Machine Learning for Signal Processing

[5LSL0]

Rik Vullings
Ruud van Sloun
Nishith Chennakeshava
Hans van Gorp

Assignment 2: Nonlinear Models

April 2022

Learning nonlinear functions for regression and classification

In this assignment we will explore some of the basic elements of machine learning in the context of optimal parameter estimation for nonlinear function approximation, regression and classification.

Linear models

Consider the following regression network:

$$f(\mathbf{x}; \mathbf{w}, b) = \mathbf{w}^T \mathbf{x} + b, \tag{1.1}$$

which we will use to map a set of input vectors $\mathbf{x}_i$ to a set of outputs y_i. The scalar outputs y_i are continuous variables stored in a vector y, and the input vectors $\mathbf{x}_i$ are stored in a matrix X (see lecture slides).

Let us assume a Gaussian distribution of prediction errors with a unity covariance matrix, i.e. $p(y_i; \theta)$ follows a normal distribution with mean $f(\mathbf{x}_i; \theta)$ and variance 1.

Q1: Derive the negative log-likelihood cost function $J(\mathbf{x}, y; \theta)$ that will yield a maximum likelihood estimator of w and b.

Q2: Given this cost function, derive an expression for the optimal weights w and bias b. (Hint: Derive and equate the gradient of the cost function with respect to the parameters to zero.)

We will now deploy our regression model and learn to predict outputs $f(\mathbf{x}; \mathbf{w}, b)$ that match those produced by some unknown process $f^*(\mathbf{x})$.

Consider the inputs $\mathbf{x} : \{[0, 0], [0.1, 1], [1, 0.2], [1, 1]\}$
and process outputs $f^*(\mathbf{x}) : \{0, 0.41, 0.18, 0.5\}$

Q3: Derive the optimal parameters w and b that describe this process. Is the process well-described by our regression model?

For the same inputs, we now measure our process with some arbitrary noisy sensor, and obtain the following outputs: $y : \{-0.0416, 0.3610, 0.1222, 0.4733\}$.
Q4: Recalculate the optimal parameters given these outputs. What happens? What would you do to obtain better estimates given such a noisy sensor?

When the noise covariance structure of our measurements is not unity, our maximum-likelihood derived cost function changes.
Q5: Derive the optimal negative log-likelihood cost criterion for a diagonal covariance matrix with $\sigma_0, ..., \sigma_i, ...\sigma_N$ on the diagonal. How does the balance between the variances of the parameter estimates play a role in the cost function?

Now let us consider learning a new function, the XOR: given a 2-element input vector with binary entries, the output will be 1 if and only if one of the two elements in the input vector is 1, and the other 0.

Q6: Calculate the optimal parameters of our regression model based on the four possible inputs and outputs of the XOR function.

Notably, our model is unable to learn and perform the (rather simple) XOR function. In fact, the same holds for many other interesting functions that we would like *a machine* to learn. As we shall see, applying a nonlinear transformation into a new domain greatly increases the flexibility of our systems.

Nonlinear functions

Before we proceed with nonlinear models, we define several nonlinear functions that are widely used in machine learning:

- **ReLU** (Rectified linear unit): $f(x) = \max(0, x)$.
- **Sigmoid**: $f(x) = \sigma(x) = 1/\left(1 + \exp(-x)\right)$.
- **Softmax**: $f(\mathbf{x})_j = \frac{\exp(x_j)}{\sum_{i=0}^{K-1} \exp(x_i)}$ for $j = 0, ..., K-1$, where K is the size of vector $\mathbf{x}$.

Q7: Derive compact expressions for the derivatives of the above nonlinearities with respect to their inputs.

As for adaptive filters, training nonlinear regression models to fulfil a specific task is done through gradient-based learning. These derivatives will come in handy at that point.

Q8: What happens to the gradients of the above three nonlinearities when the input values $x >> 0$?

Shallow (i.e. not deep...) nonlinear models

Consider a nonlinear model $f(\mathbf{x};\theta)$:

$$f(\mathbf{x};\theta) = f^{(2)}(\mathbf{h};\mathbf{w}^{(2)},b^{(2)}), \tag{1.2}$$

where

$$\mathbf{h} = f^{(1)}(\mathbf{x};\mathbf{W}^{(1)},\mathbf{b}^{(1)}) = \max(0,\mathbf{W}^{(1)}\mathbf{x}+\mathbf{b}^{(1)}), \tag{1.3}$$

nonlinearly maps $\mathbf{x}$ into a new space $\mathbf{h}$ through a ReLU.
The complete network is then:

$$f(\mathbf{x};\mathbf{W}^{(1)},\mathbf{b}^{(1)},\mathbf{w}^{(2)},b^{(2)}) = \left(\mathbf{w}^{(2)}\right)^T \max(0,\mathbf{W}^{(1)}\mathbf{x}+\mathbf{b}^{(1)}) + b^{(2)}. \tag{1.4}$$

In the lectures, we saw that it is indeed possible to replicate an XOR function with this nonlinear model.
Q9: We sequentially apply two functions: $f^{(1)}$ and $f^{(2)}$; why should there by a nonlinearity in the first? Doesn't a linear mapping to a new space h suffice?

Q10: Given the solution presented in the lecture slides (2-13), reproduce the plots on slide 2-14 in e.g. Python, showing how the input is mapped into a the latent space h. Also plot the decision boundary $f(\mathbf{x}) = 0.5$.

Q11: What does this decision boundary $f(\mathbf{x}) = 0.5$ look like for the input space x? Reconstruct it and present it in a plot.

Binary classification with logistic regression

We can turn our continuous regression problems into binary/categorical classification problems by mapping the outputs of our models to probabilities of a specific class. For the binary classification problem, we typically use a sigmoid function ($\sigma(\cdot)$) that squeezes its inputs between 0 and 1; if the probability > 0.5, the outcome is class 1, and it is 0 otherwise.

Q12: Which of the nonlinearities you know would be useful for categorical (multi-class) classification problems and why?

For the sake of simplicity, we first consider a linear model followed by a sigmoid function to convert the outputs into probabilities:

$$p = \sigma(f(\mathbf{x};\mathbf{w})) = \sigma(\mathbf{w}^T\mathbf{x}). \tag{1.5}$$

Learning the optimal parameters $\mathbf{w}$ for such a classification problem is called *logistic regression*. As for adaptive filters, we will do this in a gradient-descent fashion. We will therefore now turn to calculating the required gradients.

Since our outcomes are binary, 0 or 1, the negative log-likelihood cost function is obtained from the Bernoulli distribution; i.e. the binary cross-entropy between the predicted probabilities and the true outcomes (labels/targets):

$$J = -\sum_{i=0}^{m-1} y^{(i)}\log(p^{(i)}) + (1-y^{(i)})\log(1-p^{(i)}), \tag{1.6}$$

where m is the amount of data samples, $y^{(i)}$ is the i^{th} label, and $p^{(i)}$ is the predicted probability that the outcome is 1. In our case, $p^{(i)} = \sigma(f(\mathbf{x}^{(i)}; \mathbf{w}))$.

Q13: Derive the gradient of J with respect to p, i.e. $\partial_p J$.

Q14: Derive the gradient of p with respect to f, i.e. $\partial_f p$ (see Q7).

Q15: Derive the gradient of $f(\mathbf{x}; \mathbf{w})$ with respect to $\mathbf{w}$, i.e. $\partial_{\mathbf{w}} f(\mathbf{x}; \mathbf{w})$. To which update (linear) adaptive filter does this step relate?

Q16: Use the chain rule to derive the gradient of the cost function with respect to the weights, i.e. $\partial_{\mathbf{w}} J$

Classification with a shallow nonlinear model

We will now extend our logistic regression classifier with the more flexible nonlinear model we described earlier (Eqn. 1.4), i.e.

$$p^{(i)} = \sigma\left(f^{(2)}\left(f^{(1)}(\mathbf{x}; \mathbf{W}^{(1)}, \mathbf{b}^{(1)}); \mathbf{w}^{(2)}, b^{(2)}\right)\right) \tag{1.7}$$

Q17: Similar to Q13-Q16, use the chain rule to derive the gradient of the cost function with respect to all the model weights and biases, i.e. $\partial_{\mathbf{w}^{(2)}} J$, $\partial_{\mathbf{W}^{(1)}} J$, $\partial_{b^{(2)}} J$ and $\partial_{\mathbf{b}^{(1)}} J$.
This is a multiple choice question, with one possible answer for each of the four partial derivatives.

$$\partial_{\mathbf{w}^{(2)}} J = \begin{array}{ll} A: & \begin{cases} \mathbf{0} & \text{if} \quad \mathbf{W}^{(1)}\mathbf{x}^{(i)} + \mathbf{b}^{(1)} < 0 \\ \left(\sum_i y^{(i)} - \sigma\left(f^{(2)}\right)\right)\left(\mathbf{W}^{(1)}\mathbf{x}^{(i)} + \mathbf{b}^{(1)}\right) & \text{if} \quad \mathbf{W}^{(1)}\mathbf{x}^{(i)} + \mathbf{b}^{(1)} \geq 0 \end{cases} \\ B: & \begin{cases} \mathbf{0} & \text{if} \quad \mathbf{W}^{(1)}\mathbf{x}^{(i)} + \mathbf{b}^{(1)} < 0 \\ -\left(\sum_i y^{(i)} - \sigma\left(f^{(2)}\right)\right)\left(\mathbf{W}^{(1)}\mathbf{x}^{(i)} + \mathbf{b}^{(1)}\right) & \text{if} \quad \mathbf{W}^{(1)}\mathbf{x}^{(i)} + \mathbf{b}^{(1)} \geq 0 \end{cases} \\ C: & \left(\sum_i y^{(i)} - \sigma\left(f^{(2)}\right)\right)\left(\mathbf{W}^{(1)}\mathbf{x}^{(i)} + \mathbf{b}^{(1)}\right) \\ D: & -\left(\sum_i y^{(i)} - \sigma\left(f^{(2)}\right)\right)\left(\mathbf{W}^{(1)}\mathbf{x}^{(i)} + \mathbf{b}^{(1)}\right) \end{array} \tag{1.8}$$

$$\partial_{\mathbf{W}^{(1)}} J = \begin{array}{ll} A: & \begin{cases} \mathbf{0} & \text{if} \quad \mathbf{W}^{(1)}\mathbf{x}^{(i)} + \mathbf{b}^{(1)} < 0 \\ -\left(\sum_i y^{(i)} - \sigma\left(f^{(2)}\right)\right)\left(\mathbf{w}^{(2)}\right)^T \mathbf{x}^{(i)} & \text{if} \quad \mathbf{W}^{(1)}\mathbf{x}^{(i)} + \mathbf{b}^{(1)} \geq 0 \end{cases} \\ B: & \begin{cases} \mathbf{0} & \text{if} \quad \mathbf{W}^{(1)}\mathbf{x}^{(i)} + \mathbf{b}^{(1)} < 0 \\ \left(\sum_i y^{(i)} - \sigma\left(f^{(2)}\right)\right)\left(\mathbf{w}^{(2)}\right)^T \mathbf{x}^{(i)} & \text{if} \quad \mathbf{W}^{(1)}\mathbf{x}^{(i)} + \mathbf{b}^{(1)} \geq 0 \end{cases} \\ C: & \sum_i \left(\frac{1-y^{(i)}}{1-\sigma\left(f^{(2)}\right)} - \frac{y^{(i)}}{\sigma\left(f^{(2)}\right)}\right)\left(\mathbf{w}^{(2)}\right)^T \mathbf{x}^{(i)} \\ D: & -\sum_i \left(\frac{1-y^{(i)}}{1-\sigma\left(f^{(2)}\right)} - \frac{y^{(i)}}{\sigma\left(f^{(2)}\right)}\right)\left(\mathbf{w}^{(2)}\right)^T \mathbf{x}^{(i)} \end{array} \tag{1.9}$$

$$
\partial_{b^{(2)}} J = \begin{array}{ll}
A: & 1 \\
B: & \begin{cases} \mathbf{0} & \text{if} \quad \mathbf{x}^{(i)} < 0 \\ -\sum_i y^{(i)} - \sigma\left(f^{(2)}\right) & \text{if} \quad \mathbf{x}^{(i)} \geq 0 \end{cases} \\
C: & -\sum_i y^{(i)} - \sigma\left(f^{(2)}\right) \\
D: & \sum_i y^{(i)} - \sigma\left(f^{(2)}\right)
\end{array} \tag{1.10}
$$

$$
\partial_{\mathbf{b}^{(1)}} J = \begin{array}{ll}
A: & \begin{cases} \mathbf{0} & \text{if} \quad \mathbf{W}^{(1)}x + \mathbf{b}^{(1)} < 0 \\ \left(\sum_i y^{(i)} - \sigma\left(f^{(2)}\right)\right)\left(\mathbf{w}^{(2)}\right)^T & \text{if} \quad \mathbf{W}^{(1)}x + \mathbf{b}^{(1)} \geq 0 \end{cases} \\
B: & \begin{cases} \mathbf{0} & \text{if} \quad \mathbf{W}^{(1)}x + \mathbf{b}^{(1)} < 0 \\ -\left(\sum_i y^{(i)} - \sigma\left(f^{(2)}\right)\right)\left(\mathbf{w}^{(2)}\right)^T & \text{if} \quad \mathbf{W}^{(1)}x + \mathbf{b}^{(1)} \geq 0 \end{cases} \\
C: & 1 \\
D: & \left(\sum_i y^{(i)} - \sigma\left(f^{(2)}\right)\right)\left(\mathbf{w}^{(2)}\right)^T
\end{array} \tag{1.11}
$$

The model you have just evaluated is actually a shallow, single-layer, *neural network.* The gradient calculation strategy you derived is called *back-propagation.* As for adaptive filters, we can now update our weights as: $\mathbf{w}_{n+1} = \mathbf{w}_n + \mu\partial_{\mathbf{w}} J$, with μ being some learning rate.