Lab 7

Neural networks

Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona

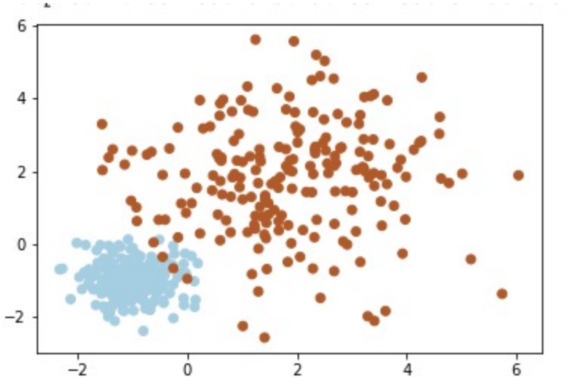
UNIVERSITAT POLITÈCNICA DE CATALUNYA

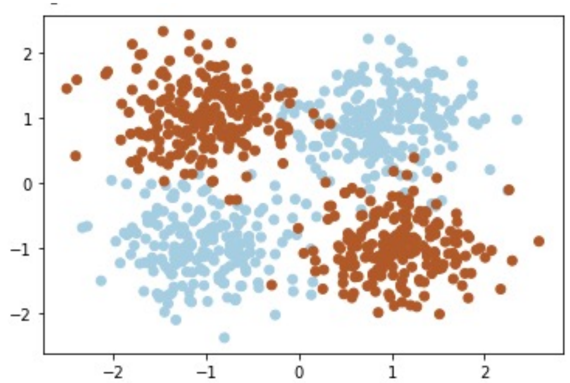
Objectives

- Learn how to train a Neural Network
- Use a Neural Network to solve the MNIST classification problem

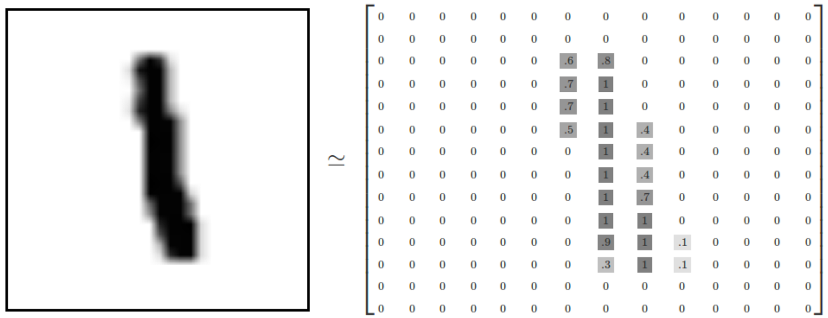
Lab7

- Toy examples: classify using a perceptron and a MLP





- MNIST: handwritten digits, 28x28 graylevel images (inverse scale);
- 60000 train set, 10000 test set (we will use subsets of 15000 and 2500 samples, resp.)



Neural Networks

- MultiLayerPerceptron model: class `MLPClassifier`

```
from sklearn.linear_model import MLPClassifier
mlp = MLPClassifier(hiden_layer_sizes=(10),solver='sgd',
learning_rate_init=0.01, max_iter=500)
mlp.fit(X, y)
mlp.get_params()
```

- Predict class labels:

```
mlp.predict(Xt)
```

- Predict probability estimates:

```
mlp.predict_proba(Xt)
```

```
{'activation': 'relu',
'alpha': 0.0001,
'batch_size': 'auto',
'beta_1': 0.9,
'beta_2': 0.999,
'early_stopping': False,
'epsilon': 1e-08,
'hidden_layer_sizes': 10,
'learning_rate': 'constant',
'learning_rate_init': 0.01,
'max_fun': 15000,
'max_iter': 500,
'momentum': 0.9,
'n_iter_no_change': 10,
'nesterovs_momentum': True,
'power_t': 0.5,
'random_state': 1,
'shuffle': True,
'solver': 'sgd',
'tol': 0.0001,
'validation_fraction': 0.1,
'verbose': False,
'warm_start': False}
```

Main parameters

- **Hidden layer sizes:** number of neurons at each layer
  - Ex1: one hidden layer 100 neurons: (100,) (default)
  - Ex2: three hidden layers, 30 neurons each: (30,30,30)
- **Activation function:** relu (default), identity, logistic, tanh
- **Solver:** sgd, adam (default)
- **Learning rate init:** initial learning rate
- **Learing rate:** schedule for updating learning rate, constant (default), invscaling, adaptive
- **Momentum**
- **Nesterov_momentum**
- **Alpha:** L2 penalty (regularization)
- **Batch size:** ‘auto’ (default: min(200, num samples))

Learned weights / loss

Attributes:

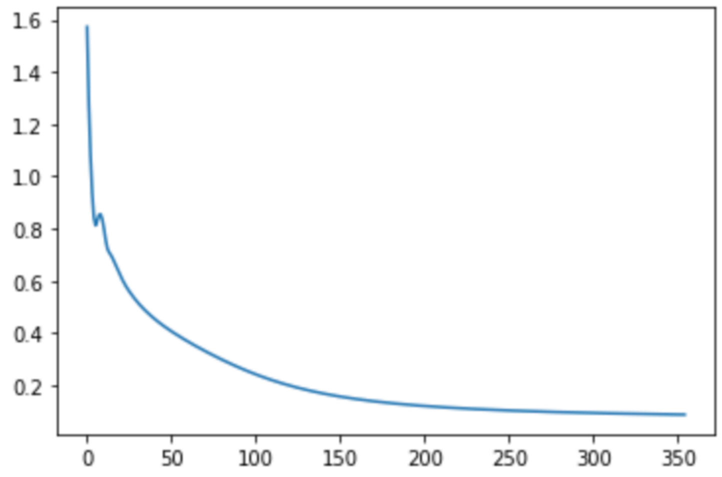
`mlp.coefs_` list of shape (n_layers - 1,), weight matrices

`mlp.intercepts_` list of shape (n_layers - 1,), bias vectors

`mlp.best_lost_` mínimum loss reached

`mlp.loss_curve_` list of shape (n_iter,) loss at each iteration

```
plt.plot(mlp.loss_curve_)
```



Hyperparameter search

Example:

```
from sklearn.model_selection import GridSearchCV
grid_search = sklearn.model_selection.GridSearchCV(estimator= model,
param_grid={"clf__hidden_layer_sizes": hidden_layer_sizes,
            "clf__activation": activation,
            "clf__learning_rate_init": learning_rate_init},
cv=sklearn.model_selection.ShuffleSplit(n_splits=1, train_size=0.75,
random_state=1)
)

grid_search.fit(X_train,y_train)
```

`grid_search.cv_results_` results of each iteration, can be imported as a DataFrame
`grid_search.best_params_` best set of parameters

```
grid_search.predict(X_train)
grid_search.predict(X_test)
```

Multi-class metrics

Classification report:

	precision	recall	f1-score	support
0	0.87	0.94	0.90	219
1	0.91	0.97	0.94	287
2	0.87	0.79	0.83	276
3	0.82	0.81	0.81	254
4	0.86	0.83	0.85	275
5	0.73	0.73	0.73	221
6	0.85	0.86	0.86	225
7	0.86	0.79	0.82	257
8	0.79	0.74	0.77	242
9	0.77	0.87	0.82	244
accuracy			0.84	2500
macro avg	0.83	0.83	0.83	2500
weighted avg	0.84	0.84	0.83	2500

Macro average: computes the metric for each class individually and averages them $\frac{1}{C}\sum_{i=1}^{C} metric\_class_i$

Weighted average: $\frac{1}{Total\_support}\sum_{i=1}^{C} support\_class_i \times metric\_class_i$

Balanced accuracy: $\frac{1}{2}(sensitivity + specificity) = \frac{1}{2}\left(\frac{TP}{pos\ samples} + \frac{TN}{neg\ samples}\right)$

If multi-class, balanced accuracy = macro average of recall

Multi-class metrics

Micro-average: aggregate contributions from all classes before calculating the metric

Micro averge accuracy: $\frac{\sum correct\ predictions}{total\ predictions}$

Micro average precisión (P_{mic}): $\frac{\sum TP_i}{\sum (TP_i + FP_i)}$

Micro average recall (R_{mic}): $\frac{\sum TP_i}{\sum (TP_i + FN_i)}$

Micro average F1: $\frac{2P_{mic}R_{mic}}{P_{mic} + R_{mic}}$

Comparison with Macro-average

Micro-average:

- Gives equal weight to each instance, favoring large classes in imbalanced datasets.
- More appropriate when **overall performance** across all samples is important.

Macro-average:

- Treats all classes equally, regardless of size.
- More appropriate when **class performance balance** is important, especially for imbalanced datasets.