Backpropagation using sklearn

*Homework 5

Franco Delgado Gerardo 226392
Machine learning, Universidad Autónoma de SLP
México, San Luis Potosí
a226392@alumnos.uaslp.mx

***Abstract*—This document shows a backpropagation implementation for a classification problem using wine dataset. The goal is to determine a NN architecture, establishes an optimizer function and choose an activation function for a right classification results.**

I. Introduction

A. Backpropagation

In machine learning, backpropagation is a widely used algorithm for training feedforward neural networks. Generalizations of backpropagation exists for other artificial neural networks (ANNs), and for functions generally. These classes of algorithms are all referred to generically as "backpropagation". In fitting a neural network, backpropagation computes the gradient of the loss function with respect to the weights of the network for a single input–output example, and does so efficiently, unlike a naive direct computation of the gradient with respect to each weight individually. This efficiency makes it feasible to use gradient methods for training multilayer networks, updating weights to minimize loss; gradient descent, or variants such as stochastic gradient descent, are commonly used. The backpropagation algorithm works by computing the gradient of the loss function with respect to each weight by the chain rule, computing the gradient one layer at a time, iterating backward from the last layer to avoid redundant calculations of intermediate terms in the chain rule.

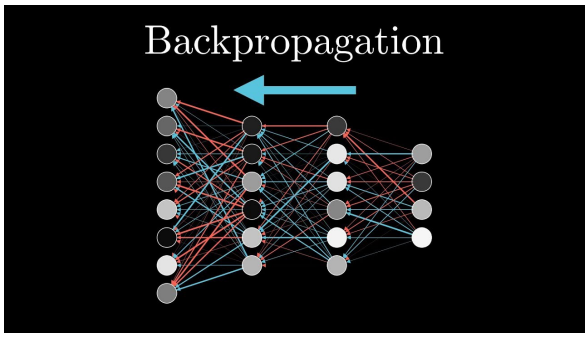

Fig. 1. **Backpropagation**

B. scikit-learn

Is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

Fig. 2. **sklearn**

C. Activation functions

1) Identity: In mathematics, an identity function, also called an identity relation or identity map or identity transformation, is a function that always returns the same value that was used as its argument. That is, for f being identity, the equality f(x) = x holds for all x.

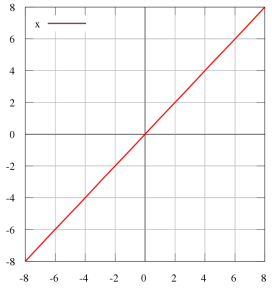

Fig. 3. **Identity activation function**

2) Logistic: The logistic function finds applications in a range of fields, including biology (especially ecology), biomathematics, chemistry, demography, economics, geoscience, mathematical psychology, probability, sociology, political science, linguistics, statistics, and artificial neural networks. A generalization of the logistic function is the hyperbolastic function of type I.

The logistic sigmoid function, returns f(x) = 1 / (1 + exp(-x))

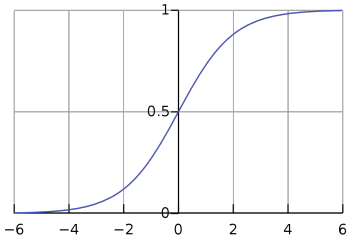

Fig. 4. **Sigmoid activation function**

3) tanh: The hyperbolic tan function, returns f(x) = tanh(x).

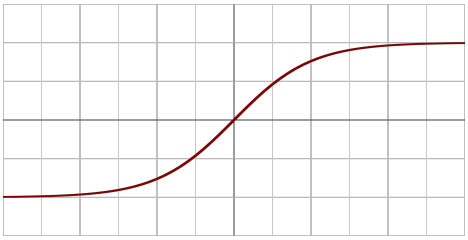

Fig. 5. **tanh activation function**

4) relu: This activation function was first introduced to a dynamical network by Hahnloser, with strong biological motivations and mathematical justifications. It was demonstrated for the first time in 2011 to enable better training of deeper networks, compared to the widely used activation functions prior to 2011, e.g., the logistic sigmoid (which is inspired by probability theory) and its more practical counterpart, the hyperbolic tangent. The rectifier is, as of 2017, the most popular activation function for deep neural networks. The rectified linear unit function, returns f(x) = max(0, x)

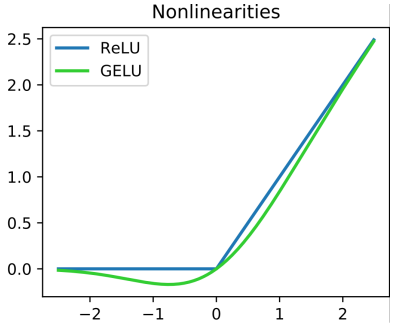

Fig. 6. **Relu activation function**

D. Optimization function

Optimization is the problem of finding a set of inputs to an objective function that results in a maximum or minimum function evaluation. It is the challenging problem that underlies many machine learning algorithms, from fitting logistic regression models to training artificial neural networks.

1) lbfgs: In numerical optimization, the Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm is an iterative method for solving unconstrained nonlinear optimization problems. Like the related Davidon–Fletcher–Powell method, BFGS determines the descent direction by preconditioning the gradient with curvature information. It does so by gradually improving an approximation to the Hessian matrix of the loss function, obtained only from gradient evaluations (or approximate gradient evaluations) via a generalized secant method.

2) sgd: Stochastic gradient descent (often abbreviated SGD) is an iterative method for optimizing an objective function with suitable smoothness properties (e.g. differentiable or subdifferentiable). It can be regarded as a stochastic approximation of gradient descent optimization, since it replaces the actual gradient (calculated from the entire data set) by an estimate thereof (calculated from a randomly selected subset of the data). Especially in high-dimensional optimization problems this reduces the computational burden, achieving faster iterations in trade for a lower convergence rate

3) adam: Adam is an optimization algorithm that can be used instead of the classical stochastic gradient descent procedure to update network weights iterative based in training data.

II. Methods

A. Dataset

These data are the results of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. The analysis determined the quantities of 13 constituents found in each of the three types of wines.
Here are the dataset features: 1) Alcohol 2) Malic acid 3) Ash 4) Alcalinity of ash 5) Magnesium 6) Total phenols 7) Flavanoids 8) Nonflavanoid phenols 9) Proanthocyanins 10)Color intensity 11)Hue 12)OD280/OD315 of diluted wines 13)Proline

With this independent variables it will necessary classify which type of wine it is.

B. Code

1) Libraries:

We are going to use the following libraries, notice that we'll use sklearn module.

```
from sklearn import datasets
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
```

Fig. 7. **Libraries**

2) Load dataset:

It will necessary import wine dataset, we could download the data from UCI Machine Learning Repository, but instead we are going to use datasets sklearn package. And declare a 'X' variable for all features and 'y' variable for the dependent variable (type of wine).

```
wine_ds = datasets.load_wine()
```

```
X = wine_ds.data
y = wine_ds.target
```

Fig. 8. **Load dataset**

3) Split dataset:

We are going to use the 80% percentage of data for training and we'll evaluate the model with only 20%. An important point is that we must shuffle the data to mix the several types of wine in the 80% of the data to have a good predictions.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=20, shuffle=True)
```

Fig. 9. **split dataset**

4) NN Architecture :

For the architecture of the neuronal network model we are going to use the network of image 11. We are using two hidden layers, the first one has 16 neurons and the second layer has 8 neurons. For the input layer we have 13 variables.

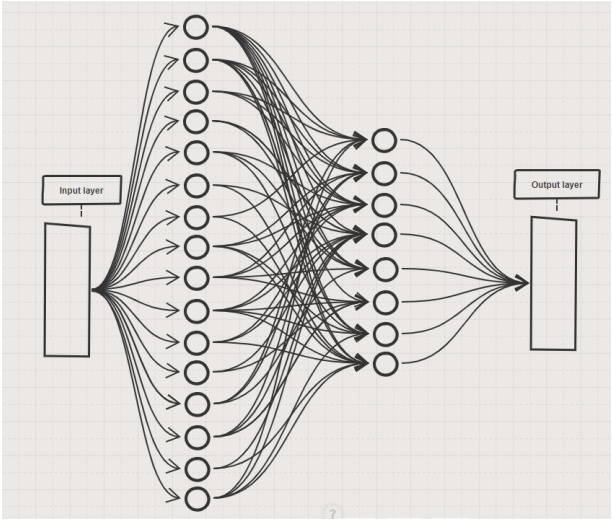

Fig. 10. **ANN architecture**

5) Model and training :

For the Multi layer perceptron classifier we are going to use a sigmoid activation function and adam optimization algorithm to train the model.

```
mlp = MLPClassifier(hidden_layer_sizes=(16,8), activation='logistic', solver='adam', max_iter=8000)
mlp.fit(X_train,y_train)
```

Fig. 11. **Training**

III. Results

A. Evaluation model

When the model is trained it'll be possible predict the test values.

```
predict_test = mlp.predict(X_test)
```

Fig. 12. **Predict X_test**

```
predict_test
```

```
array([0, 1, 0, 2, 0, 0, 0, 0, 2, 1, 0, 2, 2, 1, 1, 2, 0, 1, 2, 0])
```

```
y_test
```

```
array([0, 1, 1, 2, 0, 0, 0, 0, 2, 1, 0, 2, 2, 1, 1, 2, 0, 1, 2, 0])
```

Fig. 13. **Results**

Here is the confusion matrix of the test data:

```
cm = confusion_matrix(predict_test, y_test)
cm
```

```
array([[8, 1, 0],
       [0, 5, 0],
       [0, 0, 6]], dtype=int64)
```

Fig. 14. **Confusion matrix**

B. Model accuracy

At the end we obtained an accuracy of 95%, that means that are actually pretty good results.

```
accuracy_score(y_test, predict_test)
```

```
0.95
```

Fig. 15. **Accuracy**

IV. Conclusions

Maybe the architecture network could be changed to be less expensive, I played with different architectures and the above architecture had better results. The model had good results to identify and classify the wine type.

I hope that my code could help some readers to implement MLP in other ML scripts.

References

[1] Hastie, T., Hastie, T., Tibshirani, R., Friedman, J. H. The elements of statistical learning: Data mining, inference, and prediction. 2001, New York: Springer.

[2] Kruse, R., Borgelt, Christian,Braune, Mostaghim, Sanaz, Steinbrecher, Matthias. "Computational Intelligence: A Methodological Introduction", . 2016, London: Springer.

[3] Carlos, Juan Cuevas-Tello, Juan Carlos. (2018). Apuntes de Redes Neuronales Artificiales - Handouts for Artificial Neural Networks. 10.13140/RG.2.2.13177.57447.

[4] https://scikit-learn.org/stable/