same model, same budget, lower error.

(own computation)

NN_L2_schedules

On card fraud, the optimiser determines the speed of learning

- **In one sentence:** MLP 64-64 trained on the first 60 % of the two days, scored on the next 20 %.



Source: ULB Machine Learning Group and Worldline, credit-card transactions (OpenML 1597). Data: September 2013, 2 days; train = first 33.4 hours, validation to hour 40.3; 5 seeds.

- Area under the precision-recall curve (PR-AUC) after 1 epoch: SGD 0.004, momentum 0.488, Adam 0.592 (test hours: 0.786, 0.823, 0.803).

- **In practice:** with a rare class, plain SGD needs many epochs, momentum or Adam only one.

(own computation) NN_L2_fraud_optimizers

Comparison of optimisers

Method	Update	Extra memory	Use when
SGD	$\theta - \eta g$	none	large vision models, tuned schedule
Momentum	$v \leftarrow \beta v + g$, $\theta - \eta v$	1 vector	ill-conditioned losses
RMSProp	$\theta - \eta g / \sqrt{s}$, $s = \text{avg}(g^2)$	1 vector	recurrent nets, non-stationary losses
Adam	$\theta - \eta \hat{m} / (\sqrt{\hat{v}} + \epsilon)$	2 vectors	default first choice
AdamW	Adam + decay $(1 - \eta\lambda)\theta$	2 vectors	whenever weight decay is used

- Memory is per weight: Adam stores two extra numbers for every parameter, which matters for very large models.

- **In practice:** choose by budget and stability
  - On our data the tuned optimisers end close, Adam and AdamW are the most stable.

(Bishop & Bishop, 2024, ch. 7)

Outline

- From the loss to the gradient
- Backpropagation
- Stochastic optimisation
- Case study: value at risk with a quantile network
- Wrap-up

The pinball loss turns any network into a quantile forecaster

- **In one sentence:** penalise returns below the forecast $1-\tau$ times more than returns above it, and the minimiser is the τ-quantile.

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

- **where**
  - r_{t+1} the next-day log return in %, $\mathcal{F}_t$ the information up to day t, $\tau = 1\,\%$ for a 99 % VaR
- Network: inputs r_t, $|r_t|$, 5-day mean, 5-, 22- and 66-day volatility
  - MLP 32-32 trained with Adam on $\frac{1}{n}\sum \rho_\tau(r_{t+1} - \hat{q}_t)$.
- Train from 8 Apr 2005 to 2016, validation 2017–2019 for early stopping, test 3 Jan 2020 to 30 Oct 2026
  - Average of 5 seeds.
- Benchmarks: 250-day historical simulation and exponentially weighted moving average (EWMA) volatility with $\lambda = 0.94$ and normal quantile.

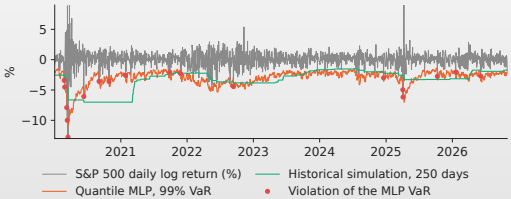
- **In practice:** the same network with another loss gives another risk measure: the loss defines what is learned.

(Koenker & Bassett, 1978)

Experiment yourself: notebook §8

Breaches of the quantile-MLP VaR during and after the 2020 pandemic are close to the nominal rate

- **In one sentence:** test period only; red dots are days when the S&P 500 lost more than the MLP's 99 % VaR.



Source: Yahoo Finance, daily quotes. Data: 3 Jan 2020 to 30 Oct 2026, trading days (test period).

- Violations over 1 716 days (expected 17.2): MLP 18, historical simulation 31, EWMA 41
  - Mean VaR 2.96 %, 3.28 %, 2.52 %.

- **In practice:** a VaR that reacts to volatility within days avoids the clusters of breaches of historical simulation.

(own computation)

NN_L2_var_backtest

The Kupiec test accepts the MLP and rejects both benchmarks

- **In one sentence:** if the VaR is right, the number of breaches is binomial with probability τ; test the observed rate against it.

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

Model	Violations	Rate (%)	$\mathrm{LR}_{\mathrm{POF}}$	p	Pinball loss
Quantile MLP (5 seeds)	18	1.05	0.04	0.840	0.0420
Historical simulation, 250 days	31	1.81	9.10	0.003	0.0577
EWMA, $\lambda = 0.94$	41	2.39	24.08	0.000	0.0473

- Individual seeds give violation rates between 1.05 % and 1.22 %; the pinball loss is the proper score for comparing quantile forecasts.

- **In practice:** a regulator looks at the breach count first
  - A risk manager should also compare pinball losses.

(Kupiec, 1995)

Experiment yourself: notebook §8

Training hardware: a laptop GPU is ten times faster than its main processor on a large MLP

- **In one sentence:** training is matrix multiplication; a graphics processing unit (GPU) runs thousands of them in parallel.
- Measured on this course's laptop (arm64, Darwin): one Adam step of an MLP 2048-2048 (4 216 833 parameters) on a batch of 2,048 real California rows takes 333.6 ms on the central processing unit (CPU, 4 threads) and 32.5 ms on the GPU through Metal Performance Shaders (MPS): 10.3 times faster.
- The small tabular nets of Lectures 1–3 train faster on one CPU thread: copying tiny batches to the GPU costs more than it saves.
- On Google Colab the same code runs on an NVIDIA GPU with `device='cuda'`
  - Set it once and move model and data with `.to(device)`.
- **In practice:** measure before you buy: for tabular models a CPU is enough, for images and text a GPU is not optional.

(own computation)

Experiment yourself: notebook §9

A default training procedure

Before training

- Standardise inputs on the training set
  - Split in time order when time matters.
- ReLU layers with He initialisation
  - Output and loss from the target (MSE, cross-entropy, pinball).
- Overfit one small batch first: if the loss does not go to zero, there is a bug.

During training

- AdamW, η on a log grid around 10^{-3}, batch 64–256, cosine schedule.
- Watch training and validation loss every epoch
  - Keep the best validation weights (Lecture 3).
- Repeat with at least 3–5 seeds before claiming a gain.

- **In practice:** write the recipe into the model documentation: it is what makes a result reproducible.

Outline

- From the loss to the gradient
- Backpropagation
- Stochastic optimisation
- Case study: value at risk with a quantile network
- Wrap-up

Key takeaways

- Training is gradient descent on the empirical risk
  - Conditioning and the learning rate set its speed and stability.
- Backpropagation is the chain rule in reverse: all gradients for about the cost of one more forward pass
  - Check custom code numerically.
- Saturating activations and bad initial scales make gradients vanish
  - ReLU with He initialisation keeps them alive.
- Momentum and adaptive steps (Adam, AdamW) reach good solutions faster and more reliably across seeds
  - Schedules lower the final error.
- The loss defines the product: with the pinball loss, an MLP gives a 99 % VaR that passes the Kupiec test (18 breaches, expected 17.2).

Key equations

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

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

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

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

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

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

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

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

Research from our group: risk forecasts without gradient training

- **Risk from language models** (Pele et al., 2026): value at risk and expected shortfall (ES) are obtained by prompting general-purpose large language models (LLM) with the return history (LLM-VaR, LLM-ES), without any task-specific gradient training — Lecture 12.
- **Foundation models for tail risk** (Jheng, Tak, Găman, Mazurencu-Marinescu-Pele & Pele, 2026): pretrained time-series foundation models forecast VaR and ES zero-shot, and conformal calibration corrects their coverage, in Central and Eastern European markets — Lecture 13.
- **How they relate to this lecture**: the same backtests (violation counts, Kupiec test, pinball loss) compare our gradient-trained quantile MLP with these pretrained models.
- Findings as stated in the papers; numbers on the other slides come from our own computations.

Explore it visually

- **TensorFlow Playground**: press the step button to see one gradient update at a time, then compare learning rates 0.03, 3 and 0.0001: the loss settles, oscillates, or barely moves.
- **3Blue1Brown, chapters 2–4**: gradient descent, the intuition of backpropagation and its calculus, with animations of the chain rule.

Further reading and resources

- **Bishop & Bishop (2024)**, chapters 7–8: gradient descent and backpropagation: bishopbook.com.
- **Dive into Deep Learning**, chapter 12 (optimisation algorithms): d2l.ai.
- **A. Karpathy, micrograd**: backpropagation in 100 lines: github.com/karpathy/micrograd.
- **PyTorch autograd tutorial**: pytorch.org/tutorials.
- **This course**: code, data, notebooks, Quantlets — github.com/danpele/NEURAL_BIZ.
- **Original papers**: Kingma & Ba (2015), Loshchilov & Hutter (2019).

Quiz 1: the stable step size

- The raw-income regression has Hessian eigenvalues 0.372 and 38.81.
  - Up to which learning rate does gradient descent converge?

Answer

- $\eta < 2/\lambda_{\max} = 0.0515$
  - With $\kappa = 104.3$ the flat direction contracts very slowly: standardise or use momentum.

Quiz 2: a dead ReLU

- In the hand-computed example, why is $\partial \ell / \partial W_{2j}^{(1)} = 0$ for both inputs?

Answer

- Unit 2 has $z_2 = -0.093 < 0$, so $\mathbb{1}\{z_2 > 0\} = 0$ and $\delta_2^{(1)} = 0$: a ReLU that is off passes no gradient.

Quiz 3: vanishing gradients

- Why does a 10-layer sigmoid network learn its first layers so slowly?

Answer

- The backward signal is multiplied by $\sigma' \leq 0.25$ at each layer
  - On our batch the first-layer gradient norm is 8.5×10^{-7}.

Quiz 4: Adam's bias correction

- At step $t = 1$, with $m_0 = 0$ and $\beta_1 = 0.9$, what are m_1 and $\hat{m}_1$?

Answer

- $m_1 = 0.1\, g_1$, biased towards zero
  - $\hat{m}_1 = m_1/(1 - 0.9) = g_1$, the correction removes the bias.

Quiz 5: backtesting a VaR

- Over 1 716 test days, how many breaches of a correct 99 % VaR do we expect, and which model passes the Kupiec test?

Answer

- About 17.2
  - The quantile MLP has 18 ($p = 0.840$), historical simulation 31 ($p = 0.003$): only the MLP passes.

References (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

References (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*.

Appendix: backpropagation in matrix form for a mini-batch

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

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

- Rows of $\mathbf{H}$, $\mathbf{Z}$, $\mathbf{\Delta}$ are the B examples of the batch
  - Every step is a matrix product, which is why graphics processors help.
- Cost: forward $\sum_l B\, n_{l-1} n_l$ multiply-adds
  - Backward about twice that.

Appendix: why Adam needs a bias correction

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

- If the gradient distribution is stationary, $m_t/(1-\beta_1^t)$ is unbiased for $\mathbb{E}[g]$
  - The same holds for v_t and $\mathbb{E}[g^2]$.
- With $\beta_2 = 0.999$, v_t without correction is 1000 times too small at $t = 1$: the first steps would be huge.

(Kingma & Ba, 2015)

Appendix: the Kupiec likelihood-ratio test

- Violations $I_t = \mathbb{1}\{r_t < -\mathrm{VaR}_t\}$
  - Under a correct model, $I_t \overset{\text{i.i.d.}}{\sim} \text{Bernoulli}(\tau)$ and $x = \sum I_t \sim \text{Bin}(n, \tau)$.
- Likelihood under H_0: $(1-\tau)^{n-x}\tau^x$
  - Under the alternative, maximised at $\hat{\pi} = x/n$.
- $\mathrm{LR} = -2\ln(L_0/L_1) \to \chi^2_1$
  - Reject at 5 % when $\mathrm{LR} > 3.84$.
- The test checks only the count
  - Independence of the violations needs Christoffersen's test (Lecture 13).

(Kupiec, 1995)