```
# netzFktn.ps1
# 27.01.2026
# Funktionen für das neuronale Netz

#    schreibeGewMat      Gewichtsmatrix in Datei schreiben
#    transpoM            Matrix transponieren
#    sigmV               Sigmoidfunktion auf Vektor anwenden
#    multMV              Matrix mit Vektor multiplizieren
#    erzeugeVektor       Aus ( String ) einen Vektor erzeugen
#    holeLernPaare       Datei lernPaare.txt auswerten
#    leseGewMat          Gewichtsmatrix aus Datei lesen
#    leseMatKonfig       Matrizenkonfiguration aus matKonfig.kfg
# ================================================================
function schreibeGewMat {
  # Ziel ./Daten/gew0.mat, ./Daten/gew1.mat usw.
  # -------------------------------------------
  param( [double[,]]$matrix, [string]$name )
  $path = "$pwd\Daten\$name"
  $zei = $matrix.GetLength(0)
  $spa = $matrix.GetLength(1)

  # dVals erzeugen
  # --------------
  $anz = $zei * $spa
  $dvals = [double[]]::new($anz)
  foreach($i in 0..($zei-1)) {
    foreach($k in 0..($spa-1)) {
      $dvals[$i*$spa+$k] = $matrix[$i,$k]
    }
  }

  # Datei schreiben
  # ---------------
  $bytes = foreach($val in $dVals) {
    [System.BitConverter]::GetBytes([double]$val)
  }
  [System.IO.File]::WriteAllBytes($path, $bytes)
} # schreibeGewMat

# ----------------------------------------------------------------
function transpoM {
  param( [double[,]]$mat )
  $dimZ = $mat.getLength(0)
  $dimS = $mat.getLength(1)
  $erg = [double[,]]::new($dimS, $dimZ)
  foreach($i in 0..($dimZ-1)) {
    foreach($k in 0..($dimS-1)) { $erg[$k,$i] = $mat[$i,$k] }
  }
  return (,$erg)
} # transpoM

# ----------------------------------------------------------------
function sigmV {
  # Sigmoid-Funktion auf einen Vektor anwenden
  # ------------------------------------------
  param( [double[]]$vek )
  $dimV = $vek.count
  $erg = [double[]]::new($dimV)
  foreach($i in 0..($dimV-1)) {
    $erg[$i] = 1 / (1 + [Math]::Exp(-$vek[$i]))
```

```
  }
  return (,$erg)
} #sigmV

# ---------------------------------------------------------------
function multMV {
  # Matrix mit Vektor multiplizieren
  # --------------------------------
  param( [double[,]]$mat, [double[]]$vek )
  $anzZ = $mat.getLength(0)
  $anzS = $mat.getLength(1)
  $dimV = $vek.count
  if($anzS -ne $dimV) { throw 'multMV: Dimensionsfehler!' }
  $erg = [double[]]::new($anzZ)      # Ergebnisvektor
  foreach($i in 0..($anzZ-1)) {
    $erg[$i] = 0.0
    foreach($k in 0..($anzS-1)) {
      $erg[$i] = $erg[$i] + $mat[$i,$k]*$vek[$k]
    }
  }
  return (,$erg)
} # multMV

# ---------------------------------------------------------------
function erzeugeVektor {
  # Vektor aus einem String ( ...) erzeugen
  # Der Vektor ist nicht leer, Typen und Werte stimmen
  # Rückgabe: Zugehöriger Double-Vektor
  # ---------------------------------------------------
  param( [string]$s )

  # Begrenzungen entfernen
  # ----------------------
  $s = $s.replace('(','')
  $s = $s.replace(')','')
  $s = $s.trim()

  # Komponentenzahl ermitteln
  # -------------------------
  $tok = @($s -split '\s+')
  $dimV = $tok.count

  # Komponenten in Double-Werte wandeln
  # -----------------------------------
  $dvals = @($tok | %{ [double]($_ -replace ',','.') })

  # Vektor erzeugen
  # ---------------
  $vektor = [double[]]::new($dimV)
  foreach($i in 0..($dimV-1)) { $vektor[$i] = $dvals[$i] }
  return (,$vektor)
} # erzeugeVektor

# ---------------------------------------------------------------
function holeLernPaare {
  # Die zu lernenden Eingabe-Ziel-Vektor-Paare aus der Textdatei
  # lernPaare.txt einlesen, die Konsistenz prüfen und die Arrays
  # evn (Eingaben) und zvn (Ziele) aufbauen.
  # Aufruf: (evn, zvn) = holeLernPaare
  # ---------------------------------------------------------------
```

```
# lernPaare.txt lesen
# -------------------
$datei = @(Get-Content ./Daten/lernPaare.txt -ea Stop)
$anzZ = $datei.count
if($anzZ -eq 0) { throw 'lernPaare.txt ist leer' }

# Zeilen mit Vektorpaaren ausblenden
# ----------------------------------
$pat = '^\s*\([^(]+\)\s*\([^(]+\)\s*'
$paare = @()
foreach($zeile in $datei) {
  if($zeile -match $pat) {
    $zeile = $zeile.trim()
    $paare += $zeile
  }
} # foreach(zeile)
$anzVP = $paare.count
if($anzVP -eq 0) { throw 'Keine Vektorpaare gefunden' }

# Gültige Paare ausblenden: Brauche Matrizen-Info
# -----------------------------------------------
$matrizen = leseMatKonfig
$gPaare = @()
$pcnt = 0
foreach($paar in $paare) {
  $pcnt++
  ($lVec,$rVec) = $paar -split '\)\s*\('
  # Untersuche lVec
  # ---------------
  $lVec = $lVec.trim()
  $lVec = $lVec -replace '^\(',''
  $lVec = $lVec.trim()

  # lVec: Komponentenzahl prüfen
  # ----------------------------
  $tok = $lVec -split '\s+'
  if($tok.count -ne $matrizen[0,1]) {
    Write-Host "holeLernPaare: lVec: Komponentenzahl-Fehler"
    continue
  }

  # lVec: Typen und Werte prüfen: Achtung: Culture deutsch
  # -----------------------------------------------------
  foreach($t in $tok) {
    if([double]::tryParse($t,[ref]$null)) {
      if(($t -le 0.0) -or ($t -ge 1.0)) {
        throw "holeLernPaare: lVec: Werte-Fehler $t"
      }
    } else { throw 'holeLernPaare: lVec: Typ-Fehler' }
  } # foreach(t)

  # Untersuche rVec
  # ---------------
  $rVec = $rVec.trim()
  $rVec = $rVec -replace '\)$',''
  $rVec = $rVec.trim()

  # rVec: Komponentenzahl prüfen
  # ----------------------------
```

```
    $tok = $rVec -split '\s+'
    if($tok.count -ne $matrizen[($matrizen.getLength(0)-1),0]) {
      Write-Host "holeLernPaare: rVec: Komponentenzahl-Fehler"
      Write-Host "holeLernPaare: rVec: next pair"
      continue
    }

    # rVec: Typen und Werte prüfen: Achtung: Culture deutsch
    # -----------------------------------------------------
    foreach($t in $tok) {
      if([double]::tryParse($t,[ref]$null)) {
        if(($t -le 0.0) -or ($t -ge 1.0)) {
          throw "holeLernPaare: rVec: Werte-Fehler $t"
        }
      } else { throw 'holeLernPaare: rVec: Typ-Fehler' }
    } # foreach(t)

    $gPaare += $paar
  } # foreach(paar)
  $anzP = $gPaare.count
  if($anzP -eq 0) { throw 'Keine gültigen Paare' }
    # Rückgabe der Arrays
    # -------------------
    $evn = [object[]]::new($anzP)
    $zvn = [object[]]::new($anzP)
    $i = -1
    foreach($zeile in $gPaare) {
      $i++
      ($lVec,$rVec) = $zeile -split '\)\s*\('
      $lVec = "$lVec)"
      $dlVec = erzeugeVektor $lVec
      $evn[$i] = $dLVec
      $rVec = "($rVec"
      $drVec = erzeugeVektor $rVec
      $zvn[$i] = $drVec
   }
   return ($evn,$zvn)
} # holeLernPaare

# ------------------------------------------------------------
function leseGewMat {
  # Gewichtsmatrix aus gleichnamiger *.mat-Datei lesen
  # Params : Matrixnummer, Zeilenzahl und Spaltenzahl
  # ------------------------------------------------------------
  param( [int]$m, [int]$r, [int]$c )   # rows und columns
  $path = "$pwd\Daten\gew$m.mat"
  $bytes = [System.IO.File]::ReadAllBytes($path)

  $vals = for($i = 0; $i -lt $bytes.count; $i += 8) {
    [System.BitConverter]::ToDouble($bytes, $i)
  }

  $mat = [double[,]]::new($r, $c)
  # lauf.Nr. = i * c + k
  # --------------------
  foreach($i in 0..($r-1)) {
    foreach($k in 0..($c-1)) {
        $mat[$i,$k] = $vals[$i*$c+$k]
    }
  }
```

```
  return (,$mat)
} # leseGewMat

# ------------------------------------------------------------
function leseMatKonfig {
  # Matrizenkonfiguration aus matrizen.kfg lesen
  # --------------------------------------------
  $datei = @(Get-Content ./Daten/matKonfig.kfg -ea Stop)
  $anzS = $datei.count
  $erg = [int[,]]::new($anzS, 2)
  foreach($i in 0..($anzS-1)) {
    ($rows,$columns) = $datei[$i] -split '\s'
    $erg[$i,0] = $rows
    $erg[$i,1] = $columns
  }
  return (,$erg)
} # leseMatKonfig
```