Universidade de São Paulo
Instituto de Matemática e Estatística
Aluna: Nara Avila Moraes
Nro. USP: 5716734

2o. semestre, 2025
Lista 03 - 21/11

MAE5911/IME: Fundamentos de Estatística e Machine Learning.
Prof.: Alexandre Galvão Patriota

Questão 01: Considere uma amostra aleatória $(Y_1, X_1), \ldots, (Y_n, X_n)$ de (Y, X) tal que a distribuição condicional $Y \mid X = x \sim \mathcal{N}\big(\mu_\theta(x),\, \sigma^2_\theta(x)\big)$, e suponha que a distribuição de X não contém informação sobre os parâmetros.

Ítem 1.1 Apresente uma **rede neural** que modele o **quantil de ordem** 75% (terceiro quartil) da distribuição condicional $Y \mid X = x$. O código deve ser generalizável para qualquer quantil.

Ítem 1.2 Mostre a aplicação do método nos seguintes dados simulados em R:

$$\text{set.seed}(32)$$

$$n = 1000$$

$$x = \text{sort}(\text{runif}(n, -4, 4))$$

$$y = \frac{3}{3 + 2|x|^3} + e^{-x^2} + \cos(x)\sin(x) + 0.3\,\varepsilon, \qquad \varepsilon \sim \mathcal{N}(0,1)$$

Sugestão: reescreva a esperança e variância condicionais em termos do quantil e construa a função de verossimilhança apropriada.

Resposta para Ítem 1.1

Definição da função de perda do modelo

Densidade condicional

Podemos descrever a média da Normal em função do quantil alvo $Q_q(x)$. O valor crítico da Normal padrão é dado por:

$$z_q = \Phi^{-1}(q),$$

A média condicional pode ser escrita como:

$$\mu(x) = Q_q(x) - \sigma(x)\, z_q.$$

Assim, o modelo probabilístico é:

$$Y \mid X = x \sim \mathcal{N}\big(\mu(x),\, \sigma^2(x)\big).$$

A densidade condicional para uma observação (x_i, y_i) é, portanto,

$$f(y_i \mid x_i; Q_q, \sigma) = \frac{1}{\sqrt{2\pi}\,\sigma(x_i)} \exp\left\{ -\frac{\big(y_i - Q_q(x_i) + \sigma(x_i) z_q\big)^2}{2\sigma^2(x_i)} \right\}.$$

Log-verossimilhança

Tomando log da expressão acima, obtemos a log-verossimilhança para uma única observação:

$$\ell_i(Q_q, \sigma) = \log f(y_i \mid x_i; Q_q, \sigma) = -\frac{1}{2}\log(2\pi) - \log \sigma(x_i) - \frac{\big(y_i - Q_q(x_i) + \sigma(x_i) z_q\big)^2}{2\sigma^2(x_i)}.$$

Somando sobre todas as observações, a log-verossimilhança total é:

$$\ell(Q_q, \sigma) = \sum_{i=1}^{n} \ell_i(Q_q, \sigma) = -\frac{n}{2}\log(2\pi) - \sum_{i=1}^{n} \log \sigma(x_i) - \sum_{i=1}^{n} \frac{\big(y_i - Q_q(x_i) + \sigma(x_i) z_q\big)^2}{2\sigma^2(x_i)}.$$

O negativo da log-verossimilhança (função de perda) é, portanto,

$$\mathcal{L}_{\mathrm{NLL}}(Q_q, \sigma) = -\ell(Q_q, \sigma) = \frac{n}{2}\log(2\pi) + \sum_{i=1}^{n} \log \sigma(x_i) + \sum_{i=1}^{n} \frac{\big(y_i - Q_q(x_i) + \sigma(x_i) z_q\big)^2}{2\sigma^2(x_i)}.$$

O termo

$$\frac{n}{2}\log(2\pi)$$

não depende de Q_q nem de $\sigma(x)$, e portanto não altera o ponto de mínimo nem os gradientes no treinamento da rede neural. Assim, podemos descartá-lo e usar a forma simplificada da log-verossimilhança:

$$\ell_i(Q_q, \sigma) = -\log \sigma(x_i) - \frac{\big(y_i - Q_q(x_i) + \sigma(x_i) z_q\big)^2}{2\sigma^2(x_i)}.$$

Portanto, a função de perda que iremos usar no treinamento da rede, negativo da log-verossimilhança, é:

$$\mathcal{L}_{\mathrm{NLL}}(Q_q, \sigma) = \sum_{i=1}^{n} \left[\log \sigma(x_i) + \frac{\big(y_i - Q_q(x_i) + \sigma(x_i) z_q\big)^2}{2\sigma^2(x_i)} \right].$$

Definição da Arquitetura

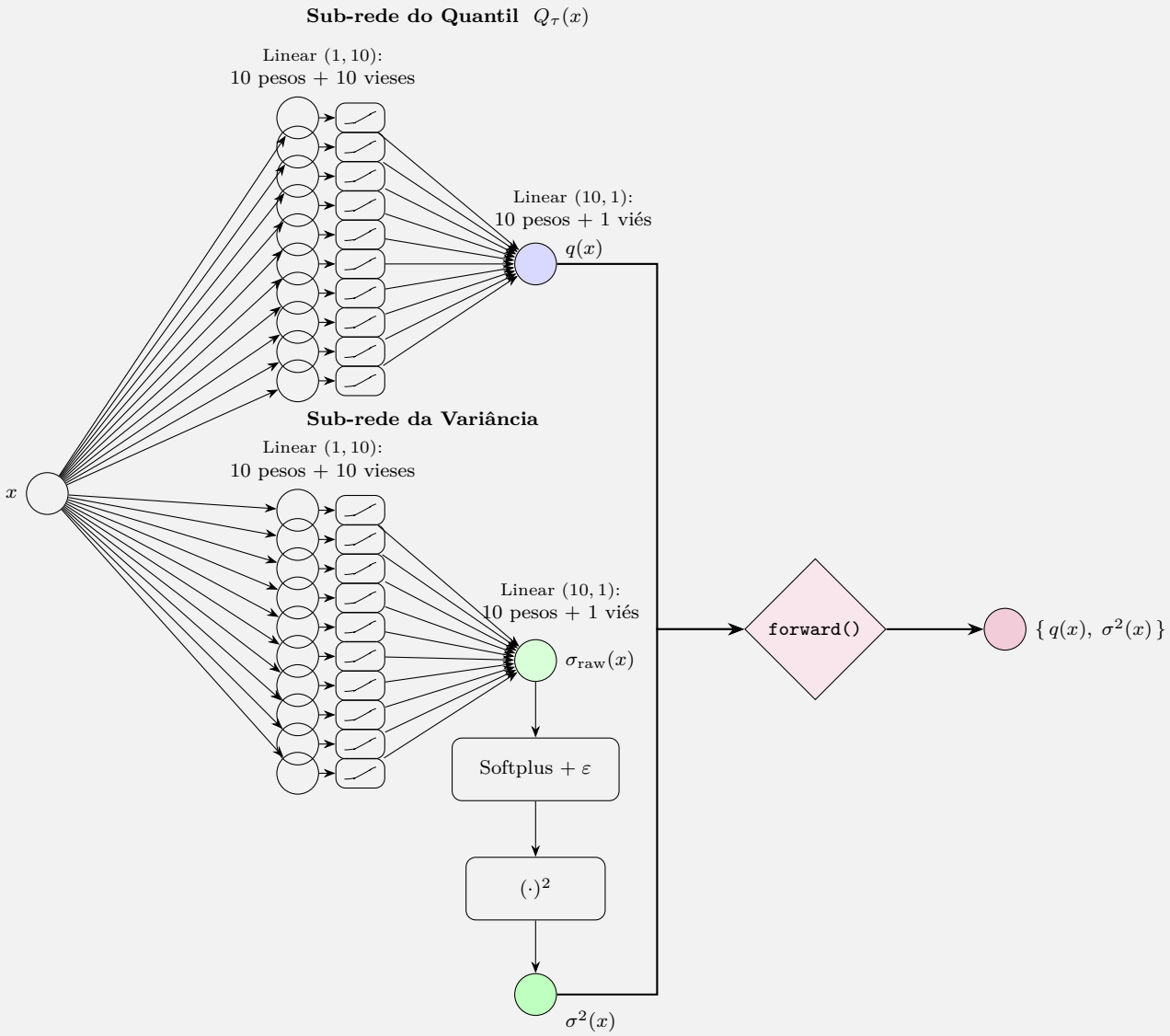

Figura 1: Arquitetura da rede neural com 10 neurônios ocultos em cada subrede.

Parâmetro	Dimensão	Nº de parâmetros
q__net.0.weight	10×1	10
q__net.0.bias	10	10
q__net.2.weight	1×10	10
q__net.2.bias	1	1
sigma__net.0.weight	10×1	10
sigma__net.0.bias	10	10
sigma__net.2.weight	1×10	10
sigma__net.2.bias	1	1
Total	—	**62**

Tabela 1: Parâmetros treináveis da rede neural utilizada no experimento.

```
library(torch)

QuantileNormalNet <- nn_module(
  "QuantileNormalNet",

  initialize = function(input_dim,
                        hidden_q     = config$hidden_mu,
                        hidden_sigma = config$hidden_sigma) {

```

```
    # Rede do quantil Q_q(x)
    self$q_net <- nn_sequential(
      nn_linear(input_dim, hidden_q),
      nn_gelu(),
      nn_linear(hidden_q, 1)
    )

    # Rede da variância sigma(x)
    self$sigma_net <- nn_sequential(
      nn_linear(input_dim, hidden_sigma),
      nn_gelu(),
      nn_linear(hidden_sigma, 1)
    )
  },

  forward = function(x) {
    # x: tensor [batch_size, 1]

    # Predição do quantil Q_q(x)
    q_pred <- self$q_net(x)

    # Predição da variância sigma(x)
    sigma_raw <- self$sigma_net(x)
    sigma     <- nnf_softplus(sigma_raw) + 1e-4
    sigma2    <- sigma$pow(2)

    list(
      q      = q_pred,
      sigma2 = sigma2
    )
  }
)
```

Listing 1: Estrutura do modelo QuantileNormalNet em `torch` para R

```
nll_normal_from_quantile <- function(Qq, sigma2, y, q_level, eps = 1e-6) {

  Qq     <- torch_tensor(Qq, dtype = torch_float())
  sigma2 <- torch_tensor(sigma2, dtype = torch_float())
  y      <- torch_tensor(y,  dtype = torch_float())

  sigma <- sigma2$sqrt()

  z_q   <- qnorm(q_level)
  z_q_t <- torch_tensor(z_q, dtype = torch_float(), device = Qq$device)

  mu <- Qq - sigma * z_q_t

  term1 <- torch_log(sigma)
  term2 <- (y - mu)$pow(2) / (2 * sigma$pow(2))

  (term1 + term2)$mean()
}
```

Listing 2: Função de perda NLL para o modelo Normal parametrizado pelo quantil $Q_q(x)$

Resposta para Ítem 1.2

Treinando o modelo proposto na Seção anterior com os dados do enunciado, obtemos os seguintes resultados:

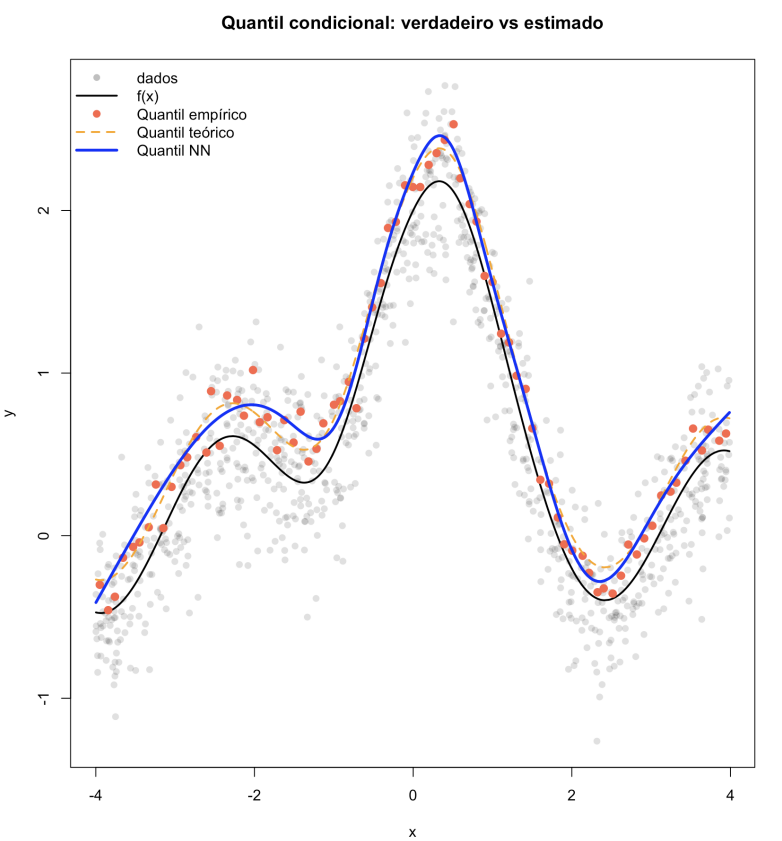

Figura 2: Quantil condicional verdadeiro vs. estimado.

A Figura 2 apresenta a comparação entre o quantil condicional verdadeiro, o quantil empírico observado nos dados e o quantil estimado pela rede neural. A curva aprendida (em azul) acompanha o comportamento oscilatório do quantil teórico. Isso evidencia que a rede foi capaz de aprender a dependência não linear de $Q_\tau(x)$.

A evolução da função de perda (NLL), cuja trajetória está ilustrada na Figura 3 mostra que as curvas de *train* e *test* permanecem próximas ao longo de todas as épocas, indicando ausência de sobreajuste e capacidade de generalização.

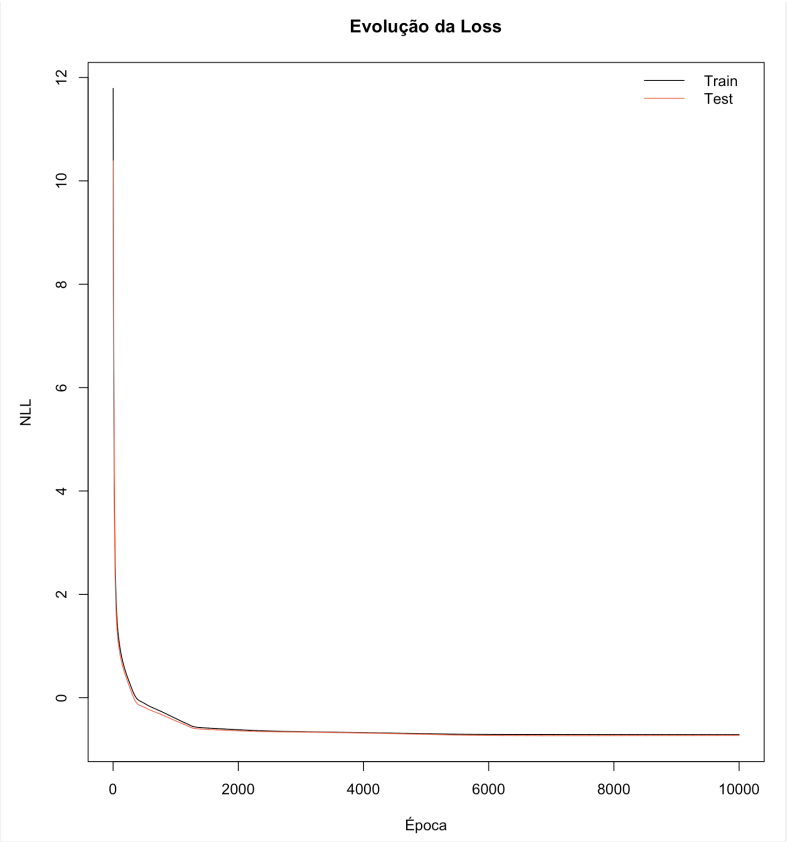

Figura 3: Evolução da NLL durante o treinamento.

```r
source("Config.R")
library(torch)

set.seed(config$seed)

n  <- config$n
x  <- sort(runif(n, config$x_min, config$x_max))

f_x <- 3/(3 + 2*abs(x)^3) +
  exp(-x^2) +
  cos(x)*sin(x)

sigma_true  <- rep(config$noise_sd,  length(x))
sigma2_true <- rep(config$noise_sd^2, length(x))

y <- f_x + rnorm(n, mean = 0, sd = sigma_true)

X_full <- torch_tensor(matrix(x, ncol = 1), dtype = torch_float())
Y_full <- torch_tensor(matrix(y, ncol = 1), dtype = torch_float())

n_train <- round(config$p_train * n)
idx <- sample(1:n)

idx_train <- idx[1:n_train]
idx_test  <- idx[(n_train + 1):n]

x_train <- x[idx_train]
y_train <- y[idx_train]

x_test  <- x[idx_test]
y_test  <- y[idx_test]

X_train <- torch_tensor(matrix(x_train, ncol = 1), dtype = torch_float())
Y_train <- torch_tensor(matrix(y_train, ncol = 1), dtype = torch_float())

X_test  <- torch_tensor(matrix(x_test,  ncol = 1), dtype = torch_float())
Y_test  <- torch_tensor(matrix(y_test, dtype = torch_float()))
```

Listing 3: Geração dos dados para o Item 1.2: modelo homoscedástico $y = f(x) + \sigma\varepsilon$.

```
library(torch)

treinar_normal_quantil <- function(model,
                                   q_level = config$q,
                                   epochs  = config$num_epochs,
                                   lr      = config$lr) {

  print_every <- config$print_every
  plot_every  <- config$plot_every
  num_bins    <- 80

  optimizer <- optim_adamw(
    model$parameters,
    lr = lr,
    weight_decay = 1e-3
  )

  loss_history_train <- numeric(epochs)
  loss_history_test  <- numeric(epochs)

  par(mfrow = c(1, 2))

  for (epoch in 1:epochs) {

    model$train()
    optimizer$zero_grad()

    out_train <- model(X_train)
    Qq_train  <- out_train$q
    sigma2_tr <- out_train$sigma2

    loss_train <- nll_normal_from_quantile(
      Qq      = Qq_train,
      sigma2  = sigma2_tr,
      y       = Y_train,
      q_level = q_level
    )

    loss_train$backward()
    optimizer$step()

    model$eval()
    out_test <- model(X_test)

    Qq_test   <- out_test$q
    sigma2_te <- out_test$sigma2

    loss_test <- nll_normal_from_quantile(
      Qq      = Qq_test,
      sigma2  = sigma2_te,
      y       = Y_test,
      q_level = q_level
    )

    loss_history_train[epoch] <- loss_train$item()
    loss_history_test[epoch]  <- loss_test$item()

    if (epoch %% print_every == 0) {
      cat("Época:", epoch,
          "- NLL train:", loss_train$item(),
          "- NLL test:",  loss_test$item(), "\n")
    }

    if (epoch %% plot_every == 0 || epoch == epochs) {

```

```
      out_full   <- model(X_full)
      Qq_full    <- as.numeric(out_full$q$squeeze())
      sigma2_hat <- as.numeric(out_full$sigma2$squeeze())
      sigma_hat  <- sqrt(sigma2_hat)

      z_q     <- qnorm(q_level)
      q_theo  <- f_x + z_q * sigma_true   # sigma_true é constante no item 1.2

      ylim_all <- range(y, f_x, q_theo, Qq_full)

      plot(x, y,
           pch  = 16,
           col  = rgb(0,0,0,0.15),
           ylim = ylim_all,
           xlab = "x", ylab = "y",
           main = "Quantil condicional: verdadeiro vs estimado")

      lines(x, f_x, col = "black", lwd = 2)

      plot_conditional_quantile(
        x, y, q_level,
        num_bins = num_bins,
        col = "tomato", pch = 19, cex = 1.1
      )

      lines(x, q_theo,  col = "orange", lwd = 2, lty = 2)
      lines(x, Qq_full, col = "blue",   lwd = 3)

      legend("topleft",
             legend = c("dados", "f(x)", "Quantil empírico",
                        "Quantil teórico", "Quantil NN"),
             col    = c(rgb(0,0,0,0.3), "black", "tomato", "orange", "blue"),
             lwd    = c(NA, 2, NA, 2, 3),
             lty    = c(NA, 1, NA, 2, 1),
             pch    = c(16, NA, 19, NA, NA),
             bty    = "n")

      plot(1:epoch, loss_history_train[1:epoch], type = "l",
           xlab = "Época", ylab = "NLL",
           ylim = range(c(loss_history_train[1:epoch],
                          loss_history_test[1:epoch])),
           main = "Evolução da Loss")

      lines(1:epoch, loss_history_test[1:epoch], col = "tomato")

      legend("topright",
             legend = c("Train", "Test"),
             col = c("black", "tomato"),
             lty = 1, bty = "n")

      Sys.sleep(0.05)
    }
  }

  invisible(list(
    model      = model,
    loss_train = loss_history_train,
    loss_test  = loss_history_test
  ))
}

plot_conditional_quantile <- function(x, y, q, num_bins = 80,
                                      col = "darkgreen", pch = 19, cex = 1.1) {

  breaks <- seq(min(x), max(x), length.out = num_bins)
  qx <- c(); qy <- c()

  for (i in 1:(length(breaks) - 1)) {
```

```r
    idx <- which(x >= breaks[i] & x < breaks[i + 1])
    if (length(idx) > 0) {
      qx <- c(qx, mean(x[idx]))
      qy <- c(qy, quantile(y[idx], q))
    }
  }

  points(qx, qy, col = col, pch = pch, cex = cex)
  invisible(list(x = qx, y = qy))
}
```

Listing 4: Função de treinamento para o modelo Normal-Quantil no caso homoscedástico (Item 1.2).

```r
config <- list(

  # ====================================================
  # Quantil alvo
  # ====================================================
  q = 0.75,

  # ====================================================
  # Hiperparâmetros de treino
  # ====================================================
  num_epochs  = 10000,
  lr          = 1e-3,
  print_every = 10L,
  plot_every  = 10L,

  # ====================================================
  # Arquitetura da rede
  # ====================================================
  input_dim    = 1L,
  hidden_mu    = 10L,   # 10 neurônios para sub-rede do quantil
  hidden_sigma = 10L,   # 10 neurônios para sub-rede da variância

  # ====================================================
  # Configuração dos dados
  # ====================================================
  seed    = 32,
  n       = 1000,
  x_min   = -4,
  x_max   =  4,
  p_train = 0.7,

  # ====================================================
  # Variância verdadeira usada na simulação
  # ====================================================
  noise_sd = 0.3,
)
```

Listing 5: Arquivo `Config.R` utilizado no Item 1.2.

Os códigos desde estudo estão disponibilizados em <http://academic-codex.github.io/MAE5911-Estatistica-e-Machine-Learning>.