Deep Learning Assignment 3 in FS 2022

Universal Function Approximator

Microsoft Forms Document:
https://forms.office.com/r/dcZ3NADdya

Manuel Günther

Distributed: Friday, March 11, 2022
Discussed: Friday, March 18, 2022

The goal of this exercise is to train a 2-layer fully-connected network to perform one-dimensional non-linear regression via gradient descent. To show the flexibility of the approach, three different functions will be approximated. First, the network and its gradient need to be implemented. Second, the training procedure of gradient descent is implemented. Third, target data for three different functions will be generated. Finally, the training procedure will be applied to the data, and the resulting approximated function will be plotted together with the data samples.

1 Network Implementation

A two-layer network is defined by parameters $\Theta = (\mathbf{W}^{(1)}, \vec{w}^{(2)})$ that are split into $\mathbf{W}^{(1)} \in \mathbb{R}^{K\times(D+1)}$ for the first layer and $\vec{w}^{(2)} \in \mathbb{R}^{K+1}$ for the second layer. In our case, since we have only a single input, we have $D = 1$. For a given input $\vec{x}$, the network is implemented in three steps:

1. Compute the first layer output, aka, the activation: $\vec{a} = \mathbf{W}^{(1)}\vec{x}$
2. Compute the activation function for each element of $\vec{a}$: $\vec{h} = g(\vec{a})$ and prepend the bias neuron $h_0 = 1$.
3. Compute the output of the network: $y = \vec{w}^{(2)\mathrm{T}}\vec{h}$.

Task 1 Implement a function that returns the network output y and the output of the hidden neurons $\vec{h}$ for a given input $\vec{x} = (1, x)^{\mathrm{T}}$ and parameters $\Theta = (\mathbf{W}^{(1)}, \vec{w}^{(2)})$ that represent a variable number of hidden units K. Here, we use the logistic activation function:

$$g(a) = \frac{1}{1 + e^{-a}} \tag{1}$$

Test 1 When selecting all weights to be 0 in the first layer, i.e., $\mathbf{W}^{(1)} = \mathbf{0}^{K\times 2}$ and 1 in the second layer $\vec{w}^{(2)} = \mathbf{1}^{K+1}$, what is the expected output of the network for any $\vec{x}$? Implement a test case that assures that the function from Task 1 actually returns this output for those weights.

2 Gradient Implementation

In order to perform gradient descent, we need to define a loss function. As provided in the lecture, the $\mathcal{J}^{L_2}$ loss function is defined over a dataset $X = \{(\vec{x}^{[n]}, t^{[n]}) \mid n \in [1, N]\}$ that is defined as a list of tuples, as follows:

$$\mathcal{J}^{L_2} = \frac{1}{N}\sum_{n=1}^{N}(y^{[n]} - t^{[n]})^2 \tag{2}$$

where $y^{[n]}$ is the output of the network from Task 1 when inputting $\vec{x}^{[n]}$. Interestingly, however, we never explicitly need to compute the output of the loss function. It is only used to analytically compute the gradient as shown in the lecture.

The gradient is composed of two items, one for each layer. Particularly, for a given dataset X, the gradient of loss $\mathcal{J}^{L_2}$ is defined as:

$$\frac{\partial \mathcal{J}}{\partial w_{kd}^{(1)}} = \frac{2}{N}\sum_{n=1}^{N}(y^{[n]} - t^{[n]})w_k^{(2)} h_k^{[n]}(1 - h_k^{[n]})x_d^{[n]} \tag{3}$$

$$\frac{\partial \mathcal{J}}{\partial w_{k}^{(2)}} = \frac{2}{N}\sum_{n=1}^{N}(y^{[n]} - t^{[n]})h_k^{[n]} \tag{4}$$

Task 2 Implement a function that returns the gradient as defined in (3) and (4) for a given dataset X, and given weights $\Theta = (\mathbf{W}^{(1)}, \vec{w}^{(2)})$. Make sure that both parts of the gradient are computed. Make use of the function implemented in Task 1 where appropriate.

3 Gradient Descent

The procedure of gradient decent is the repeated application of two steps. First, the gradient of the loss $\nabla_\Theta \mathcal{J}^{L_2}$ is computed based on the current value of the parameters $\Theta = (\mathbf{W}^{(1)}, \vec{w}^{(2)})$. Second, the weights are updated by moving a small step into the direction of the negative gradient:

$$\Theta = \Theta - \eta \nabla_\Theta \mathcal{J} \tag{5}$$

As stopping criterion, we select the number of training epochs to be 10000.

Task 3 Implement a function that performs gradient descent with a given dataset X, given initial parameters Θ and a given learning rate η and returns the optimized parameters Θ^*.

4 Datasets

In total, we will test our gradient descent function with three different datasets. Particularly, we approximate

1. $X_1 : t = \cos(3x)$ for $x \in [-2, 2]$
2. $X_2 : t = e^{-x^2}$ for $x \in [-2, 2]$
3. $X_3 : t = x^5 + 3x^4 - 11x^3 - 27x^2 + 10x + 64$ for $x \in [-4.5, 3.5]$

Task 4 Generate dataset X_1 for $N = 50$ samples randomly drawn from range $x \in [-2, 2]$. Generate dataset X_2 for $N = 30$ samples randomly drawn from range $x \in [-2, 2]$. Generate dataset X_3 for $N = 200$ samples randomly drawn from range $x \in [-4.5, 3.5]$. Implement all three datasets as lists of tuples: $\{(\vec{x}^{[n]}, t^{[n]}) \mid 1 \leq n \leq N\}$.

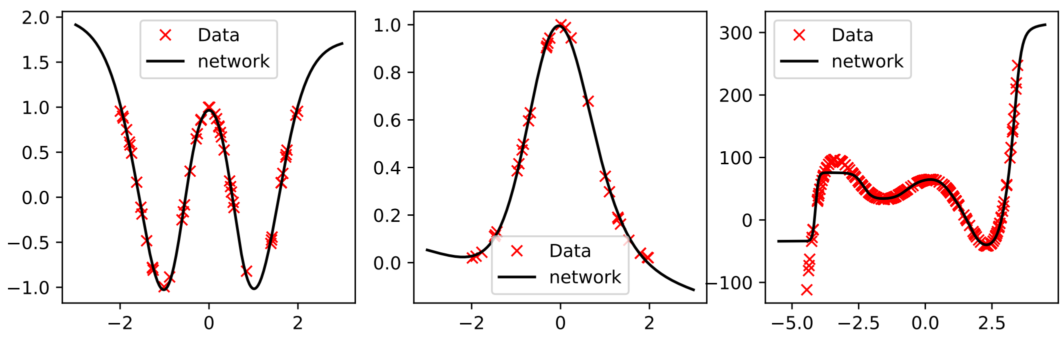

Figure 1: Exemplary solutions for the plotting Task 8.

5 Function Approximation

Finally, we want to make use of our gradient descent implementation to approximate our functions. In order to see our success, we want to plot the functions together with the data.

Task 5 (theoretical question) When looking at the example plots in Figure 1, how many hidden neurons K do we need in order to approximate the functions? Is there any difference between the three target functions?

Task 6 For each of the datasets, randomly initialize the parameters $\Theta_1, \Theta_2, \Theta_3 \in [-1, 1]$ according to the number of hidden neurons estimated in Task 7.

Task 7 Call the gradient descent function from Task 3 using each of the datasets X_1, X_2, X_3, the according created parameters $\Theta_1, \Theta_2, \Theta_3$ and a learning rate of $\eta = 0.1$. Store the resulting optimized weights $\Theta_1^*, \Theta_2^*, \Theta_3^*$ and the loss values.

6 Data and Function Plotting

Task 8 Implement a plotting function that takes a given dataset X, given parameters Θ and a defined range R. Each data sample $(x^{[n]}, t^{[n]})$ of the dataset is plotted as an "x". In order to plot the function that is approximated by the network, generate sufficient equally-spaced input values $x \in R$, compute the network output y for these inputs, and plot them with a line.

Task 9 For each of the datasets and their according optimized parameters, call the plotting function from Task 8. Use range $R = [-3, 3]$ for dataset X_1 and X_2, and range $R = [-5.5, 4.5]$ for dataset X_3. Note that the first element of range R should be the lowest x-location, and the second element of R the highest value for x. Did the networks approximate the functions? What can we do if not?