Homework 3 - Group 076

Aprendizagem 2021/2022

1 Pen and Paper

1) Let $b^{[l]}$, $net^{[l]}$ and $a^{[l]} = \phi(net^{[l]})$ denote the vector of biases, net values and activations of the l-th layer, respectively (where ϕ is the activation function such that $\phi_i(net^{[l]}) = \tanh(net_i^{[l]})$). Let $W^{[l]} = [w_{ij}]$ be the matrix of weights w_{ij} that connect the j-th activation of layer $l-1$ to the i-th net of layer l.

- **Forward Propagation**

  Given that $a^{[l]} = \phi(net^{[l]}) = \phi(W^{[l]}a^{[l-1]} + b^{[l]})$ (for $i \in \{1, 2, 3\}$) and considering that $a^{[0]} = \mathbf{x}$:

$$a^{[1]} = \phi(W^{[1]}a^{[0]} + b^{[1]}) = \phi\left(\begin{bmatrix}1.0 & 1.0 & 1.0 & 1.0 & 1.0\\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0\\ 1.0 & 1.0 & 1.0 & 1.0 & 1.0\end{bmatrix}\begin{bmatrix}1\\1\\1\\1\\1\end{bmatrix} + \begin{bmatrix}1.0\\1.0\\1.0\end{bmatrix}\right)$$

$$= \phi\left(\begin{bmatrix}6.0\\1.0\\6.0\end{bmatrix}\right) = \begin{bmatrix}\tanh(6.0)\\ \tanh(1.0)\\ \tanh(6.0)\end{bmatrix} = \begin{bmatrix}0.99999\\0.76159\\0.99999\end{bmatrix}$$

$$a^{[2]} = \phi(W^{[2]}a^{[1]} + b^{[2]}) = \phi\left(\begin{bmatrix}1.0 & 1.0 & 1.0\\ 1.0 & 1.0 & 1.0\end{bmatrix}\begin{bmatrix}0.99999\\0.76159\\0.99999\end{bmatrix} + \begin{bmatrix}1.0\\1.0\end{bmatrix}\right) = \phi\left(\begin{bmatrix}3.76157\\3.76157\end{bmatrix}\right) = \begin{bmatrix}0.99892\\0.99892\end{bmatrix}$$

$$a^{[3]} = \phi(W^{[3]}a^{[2]} + b^{[3]}) = \phi\left(\begin{bmatrix}0.0 & 0.0\\ 0.0 & 0.0\end{bmatrix}\begin{bmatrix}0.99892\\0.99892\end{bmatrix} + \begin{bmatrix}0.0\\0.0\end{bmatrix}\right) = \phi\left(\begin{bmatrix}0.0\\0.0\end{bmatrix}\right) = \begin{bmatrix}0.0\\0.0\end{bmatrix}$$

- **Backward propagation**

  Consider the squared error loss $E = \frac{1}{2}\sum_{i=1}^{2}(z_i - \hat{z}_i)^2 = \frac{1}{2}\sum_{i=1}^{2}\left(z_i - a_i^{[3]}\right)^2 = \frac{1}{2}\|\mathbf{z} - a^{[3]}\|^2$ and define $\delta^{[l]} = \nabla_{net^{[l]}}E$. By the chain rule of derivation, we have, for layer $l \in \{1, 2\}$:

$$\begin{aligned}
\delta^{[l]} &= \nabla_{net^{[l]}}E = \nabla_{net^{[l]}}a^{[l]}\nabla_{a^{[l]}}net^{[l+1]}\nabla_{net^{[l+1]}}E\\
&= \operatorname{diag}(\tanh'(net_1^{[l]}), ..., \tanh'(net_{n_l}^{[l]}))(W^{[l+1]})^T\delta^{[l+1]}\\
&= \begin{bmatrix}\tanh'(net_1^{[l]}) & \cdots & \tanh'(net_{n_l}^{[l]})\end{bmatrix}^T \circ \left((W^{[l+1]})^T\delta^{[l+1]}\right)\\
&= \begin{bmatrix}1-\tanh^2(net_1^{[l]}) & \cdots & 1-1-\tanh^2(net_{n_l}^{[l]})\end{bmatrix}^T \circ \left((W^{[l+1]})^T\delta^{[l+1]}\right)\\
&= \begin{bmatrix}1-(a_1^{[l]})^2 & \cdots & 1-(a_{n_l}^{[l]})^2\end{bmatrix}^T \circ \left((W^{[l+1]})^T\delta^{[l+1]}\right)
\end{aligned} \tag{1}$$

  where n_l denotes the number of units in layer l and in the two last steps the equalities $\tanh'(x) = 1 - \tanh^2(x)$ and $a_i^{[l]} = \tanh(net_i^{[l]})$ were used. For the output layer ($l = 3$), using the equality $\nabla_x\left(\frac{1}{2}\|x\|^2\right) = x$ to compute $\nabla_{a^{[3]}}E$, we have:

$$\delta^{[3]} = \nabla_{net^{[3]}}a^{[3]}\nabla_{a^{[3]}}E = \begin{bmatrix}1-(a_1^{[3]})^2 & 1-(a_2^{[3]})^2\end{bmatrix}^T \circ (\mathbf{z} - a^{[3]}) \tag{2}$$

  Computing the deltas:

$$\delta^{[3]} = \begin{bmatrix}1.0\\1.0\end{bmatrix} \circ \begin{bmatrix}-1.0\\1.0\end{bmatrix} = \begin{bmatrix}-1.0\\1.0\end{bmatrix} \quad \delta^{[2]} = \begin{bmatrix}0.0\\0.0\end{bmatrix} \quad \delta^{[1]} = \begin{bmatrix}0.0\\0.0\\0.0\end{bmatrix}$$

  since $W^{[3]}$ is a null matrix and $\delta^{[2]}$ is a null vector (so $\delta^{[2]}$ and $\delta^{[2]}$ are null according to (1)).

- **Parameter update**

  To perform gradient descent on the weights and biases, we calculate the gradient of the error with respect to these parameters (for $l \in \{1, 2, 3\}$):

$$\frac{\partial E}{\partial w_{ij}^{[l]}} = \frac{\partial E}{\partial net_i^{[l]}} \frac{\partial net_i^{[l]}}{\partial w_{ij}^{[l]}} = \delta_i^{[l]} a_j^{[l-1]} \Rightarrow \nabla_{W^{[l]}} E = \delta^{[l]} (a^{[l-1]})^T \tag{3}$$

$$\nabla_{b^{[l]}} E = \nabla_{b^{[l]}} net^{[l]} \nabla_{net^{[l]}} E = \mathbb{I}^{n_l \times n_l} \delta^{[l]} = \delta^{[l]} \tag{4}$$

  since for $x, y \in \mathbb{R}^n$, we have that $xy^T = [x_i y_j]$. Noting that deltas in layers 1 and 2 are zero valued vectors, we conclude that the gradients with respects to parameters in these layers are also zero valued (according to (3) and (4)), and so they don't change:

$$(W^{[1]})^{\text{new}} = (W^{[1]})^{\text{old}} = \begin{bmatrix} 1.0 & 1.0 & 1.0 & 1.0 & 1.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 1.0 & 1.0 & 1.0 & 1.0 & 1.0 \end{bmatrix} \quad (b^{[1]})^{\text{new}} = (b^{[1]})^{\text{old}} = \begin{bmatrix} 1.0 \\ 1.0 \\ 1.0 \end{bmatrix}$$

$$(W^{[2]})^{\text{new}} = (W^{[2]})^{\text{old}} = \begin{bmatrix} 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \end{bmatrix} \quad (b^{[2]})^{\text{new}} = (b^{[2]})^{\text{old}} = \begin{bmatrix} 1.0 \\ 1.0 \end{bmatrix}$$

  And, setting $\eta = 0.1$, for the output layer, we have:

$$\nabla_{W^{[3]}} E = \begin{bmatrix} -1.0 \\ 1.0 \end{bmatrix} \begin{bmatrix} 0.99892 & 0.99892 \end{bmatrix} = \begin{bmatrix} -0.99892 & -0.99892 \\ 0.99892 & 0.99892 \end{bmatrix}$$

$$\nabla_{b^{[3]}} E = \begin{bmatrix} -1.0 \\ 1.0 \end{bmatrix}$$

$$(W^{[3]})^{\text{new}} = (W^{[3]})^{\text{old}} - \eta \nabla_{W^{[3]}} E = \begin{bmatrix} 0.09989 & 0.09989 \\ -0.09989 & -0.09989 \end{bmatrix}$$

$$(b^{[3]})^{\text{new}} = (b^{[3]})^{\text{old}} - \eta \nabla_{b^{[3]}} E = \begin{bmatrix} 0.1 \\ -0.1 \end{bmatrix}$$

2)

- **Forward Propagation**

  The only change in this stage occurs in the activation function in the last layer, which is now softmax:

$$a^{[3]} = \text{softmax}(net^{[3]}) = \sigma(net^{[3]}) = \left(\frac{e^{net_1^{[3]}}}{\sum_{j=1}^{2} e^{net_j^{[3]}}}, \frac{e^{net_2^{[3]}}}{\sum_{j=1}^{2} e^{net_j^{[3]}}} \right)$$

  And so we have:

$$a^{[1]} = \begin{bmatrix} 0.99999 \\ 0.76159 \\ 0.99999 \end{bmatrix} \quad a^{[2]} = \begin{bmatrix} 0.99892 \\ 0.99892 \end{bmatrix} \quad a^{[3]} = \sigma(net^{[3]}) = \sigma\left(\begin{bmatrix} 0.0 \\ 0.0 \end{bmatrix}\right) = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$$

- **Backward propagation**

  With the change in the output layer activation function, we only need to re-compute $\delta^{[3]}$. Given the cross entropy loss $E = -\sum_{k=1}^{2} z_k \log(a_k^{[3]})$ and the fact that:

$$\frac{\partial \sigma_j(\mathbf{x})}{\partial x_i} = \begin{cases} -\sigma_i(\mathbf{x})\sigma_j(\mathbf{x}), & i \neq j \\ \sigma_i(\mathbf{x})(1 - \sigma_i(\mathbf{x})), & i = j \end{cases}$$

  we have:

$$\frac{\partial E}{\partial net_i^{[3]}} = \sum_{k=1}^{2} \frac{\partial E}{\partial a_k^{[3]}} \frac{\partial a_k^{[3]}}{\partial net_i^{[3]}} = \sum_{\substack{k=1 \\ k \neq i}}^{2} \left(\frac{z_k}{a_k^{[3]}} a_k^{[3]} a_i^{[3]} \right) - \frac{z_i}{a_i^{[3]}} a_i^{[3]} (1 - a_i^{[3]}) = a_i^{[3]} \left(\sum_{k=1}^{2} z_k \right) - z_i = a_i^{[3]} - z_i$$

which implies that $\nabla_{net^{[3]}} E = a^{[3]} - \mathbf{z}$. Computing the deltas, we have:

$$\delta^{[3]} = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} - \begin{bmatrix} 1.0 \\ 0.0 \end{bmatrix} = \begin{bmatrix} -0.5 \\ 0.5 \end{bmatrix} \quad \delta^{[2]} = \begin{bmatrix} 0.0 \\ 0.0 \end{bmatrix} \quad \delta^{[1]} = \begin{bmatrix} 0.0 \\ 0.0 \\ 0.0 \end{bmatrix}$$

where we shorten the computation of $\delta^{[1]}$ and $\delta^{[2]}$ following the same arguments used in the previous question.

- **Parameter update**

  Using the same argument as before, the following parameters don't register changes:

$$(W^{[1]})^{\text{new}} = (W^{[1]})^{\text{old}} = \begin{bmatrix} 1.0 & 1.0 & 1.0 & 1.0 & 1.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 1.0 & 1.0 & 1.0 & 1.0 & 1.0 \end{bmatrix} \quad (b^{[1]})^{\text{new}} = (b^{[1]})^{\text{old}} = \begin{bmatrix} 1.0 \\ 1.0 \\ 1.0 \end{bmatrix}$$

$$(W^{[2]})^{\text{new}} = (W^{[2]})^{\text{old}} = \begin{bmatrix} 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \end{bmatrix} \quad (b^{[2]})^{\text{new}} = (b^{[2]})^{\text{old}} = \begin{bmatrix} 1.0 \\ 1.0 \end{bmatrix}$$

  According to (3) and (4), for a learning rate of $\eta = 0.1$, we have:

$$\nabla_{W^{[3]}} E = \begin{bmatrix} -0.5 \\ 0.5 \end{bmatrix} \begin{bmatrix} 0.99892 & 0.99892 \end{bmatrix} = \begin{bmatrix} -0.49946 & -0.49946 \\ 0.49946 & 0.49946 \end{bmatrix}$$

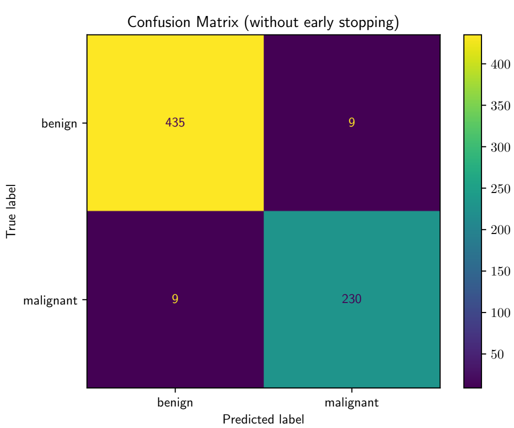
$$\nabla_{b^{[3]}} E = \begin{bmatrix} -0.5 \\ 0.5 \end{bmatrix}$$

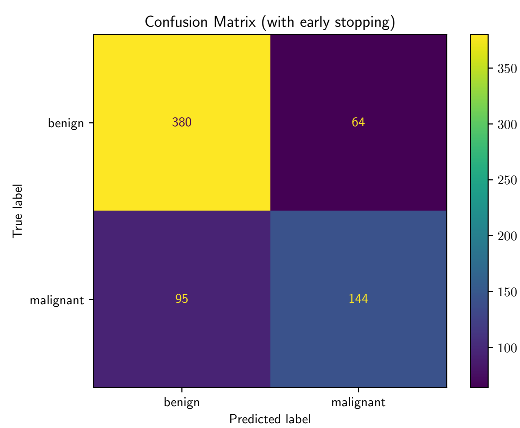
$$(W^{[3]})^{\text{new}} = (W^{[3]})^{\text{old}} - \eta \nabla_{W^{[3]}} E = \begin{bmatrix} 0.04995 & 0.04995 \\ -0.04995 & -0.04995 \end{bmatrix}$$

$$(b^{[3]})^{\text{new}} = (b^{[3]})^{\text{old}} - \eta \nabla_{b^{[3]}} E = \begin{bmatrix} 0.05 \\ -0.05 \end{bmatrix}$$

2 Programming and critical analysis

2)

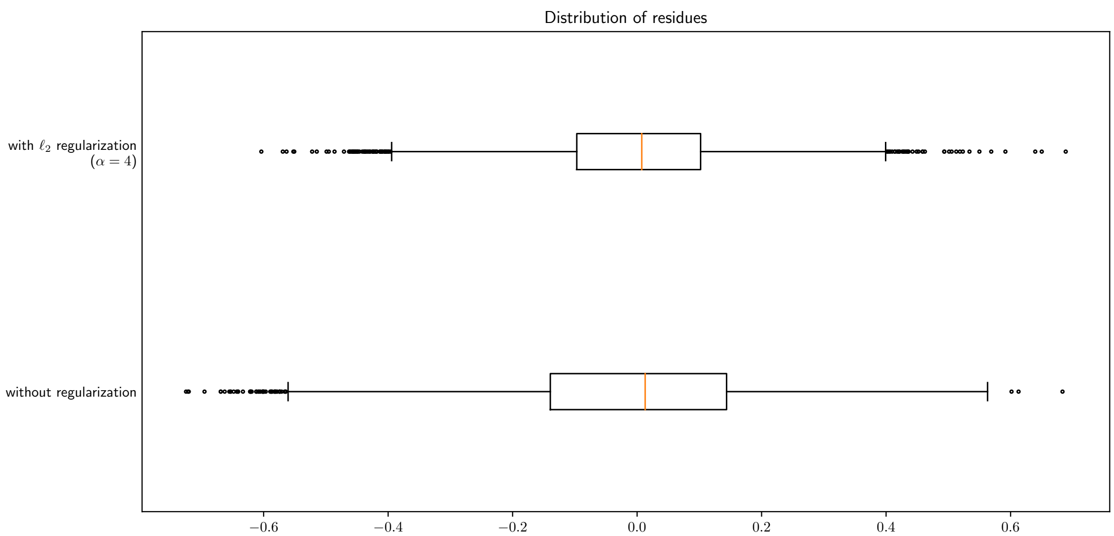




The confusion matrices above reveal a decrease in accuracy when there is early stopping (there are fewer true positives and true negatives). This behaviour may be due to:

- the presence of overfitting to the validation sets across dataset folds, since early stopping aims to maximize accuracy over these subsets;
- the scarcity of instances that cover all possible values for each feature, which is exacerbated by creating yet another subset of the dataset in each fold. Consequently, some parameters may be left undertrained, leading to more misclassifications.

3)



In order to reduce the observed error, we can resort to strategies that use k-fold cross-validation to try out and select the best values for the following network hyperparameters:

- regularization penalty constant α, given the decrease in residue variability and median absolute value shown above;
- learning rate η (absolute value and/or decay rate), so as to avoid error function local minima (note that in the presence of regularization, the error function is more likely to assess performance over the whole population);
- random seeds, which determine initial weights and biases, and, consequently, the quality of the found error function minimum;
- variance of zero-mean Gaussian noise added to the weights (for the purpose of reducing overfitting).

3 Appendix

```python
import numpy as np
import pandas as pd
from scipy.io import arff
from sklearn import neural_network, model_selection, metrics
import matplotlib.pyplot as plt
plt.rcParams["text.usetex"] = True

def load_data(filename):
    dataset = arff.loadarff(filename)
    dataset = pd.DataFrame(dataset[0])
    str_columns = [col for col in dataset.columns if dataset[col].dtype == "object"]
    dataset[str_columns] = dataset[str_columns].apply(lambda x: x.str.decode('utf8'))
    dataset = dataset.dropna()
    return dataset

def mlp_predict(mlp_model ,inputs, outputs, folds, early_stp, alpha):
    clf = mlp_model(activation = 'relu',    hidden_layer_sizes = (3, 2), early_stopping = early_stp,\
                                            alpha = alpha, random_state = 76, max_iter = 1500)
    return model_selection.cross_val_predict(clf, inputs, outputs, cv = folds)

def mlp_conf_matrix(inputs, outputs, folds, early_stp):
    outputs_pred = mlp_predict(neural_network.MLPClassifier, inputs, outputs, folds, early_stp, 0.4)
    disp = metrics.ConfusionMatrixDisplay.from_predictions(outputs, outputs_pred)
    disp.ax_.set(title=f'Confusion Matrix (with early stopping)' if early_stp else \
                       f'Confusion Matrix (without early stopping)')
    plt.savefig(f"output/mlp_conf_matrix_{early_stp}.pdf")

def residue_dist_bp(inputs, outputs, folds):
    res_regularized = outputs - mlp_predict(neural_network.MLPRegressor, \
                                            inputs, outputs, folds, True, 4)
    res_nonregularized = outputs - mlp_predict(neural_network.MLPRegressor, \
                                            inputs, outputs, folds, True, 0)
    fig, ax = plt.subplots()
    bp = ax.boxplot([res_regularized, res_nonregularized], \
                    vert = False, flierprops={'marker': 'o', 'markersize': 2})
    ax.set_yticklabels(['without regularization','with $\\ell_2$ regularization \n ($\\alpha = 4$)'])
    ax.tick_params(axis = u'y', length = 0)
    ax.set(title=f"Distribution of residues")
    fig.set_size_inches(12, 6)
    plt.savefig("output/residue_boxplot.pdf")

kf = model_selection.KFold(n_splits = 5, shuffle = True, random_state = 0)
breast_data = load_data("../data/breast.w.arff")
inputs_breast = breast_data.iloc[:, :-1].to_numpy()
outputs_breast = breast_data.iloc[:, [-1]].to_numpy().T.flatten()
mlp_conf_matrix(inputs_breast, outputs_breast, kf, True)
mlp_conf_matrix(inputs_breast, outputs_breast, kf, False)
kin_data = load_data("../data/kin8nm.arff")
inputs_kin = kin_data.iloc[:, :-1].to_numpy()
outputs_kin = kin_data.iloc[:, [-1]].to_numpy().T.flatten()
residue_dist_bp(inputs_kin, outputs_kin, kf)
```