Statistical Machine Learning 2018

Exercises and answers, week 10

23 November 2018

TUTORIAL

Exercise 1 – Classification with Labeling Errors

(See Exercise 5.4 in Bishop) Consider a binary classification problem in which the target values are $t \in \{0, 1\}$, with a network output $y(\mathbf{x}, \mathbf{w})$ that represents $p(t = 1|\mathbf{x}, \mathbf{w})$.

1. Let us consider a training set of independent and identically distributed observations $\mathcal{D} = \{(\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2), ..., (\mathbf{x}_N, t_N)\}$. Derive the expression for the negative log-likelihood of this data set and show that it is equivalent to the *cross-entropy* error function defined in Bishop, Equation (5.21).

   ANSWER: The conditional distribution of targets given inputs can be written as a Bernoulli distribution of the form

   $$p(t|\mathbf{x}, \mathbf{w}) = y(\mathbf{x}, \mathbf{w})^t \{1 - y(\mathbf{x}, \mathbf{w})\}^{1-t}.$$

   Given that the data points are independent, the likelihood can be written as

   $$p(\mathbf{t}|\mathbf{X}, \mathbf{w}) = \prod_{n=1}^{N} p(t_n|\mathbf{x}_n, \mathbf{w}) = \prod_{n=1}^{N} y(\mathbf{x}_n, \mathbf{w})^{t_n} \{1 - y(\mathbf{x}_n, \mathbf{w})\}^{1-t_n}.$$

   The negative log-likelihood is then

   $$-\log p(\mathbf{t}|\mathbf{X}, \mathbf{w}) = -\sum_{n=1}^{N} \{t_n \log y(\mathbf{x}_n, \mathbf{w}) + (1 - t_n) \log(1 - y(\mathbf{x}_n, \mathbf{w}))\}.$$

   If we denote $y(\mathbf{x}_n, \mathbf{w})$ by y_n, we immediately obtain the requested equivalence:

   $$E(\mathbf{w}) = -\log p(\mathbf{t}|\mathbf{X}, \mathbf{w}) = -\sum_{n=1}^{N} \{t_n \log y_n + (1 - t_n) \log(1 - y_n)\}.$$

2. In the expression of the *cross-entropy* error function there is no analogue for the target noise precision β, which we have previously encountered in Bayesian linear regression (Chapter 3). This is because the target values $\mathbf{t}$ are assumed to be correctly labeled. Suppose now that there is a probability ϵ that the class label on a training data point has been incorrectly set. If we denote the true label of the data point corresponding to input $\mathbf{x}_n$ by t_n and the given label by $\tilde{t}_n$, then show that the probability of a single 'noisy' observation can be written as

   $$p(\tilde{t}_n|\mathbf{x}_n, \mathbf{w}) = y_n^{\tilde{t}_n}(1 - y_n)^{1-\tilde{t}_n}(1 - \epsilon) + y_n^{1-\tilde{t}_n}(1 - y_n)^{\tilde{t}_n}\epsilon.$$

Hint: Use the same expression for the probability of a correctly labeled data point and then relate the true label to the noisy (given) label.

ANSWER: We can distinguish between two separate cases: (i) the given label is correct, i.e., $\tilde{t}_n = t_n$ and (ii) the given label is incorrect, i.e., $\tilde{t}_n \neq t_n$. Using the sum and product rules, we can then write the probability of observing a noisy label as:

$$\begin{aligned} p(\tilde{t}_n|\mathbf{x}_n, \mathbf{w}) &= p(\tilde{t}_n, \tilde{t}_n = t_n|\mathbf{x}_n, \mathbf{w}) + p(\tilde{t}_n, \tilde{t}_n \neq t_n|\mathbf{x}_n, \mathbf{w}) \\ &= p(\tilde{t}_n|\tilde{t}_n = t_n, \mathbf{x}_n, \mathbf{w})p(\tilde{t}_n = t_n|\mathbf{x}_n, \mathbf{w}) + p(\tilde{t}_n|\tilde{t}_n \neq t_n, \mathbf{x}_n, \mathbf{w})p(\tilde{t}_n \neq t_n|\mathbf{x}_n, \mathbf{w}) \\ &= p(\tilde{t}_n|\tilde{t}_n = t_n, \mathbf{x}_n, \mathbf{w})p(\tilde{t}_n = t_n) + p(1 - \tilde{t}_n|\tilde{t}_n = t_n, \mathbf{x}_n, \mathbf{w})p(\tilde{t}_n \neq t_n) \\ &= y_n^{\tilde{t}_n}(1 - y_n)^{1-\tilde{t}_n}(1 - \epsilon) + y_n^{1-\tilde{t}_n}(1 - y_n)^{\tilde{t}_n}\epsilon \end{aligned}$$

In the derivation, we used the fact that the probability of mislabeling is independent of the inputs, i.e., $p(\tilde{t}_n = t_n|\mathbf{x}_n, \mathbf{w}) = p(\tilde{t}_n = t_n)$. We also used the fact that $\tilde{t}_n \neq t_n \iff \tilde{t}_n = 1 - t_n$. Note that if we set $\epsilon = 0$ in the last expression, we obtain $p(\tilde{t}_n|\mathbf{x}_n, \mathbf{w}) = p(t_n|\mathbf{x}_n, \mathbf{w})$.

3. Show that the previous probability can be written equivalently as

$$p(\tilde{t}_n|\mathbf{x}_n, \mathbf{w}) = \tilde{y}_n^{\tilde{t}_n}(1 - \tilde{y}_n)^{1-\tilde{t}_n},$$

where $\tilde{y}_n = \tilde{y}(\mathbf{x}_n, \mathbf{w}, \epsilon) = y_n + \epsilon(1 - 2y_n)$. Then, assuming again independent and identically distributed data, write down the error function corresponding to the negative log-likelihood. Verify that the obtained error function is just the *cross-entropy* function when $\epsilon = 0$.

ANSWER: We use the fact that there are only two possible labels for $\tilde{t}_n$:

$$\begin{aligned} p(\tilde{t}_n = 1|\mathbf{x}_n, \mathbf{w}) &= y_n(1 - \epsilon) + (1 - y_n)\epsilon = y_n + \epsilon(1 - 2y_n) = \tilde{y}_n; \\ p(\tilde{t}_n = 0|\mathbf{x}_n, \mathbf{w}) &= (1 - y_n)(1 - \epsilon) + y_n\epsilon = 1 - y_n - \epsilon(1 - 2y_n) = 1 - \tilde{y}_n. \end{aligned}$$

Like before, we can then write these two values compactly following the Bernoulli form:

$$p(\tilde{t}_n|\mathbf{x}_n, \mathbf{w}) = \tilde{y}_n^{\tilde{t}_n}(1 - \tilde{y}_n)^{1-\tilde{t}_n}. \quad \square$$

The negative log-likelihood of the data set immediately follows:

$$\tilde{E}(\mathbf{w}) = -\log p(\tilde{\mathbf{t}}|\mathbf{X}, \mathbf{w}) = -\sum_{n=1}^{N}\{\tilde{t}_n \log \tilde{y}_n + (1 - \tilde{t}_n)\log(1 - \tilde{y}_n)\}.$$

If we have labeling without error, then $\tilde{y}_n = y_n$, and therefore we immediately obtain the *cross-entropy* function in the last expression.

4. The error function obtained for noisy observations makes the model robust to incorrectly labeled data, in contrast to the usual *cross-entropy* error function. To see this, consider the case when our network function $y(\mathbf{x}, \mathbf{w}) = p(t|\mathbf{x}, \mathbf{w})$ (not $\tilde{y}$) takes the values one and zero, i.e., when we are certain that the data point belongs to one class or the other. What is the corresponding value for $\tilde{y}(\mathbf{x}, \mathbf{w})$ and how does it account for the error in labeling? What happens to $\tilde{y}$ when $y(\mathbf{x}, \mathbf{w}) = 0.5$, which indicates maximum uncertainty in the classification?

ANSWER: If $y = 1$, then $\tilde{y} = p(\tilde{t} = 1|\mathbf{x}, \mathbf{w}) = 1 + \epsilon(1 - 2 \cdot 1) = 1 - \epsilon$. If $y = 0$, then $\tilde{y} = 0 + \epsilon(1 - 2 \cdot 0) = \epsilon \implies 1 - \tilde{y} = p(\tilde{t} = 0|\mathbf{x}, \mathbf{w}) = 1 - \epsilon$. We see that the network function $\tilde{y}$ naturally incorporates the uncertainty in the labeling. The interpretation is that we can only be at most $(1 - \epsilon)$ certain in our classification given the 'noisy' labeling, since there is a ϵ chance of mislabeling. If $y = 0.5$, then $\tilde{y} = 0.5 + 2(1 - 0.5) = 0.5$, so we are still in the situation of maximum uncertainty.

Exercise 2

We look at backpropagation in a neural network. (see §5.3) In figure 1 a two-layer neural network is depicted with one input node x, one output node y and three hidden nodes z_1, z_2 and z_3. In addition two fixed value bias nodes have been included, $x_0 = 1$ and $z_0 = 1$. The links between them represent weight parameters, with $w_3^{(1)}$ indicating a weight between x and node z_3, and with $b_1^{(2)}$ representing a weight between bias node z_0 and node y. (*Note:* the distinction between 'normal' and bias nodes/weights is purely for notational purposes.) The network is used in a regression problem to learn the function $f(x) = x^2$. The output node has a linear activation function ($y = \sum_j w_j^{(2)} z_j + b_1^{(2)}$), and the z_j have $\tanh(\cdot)$ activation functions.

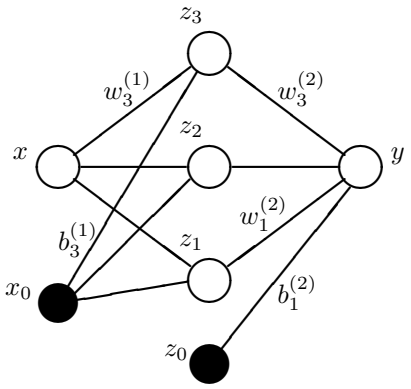

Figure 1: Neural network, 3 hidden nodes

After initialization of the weights, training of the network essentially involves three steps

- Propagate new input value x through network to calculate output y (eq. 5.48/49)
- Backpropagate error $\delta = (y - t)$ to establish the derivatives $\frac{\partial E}{\partial w_{ji}}$ (eq. 5.53/56)
- Use the estimated derivatives in gradient descent (or similar) to update weights (eq. 5.43)

and repeat until convergence.

We want to study how the network adapts to new input/output data through backpropagation.

1. Why is it not a good idea to start with a network in which all the weights have been initialized to zero?

   ANSWER: With zero weights, in the first forward propagation phase all outputs z_i from non-bias nodes in the network are zero. However, the main reason why the network cannot learn in this situation is that in the subsequent phase all backpropagated errors δ_i are also zero. As a result, all derivatives $\frac{\partial E_n}{\partial w_{ji}}$ become zero, with the exception of the gradients for the bias parameter(s) to the output node(s). Therefore there is no update in a gradient descent method for any other node, and the network is only able to learn the constant output bias. Note: Similarly, initializing all weights to the same value is not helpful (all weights are updated the same, making it essentially a single hidden node), nor choosing initial values that are very large (saturation: weights are hardly updated at all in each step).

The weights are set to $\mathbf{w}^{(1)} = [1, 0.1, -1]$, $\mathbf{w}^{(2)} = [-1, 0.1, -1]$ and for the bias $\mathbf{b}^{(1)} = [1, 0, 1]$ and $\mathbf{b}^{(2)} = [2]$. After this initialization, the first point offered to the network is $\{x_1, t_1\} = \{0.5, 0.25\}$.

2. Calculate one full training cycle of the network for this datapoint. Assume a sum-of-squares error function $E(\mathbf{w})$, and use a learning rate parameter of $\eta = 0.5$ in updating the weights.

Check whether or not the updated network has improved the output.

ANSWER: Forward propagating the first data point $\{x_1, t_1\} = \{0.5, 0.25\}$ in the network of fig.1, using $a_j = \sum_i w_{ji} z_i$ (eq.5.48), the inputs a_j to the hidden nodes are

$$\mathbf{a} = [(0.5 + 1), (0.05 + 0), (-0.5 + 1)] = [1.5, 0.05, 0.5]$$

For $\tanh(\cdot)$ activation functions the subsequent outputs become

$$\mathbf{z} = [\tanh(1.5), \tanh(0.05), \tanh(0.5)] \approx [0.905, 0.05, 0.462]$$

This results in a initial value for output node y of

$$\mathbf{y} \approx [-0.905 + 0.005 + 0.462 + 2] = [0.638]$$

Backpropagating the errors we find for the output node

$$\delta_y = y - t = 0.638 - 0.25 = 0.388$$

and, using $\delta_j = \tanh'(a_j) \sum_k w_{kj} \delta_k$ (eq.5.56), for the hidden nodes

$$\delta_z = [(1 - (-0.905)^2) \cdot (-1) \cdot (0.388), ..., ...] \approx [-0.07, 0.0387, -0.305]$$

where we used $\tanh'(a_j) = (1 - \tanh^2(a_j)) = (1 - z_j^2)$.
Next comes the task of calculating the derivatives using $\dfrac{\partial E_n}{\partial w_{ji}} = \delta_j z_i$ (eq.5.53), to give

$$\frac{\partial E_1}{\partial \mathbf{w}^{(1)}} = [(-0.07) \cdot (0.5), (0.0387) \cdot (0.5), (-0.305) \cdot (0.5)] = [-0.035, 0.0193, -0.1525]$$

Similarly,

$$\begin{aligned} \frac{\partial E_1}{\partial \mathbf{b}^{(1)}} &= [-0.07, 0.0387, -0.305] \\ \frac{\partial E_1}{\partial \mathbf{w}^{(2)}} &= [0.35, -0.0194, 0.18] \\ \frac{\partial E_1}{\partial \mathbf{b}^{(2)}} &= [0.388] \end{aligned}$$

Using these in a stochastic gradient descent update $\mathbf{w}^{(new)} = \mathbf{w}^{(old)} - \eta \nabla E_n(\mathbf{w}^{(old)})$ with learning parameter $\eta = 0.5$ then gives for the updated values

$$\begin{aligned} \mathbf{w}^{(1)} &= [1.018, 0.0903, -0.924] \\ \mathbf{b}^{(1)} &= [1.035, -0.0193, 1.153] \\ \mathbf{w}^{(2)} &= [-1.176, 0.0903, -1.090] \\ \mathbf{b}^{(2)} &= [1.806] \end{aligned}$$

Forward propagating the same point once more using the updated weights gives for the output $\mathbf{y} \approx [0.084]$ for an error of $\delta_y = -0.166$. Clearly, after one complete update cycle the error in the output of the network has indeed been reduced. In this case, a learning rate parameter of slightly over 0.7 is already enough to make the network unstable and prevent convergence. However, for $\eta \approx 0.33$, convergence up to 5 digits is already achieved in just 4 steps.

In this regression network the output node y has a linear activation function. If we wanted to do classification we would choose a sigmoidal function. However, this freedom in the choice of activation function does not apply to the hidden nodes z_i.

3. Describe why the nonlinearity of the activation functions for the hidden nodes is crucial for a network to learn more than mainly trivial regression or classification problems.
   Hint: Consider what regression/classification such a network would be able to perform.

   ANSWER: Any network with linear activation functions for nodes in the hidden layer(s) is equivalent to a network without any hidden nodes (why?). Therefore, such a network can essentially perform only linear regression / classification and is not capable to deal effectively with problems that are not linearly separable. It mostly resembles the K-class linear discriminant of §4.1.2.
   All the power and flexibility of neural networks as 'universal approximators' stems from the nonlinearity of the activation functions for hidden nodes in intermediate layers.

In a sequential learning procedure, each time one of the points is selected at random and processed through one full training cycle of the network. Batch mode takes all available data points at once before making an update and repeating the process. Suppose the entire data set consists of 50 data points $\{x_i, t_i\}$ sampled over the interval $[-1, 1]$ from a noisy quadratic function: $t = x^2 + \epsilon$, with $\epsilon \sim \mathcal{N}(\epsilon|0, 0.2)$.

4. Will the sequential procedure converge for this data set? If so, is the final weight distribution the same as when a batch procedure would have been used? If not, does batch mode updating suffer from the same drawback? Explain your answer.

   ANSWER: Both answers are possible. If we just used sequential gradient descent method with some small but fixed η the procedure would not converge. Instead, it would keep on changing within a relatively small volume of parameter space because the noisy data would ensure that for each step there remains a non-zero error component in the output that gets propagated back through the network to update the weight vector (assuming there are fewer hidden nodes than datapoints). A batch procedure does not suffer from the same problem, as the contribution of all data points to the error is aggregated before the weights are updated, and therefore once a (local) minimum is reached the weight vector remains the same.
   However, from the Robbins-Monro algorithm §2.3.5, we know that a stochastic gradient descent algorithm will converge, provided the successive update-steps satisfy eq.2.130-2. Therefore, if the learning rate parameter η is suitably reduced in each step, even the sequential procedure will converge. It is not guaranteed that this minimum is the same as when a batch procedure is used (as the latter can depend on the sarting point) , but it is guaranteed to correspond to a local minimum for the batch procedure (why?). However, in some cases the random fluctuation in sequential updates can actually be beneficial, as it is easier for the algorithm to escape from a (near) local minimum compared to the more stable batch update ...

BONUS PRACTICE

Exercise 3

In neural networks, the nonlinear activation functions $h(\cdot)$ for the hidden units are generally chosen to be 'S-shaped' functions, such as the logistic sigmoid $\sigma(a)$ and the $\tanh(a)$ function, where

$$\sigma(a) = \frac{1}{1+\exp(-a)} \tag{1}$$

$$\tanh(a) = \frac{\exp(a)-\exp(-a)}{\exp(a)+\exp(-a)} \tag{2}$$

1. Show that $\tanh(a)$ is just a stretched, squashed and shifted sigmoid:

$$\tanh(a) = 2\sigma(2a) - 1$$

ANSWER: Use that $\exp(-a)/\exp(a) = \exp(-2a)$ to get

$$\begin{aligned}
\tanh(a) &= \frac{\exp(a)-\exp(-a)}{\exp(a)+\exp(-a)} \\
&= \frac{1-\exp(-2a)}{1+\exp(-2a)} \\
&= \frac{2-(1+\exp(-2a))}{1+\exp(-2a)} \\
&= \frac{2}{1+\exp(-2a)} - \frac{1+\exp(-2a)}{1+\exp(-2a)} \\
&= 2\sigma(2a) - 1
\end{aligned}$$

So tanh is a sigmoid that is stretched by a factor 2 in the vertical direction, squashed by a factor 2 along the horizontal and shifted down by minus 1.

2. Show that the derivative $h'(a)$ of the $\tanh(\cdot)$ activation function can be expressed in terms of the function itself:

$$\frac{d\tanh(a)}{da} = 1 - \tanh^2(a) \tag{3}$$

ANSWER: Apply quotient rule $(f/g)' = (f'g - fg')/g^2$

$$\begin{aligned}
\frac{d\tanh(a)}{da} &= \frac{d}{da}\left(\frac{\exp(a)-\exp(-a)}{\exp(a)+\exp(-a)}\right) \\
&= \frac{(\exp(a)+\exp(-a))^2 - (\exp(a)-\exp(-a))^2}{(\exp(a)+\exp(-a))^2} \\
&= 1 - \frac{(\exp(a)-\exp(-a))^2}{(\exp(a)+\exp(-a))^2} \\
&= 1 - \tanh^2(a)
\end{aligned}$$

3. Create a Taylor expansion of (2) to argue that for small a, i.e. $a \approx 0$, the tanh activation function is essentially linear, and well approximated by

$$\tanh(a) \approx a$$

ANSWER: The second order Taylor expansion of a 1-dimensional function around u is:

$$f(u+x) = f(u) + xf'(u) + \frac{1}{2}x^2 f''(u) + \mathcal{O}(x^3)$$

Filling in for $u = 0$ we have $\tanh(0) = 0$ and $\tanh'(0) = 1 - \tanh^2(0) = 1$. Differentiating (3) once more (chain rule) gives $\tanh''(0) = -2\tanh(0)(1 - \tanh^2(0)) = 0$. So, second order term in Taylor expansion of (2) around zero vanishes, and we end up with

$$\tanh(a) = 0 + a + 0 + \mathcal{O}(a^3) \tag{4}$$

(See page XI of Bishop for the meaning of the $\mathcal{O}$-symbol). In other words, for small a, $\tanh(a) \approx a$ is a good approximation, with error of order a^3.

Exercise 4

(Exercise 5.25 in Bishop)

Consider a quadratic error function of the form:

$$E(\mathbf{w}) = E_0 + \frac{1}{2}(\mathbf{w} - \mathbf{w}^*)^{\mathrm{T}}\mathbf{H}(\mathbf{w} - \mathbf{w}^*), \tag{5}$$

where $\mathbf{w}^*$ minimizes the error function, and the Hessian matrix $\mathbf{H}$ is positive definite and constant. Suppose the initial weight vector $\mathbf{w}^{(0)}$ is chosen to be at the origin and is updated using simple gradient descent:

$$\mathbf{w}^{(\tau)} = \mathbf{w}^{(\tau-1)} - \rho\nabla E \tag{6}$$

where τ denotes the step number, and ρ is the learning rate, which is assumed to be small. Show that, after τ steps, the components of the weight vector parallel to the eigenvectors of $\mathbf{H}$ can be written as:

$$w_j^{(\tau)} = \left[1 - (1 - \rho\eta_j)^{\tau}\right] w_j^*, \tag{7}$$

where $w_j = \mathbf{w}^{\mathrm{T}}\mathbf{u}_j$, and $\mathbf{u}_j$ and η_j are the eigenvectors and eigenvalues, respectively, of $\mathbf{H}$, so that

$$\mathbf{H}\mathbf{u}_j = \eta_j\mathbf{u}_j. \tag{8}$$

Show that as $\tau \to \infty$, this gives $\mathbf{w}^{(\tau)}$ as expected, provided $|1 - \rho\eta_j| < 1$. Now suppose that training is halted after a finite number τ of steps. Show that the components of the weight vector parallel to the eigenvectors of the Hessian satisfy

$$w_j^{(\tau)} \simeq w_j^* \quad \text{when} \quad \eta_j \gg (\rho\tau)^{-1},$$
$$|w_j^{(\tau)}| \ll |w_j^*| \quad \text{when} \quad \eta_j \ll (\rho\tau)^{-1}.$$

Show that $(\rho\tau)^{-1}$ is analogous to the simple weight decay regularization parameter λ (see Equation (5.112) in Bishop).

ANSWER:

The gradient of the error function in (5) is

$$\nabla E = \mathbf{H}(\mathbf{w} - \mathbf{w}^*),$$

so we update formula (6) accordingly to obtain:

$$\mathbf{w}^{(\tau)} = \mathbf{w}^{(\tau-1)} - \rho\mathbf{H}(\mathbf{w}^{(\tau-1)} - \mathbf{w}^*).$$

By pre-multiplying both sides of the previous result with $\mathbf{u}_j^{\mathrm{T}}$, we get

$$\begin{aligned} w_j^{(\tau)} &= \mathbf{u}_j^{\mathrm{T}}\mathbf{w}^{(\tau)} \\ &= \mathbf{u}_j^{\mathrm{T}}\mathbf{w}^{(\tau-1)} - \rho\mathbf{u}_j^{\mathrm{T}}\mathbf{H}(\mathbf{w}^{(\tau-1)} - \mathbf{w}^*) \\ &= w_j^{(\tau-1)} - \rho\eta_j\mathbf{u}_j^{\mathrm{T}}(\mathbf{w} - \mathbf{w}^*) \\ &= w_j^{(\tau-1)} - \rho\eta_j(w_j^{(\tau-1)} - w_j^*), \end{aligned} \tag{9}$$

where we have used Equation (8).

To show that Equation (7) holds for $\tau = 1, 2, ...$, we can use proof by induction. For $\tau = 1$, we recall that $\mathbf{w}^{(0)} = \mathbf{0}$ an insert this into (9), which then gives

$$\begin{aligned} w_j^{(1)} &= w_j^{(0)} - \rho\eta_j(w_j^{(0)} - w_j^*) \\ &= \rho\eta_j w_j^* \\ &= [1 - (1 - \rho\eta_j)]w_j^*. \quad \square \end{aligned}$$

Now we assume that the result holds for $\tau = N - 1$ and then again make use of (9) to get:

$$\begin{aligned} w_j^{(N)} &= w_j^{(N-1)} - \rho\eta_j(w_j^{(N-1)} - w_j^*) \\ &= w_j^{(N-1)}(1 - \rho\eta_j) + \rho\eta_j w_j^* \\ &= [1 - (1 - \rho\eta_j)^{N-1}]w_j^*(1 - \rho\eta_j) + \rho\eta_j w_j^* \\ &= [(1 - \rho\eta_j) - (1 - \rho\eta_j)^N]w_j^* + \rho\eta_j w_j^* \\ &= [1 - (1 - \rho\eta_j)^N]w_j^*. \quad \square \end{aligned}$$

Provided that $|1 - \rho\eta_j| < 1$, we have $(1 - \rho\eta_j)^\tau \to 0$ as $\tau \to \infty$, and hence $[1 - (1 - \rho\eta_j)^N] \to 1$ and $\mathbf{w}^{(\tau)} \to \mathbf{w}^*$. If τ is finite, but $\eta_j \gg (\rho\tau)^{-1}$, τ must still be large, since $\eta_j\rho\tau \gg 1$, even though $|1 - \rho\eta_j| < 1$. If τ is large, it follows from the argument above that $w_j^{(\tau)} \simeq w_j^*$. If, on the other hand, $\eta_j \ll (\rho\tau)^{-1}$, this means that $\rho\eta_j$ must be small, since $\rho\eta_j\tau \ll 1$ and τ is an integer greater than or equal to one. If we expand,

$$(1 - \rho\eta_j)^\tau = 1 - \tau\rho\eta_j + O(\rho\eta_j^2)$$

and insert this into Equation (7), we get

$$\begin{aligned} |w_j^{(\tau)}| &= |[1 - (1 - \rho\eta_j)^\tau]w_j^*| \\ &= |[1 - (1 - \tau\rho\eta_j + O(\rho\eta_j^2))]w_j^*| \\ &\simeq \tau\rho\eta_j|w_j^*| \ll |w_j^*| \end{aligned}$$

$\rho\tau$ plays an analogous role a regularization parameter. When the regularization parameter is much larger than one of the eigenvalues, then the corresponding parameter value w_i will be close to zero. Conversely, when the regularization parameter is much smaller than one of the eigenvalues, then w_i will be close to its maximum likelihood value.