NeuralGas Vignette

Josh Taylor *Rice University* June 3, 2021

R Interface for NeuralGas Prototype Learning

Contents

Preface

`NeuralGas` is an R package for Neural Gas prototype learning. The main package features are:

- Exposure of both online and batch learning algorithms
- Fast and efficient C++ implementations of the above (based on Rcpp and RcppArmadillo)
- Optional parallel computation during training via RcppParallel

1 The Neural Gas Algorithm

Given N data vectors $X = \{x_i\} \in \mathbb{R}^d$, Neural Gas [**martinetz:gas**] finds, via competitive and cooperate learning, M **prototypes** $W = \{w_j\} \in \mathbb{R}^d$ by minimizing the following cost function:

$$NG_{Cost} = \sum_i \sum_j h_{ij} \times d(x_i, w_j). \tag{1}$$

Here, $d(x_i, w_j)$ is standard Euclidean distance from datum x_i to prototype w_j. The **neighborhood function** $h_{ij} = \exp(-k_{ij}/\lambda)$ controls the degree of cooperation among prototypes during learning. The term k_{ij} represents the ascending **rank** of the distance from x_i to w_j with respect to the distances to all other prototypes w_k, i.e.,

$$k_{ij} = |\{w_j | d(x_i, w_j) < d(x_i, w_k)\}|.$$

By convention, $k_{ij} \in \{0, 1, \ldots, M-1\}$. The user-supplied parameter λ controls the size of the cooperative neighborhood; $\lambda \to 0$ induces less cooperation among prototypes during learning while $\lambda \to \infty$ induces a greater degree of cooperation. λ should be annealed during training to ensure network convergence.

As the prototypes W induce a vector quantization (VQ) of X we present some common terms related to a VQ mapping and quality which will be referenced below:

- The **B**est **M**atching **U**nit of a datum x_i is the index of its closest prototype:

$$BMU_i = \arg\min_j d(x_i, w_j)$$

.

- The quantization error of x_i by its BMU is:

$$QE_i = d(x_i, w_{BMU_i})$$

.

- The **M**ean **Q**uantization **E**rror made by quantizing all of X by W is:

$$MQE = \frac{1}{N} \sum_i QE_i$$

.

- The **R**eceptive **F**ield of prototype w_j is the set of x_i mapped to w_j:

$$RF_j = \{x_i \,|\, BMU_i = j\}$$

- The size (cardinality) of a RF is denoted $|RF_j|$.

- The (normalized) Shannon entropy of the discretization of X by W is:

$$Entropy = \frac{1}{\log(M)} \sum_j \frac{|RF_j|}{N} \times \log\left(\frac{|RF_j|}{N}\right)$$

If each datum x_i has an associated label ℓ_i we can compute the additional quantities:

- The label of each RF (equivalently, prototype) is obtained by plurality vote of the data labels it contains:

$$WL_j = \arg\max_\ell p_{RFL_j}(\ell),$$

where $p_{RFL_j}(\cdot)$ is the empirical distribution of data labels in RF_j.

- The Purity of each RF π_j measures how much label confusion exists in the RF:

$$\pi_j = \max_\ell p_{RFL_j}(\ell),$$

Ideally, if the labels indicate well separated classes / clusters we would have $\pi_j = 1 \, \forall \, j$.

- The overall purity of the mapping is the weighted overall average of each RF's purity score, weighted by its size:

$$\Pi = \frac{1}{N} \sum_j |RF_j| \times \pi_j$$

- The Hellinger distance between the empirical categorical distributions of the data labels

($p_{XL}(\ell)$) and RF labels ($p_{RFL}(\ell)$) measures how well the RF labeling represents the data labeling:

$$Hellinger = \left(1 - \sum_{\ell} \sqrt{p_{XL}(\ell)\, p_{RFL}(\ell)}\right)^{\frac{1}{2}}$$

1.1 *Online Learning*

Online learning minimizes (**??**) over W via stochastic gradient descent according to the update rule:

$$w_j(s+1) = w_j(s) + \alpha h_{ij}(x_i - w_j)$$

where s indicates the current learning iteration and α is the user-supplied learning rate controlling the SGD step sizes. For online learning, s increments after presentation of a single datum to the network for training.

1.2 *Batch Learning*

Batch learning minimizes (**??**) over W via gradient descent with prototype updates at each epoch t defined by

$$w_j(t) = \frac{\sum_i h_{ij} x_i}{\sum_i h_{ij}}.$$

The batch update rule was shown in [**cottrell2006**] to be equivalent to minimization of (**??**) via Newton's method. This second order optimization was shown in [**cottrell2006**] to be much faster than online Neural Gas learning (i.e., requires fewer training iterations) with no impact on the quality of the resulting vector quantization. The denominator in the batch update rule supplants the need for a user-supplied learning rate. For batch learning, t increments after presentation of all N data vectors to the network (an epoch).

1.3 *Prototype Initialization*

Both batch and online learning update the prototypes W iteratively, requiring an initialization of W_0. Typically this occurs randomly (uniformly) in the range of X, which is the default behavior in the `NeuralGas` package. A user-supplied initialization is also accepted, as demonstrated in the Iris example User-Supplied Prototype Initializations.

See `NGWInitialization` for more information.

1.4 *Annealing*

λ and α (for online learning) should both be annealed during training to ensure its stability and convergence, which requires them to be non-increasing functions of the learning iteration s or epoch t. The `NeuralGas` package allows for both multiplicative and scheduled decay of both of these rates. Multiplicative decay is of the form

$$\lambda^M(t) = \lambda_0 \times \eta_\lambda^{t-1}, \quad \alpha^M(t) = \alpha_0 \times \eta_\alpha^{t-1}$$

where $\eta_* < 1$ is a decay factor. Scheduled decay allows step-wise decreases to the rates at a series of user-supplied iterations. For example, we could specify the following piecewise constant schedule for α:

$$\alpha^S(s) = \begin{cases} 0.9 & s \leq 5 \\ 0.5 & 5 < s \leq 10 \\ 0.1 & s > 10 \end{cases}$$

1.5 *Monitoring Learning*

After every training epoch t the following fitness measures of the NG network's vector quantization are computed and stored. Note that for online learning, we consider an "epoch" to constitute N individual training iterations. This **learning history** is stored in the `LearnHist` data frame of the list returned after convergence

- **Epoch** - the epoch for which measures are reported
- **alpha** - the prevailing learning rate (for online learning)
- **lambda** - the previaling neighborhood parameter
- **Cost** - the prevailing value of (**??**), divided by N (division is done to place Cost on a similar scale to MQE)
- **MQE** - Mean Quantization Error of all data
- **NhbEff** - the effect of the current value of λ on the network, defined as $Cost/MQE$. NhbEff is always ≥ 1, with values $= 1$ indicating no neighborhood effect, which occurs as $\lambda \to 0$.
- **delCost** - the relative absolute percent change in Cost from the previous epoch:

$$delCost(t) = \frac{Cost(t) - Cost(t-1)}{Cost(t-1)} \times 100$$

- **delMQE** - the relative absolute percentage change in the Mean Quantization Error from the previous epoch:
- **delBMU** - the proportion of data whose BMU has changed from the previous epoch:

$$delBMU(t) = \frac{1}{N} \sum_i I\left[BMU_t(x_i) \neq BMU_{t-1}(x_i)\right] \times 100$$

- **Entropy** - Normalized Shannon Entropy of the VQ mapping

If data labels are given, the learning history also reports

- **PurityWOA** - the Weighted Overall Average of each RF's purity score
- **WLUnq** - the number of unique prototype labels which, when compared to the number of unique data labels, can help diagnose how well the prototypes represent rare classes
- **WLHell** - the Hellinger distance between the categorical distributions of the data and prototype labels

See `?NGLearnHist` for more information.

1.6 Convergence

When training with the `NeuralGas` package, minimization of (**??**) proceeds until the first occurrence of either of the following events:

- The number of training epochs exceeds a user-supplied parameter `max_epochs`
- The VQ becomes stable, by which we mean `delBMU` drops below a user-supplied tolerance `tol_delBMU`, AND `delMQE` drops below a user-supplied tolerance `tol_delMQE`, FOR 3 CONSECUTIVE EPOCHS. Combined, these measures capture the stability of the VQ.

By default, `max_epochs = 999999`, `tol_delBMU = 1` and `tol_delMQE = 0.1` To train for a fixed number of epochs (regardless of the values of the stability measures), set `max_epochs` as desired and `tol_delBMU = tol_delMQE = 0`. Again, for online learning we consider an epoch to comprise N individual (stochastic) learning iterations s.

During training, early termination can be achieved by user interruption (typing CTRL-C in the R console window). Upon detecting early termination, both batch and online learning will return the current state of the network (existing values of the prototypes and any training history logged before user interruption).

See `?NGConvergence` for more information.

2 Example: Learning Iris

We use Fisher's iris data to demonstrate the learning functionality of Neural Gas. To begin, strip out the iris measurements as a matrix X and their labels as a vector L:

```
# Load library
library(NeuralGas)
```

```
# Store iris data as a matrix
X = as.matrix(iris[,1:4])
str(X)
##  num [1:150, 1:4] 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr [1:4] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width"

# Store iris labels
L = iris[,5]
str(L)
##  Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

2.1 Learning

We will demonstrate batch Neural Gas learning of iris via The `NGBatch` function (online training is almost identical, except the learning rate α for SGD is also a required input; see `?NGOnline`). Its arguments, with defaults, are

```
# View argument list of NGBatch function
args(NGBatch)
## function (X, W, lambda0 = 0.25 * nrow(W), lambda_decay = 0.9,
##     lambda_schedule = NULL, tol_delBMU = 1, tol_delMQE = 0.1,
##     max_epochs = 999999, XL = NULL, parallel = TRUE, verbose = TRUE)
## NULL
```

We have already prepared our data `X` and its label `L` above (if labels are unavailable simply do not supply a value for the argument `XL`). We will train 30 neural gas prototypes, initialized randomly (uniformly) in the range of `X` with a fixed seed. We invoke this behavior by passing `W = c(30, 123)` to `NGBatch` (first element of the vector sets the number of prototypes, last sets the random seed at which their are drawn).

For this learning we will leave use multiplicative annealing and default values of `lambda0` (25% of the number of prototypes), its decay factor (0.9), and the convergence tolerances (`tol_delBMU = 1`, `tol_delMQE = 1`, `max_epochs = -1`). Parallel processing is enabled by default (see `?RcppParallel::setThreadOptions` to change the number of threads used for parallel computation). The `verbose` flag controls whether the learning histories are printed to the console during training; we disable this here to save space. See `?NGBatch` for a more complete overview of this functionality.

```
# Batch iris training
```

```
ng.iris = NGBatch(X = X, W = c(30, 123), XL = L, verbose = F)
str(ng.iris)
## List of 12
##  $ W              : num [1:30, 1:4] 5.66 5.62 5.2 4.98 6.49 ...
##  $ epochs         : num 37
##  $ lambda_start   : num 7.5
##  $ lambda_end     : num 0.169
##  $ lambda_decay   : num 0.9
##  $ lambda_schedule: NULL
##  $ tol_delBMU     : num 1
##  $ tol_delMQE     : num 0.1
##  $ max_epochs     : int 999999
##  $ exec_time      : num 0.000148
##  $ converged      : logi TRUE
##  $ LearnHist      :'data.frame': 37 obs. of  12 variables:
##   ..$ Epoch    : num [1:37] 1 2 3 4 5 6 7 8 9 10 ...
##   ..$ lambda   : num [1:37] 7.5 6.75 6.08 5.47 4.92 ...
##   ..$ Cost     : num [1:37] 2.007 1.039 0.829 0.691 0.578 ...
##   ..$ MQE      : num [1:37] 0.1218 0.0749 0.0649 0.0622 0.0595 ...
##   ..$ NhbEff   : num [1:37] 16.48 13.87 12.76 11.11 9.72 ...
##   ..$ delCost  : num [1:37] NaN 48.3 20.2 16.6 16.3 ...
##   ..$ delMQE   : num [1:37] NaN 38.52 13.26 4.25 4.31 ...
##   ..$ delBMU   : num [1:37] NaN 62 32.7 18 20.7 ...
##   ..$ Entropy  : num [1:37] 0.591 0.754 0.812 0.857 0.872 ...
##   ..$ PurityWOA: num [1:37] 0.893 0.96 0.967 0.96 0.967 ...
##   ..$ WLUnq    : num [1:37] 3 3 3 3 3 3 3 3 3 3 ...
##   ..$ WLHell   : num [1:37] 0.1688 0.1507 0.0856 0.0871 0.0871 ...
```

Looking at the structure of the list returned by `NGBatch` we see:

- the matrix of learned prototypes in slot `W` (one prototype per row)
- the number of epochs required to reach convergence = 37
- the starting, ending and decay used for λ (notice $\lambda_t = \lambda_{t-1} \times 0.9$, which is the default multiplicative decay rate)
- the tolerances which controlled the convergence of this learning
- the execution time (in minutes)
- the `LearnHist` data frame, see `?NGLearnHist` for more details. We examine the learning history for the first and last 5 epochs below, formatted with the `kableExtra` package for easier reading:

```
# View a tidy version of LearnHist
library(kableExtra)
view_rows = c(1:5, (ng.iris$epochs-5+1):ng.iris$epochs)
kbl(ng.iris$LearnHist[view_rows,], row.names = F,
    digits = c(0, rep(3,9), 0, 3), booktabs = T) %>%
  kable_styling(latex_options = c('scale_down',"hold_position"))
```

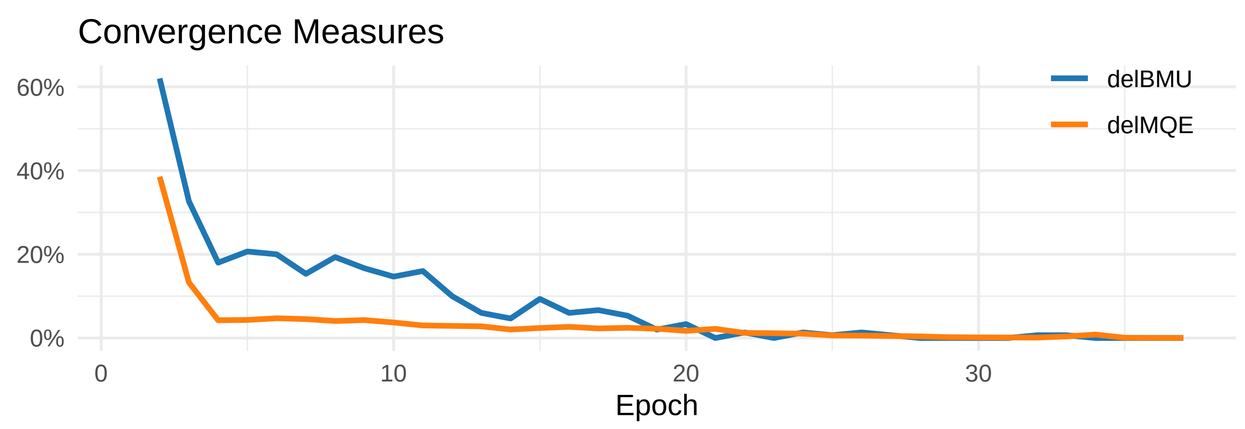
Epoch	lambda	Cost	MQE	NhbEff	delCost	delMQE	delBMU	Entropy	PurityWOA	WLUnq	WLHell
1	7.500	2.007	0.122	16.477	NaN	NaN	NaN	0.591	0.893	3	0.169
2	6.750	1.039	0.075	13.870	48.251	38.525	62.000	0.754	0.960	3	0.151
3	6.075	0.829	0.065	12.758	20.218	13.262	32.667	0.812	0.967	3	0.086
4	5.468	0.691	0.062	11.107	16.640	4.247	18.000	0.857	0.960	3	0.087
5	4.921	0.578	0.060	9.717	16.287	4.315	20.667	0.872	0.967	3	0.087
33	0.258	0.036	0.034	1.036	1.991	0.380	0.667	0.967	0.967	3	0.029
34	0.232	0.035	0.034	1.024	2.020	0.827	0.000	0.967	0.967	3	0.029
35	0.209	0.034	0.034	1.015	0.945	0.061	0.000	0.967	0.967	3	0.029
36	0.188	0.034	0.034	1.009	0.628	0.030	0.000	0.967	0.967	3	0.029
37	0.169	0.034	0.034	1.005	0.402	0.022	0.000	0.967	0.967	3	0.029

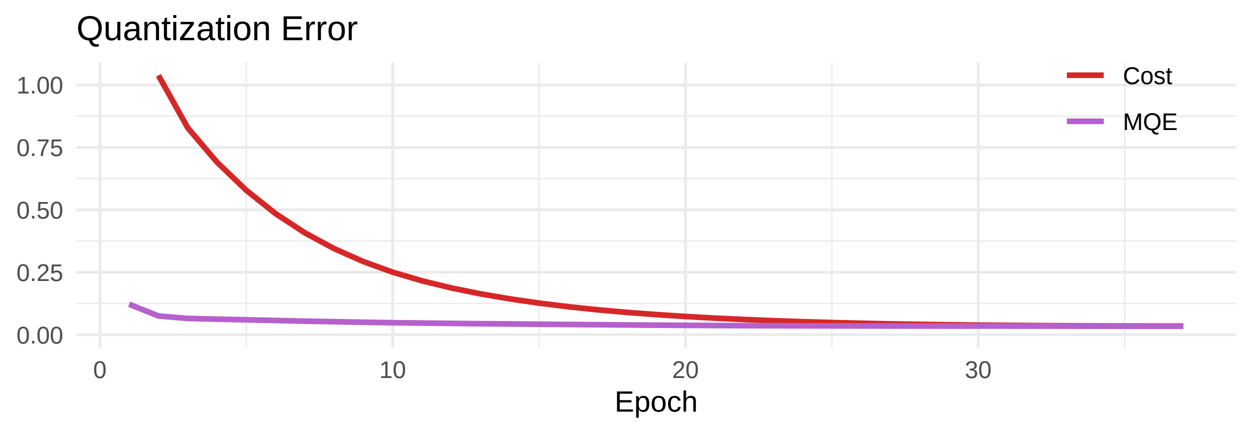
Learning appears successful, with all VQ measures improving as training progressed. In particular note the values of `delMQE` and `delBMU`, which dropped below their tolerances for the last three training epochs (which triggered convergence and terminated learning).

2.2 *Visualizations*

We can visualize the learning curves related to convergence with the `vis_NGLearnHist` function included in the package (which requires the `ggplot2` package, see `?vis_NGLearnHist`)

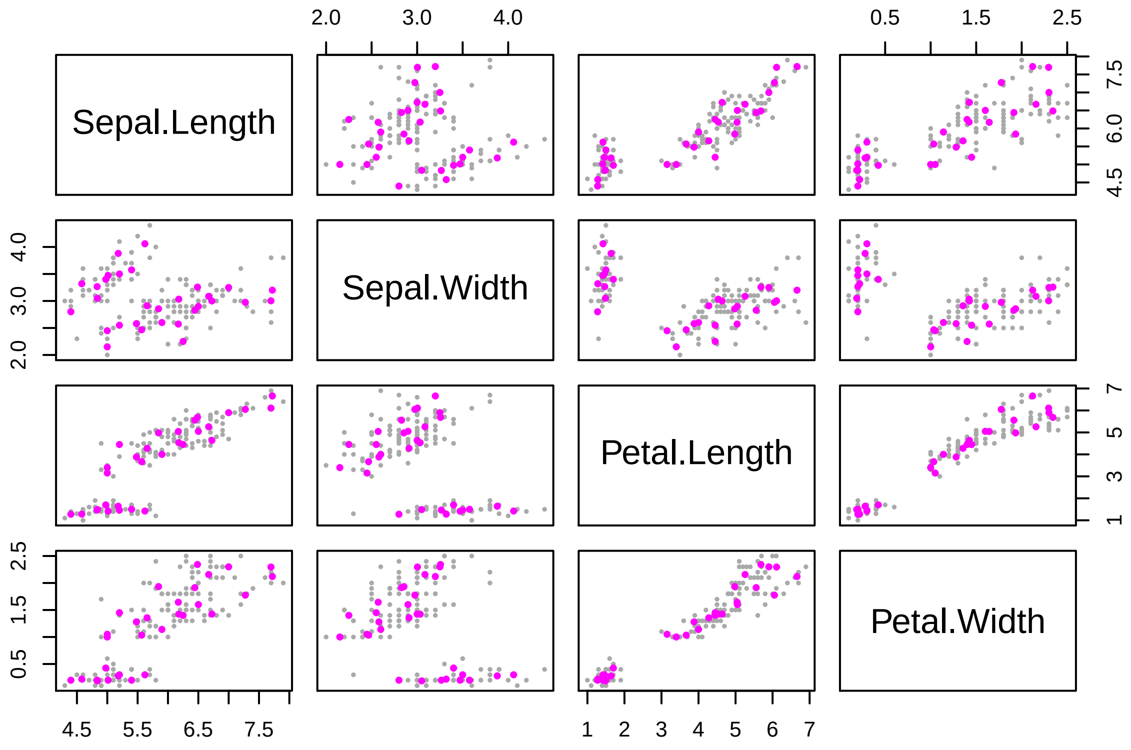
```
# Visualize the learning history
vis_NGLearnHist(ng.iris$LearnHist)
## Warning: Removed 2 row(s) containing missing values (geom_path).
## Warning: Removed 1 row(s) containing missing values (geom_path).
```





To inspect the actual prototype placement, we can look at pairs plots of data (gray) + prototypes (magenta) across the 4 dimensions of iris:

```
# View prototypes in data space
# First, combine the data & prototypes into one matrix for plotting
# Then specify different sizes & colors for prototype vs. data markers
pairs_data = rbind(X, ng.iris$W)
pairs_sizes = c(rep(0.5,nrow(X)), rep(0.75, nrow(ng.iris$W)))
pairs_colors = c(rep('darkgray',nrow(X)), rep('magenta', nrow(ng.iris$W)))
pairs(pairs_data, cex = pairs_sizes, col = pairs_colors, pch=16)
```



2.3 *Scheduled Annealing*

The above learning used multiplicative decay to anneal λ as training progressed. Alternatively, an annealing schedule may be supplied as a named vector whose elements give the annealed values, and whose names specify the epoch **through which** the respective values are effective. For example, we can specify the following annealing schedule

$$\lambda(t) = \begin{cases} 7 & t \leq 1 \\ 5 & 1 < t \leq 3 \\ 3 & 3 < t \leq 5 \\ 1 & t > 5 \end{cases}$$

as a named vector `lambda_schedule`:

```
# Define a named vector for scheduled annealing
lambda_schedule = c(7, 5, 3, 1)  # the desired lambda values
schedule_epochs = c(1, 3, 5, 6)  # the epochs after which lambda changes
names(lambda_schedule) = schedule_epochs
```

The last λ value in the schedule will be repeated as needed if training extends beyond `max(schedule_epochs)` (so in this case, $\lambda = 1$ is used for any training epoch beyond 5). Once built (as shown above), this schedule can be passed to the `lambda_schedule` argument of the `NGBatch` (or `NGOnline`) function.

If using scheduled annealing, do not supply any values to the multliplicative annealing arguments (i.e., `lambda0`, `lambda_decay`, `alpha0`, `alpha_decay`). Checks will be performed to ensure the scheduling occurs at integers, and that `schedule_epochs` were supplied in non-decreasing order. The result of learning with our schedule specified above is:

```
# Batch learning with scheduled annealing
ng.iris.sched = NGBatch(X = X, W = c(30, 123), lambda_schedule = lambda_schedule,
                        XL = L, verbose = F)

# View the LearnHist
kbl(ng.iris.sched$LearnHist, row.names = F,
    digits = c(0, rep(3,9), 0, 3), booktabs = T) %>%
  kable_styling(latex_options = c('scale_down',"hold_position"))
```

Epoch	lambda	Cost	MQE	NhbEff	delCost	delMQE	delBMU	Entropy	PurityWOA	WLUnq	WLHell
1	7	1.848	0.122	15.176	NaN	NaN	NaN	0.591	0.893	3	0.169
2	5	0.668	0.072	9.331	63.862	41.228	66.000	0.775	0.967	3	0.069
3	5	0.595	0.059	10.132	10.990	18.023	38.000	0.846	0.973	3	0.047
4	3	0.280	0.057	4.925	52.992	3.300	19.333	0.891	0.973	3	0.054
5	3	0.266	0.050	5.367	4.921	12.745	33.333	0.916	0.973	3	0.018
6	1	0.083	0.047	1.757	68.918	5.082	21.333	0.945	0.973	3	0.017
7	1	0.074	0.039	1.886	9.864	16.026	19.333	0.968	0.973	3	0.029
8	1	0.073	0.038	1.937	2.112	4.652	8.000	0.970	0.973	3	0.029
9	1	0.072	0.037	1.945	0.629	1.046	4.667	0.972	0.973	3	0.029
10	1	0.072	0.037	1.955	0.471	1.012	6.000	0.975	0.973	3	0.029
11	1	0.072	0.036	1.971	0.411	1.188	5.333	0.976	0.973	3	0.029
12	1	0.071	0.036	1.986	0.409	1.169	2.000	0.976	0.973	3	0.029
13	1	0.071	0.036	1.992	0.105	0.397	0.667	0.975	0.973	3	0.029
14	1	0.071	0.036	1.993	0.048	0.119	0.667	0.974	0.973	3	0.029
15	1	0.071	0.036	1.995	0.048	0.137	0.000	0.974	0.973	3	0.029
16	1	0.071	0.036	1.993	0.035	0.050	0.667	0.975	0.980	3	0.029
17	1	0.071	0.036	1.992	0.012	0.079	0.667	0.975	0.980	3	0.029
18	1	0.071	0.036	1.993	0.043	0.100	0.000	0.975	0.980	3	0.029
19	1	0.071	0.036	1.992	0.021	0.007	0.667	0.975	0.980	3	0.029
20	1	0.071	0.036	1.992	0.005	0.004	0.667	0.974	0.980	3	0.029
21	1	0.071	0.036	1.994	0.048	0.129	0.000	0.974	0.980	3	0.029
22	1	0.071	0.036	1.994	0.015	0.006	0.000	0.974	0.980	3	0.029
23	1	0.071	0.036	1.994	0.006	0.012	0.000	0.974	0.980	3	0.029
24	1	0.071	0.036	1.994	0.010	0.027	0.000	0.974	0.980	3	0.029

We can see convergence occurred sooner (in 24 epochs vs. 37) with minimal impact on final MQE (0.036 vs. 0.034).

For online learning the schedule should be set in terms of learning iterations s (not epochs t). Additionally, if using online learning with scheduled annealing, both α and λ must be scheduled (i.e., either both are scheduled, or neither). See `NGOnline` for details.

2.4 *User-Supplied Prototype Initializations*

Above we set the initial value of NG prototypes as random vectors selected uniformaly in the range of our data. Alternatively, a user can supply a fixed matrix of prototypes to initialize. For example, initializing at randomly selected vectors of X is common; we show how to achieve this below:

```
## Sample 30 vectors X to set initial prototypes
set.seed(123)
use_these = sample.int(n = nrow(X), size = 30, replace = F)
W0 = X[use_these,]

## Train with default lambda and annealing
ng.iris = NGBatch(X = X, W = W0, XL = L, verbose = F)
str(ng.iris)
## List of 12
##  $ W              : num [1:30, 1:4] 4.82 5.34 7.69 4.4 6.76 ...
##  $ epochs         : num 35
##  $ lambda_start   : num 7.5
##  $ lambda_end     : num 0.209
##  $ lambda_decay   : num 0.9
##  $ lambda_schedule: NULL
##  $ tol_delBMU     : num 1
##  $ tol_delMQE     : num 0.1
##  $ max_epochs     : int 999999
##  $ exec_time      : num 8.54e-05
##  $ converged      : logi TRUE
##  $ LearnHist      :'data.frame': 35 obs. of  12 variables:
##   ..$ Epoch    : num [1:35] 1 2 3 4 5 6 7 8 9 10 ...
##   ..$ lambda   : num [1:35] 7.5 6.75 6.08 5.47 4.92 ...
##   ..$ Cost     : num [1:35] 1.226 0.975 0.819 0.688 0.577 ...
##   ..$ MQE      : num [1:35] 0.0395 0.0718 0.0649 0.0624 0.0597 ...
##   ..$ NhbEff   : num [1:35] 31.02 13.59 12.63 11.03 9.67 ...
##   ..$ delCost  : num [1:35] NaN 20.4 16 16 16.1 ...
##   ..$ delMQE   : num [1:35] NaN 81.59 9.59 3.89 4.26 ...
##   ..$ delBMU   : num [1:35] NaN 77.3 30 19.3 20.7 ...
##   ..$ Entropy  : num [1:35] 0.956 0.826 0.847 0.873 0.89 ...
##   ..$ PurityWOA: num [1:35] 0.967 0.947 0.96 0.967 0.973 ...
##   ..$ WLUnq    : num [1:35] 3 3 3 3 3 3 3 3 3 3 ...
##   ..$ WLHell   : num [1:35] 0.0289 0.1609 0.1282 0.0776 0.0681 ...
```