A Library for Locally Weighted Projection Regression — Supplementary Documentation —

Stefan Klanke and Sethu Vijayakumar

June 23, 2008

Contents

1 Introduction

Locally weighted projection regression (LWPR) is an algorithm that achieves nonlinear function approximation in high dimensional spaces even in the presence of redundant and irrelevant input dimensions [1]. At its core, it uses locally linear models

$$\psi_k(\mathbf{x}) = \beta_0 + \sum_{i=1}^{R} \beta_i s_i \tag{1}$$

spanned by a small number of univariate regressions in selected directions in input space, and it employs weighted partial least squares (PLS) to detect those directions and the corresponding slopes β_i. This nonparametric local learning system learns rapidly with second order learning methods based on incremental training. LWPR uses statistically sound stochastic cross validation for automatically updating the distance metrics of each local model (the size of its "receptive field"). In particular, it minimises

$$J = \frac{1}{\sum_{i=1}^{M} w_i} \sum_{i=1}^{M} \frac{w_i (y_i - \hat{y}_i)^2}{(1 - w_i \mathbf{s}_i^T \mathbf{P}_s \mathbf{s}_i)^2} + \frac{\gamma}{N} \sum_{i,j=1}^{N} D_{ij}^2 \tag{2}$$

by stochastic gradient descent. It is therefore necessary to provide LWPR with a large enough number of training data, so that the algorithm can properly detect the local dimensionality (number R of PLS projections) and the scale on which the regression function is locally linear. If the training set is small, the samples need to be presented to LWPR multiple times in random order.

2 When to use LWPR (and when not)

LWPR is particularly suited for the following regression problems:

- The function to be learnt is **non-linear**. Otherwise having multiple local models is a waste of resources, and you should rather use ordinary linear regression, or partial least squares (PLS) for the case of high-dimensional or irrelevant inputs.
- There are **large amounts of training data**. If you desire good generalization from only relatively few samples (say, less than 2000), you are probably better off with Gaussian Processes (GP).
- Your application requires **incremental, online** training. If you can afford to collect the data beforehand, and the time required for batch learning is not critical, LWPR loses its edge against SVM regression, or (Sparse) GP regression. When compared to global function approximators like multi-layer neural networks, LWPR has the tremendous advantage that its local models learn *independently* and without interference.
- The input space is **high-dimensional**, but the data lies on low-dimensional manifolds. LWPR places local models only where they are needed, and can detect the local dimensionality through PLS, yielding robust estimates of the regression coefficients. The latter feature sets off LWPR against previous (but otherwise similar) algorithms as Receptive Field Weighted Regression.
- The model may require **adaptation**, since the target mapping may **change over time**. This suits LWPR very well because a built-in forgetting factor can be tuned to match the expected time scale at which such changes occur. The adaptation then usually happens quite fast, since the overall placement of receptive fields, their size, and the local PLS directions of a well-trained model can often be kept, while the regression parameters β get re-adjusted.

3 How to use LWPR

Despite all the nice properties and the efficiency of LWPR, applying the algorithm to a complex learning task – in a way that a good learning performance is achieved – can be quite involved. While we put a lot of effort into selecting reasonable defaults for all the learning parameters, there are still some parameters that need to be tuned to the task at hand. Probably the most important parameter is init_D, the distance metric assigned initially to newly created local models, which is also tightly connected to the normalisation of the input data (norm_in). For tuning this parameter, you should consider the following *rule of thumb*:

- Collect a batch set of training data. This does not need to cover the complete range of data you expect, but it should be a "typical" subset. Split this into a training and a test set.
- Set the input normalization (norm_in) to the expected range of the input data – do *not* "blindly" normalize by the standard deviation in each direction of the input space.
- Fit an LWPR model to the training set, starting with *fixed* distance metrics (update_D=0) and rather wide (spherical) distance metrics, that is, set init_D to $r\mathbf{I}$, where r is small (e.g., 0.05).
- Check the prediction performance on the test set, and retrain the model with an increased r (yielding smaller receptive fields) until you get a satisfying performance.
- Once you have a good init_D, enable distance metric adaptation (set updata_D=1). Optionally, tune the learning rate init_alpha. For this please see the description of that parameter on page 4.
- Hint: If you know that the target function is linear in a certain subset of the inputs[1], you can try keeping the corresponding diagonal elements of init_D at very small values.

4 Top-level elements of the LWPR model

4.1 Basic elements

nIn (integer)

Number of input dimensions. Must be specified when creating a new model.

nOut (integer)

Number of output dimensions. Must be specified when creating a new model.

name (string)

Optional description string.

Tune! norm_in (double vector, N)

Component-wise input normalization factors useful for making the inputs dimensionless. In general, it is dangerous to divide each dimension by its variance without considering the physical properties of the input values since some input dimensions may be actually moving very little relative to its range. Ideally, one should know the range of possible inputs in each dimension and try to normalize each input by that. Note also that distances between receptive fields and input data are calculated *after* the normalization, that is, norm_in and init_D are closely connected tuning parameters.

norm_out (double vector, N_{out})

Component-wise output normalization factors useful for making the outputs dimensionless. Since in the current implementation of LWPR all output dimensions are learned separately, tuning of this parameter has only little effect.

kernel (string or enum'ed constants, either Gaussian (default) or BiSquare)

This field determines which locality kernel should be used as the receptive fields' activation function. The Bisquare kernel might yield some gaim in computational efficiency, but as long as non-zero cutoff values are use the difference should be negligible.

[1]this is often the case for learning forward dynamics of torque-controlled robot arms

4.2 Elements related to distance metrics

diag_only (boolean flag, default = True)

If this flag is True, the distance metric and all related quantites (e.g., memory terms, learning rates, ...) are treated as diagonal matrices. This usually yields a big speed-up especially in higher dimensions. However, using diagonal distance metrics might not be sufficient for complicated learning problems. As a general rule, try learning with diag_only=1 first.

update_D (boolean flag, default = True)

This flag determines whether the distances metrics of receptive fields are updated or kept fixed. The latter is faster, but learning performance depends completely on a suitable choice of init_D.

Tune! init_alpha (double matrix, $N \times N$)

Component-wise distance metric learning rate initialization for gradient descent. If you see instability in convergence, you have too large a learning rate. If the MSE is decreasing but the convergence is slow, you might try increasing the learning rate. In theory, using meta learning meta=1 should alleviate the need to tune this parameter.

meta (boolean flag, default = False)

If this flag is True, updates to the distances metrics use second order adaptation of learning rates by the Incremental Delta Bar Delta (IDBD) algorithm. This is slightly more expensive, but usually results in a better (at least faster) adaptation of the local models to the data.

meta_rate (double)

Second order learning rate, i.e. the rate that governs the step size of changes to the distance metrics. Default value = 250.0

Tune! penalty (double)

Multiplication factor γ for the regularization penalty term in the optimization functional (2). Larger values enforce smaller distance metrics, corresponding too wider receptive fields, which in turn implies smoother functions.

Tune! init_D (double matrix, $N \times N$)

Initial distance metric (must be symmetric positive definite) assigned to newly created receptive fields. The distance metric automatically adjusts itself if the distance metric learning is enabled update_D=1. However, convergence properties and speed are strongly dependent on a good initialization. What can bad choices do:

- Too small value of D (large receptive fields) can lead to local minima and delay convergence
- Too large value of D (small receptive fields) can lead to allocation of too many receptive fields and overfitting

Theoretically, the learning mechanism takes care of thes problems, but there is nothing like a good initialization to make things easier for the algorithm! One way of guessing a good initialization is to guess the Hessian of the function being approximated and put a conservatively big initialization of receptive field based on the curvature.

init_M (double matrix, $N \times N$)

Cholesky factors of initial distance metric init_D. This is automatically calculated when you modify init_D. You should not tweak this field directly.

4.3 Elements controlling the local regression

w_gen (double, default = 0.1)

Weight activation threshold. A new local model is generated if no existing model elicits response (activation) greater than w_gen for a training sample.

w_prune (double, default = 0.9)

If a training sample elicits responses greater than w_prune from 2 local models, then the one with the smaller receptive field, that is, the one with the larger Frobenius norm of its distance metric, is pruned.

init_S2 (double, default = 10^{-10})

Initial value for the covariance computation of the data (receptive field element SSs2), useful to handle the case when no data has been seen so far.

add_threshold (double, default = 0.5)

The mean squared error of the current regression dimension is compared against the previous one. Only if the ratio of $\frac{nMSE[R]}{nMSE[R-1]}$ < add_threshold, a new regression direction is added. The criterion is used in conjunction with some other checks to ensure that the decision is based on enough data support. See also Section 6.2.

init_lambda (double, default = 0.999)

Initial forgetting factor.

final_lambda (double, default = 0.99999)

Final forgetting factor.

tau_lambda (double, default = 0.9999)

Annealing constant for the forgetting factor.

4.4 Read-only elements for inspection

n_data (integer)

Number of samples the LWPR model has been trained on so far.

mean_x (double vector, $N \times N$)

Mean of all input training data.

var_x (double vector, $N \times N$)

Variance of all input training data.

5 Elements of a Receptive Field

The following elements are ordered alphabetically. You should not modify any of them by hand, but rather treat them as read-only variables for inspection purposes.

alpha (double matrix, $N \times N$)

Learning rates for updates to M. When using meta learning, this field itself gets updated using Incremental Delta Bar Delta (IDBD). Initially it is set to model.init_alpha.

b (double matrix, $N \times N$)

Memory terms for 2nd order updates to M, as part of the IDBD algorithm.

beta (double vector, R)

PLS regression coefficients as used in (1). These get calculated from sufficient statistics SSYres.

beta_0 (double)

Intercept (or offset) of the local linear model, corresponding to β_0 in (1). This parameter is simply estimated as the weighted mean of all output data the receptive fields is trained on.

c (double vector, N)

Center of the receptive field. This vector is set to the current training input $\mathbf{x}$ when a new receptive field is created. It is not modified afterwards (independence property of local learning).

D (double matrix, $N \times N$)

Distance metric of the receptive field. Gets updated through its Cholesky decomposition M. When a new receptive field is created, its distance metric is set either to the model.init_D, or the distance metric of the closest existing receptive field.

H (double vector, R)

Sufficient statistics for distance metric updates.

h (double matrix, $N \times N$)

Sufficient statistics for 2nd order distance metric updates, part of IDBD.

lambda (double vector, R)

Forgetting factor for each PLS direction.

M (double matrix, $N \times N$)

Cholesky decomposition of the distance metric D. If model.update_D is non-zero, then this field gets updated by gradient descent, or – if also meta learning is active (model.meta) – by the Incremental Delta Bar Delta algorithm.

mean_x (double vector, N)

Weighted mean of the training data this RF has seen.

n_data (double vector, R)

Weighted number of training data each PLS direction has seen.

P (double matrix, $N \times R$)

PLS projection axes (R vectors of length N), gets calculated from SSXres.

r (double vector, R)

Sufficient statistics for distance metric updates.

s (double vector, R)

Current PLS loadings.

SSp (double)

Sufficient statistics used for calculating confidence bounds.

SSs2 (double vector, R)

Accumulated statistics (covariance) of PLS factor loadings s.

SSXres (double matrix, $N \times R$)

Sufficient statistics for the PLS projection axes P.

SSYres (double matrix, $N \times R$)

Contains sufficient statistics for the PLS regression axes U.

sum_e2 (double)

This field holds the accumulated prediction error on the training data, in its non-cross-validated form.

sum_e_cv2 (double vector, R)

Accumulated CV-error on training data.

sum_w (double vector, R)

Accumulated activation w per PLS direction.

SXresYres (double matrix, $N \times R$)

Contains sufficient statistics for the PLS regression axes U.

trustworthy (boolean flag)

This field reports whether a receptive field has "seen" enough training data so that predictions from it can be trusted. In the current implementation the correspong threshold for the weighted number of data (n_data) is set to $2N$, where N is the input dimensionality.

U (double matrix, $N \times R$)

PLS regression axes (R vectors of length N), gets calculated from SXresYres by normalizing the columns to unit-length.

var_x (double vector, N)

Weighted variance of the training data this RF has seen.

6 Heuristics and numerical safety measures

This section lists the strategies and heuristics employed to make the LWPR algorithm more robust against both ill-conditioned learnings tasks and numerical problems. The code snippets below are taken from the Matlab implementation.

6.1 Initialisation of receptive fields

- `SSs2` is set to the initial variance given by `init_S2` ($= 10^{-10}$ by default).
- `sum_w` is set to 10^{-10}.
- `n_data` is set to 10^{-10}.

6.2 Check whether to add a new PLS projection

```
mse_n_reg   = rf.sum_e_cv2(n_reg)  / rf.sum_w(n_reg) + 1.e-10;
mse_n_reg_1 = rf.sum_e_cv2(n_reg-1)/ rf.sum_w(n_reg-1) + 1.e-10;
```

The above lines calculate a mean squared error (cross-validated) statistics for the two latest PLS projections. The error is weighted by the accumulated activations. A small value is added to prevent division by zero in the following `if`-clause:

```
if (mse_n_reg/mse_n_reg_1 < model.add_threshold & ...
   rf.n_data(n_reg)/rf.n_data(1) > 0.99 & ...
   rf.n_data(n_reg)*(1-rf.lambda(n_reg)) > 0.5),
```

$\Rightarrow$ A new PLS projection is added

- if the quotient of the error statistics is below a threshold, indicating that the latest PLS projection significantly contributes to the prediction accuracy, *and*
- if the latest PLS projection has already seen 99% of the data the receptive field has seen *and*
- if the latest PLS projection has seen sufficient data, as determined by $n > \frac{0.5}{1-\lambda}$.

6.3 Adding a new PLS projection

- `SSs2` is set to an initial variance of `init_S2` ($= 10^{-10}$ by default).
- `sum_w` is set to 10^{-10}.
- `n_data` is set to 0. This is in contrast to the initialisation of a new RF (see above), but still ok, since only the first element of `n_data` is used in divisions.

6.4 Updates to the regression parameters

- Sufficient statistics for the PLS projections `U`, i.e. `SXresYres`, get multiplied by `lambda_slow` $=$ $1.0-(1.0-$`lambda`$)/10$, but sufficient statistics for the residuals `SXres` and `SYres` are multiplied by the normal `lambda`.
- Sufficient statistics for confidence bounds (`SSp`) are updated with the squared activation w^2 instead of w.
- Accumulated errors and CV-errors are only updated if "sufficient data" has been seen:

  ```
  if rf.n_data(1) > 0.1./(1.-rf.lambda(1))
     rf.sum_e_cv2 = rf.sum_e_cv2.*rf.lambda + w*e_cv.^2;
     rf.sum_e2    = rf.sum_e2*rf.lambda(end) + w*e^2;
  end
  ```

  Note that the condition is different from the check for a new PLS projection. `sum_e2` is only used for calculating a transient multiplier, which acts as an additional learning rate for distance metric updates.

6.5 Distance metric updates

- An update is only performed if the RF has seen sufficient data. The condition is the same as for the (CV-)error accumulation (see above).
- A `transient_multiplier` gets computed by $\left(\texttt{sum\_e2}/(\texttt{sum\_e\_cv2(end)} + 10^{-10})\right)^4$. This serves as an additional factor in the learning rate.
- In meta (second order) updates, the parameter `b` is clipped to $[-10; 10]$, and updates to `b` are clipped to $[-0.1; 0.1]$.
- Updates to $\mathbf{M}$, that is, elements of $\Delta\mathbf{M}$, are compared against $0.1\max(\mathbf{M})$. If any element Δm_{ij} is larger, the learning rate α_{ij} is divided by 2, and **no** update is performed for this element.

- The memory term `r` gets updated with a term proportional to the squared activation w^2 instead of w.

6.6 Predictions

- Depending on the parameter `cutoff`, only receptive fields with sufficient activation contribute to the prediction. This is mainly for speeding up predictions.
- Furthermore, only `trustworthy` receptive fields contribute, that is, receptive fields having seen more than $2N$ training samples.

6.7 Updating receptive fields

- Only receptive fields with sufficient activation, $w > 0.001$ are updated.

6.8 Adding and pruning receptive fields

- Creation of a new RF depends on the threshold parameter `w_gen`. If the currently most active RF is `trustworthy` and has "reasonable" activation, it is used as a template.

```
if (wv(3) <= model.w_gen)
   if (wv(3) > 0.1*model.w_gen & sub.rfs(iv(3)).trustworthy)
      sub.rfs(numrfs+1) = lwpr_x_init_rf(model, sub.rfs(iv(3)), xn, yn);
   else
      sub.rfs(numrfs+1) = lwpr_x_init_rf(model, [], xn, yn);
   end
end
```

- If two receptive fields have activation $w >$ `w_prune`, one of them is removed. The selection criterion in this case is the width of the kernel, which in this implementation is measured by the trace (= sum of eigenvalues) of `D`.

References

[1] S. Vijayakumar, A. D'Souza, and S. Schaal. Incremental online learning in high dimensions. *Neural Computation*, 17:2602–2634, 2005.