```
# trainieren.ps1 (Version_0)
# 19.01.2026
# Trainieren des neuronalen Netzes
#   Aufruf: trainieren [Runs]   { Voreinstellung 1 }
# Runs ist die Anzahl an Lernschritten, die das Netz mit den zu
# lernenden EingabeZielPaaren durchführen soll
# ================================================================
param( [int]$Runs=1 )
Set-StrictMode -Version Latest

# Funktionen einbinden
# --------------------
. ./Tools/netzFktn.ps1
. ./Tools/alphaHolen.ps1

function main {
  param( [int]$Runs )

  # Parameter-Check
  # ---------------
  if($Runs -lt 1) { throw "Nur positive Zahlen" }
  Write-Host "Runs             : $Runs"

  # Netzkonfiguration aus matrizen.kfg lesen: Int[,]
  #    anzM: Anzahl an Matrizen
  # ------------------------------------------------
  $matrizen = leseMatKonfig
  $anzM = $matrizen.getLength(0)

  # Ausgabeaufbereitung
  # -------------------
  $s = ''
  foreach($i in 0..($anzM-1)) {
    $s += "($($matrizen[$i,0])x$($matrizen[$i,1]))+"
  }
  $s = $s -replace '\+$',''
  Write-Host "Gewichtsmatrizen : $s"

  # anzM Gewichtsmatrizen im Anfangszustand einlesen
  # Dateinamen sind gew0.mat[.start], ...
  # ------------------------------------------------
  foreach($m in 0..($anzM-1)) {
    cp ./Daten/gew$m.mat.start ./Daten/gew$m.mat -ea Stop
  }

  $gew = [object[]]::new($anzM)
  foreach($i in 0..($anzM-1)) {
    $gew[$i] = leseGewMat $i $matrizen[$i,0] $matrizen[$i,1]
  }

  # Lernschnittstelle
  # Belegt werden die Vektor-Arrays evn und zvn mit Lernpaaren,
  # deren Komponenten jeweils einen Eingabe- und den zugehörigen
  # Zielvektor enthalten. Die Paare kommen aus der Datei
  # lernPaare.txt. Die Funktion holeLernPaare gibt wenigstens
  # ein Lernpaar zurück.
  # ------------------------------------------------------------
  ($evn, $zvn) = holeLernPaare
  $anzP = $evn.getLength(0)
  Write-Host "Lernschnittstelle: $anzP Vektorpaare"
```

```
# Schrittweite für den Gradientenabstieg
# --------------------------------------
$alpha = holeAlpha
Write-Host "Schrittweite     : $alpha"

# Lernvorgang: Ausgabevektor av
# -----------------------------
$ev = [object[]]::new($anzM+1)
foreach($run in 1..$runs) {
  foreach($paar in 0..($anzP-1)) {
    $iv = $evn[$paar]                    # Eingabevektor
    $zv = $zvn[$paar]                    # Zielvektor

    # Eingabepropagierung
    # -------------------
    $ev[0] = $iv
    foreach($m in 0..$($anzM-1)) {
      $hlp = multMV $gew[$m] $ev[$m]
      $ev[$m+1] = sigmV $hlp
    }
    $av = $ev[$anzM]

    # Bestimmung des Ausgabefehlervektors afv
    # ---------------------------------------
    $dimZV = $zv.count
    $afv = [double[]]::new($dimZV)       # Ausgabefehler
    foreach($i in 0..($dimZV-1)) {
      $afv[$i] = $zv[$i]-$ev[$m+1][$i]
    }

    # Fehlerrückpropagierung (fv[0] wird nicht benötigt)
    # --------------------------------------------------
    $fv = [object[]]::new($anzM+1)       # Fehlervektoren
    $fv[$anzM] = $afv
    foreach($m in $anzM..1) {
      if($m -eq 1) { break }
      $trans = transpoM $gew[$m-1]
      $fv[$m-1] = multMV $trans $fv[$m]
    }

    # Gewichtsanpassungen
    # -------------------
    foreach($m in 0..($anzM-1)) {
      $dimEV = $matrizen[$m, 1]
      $dimAV = $matrizen[$m, 0]

      foreach($j in 0..($dimEV-1)) {
        foreach($k in 0..($dimAV-1)) {
          $gew[$m][$k,$j] += $alpha * $fv[$m+1][$k] *
                             $ev[$m+1][$k] *
                             (1 - $ev[$m+1][$k]) *
                             $ev[$m][$j]
        }
      }
    }
  } # paar
} # run

foreach($m in 0..($anzM-1)) {
```

```
    schreibeGewMat $gew[$m] "gew$m.mat"
  }
  Write-Host 'Lernvorgang      : beendet'
} # main

# ================
main $Runs
# ================
```