TÉCNICO LISBOA

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 (with $\phi_i(net^{[l]}) = \tanh(net_i^{[l]})$ being the activation function). 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 $a^{[l]} = \phi(net^{[l]}) = \phi(W^{[l]}a^{[l-1]} + b^{[l]})$ $(i \in \{1,2,3\})$, 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 $E = \frac{1}{2}\sum_{i=1}^{2}(\mathbf{z}_i - \hat{\mathbf{z}}_i)^2 = \frac{1}{2}\sum_{i=1}^{2}\left(\mathbf{z}_i - a_i^{[3]}\right)^2 = \frac{1}{2}\|\mathbf{z} - a^{[3]}\|^2$ and define $\delta^{[l]} = \nabla_{net^{[l]}}E$. By the chain rule of derivation, we have, for layer $l \in \{1,2\}$, with n_l denoting the number of units in layer l:

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

  where 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:

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

  Computing the deltas:

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

  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 ($l \in \{1, 2, 3\}$):

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

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

since for $x, y \in \mathbb{R}^n$, we have that $xy^T = [x_i y_j]$. Thus, setting the learning rate η to 0.1:

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

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

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

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

As deltas in layers 1 and 2 are zero valued vectors, the error gradient with respect to the weights and biases of these layers is also zero valued, acordding to (3) and (4). Thus, these parameters don't change:

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

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

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:

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

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

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

$$\begin{aligned}
\nabla_{W^{[3]}} E &= \begin{bmatrix} -0.5 \\ 0.5 \end{bmatrix} \begin{bmatrix} 0.99892 & 0.99892 \end{bmatrix} = \begin{bmatrix} -0.49946 & -0.49946 \\ 0.49946 & 0.49946 \end{bmatrix} \\
\nabla_{b^{[3]}} E &= \begin{bmatrix} -0.5 \\ 0.5 \end{bmatrix} \\
(W^{[3]})^{\text{new}} &= (W^{[3]})^{\text{old}} - \eta \nabla_{W^{[3]}} E = \begin{bmatrix} 0.04995 & 0.04995 \\ -0.04995 & -0.04995 \end{bmatrix} \\
(b^{[3]})^{\text{new}} &= (b^{[3]})^{\text{old}} - \eta \nabla_{b^{[3]}} E = \begin{bmatrix} 0.05 \\ -0.05 \end{bmatrix}
\end{aligned}$$

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

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

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

2 Programming and critical analysis

4) Inserir aqui as respostas

3 Appendix

```
import numpy as np
import pandas as pd
from scipy.io import arff
from sklearn import neural_network, model_selection, preprocessing, 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

#  MLP with l2 regularization, RELU activation function, 2 hidden layers of size 3,2 and remaining default pa
def mlp_predict(mlp_model ,inputs, outputs, folds, early_stopping, alpha):
    clf = mlp_model(activation = 'relu',    hidden_layer_sizes = (3, 2), \
                                            random_state = 76, \
                                            early_stopping = early_stopping, \
                                            alpha = alpha, \
                                            max_iter = 1500)
    return model_selection.cross_val_predict(clf, inputs, outputs, cv = folds)

# Plots the confusion matrix for a given MLP with/without early stopping
def mlp_conf_matrix(inputs, outputs, folds, early_stopping):
    outputs_pred = mlp_predict(neural_network.MLPClassifier, inputs, outputs, folds, early_stopping, 1)
    disp = metrics.ConfusionMatrixDisplay.from_predictions(outputs, outputs_pred)
    disp.ax_.set(title=f'Confusion Matrix {"(with early stopping)" if early_stopping else "(without early sto
    plt.savefig(f"output/mlp_conf_matrix_{early_stopping}.pdf")

# Returns the residues of the distribution with/without regularization
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)
    #np.savez("res.txt", res_regularized, res_nonregularized)

    #npzfile = np.load("res.txt.npz")
    #res_regularized = npzfile["arr_0"]
    #res_nonregularized = npzfile["arr_1"]

    fig, ax = plt.subplots()
    bp = ax.boxplot([res_regularized, res_nonregularized], vert = False, flierprops={'marker': 'o', 'markers

    ax.set_yticklabels(['without regularization','with regularization \n ($\\alpha = 4$)'])
    ax.tick_params(axis = u'y', length = 0)
    fig.set_size_inches(12, 6)

    plt.savefig("output/residue_boxplot.pdf")
```

```
def main():
    kf = model_selection.KFold(n_splits = 5, shuffle = True, random_state = 0)

    # 2) =====
    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)

    # 3) =====
    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)

<<<<<<< HEAD
=======
    fig, disp = plt.subplots()
    disp.boxplot(residues, vert=False, patch_artist=True)
    disp.set_yticklabels(['with regularization','without regularization'])
    disp.set_title('Comparison of residues in the presence and absence of regularization')
    plt.show()
>>>>>>> 329bb80e7f23b930bb831691974740f7b1df11c3

if __name__ == "__main__":
    main()
```