TÉCNICO LISBOA

Aprendizagem 2021/22

Homework III – Group 010

I. Pen-and-paper

1) a)

Forward Propagation

$Z^{[1]} = W^{[1]}X^{[0]} + b^{[1]} = [6\ 1\ 6]^T$ $\quad X^{[1]} = tanh(Z^{[1]}) = [0.99998771\ \ 0.76159416\ \ 0.99998771]^T$

$Z^{[2]} = W^{[2]}X^{[1]} + b^{[2]} = [3.76156958\ \ 3.76156958]^T$ $\quad X^{[2]} = tanh(Z^{[2]}) = [0.99891972\ 0.9989197]^T$

$Z^{[3]} = W^{[3]}X^{[2]} + b^{[3]} = [0\ 0]^T$ $\quad X^{[3]} = tanh(Z^{[3]}) = [0\ 0]^T$

Stochastic gradient descent update

Updates: $W^{[i]}_{new} = W^{[i]}_{old} - \eta \frac{\partial E}{\partial W^{[i]}}$

$b^{[i]}_{new} = b^{[i]}_{old} - \eta \frac{\partial E}{\partial b^{[i]}}$

Hyperbolic tangent derivative: $\frac{\partial}{\partial x} tanh(x) = 1 - tanh(x)^2$

Derivatives: $\frac{\partial Z^{[i]}}{\partial X^{[i-1]}} = W^{[i]}$ $\quad \frac{\partial X^{[i]}}{\partial Z^{[i]}} = 1 - tanh(Z^{[i]})^2$

$\frac{\partial Z^{[i]}}{\partial W^{[i]}} = X^{[i-1]}$ $\quad \frac{\partial Z^{[i]}}{\partial b^{[i]}} = 1$

$\frac{\partial E}{\partial X^{[i]}} = X^{[i]} - t$

Loss function: $E = \frac{1}{2}(X^{[3]} - t)^2$

Delta function: $\delta^{[i]} = \frac{\partial E}{\partial X^{[i]}} \circ \frac{\partial X^{[i]}}{\partial Z^{[i]}} \rightarrow last\ layer$

$\delta^{[i]} = \frac{\partial Z^{[i+1]}}{\partial X^{[i]}}^T \cdot \delta^{[i+1]} \circ \frac{\partial X^{[i]}}{\partial Z^{[i]}} \rightarrow intermediate\ layers$

- **Delta computation**

$$\delta^{[3]} = (X^{[3]} - t) \circ (1 - tanh(Z^{[3]})^2) = [-1\ \ 1]^T$$

$$\delta^{[2]} = \frac{\partial Z^{[3]}}{\partial X^{[2]}}^T \cdot \delta^{[3]} \circ \frac{\partial X^{[2]}}{\partial Z^{[2]}} = W^{[3]^T} \cdot \delta^{[3]} \circ (1 - tanh(Z^{[2]})^2) = [0\ \ 0]^T$$

$$\delta^{[1]} = \frac{\partial Z^{[2]}}{\partial X^{[1]}}^T \cdot \delta^{[2]} \circ \frac{\partial X^{[1]}}{\partial Z^{[1]}} = W^{[2]^T} \cdot \delta^{[2]} \circ (1 - tanh(Z^{[1]})^2) = [0\ \ 0]^T$$

- **Weights and biases update**

$\frac{\partial E}{\partial W^{[1]}} = \delta^{[1]} \cdot \frac{\partial Z^{[1]}}{\partial W^{[1]}}^T = \delta^{[1]} \cdot X^{[0]^T}$ $\quad \frac{\partial E}{\partial b^{[1]}} = \delta^{[1]} \cdot \frac{\partial Z^{[1]}}{\partial b^{[1]}}^T = \delta^{[1]} \cdot 1 = \delta^{[1]}$

$W^{[1]}_{new} = W^{[1]}_{old} - \eta \frac{\partial E}{\partial W^{[1]}} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$ $\quad b^{[1]}_{new} = b^{[1]}_{old} - \eta \frac{\partial E}{\partial b^{[1]}} = [1\ \ 1\ \ 1]^T$

$\frac{\partial E}{\partial W^{[2]}} = \delta^{[2]} \cdot \frac{\partial Z^{[2]}}{\partial W^{[2]}}^T = \delta^{[2]} \cdot X^{[1]^T}$ $\quad \frac{\partial E}{\partial b^{[2]}} = \delta^{[2]} \cdot \frac{\partial Z^{[3]}}{\partial b^{[2]}}^T = \delta^{[2]} \cdot 1 = \delta^{[2]}$

$W^{[2]}_{new} = W^{[2]}_{old} - \eta \frac{\partial E}{\partial W^{[2]}} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$ $\quad b^{[2]}_{new} = b^{[2]}_{old} - \eta \frac{\partial E}{\partial b^{[2]}} = [1\ \ 1]^T$

$\frac{\partial E}{\partial W^{[3]}} = \delta^{[3]} \cdot \frac{\partial Z^{[3]}}{\partial W^{[3]}}^T = \delta^{[3]} \cdot X^{[2]^T}$ $\quad \frac{\partial E}{\partial b^{[3]}} = \delta^{[3]} \cdot \frac{\partial Z^{[3]}}{\partial b^{[3]}}^T = \delta^{[3]} \cdot 1 = \delta^{[3]}$

$W^{[3]}_{new} = W^{[3]}_{old} - \eta \frac{\partial E}{\partial W^{[3]}} = \begin{bmatrix} 0.09989197 & 0.09989197 \\ -0.09989197 & -0.09989197 \end{bmatrix}$ $\quad b^{[3]}_{new} = b^{[3]}_{old} - \eta \frac{\partial E}{\partial b^{[3]}} = [0.1\ \ -0.1]^T$

Aprendizagem 2021/22

Homework III – Group 010

b)

Forward Propagation

$Z^{[1]} = W^{[1]}X^{[0]} + b^{[1]} = [6\ 1\ 6]^T$ $\quad X^{[1]} = tanh(Z^{[1]}) = [0.99998771\ \ 0.76159416\ \ 0.99998771]^T$

$Z^{[2]} = W^{[2]}X^{[1]} + b^{[2]} = [3.76156958\ \ 3.76156958]^T$ $\quad X^{[2]} = tanh(Z^{[2]}) = [0.99891972\ 0.9989197]^T$

$Z^{[3]} = W^{[3]}X^{[2]} + b^{[3]} = [0\ 0]^T$ $\quad X^{[3]} = softmax(Z^{[3]}) = [0.5\ \ 0.5]^T$

Stochastic gradient descent update

Updates: $W^{[i]}_{new} = W^{[i]}_{old} - \eta\frac{\partial E}{\partial W^{[i]}}$

$b^{[i]}_{new} = b^{[i]}_{old} - \eta\frac{\partial E}{\partial b^{[i]}}$

Loss function: $E = -\sum_{i=1}^{d} t_i log\, x_i^{[3]}$

Delta function: $\delta^{[3]}_j = \frac{\partial E}{\partial z_i} = \frac{\partial}{\partial z_i}(-\sum_{i=1}^{d} t_i log x_i^{[3]}) \rightarrow last\ layer$

$\delta^{[i]} = \frac{\partial Z^{[i+1]}}{\partial X^{[i]}}^T \cdot \delta^{[i+1]} \circ \frac{\partial X^{[i]}}{\partial Z^{[i]}} \rightarrow intermediate\ layers$

Hyperbolic tangent derivative: $\frac{\partial}{\partial x}tanh(x) = 1 - tanh(x)^2$

Derivatives: $\frac{\partial Z^{[i]}}{\partial X^{[i-1]}} = W^{[i]}$

$\frac{\partial Z^{[i]}}{\partial W^{[i]}} = X^{[i-1]}$

$\frac{\partial X^{[i]}}{\partial Z^{[i]}} = 1 - tanh(Z^{[i]})^2$

$\frac{\partial Z^{[i]}}{\partial b^{[i]}} = 1$

Softmax activation function:

In the softmax activation function each unit's output depends not only on its input but on the inputs of the other units.

$$softmax([z_1 \dots z_n]) = [x_1 \dots x_n] \quad , where\ x_i = \frac{exp(z_i)}{\sum_{k=1}^{i} z_k}$$

The derivatives of x_i with respect to a z_i are as follows:

$$\frac{\partial x_i}{\partial z_i} = x_i(1 - x_i)\ , when\ i = j \qquad \frac{\partial x_i}{\partial z_i} = -\ x_i x_j\ , when\ i \neq j$$

- **Delta computation**

  The delta from the last layer will differ because the error function is now the cross-entropy and softmax is the output activation function. The other deltas end up being the same as the ones in the last exercise because $W^{[3]}$ is a null matrix and the difference in $\delta^{[3]}$ becomes irrelevant for their determination.

$$\delta^{[3]}_j = \frac{\partial E}{\partial z_i} = \frac{\partial}{\partial z_i}(-\sum_{k=1}^{d} t_k log x_k^{[3]}) = -\sum_{k=1}^{d} t_k \frac{\partial}{\partial z_i} log x_k^{[3]} = -\sum_{k=1}^{d} t_k \frac{1}{x_k^{[3]}} \frac{\partial x_k^{[3]}}{\partial z_i} =$$

$$= -\sum_{k=i}^{d} t_k \frac{1}{x_k^{[3]}}(x_k^{[3]}(1 - x_k^{[3]})) - \sum_{k \neq i}^{d} t_k \frac{1}{x_k^{[3]}}(-x_i^{[3]} x_k^{[3]}) = -t_i + t_i x_i^{[3]} + \sum_{k \neq i}^{d} t_k x_i^{[3]} = -t_i + x_i^{[3]} + \sum_{k=1}^{d} t_k =$$

$$= x_i^{[3]} - t_i$$

$$\delta^{[3]} = (X^{[3]} - t) = [-\ 0.5\ \ 0.5]^T$$

$$\delta^{[2]} = \frac{\partial Z^{[3]}}{\partial X^{[2]}}^T \cdot \delta^{[3]} \circ \frac{\partial X^{[2]}}{\partial Z^{[2]}} = W^{[3]^T} \cdot \delta^{[3]} \circ (1 - tanh(Z^{[2]})^2) = [0\ \ 0]^T$$

$$\delta^{[1]} = \frac{\partial Z^{[2]}}{\partial X^{[1]}}^T \cdot \delta^{[2]} \circ \frac{\partial X^{[1]}}{\partial Z^{[1]}} = W^{[2]^T} \cdot \delta^{[2]} \circ (1 - tanh(Z^{[1]})^2) = [0\ \ 0]^T$$

- **Weights and biases update**

$$\frac{\partial E}{\partial W^{[1]}} = \delta^{[1]} \cdot \frac{\partial Z^{[1]}}{\partial W^{[1]}}^T = \delta^{[1]} \cdot X^{[0]^T}$$

$$W^{[1]}_{new} = W^{[1]}_{old} - \eta \frac{\partial E}{\partial W^{[1]}} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$\frac{\partial E}{\partial b^{[1]}} = \delta^{[1]} \cdot \frac{\partial Z^{[1]}}{\partial b^{[1]}}^T = \delta^{[1]} \cdot 1 = \delta^{[1]}$$

$$b^{[1]}_{new} = b^{[1]}_{old} - \eta \frac{\partial E}{\partial b^{[1]}} = [1\ \ 1\ \ 1]^T$$

$$\frac{\partial E}{\partial W^{[2]}} = \delta^{[2]} \cdot \frac{\partial Z^{[2]}}{\partial W^{[2]}}^T = \delta^{[2]} \cdot X^{[1]^T}$$

$$W^{[2]}_{new} = W^{[2]}_{old} - \eta \frac{\partial E}{\partial W^{[2]}} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

$$\frac{\partial E}{\partial b^{[2]}} = \delta^{[2]} \cdot \frac{\partial Z^{[3]}}{\partial b^{[2]}}^T = \delta^{[2]} \cdot 1 = \delta^{[2]}$$

$$b^{[2]}_{new} = b^{[2]}_{old} - \eta \frac{\partial E}{\partial b^{[2]}} = [1\ \ 1]^T$$

$$\frac{\partial E}{\partial W^{[3]}} = \delta^{[3]} \cdot \frac{\partial Z^{[3]}}{\partial W^{[3]}}^T = \delta^{[3]} \cdot X^{[2]^T}$$

$$W^{[3]}_{new} = W^{[3]}_{old} - \eta \frac{\partial E}{\partial W^{[3]}} = \begin{bmatrix} 0.04994599 & 0.04994599 \\ -0.04994599 & -0.04994599 \end{bmatrix}$$

$$\frac{\partial E}{\partial b^{[3]}} = \delta^{[3]} \cdot \frac{\partial Z^{[3]}}{\partial b^{[3]}}^T = \delta^{[3]} \cdot 1 = \delta^{[3]}$$

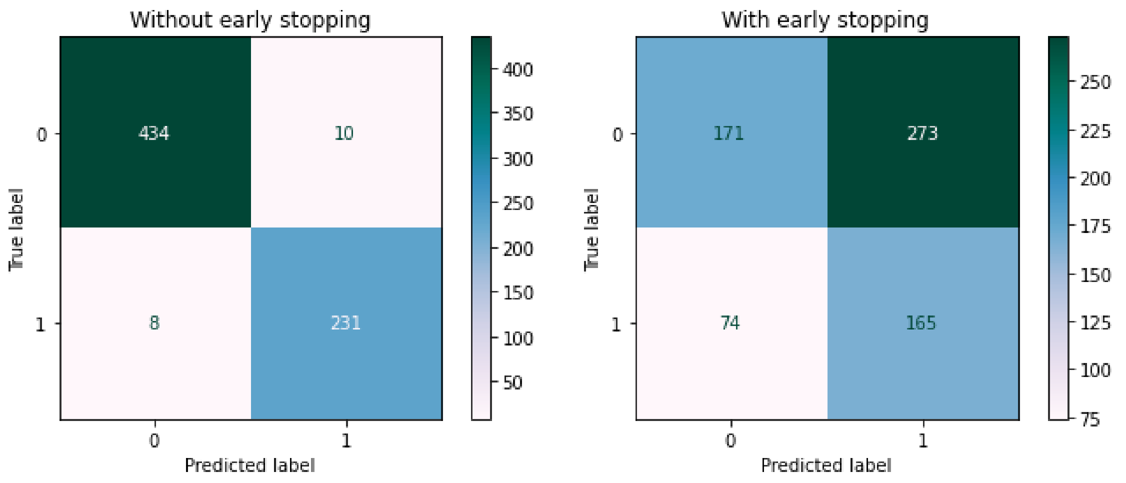
$$b^{[3]}_{new} = b^{[3]}_{old} - \eta \frac{\partial E}{\partial b^{[3]}} = [0.05\ -\ 0.05]^T$$

Aprendizagem 2021/22

Homework III – Group 010

II. Programming and critical analysis

2) Early stopping is an optimization technique used to avoid overfitting and improve generalization of neural networks. The main idea behind early stopping is to train on the train set but stop training at the point when performance on a validation set starts to degrade, in other words, stop training before the model starts to overfit.

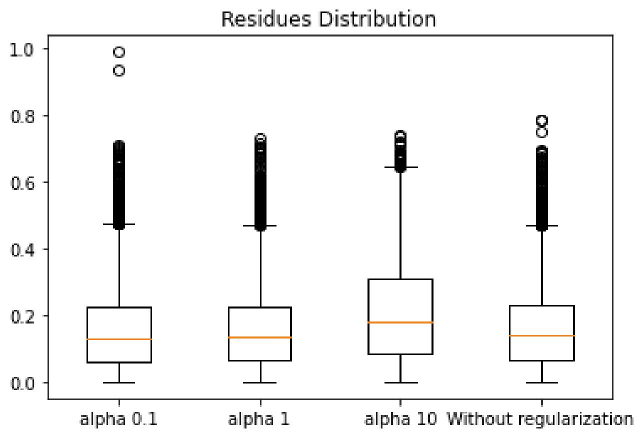


However, the plots above do not show what was expected. The model trained without early stopping is more accurate than the one with early stopping. This may be happening for two reasons:

- In small datasets, when early stopping is used, the amount of data used for training gets even smaller because some part of it goes to validation. Because of this, the model may not have enough data to train and generalize the population well enough, which leads to poorer performance on the train set.

- We are looking at a simple neural network that is not predisposed to overfitting. In this case, using early stopping is not needed and will only make the train set smaller and have the consequences stated above.

3)



Aprendizagem 2021/22

Homework III – Group 010

The MLP regressor error be minimized by multiple factors:

- The **regularization factor** is very important when trying to minimize the overfit of the training data. If the weights of a network are too large it may be an unstable network where small changes to the input can lead to large changes in the output. A solution to this problem is keeping the weights small by using regularization. Instead of adding each weight to the penalty directly, they can be weighted using a hyperparameter called alpha that controls how much attention should be paid to the penalty in the learning process. If the penalty is too strong, the model will underestimate the weights and underfit the problem. If the penalty is too weak, the model will be allowed to overfit the training data. For this MLP Regressor the differences of performance for different regularization terms are not statistically significant. We can see that the error using alpha = 10 is, to some extent, higher, maybe because it is too high of a value and makes the model underfit the train test but this difference is not significant. By looking at the plots we can conclude that changing the regularization term is not the most effective way to optimize this specific MLP Regressor and knowing this, the next points may be better approaches to optimize this model.

- Choosing **good initial weights** values is essential for the optimization of the loss function, which is fundamental for the regressor to make good predictions. Starting the training of the neural network from a good initial state makes it more probable to reach a deep minimum of the error function, instead of getting stuck on a local minimum. Because of this, often, the learning process is repeated with various initial weights. Also, some studies show that introducing prior knowledge in the initial weights may in some cases improve generalization performance.

- The **activation functions** chosen are also critical to control how well the model learns the train dataset. The most common activation functions used in hidden layers are ReLU, Sigmoid and Tanh and these last two can make the model more susceptible to problems during training, via the so-called vanishing gradients problem. Despite that, this model is not very complex and the vanishing problem becomes absent so, perhaps, using one of these two last activation functions could improve performance. For the output layer, the choice is made based on the type of prediction problem that is being solved and the most common ones are Sigmoid, Softmax for classification and Linear for regression. For this specific problem, using a no-op activation would result in a lower error, although it is not significant.

- The number of layers and nodes- **the network's complexity** - is also important to secure the ability of the network to generalize. This network may be too simple because it has only two hidden layers with little nodes and we know that the dataset is highly non-linear. In order to achieve the non-linearity necessary to generalize this dataset well we might need a more complex model.

- A simple way for minimizing the error of any model is to **increase the train set**. Having more data to train only makes the model generalize better to the population since it has more information and, as a consequence, it can never become worse, only more accurate. Nevertheless, gathering more data is not always easy and sometimes is even impossible.

Aprendizagem 2021/22

Homework III – Group 010

III. APPENDIX

```
import pandas as pd
from scipy.io.arff import loadarff
from sklearn.model_selection import KFold, StratifiedKFold
import numpy as np
from sklearn.neural_network import MLPClassifier, MLPRegressor
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

#----------------------- Confusion matrices w/ and w/o early stopping --------------------------#
raw_data = loadarff('breast.w.arff')
df_data = pd.DataFrame(raw_data[0]).dropna()  # converting data to a pandas DataFrame
df_data['Class'].replace({b'malignant': 1, b'benign': 0}, inplace=True)
data, target = df_data.drop(columns='Class').values, df_data['Class'].values

predicted_targets1, predicted_targets2, actual_targets = np.array([]), np.array([]), np.array([])

# creating classifiers w/ and w/o early stopping and adjusting max_iter for the model to converge
clasf1 = MLPClassifier(hidden_layer_sizes=(3,2), early_stopping=False, alpha = 0.1, max_iter = 
2000)
clasf2 = MLPClassifier(hidden_layer_sizes=(3,2), early_stopping=True, alpha = 0.1)

# Stratified CV is better for classification
kf = StratifiedKFold(n_splits=5, shuffle=True, random_state=0)

for train_subset, test_subset in kf.split(data, target):
  X_train, X_test = data[train_subset], data[test_subset]
  Y_train, Y_test = target[train_subset], target[test_subset]

  # train classifier without early stopping
  c1 = clasf1.fit(X_train, Y_train)
  # test and store predicted values
  predicted_targets1 = np.append(predicted_targets1, c1.predict(X_test))

  # train classifier with early stopping
  c2 = clasf2.fit(X_train, Y_train)
  # test and store predicted values
  predicted_targets2 = np.append(predicted_targets2, c2.predict(X_test))

  actual_targets = np.append(actual_targets, Y_test) # store the actual values

cnf_matrix1 = confusion_matrix(actual_targets, predicted_targets1, labels=clasf1.classes_)
ConfusionMatrixDisplay(cnf_matrix1,
display_labels=clasf1.classes_).plot(cmap=plt.get_cmap('PuBuGn'))
plt.title("Without early stopping")
plt.show()

cnf_matrix2 = confusion_matrix(actual_targets, predicted_targets2, labels=clasf2.classes_)
ConfusionMatrixDisplay(cnf_matrix2,
display_labels=clasf2.classes_).plot(cmap=plt.get_cmap('PuBuGn'))
plt.title("With early stopping")
plt.show()

#-----------------  Distribution of residues for different regularization terms ---------------#
raw_data = loadarff('kin8nm.arff')
df_data = pd.DataFrame(raw_data[0])  # converting data to a pandas DataFrame
data, target = df_data.drop(columns='y').values, df_data['y'].values

kf = KFold(n_splits=5, shuffle=True, random_state=0)
```

```
residues1, residues2, residues3, residues4 = [], [], [], []

# Creating MLP Regressors with different regularization terms and with no regularization
regr1 = MLPRegressor(hidden_layer_sizes=(3,2), alpha = 0.1)
regr2 = MLPRegressor(hidden_layer_sizes=(3,2), alpha = 1)
regr3 = MLPRegressor(hidden_layer_sizes=(3,2), alpha = 10)
regr4 = MLPRegressor(hidden_layer_sizes=(3,2), alpha = 0)

for train_subset, test_subset in kf.split(data):
  X_train, X_test = data[train_subset], data[test_subset]
  Y_train, Y_test = target[train_subset], target[test_subset]
  # train
  regr1.fit(X_train, Y_train)
  regr2.fit(X_train, Y_train)
  regr3.fit(X_train, Y_train)
  regr4.fit(X_train, Y_train)
  # store the residue for each prediction: absolute value of (actual - predicted)
  residues1.extend(np.absolute(np.subtract(Y_test, regr1.predict(X_test))))
  residues2.extend(np.absolute(np.subtract(Y_test, regr2.predict(X_test))))
  residues3.extend(np.absolute(np.subtract(Y_test, regr3.predict(X_test))))
  residues4.extend(np.absolute(np.subtract(Y_test, regr4.predict(X_test))))

fig, ax = plt.subplots()
ax.set_title("Residues Distribution")
ax.boxplot([residues1, residues2, residues3, residues4], labels=["alpha 0.1", "alpha 1", "alpha
10", "Without regularization"])
plt.show()
```

END