Linear Solutions for an Extreme Learning Machine

Pseudoinverse, least squares, and ridge regularization

1 ELM notation

Consider an Extreme Learning Machine (ELM) with one scalar input, m hidden neurons, and n known data points (x_j, y_j). The parameters of the hidden layer are randomly selected and then held fixed. Its hidden-layer output is

$$M_{ij} = \sigma(w_i^{(1)} x_j + b_i^{(1)}), \qquad i = 1, \ldots, m, \quad j = 1, \ldots, n. \tag{1}$$

We use the convention that hidden neurons are rows and data points are columns:

$$M \in \mathbb{R}^{m \times n}. \tag{2}$$

This retains the familiar neural-network form

$$M = \sigma\left(\boldsymbol{w}^{(1)} \boldsymbol{x} + \boldsymbol{b}^{(1)}\right). \tag{3}$$

In ELM, the "model matrix" M is **fixed** !

The output of the network at the training points is

$$\widehat{y}_j = \sum_{i=1}^{m} w_i^{(2)} M_{ij}. \tag{4}$$

Using column vectors $\boldsymbol{w} \equiv \boldsymbol{w}^{(2)} \in \mathbb{R}^m$ and $\boldsymbol{y} \in \mathbb{R}^n$, this is

$$\boxed{\widehat{\boldsymbol{y}} = M^T \boldsymbol{w}.} \tag{5}$$

Equivalently, if $\boldsymbol{y}^T$ and $\boldsymbol{w}^T$ are regarded as row vectors,

$$\widehat{\boldsymbol{y}}^{,T} = \boldsymbol{w}^T M. \tag{6}$$

The relevant shapes are

Quantity	Meaning	Shape
M	hidden-layer matrix	$m \times n$
M^T	least-squares design matrix	$n \times m$
$\boldsymbol{w}$	output weights	m
$\boldsymbol{y}$	observed outputs	n

Training the output layer is therefore a linear problem. The three solution methods discussed below differ in how this problem is formulated or regularized; the random hidden layer is the same in all cases.

2 Ordinary least squares

When an exact solution of $M^T \boldsymbol{w} = \boldsymbol{y}$ does not exist, the natural choice is to minimize the sum of squared residuals,

$$L(\boldsymbol{w}) = \left\| M^T \boldsymbol{w} - \boldsymbol{y} \right\|_2^2. \tag{7}$$

Expanding the objective gives

$$\begin{aligned} L(\boldsymbol{w}) &= (M^T \boldsymbol{w} - \boldsymbol{y})^T (M^T \boldsymbol{w} - \boldsymbol{y}) && (8) \\ &= \boldsymbol{w}^T M M^T \boldsymbol{w} - 2\boldsymbol{y}^T M^T \boldsymbol{w} + \boldsymbol{y}^T \boldsymbol{y}. && (9) \end{aligned}$$

Its gradient is

$$\nabla_{\boldsymbol{w}} L = 2MM^T \boldsymbol{w} - 2M\boldsymbol{y}. \tag{10}$$

At a stationary point,

$$\boxed{MM^T \boldsymbol{w} = M\boldsymbol{y}.} \tag{11}$$

These are the normal equations. If MM^T is nonsingular, they give

$$\boxed{\boldsymbol{w} = (MM^T)^{-1} M\boldsymbol{y}.} \tag{12}$$

Equation (12) is useful as a derivation, but it is usually not the best numerical algorithm. Forming MM^T squares the spectral condition number:

$$\kappa_2(MM^T) = \kappa_2(M)^2 \tag{13}$$

when the relevant singular values are nonzero. A direct least-squares routine based on a rank-revealing factorization is preferable.

NumPy's interface solves

$$\min_{\boldsymbol{z}} \|A\boldsymbol{z} - \boldsymbol{b}\|_2. \tag{14}$$

Identifying $A = M^T$, $\boldsymbol{z} = \boldsymbol{w}$, and $\boldsymbol{b} = \boldsymbol{y}$ gives

```
w2, residuals, rank, s = np.linalg.lstsq(
    M.T, y, rcond=None
)
```

The first returned object is the fitted vector $\boldsymbol{w}$. The other objects report residual information, numerical rank, and singular values.

3 The Moore–Penrose pseudoinverse

The Moore–Penrose pseudoinverse generalizes the matrix inverse to rectangular and rank-deficient matrices. Let the compact singular value decomposition of a matrix A be

$$A = U\Sigma V^T, \tag{15}$$

where the nonzero diagonal entries of Σ are the singular values $s_i > 0$. The pseudoinverse is

$$A^+ = V\Sigma^+ U^T, \tag{16}$$

where

$$(\Sigma^+)_{ii} = \frac{1}{s_i} \tag{17}$$

for retained singular values, while singular values treated as numerically zero contribute zero.

For the ELM equation (5), the pseudoinverse solution is

$$\boxed{\boldsymbol{w} = (M^T)^+ \boldsymbol{y}.} \tag{18}$$

The identity

$$(M^T)^+ = (M^+)^T \tag{19}$$

shows how this relates to the original row-vector expression. Transposing Eq. (18) gives

$$\boxed{\boldsymbol{w}^T = \boldsymbol{y}^T M^+.} \tag{20}$$

This is implemented by

```
w2 = y @ np.linalg.pinv(M)
```

when `y` is a one-dimensional NumPy array. The result satisfies

$$\boldsymbol{w} = \underset{\boldsymbol{z}}{\operatorname{argmin}} \|\boldsymbol{z}\|_2 \quad \text{among all vectors minimizing} \quad \|M^T\boldsymbol{z} - \boldsymbol{y}\|_2. \tag{21}$$

Thus the pseudoinverse supplies two properties:

1. it minimizes the residual norm;
2. if several vectors give the same minimum residual, it selects the one with minimum Euclidean norm.

For $m > n$, as in an ELM with more hidden neurons than training points, the system $M^T\boldsymbol{w} = \boldsymbol{y}$ is underdetermined. If it is consistent, there are generally infinitely many interpolating output-weight vectors. The pseudoinverse selects the minimum-norm interpolant.

4 Why `pinv` and `lstsq` give the same mathematical solution

For any matrix A, the minimum-norm least-squares solution is

$$\boldsymbol{z} = A^{+}\boldsymbol{b}. \tag{22}$$

Taking $A = M^T$ gives

$$\boldsymbol{w} = (M^T)^{+}\boldsymbol{y} = (M^{+})^T\boldsymbol{y}. \tag{23}$$

After transposition,

$$\boldsymbol{w}^T = \boldsymbol{y}^T M^{+}. \tag{24}$$

Therefore

```
w2 = y @ np.linalg.pinv(M)
```

and

```
w2 = np.linalg.lstsq(M.T, y, rcond=None)[0]
```

represent the same mathematical solution. Tiny numerical differences can occur because the routines may use different algorithms or effective singular-value cutoffs. The `lstsq` expression is normally preferable because it states and solves the least-squares problem directly instead of explicitly forming M^{+}.

If `y` is stored as a row array with shape `(1,n)`, the corresponding call is

```
w2 = np.linalg.lstsq(M.T, y.T, rcond=None)[0].T
```

5 SVD interpretation and large output weights

Write the singular value decomposition of the design matrix as

$$M^T = U\Sigma V^T. \tag{25}$$

The minimum-norm least-squares solution becomes

$$\boldsymbol{w} = V\Sigma^{+}U^T\boldsymbol{y} = \sum_{i:s_i>0} \frac{\boldsymbol{u}_i^T\boldsymbol{y}}{s_i}\,\boldsymbol{v}_i. \tag{26}$$

Equation (26) explains why a visually excellent ELM fit can have very large output weights. If two or more hidden neurons produce nearly the same feature, M^T has a small singular value. Division by this s_i strongly amplifies the corresponding component of the data.

Large positive and negative weights may then cancel:

$$10^6\phi_1(x) - 10^6\phi_2(x), \tag{27}$$

leaving a function of ordinary size when ϕ_1 and ϕ_2 are almost identical. Such cancellation is sensitive to perturbations in the data, random features, and floating-point arithmetic.

The singular values can be inspected with

```
s = np.linalg.svd(M.T, compute_uv=False)
print("singular values:", s)
print("condition number:", s[0] / s[-1])
```

Input scaling and a suitable distribution of random hidden parameters can improve the feature matrix, but they do not remove the basic possibility of small singular values.

6 Ridge-regularized least squares

Ridge regression adds a quadratic penalty on the output weights:

$$L_\lambda(\boldsymbol{w}) = \left\| M^T\boldsymbol{w} - \boldsymbol{y} \right\|_2^2 + \lambda \|\boldsymbol{w}\|_2^2, \qquad \lambda \geq 0. \tag{28}$$

The first term rewards agreement with the data; the second discourages large and mutually cancelling coefficients. Differentiation gives

$$\nabla_{\boldsymbol{w}} L_\lambda = 2M(M^T\boldsymbol{w} - \boldsymbol{y}) + 2\lambda\boldsymbol{w} \tag{29}$$

$$= 2(MM^T + \lambda I)\boldsymbol{w} - 2M\boldsymbol{y}. \tag{30}$$

Setting the gradient to zero produces the ridge normal equations,

$$\boxed{(MM^T + \lambda I)\boldsymbol{w}_\lambda = M\boldsymbol{y}.} \tag{31}$$

For $\lambda > 0$, the matrix $MM^T + \lambda I$ is positive definite even when M is rank deficient. Formally,

$$\boxed{\boldsymbol{w}_\lambda = (MM^T + \lambda I)^{-1} M\boldsymbol{y}.} \tag{32}$$

A direct implementation is

```
nhid = M.shape[0]
w2 = np.linalg.solve(
    M @ M.T + lam * np.eye(nhid),
    M @ y
)
```

but this again forms MM^T. A numerically preferable formulation observes that Eq. (28) can be written as one augmented least-squares problem:

$$L_\lambda(\boldsymbol{w}) = \left\| \begin{pmatrix} M^T \\ \sqrt{\lambda} I \end{pmatrix} \boldsymbol{w} - \begin{pmatrix} \boldsymbol{y} \\ \boldsymbol{0} \end{pmatrix} \right\|_2^2. \tag{33}$$

Indeed, squaring the norm in Eq. (33) gives

$$\|M^T \boldsymbol{w} - \boldsymbol{y}\|_2^2 + \|\sqrt{\lambda}\boldsymbol{w}\|_2^2, \tag{34}$$

which is exactly Eq. (28). The augmented implementation is

```
nhid = M.shape[0]

A_aug = np.vstack((
    M.T,
    np.sqrt(lam) * np.eye(nhid)
))

y_aug = np.concatenate((
    y,
    np.zeros(nhid)
))

w2 = np.linalg.lstsq(A_aug, y_aug, rcond=None)[0]
```

This solves the ridge problem without explicitly constructing MM^T.

7 Ridge regression in the singular-vector basis

Again let

$$M^T = U\Sigma V^T. \tag{35}$$

Since

$$MM^T = V\Sigma^T\Sigma V^T, \tag{36}$$

Eq. (32) becomes

$$\boxed{\boldsymbol{w}_\lambda = V \operatorname{diag}\left(\frac{s_i}{s_i^2 + \lambda}\right) U^T \boldsymbol{y}.} \tag{37}$$

Ordinary least squares uses the factor

$$\frac{1}{s_i}, \tag{38}$$

whereas ridge regression uses

$$\frac{s_i}{s_i^2 + \lambda}. \tag{39}$$

For a well-determined direction, $s_i^2 \gg \lambda$, so

$$\frac{s_i}{s_i^2 + \lambda} \approx \frac{1}{s_i}. \tag{40}$$

For a poorly determined direction, $s_i^2 \ll \lambda$, so

$$\frac{s_i}{s_i^2 + \lambda} \approx \frac{s_i}{\lambda}. \tag{41}$$

Ridge regression therefore leaves large-singular-value directions nearly unchanged while suppressing the unstable small-singular-value directions.

The limits are

$$\lambda \to 0^+ : \quad \boldsymbol{w}_\lambda \to (M^T)^+ \boldsymbol{y}, \quad \text{under the usual rank assumptions,} \tag{42}$$

$$\lambda \to \infty : \quad \boldsymbol{w}_\lambda \to \boldsymbol{0}. \tag{43}$$

8 Comparison of the three expressions

Method	Mathematical expression	NumPy form
Pseudoinverse	$\boldsymbol{w} = (M^T)^+ \boldsymbol{y}$, or $\boldsymbol{w}^T = \boldsymbol{y}^T M^+$	`y @ pinv(M)`
Direct least squares	$\arg\min_{\boldsymbol{w}} \|M^T \boldsymbol{w} - \boldsymbol{y}\|_2$	`lstsq(M.T,y)[0]`
Ridge least squares	$\arg\min_{\boldsymbol{w}} [\|M^T \boldsymbol{w} - \boldsymbol{y}\|_2^2 + \lambda \|\boldsymbol{w}\|_2^2]$	augmented `lstsq`

The pseudoinverse and direct least-squares calls represent the same minimum-norm least-squares solution. Ridge regression deliberately changes the optimization problem: it accepts some additional fitting error in exchange for smaller and more stable output weights.

9 Practical scaling remarks

For a stable ELM experiment, scale the input using statistics obtained from the training data:

```
x_mean = x.mean()
x_std = x.std()
xs = (x - x_mean) / x_std
```

Prediction data must use the same `x_mean` and `x_std`. One should not normalize the already fitted $\boldsymbol{w}$: doing so changes the predicted function. Instead, improve the scaling of the inputs and hidden features, or control the weights through the ridge penalty.

For comparisons between different hidden-layer widths, random-feature theory often uses

$$M_{ij} = \frac{1}{\sqrt{m}} \sigma(w_i^{(1)} x_j + b_i^{(1)}). \tag{44}$$

This keeps the induced kernel

$$k_m(x, x') = \frac{1}{m} \sum_{i=1}^{m} \phi_i(x) \phi_i(x') \tag{45}$$

of order one as m grows. This normalization is important when comparing a fixed value of λ across widths. It does not by itself cure an ill-conditioned feature matrix; ridge regularization addresses that issue.

10 Summary

For the ELM convention $M \in \mathbb{R}^{m \times n}$, training the linear output layer means solving $M^T \boldsymbol{w} \approx \boldsymbol{y}$.

- `y @ pinv(M)` explicitly applies the Moore–Penrose pseudoinverse.
- `lstsq(M.T,y)[0]` obtains the same minimum-norm least-squares solution without explicitly constructing the pseudoinverse.
- Ridge regression adds $\lambda\|\boldsymbol{w}\|_2^2$ and replaces the unstable SVD factor $1/s_i$ by $s_i/(s_i^2 + \lambda)$.
- Very large ELM output weights signal cancellation associated with small singular values, even when the plotted training fit looks excellent.