Machine Learning for Agricultural Applications

Assignment 3

Prof. Dr. Niels Landwehr
Dr. Julian Adolphs

Summer Term 2020

Release: 21.05.2020
Discussion: 28.05.2020

Task 1 – Artificial Neural Network [10 points]

Consider a simple artificial neural network (ANN):

$$\begin{aligned} f:\ & \mathbb{R} \to \mathbb{R} \\ x\ & \mapsto\ \mathbf{W}_2\, \sigma(\mathbf{W}_1 x + \mathbf{b}_1) + b_2\,. \end{aligned}$$

It consists of one hidden layer $\mathbf{W}_1 \in \mathbb{R}^{k\times 1}$, one output layer $\mathbf{W}_2 \in \mathbb{R}^{1\times k}$ and the two biases $\mathbf{b}_1 \in \mathbb{R}^k$ and $b_2 \in \mathbb{R}$. The activation function σ is given by the sigmoid function

$$\begin{aligned} \sigma:\ & \mathbb{R} \to \mathbb{R} \\ t\ & \mapsto\ \frac{1}{1+e^{-t}} \end{aligned}$$

and is applied element-wise to the the vector $\mathbf{W}_1 x + \mathbf{b}_1$ (i.e. on every entry).

The given training data are $\mathcal{Z} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$, x_i, $y_i \in \mathbb{R}$. The loss for the i-th sample pair is given by the mean squared error

$$\ell(x_i, y_i) = (f_\theta(x_i) - y_i)^2\ .$$

Hence, the setting is a nonlinear regression task.

Remark: The derivative of $\sigma(t) = \frac{1}{1+e^{-t}}$ holds $\sigma'(t) = \sigma(t)(1-\sigma(t))$. Also $\sigma'(t)$ is applied element-wise to a matrix, i.e. the product $\sigma(t)(1-\sigma(t))$ is meant element-wise (the so-called Hadamar product $\circ$, in python the * operator).

a) Draw a schematic picture of the ANN.

b) Show for $k = 2$ that the partial derivatives $\frac{\partial \ell}{\partial \mathbf{W}_1}$, $\frac{\partial \ell}{\partial \mathbf{W}_2}$, $\frac{\partial \ell}{\partial \mathbf{b}_1}$, $\frac{\partial \ell}{\partial b_2}$ of the loss function hold:

$$\begin{aligned} \frac{\partial \ell}{\partial \mathbf{W}_1} &= 2\,[f(x) - y]\, \mathbf{W}_2^T \circ \sigma'(\mathbf{W}_1 x + \mathbf{b}_1)\, x \\ \frac{\partial \ell}{\partial \mathbf{W}_2} &= 2\,[f(x) - y]\ (\sigma(\mathbf{W}_1 x + \mathbf{b}_1))^T \\ \frac{\partial \ell}{\partial \mathbf{b}_1} &= 2\,[f(x) - y]\, \mathbf{W}_2^T \circ \sigma'(\mathbf{W}_1 x + \mathbf{b}_1) \\ \frac{\partial \ell}{\partial b_2} &= 2\,[f(x) - y], \end{aligned}$$

where $\mathbf{W}_i^T$ is the transposed of matrix $\mathbf{W}_i$.

Hint: Use $k = 2$ and write all matrices explicitly, i.e.:

$$\mathbf{W}_1 = \begin{pmatrix} W_1^{(1)} \\ W_1^{(2)} \end{pmatrix},\quad \mathbf{b}_1 = \begin{pmatrix} b_1^{(1)} \\ b_1^{(2)} \end{pmatrix},\quad \mathbf{W}_2 = (W_2^{(1)}\ W_2^{(2)}).$$

Task 2 – Implement Simple Neural Network [15 points]

Write a python program, that implements the simple neural network from **task 1**. The ANN has one hidden layer with k nodes and 1 node in the output layer. Your neural network has to solve a regression problem ('curve fitting') using the quadratic loss as objective function. The data points are constructed with uniformly distributed random values for $x \in [-1, +1]$ and with the function $f_{\text{noisy}}(x)$, which is the Gauss-shaped function $f_{\text{org}}(x)$ with additional normal distributed noise:

$$f_{\text{org}}(x) = \frac{1}{1 + \exp(-10\,x^2)} - \frac{1}{2}$$
$$f_{\text{noisy}}(x) = f_{\text{org}}(x) + s\, f_{\text{normal}}(x)\,.$$

Use the derivatives from task 1 for the calculation of the backward steps and update the weights with that. Plot the loss function over the epochs and plot the data points and the function your neural net fitted to the data points. Try how different numbers of nodes of the hidden layer influence the fit result. You could also play with different functions for $f_{\text{org}}(x)$.

Hints:

```
def get_random_x(num_x):
    return np.random.rand(num_x)*2 - 1

def f_org(x):
    return 1.0/(np.exp(-10*x**2)+1) - 0.5

def f_noisy(x, s = 0.1):
    return f_org(x) + s*np.random.randn(len(x))
```

Task 3 – AutoDiff [15 points]

Write down and draw the graph of primitive operations for the function

$$f(x_0, x_1, w_0, w_1, w_2) = \left[\tanh\left(\frac{1}{w_0 x_0 + w_1 x_1 + w_2}\right) - 1\right]^2, \qquad x_0, x_1, w_0, w_1, w_2 \in \mathbb{R}.$$

Write down all local gradients and how to calculate the forward- and backward pass as shown in the lecture. Use the values $x_0 = 1.2$, $x_1 = 3.1$, $w_0 = 1.5$, $w_1 = -1.5$, $w_2 = 2.0$ and insert the results in your graph representation.

Also calculate the gradient analytically. For the latter show that $\frac{d}{dx}\tanh(x) = 1 - \tanh(x)^2$.

Task 4 – Implement AutoDiff [10 points]

Write a python script, that does the forward and backward pass calculations from **Task 3**. Use the values $x_0 = 1.2$, $x_1 = 3.1$, $w_0 = 1.5$, $w_1 = -1.5$, $w_2 = 2.0$ and compare the outputs with the result of the analytic derivative (do that also in your script).