Session 3

In this third session we will apply Logistic Regression to some classification tasks. A simple implementation of Logistic Regression and its application is provided. The final purpose of this session is to apply Logistic Regression to MyDigits dataset to train a matrix of weights that will be used in an application for Handwritten Digit Classification.

You may need to run this code if this is the first time you are running this notebook:

In [ ]:
```
!pip install seaborn scikit-learn pandas pillow gradio matplotlib
```

Logistic regression

In [1]:
```python
import numpy as np
```

OneHotEncoding: function to convert class labels into one-hot encoding representation

$$\text{one-hot}(y) = \boldsymbol{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_C \end{pmatrix} = \begin{pmatrix} \mathbb{I}(y=1) \\ \vdots \\ \mathbb{I}(y=C) \end{pmatrix} \in \{0,1\}^C \quad \text{with} \quad \sum_c y_c = 1$$

where y is a categorical variable that takes a value among C possible categories, $\{1, \ldots, C\}$

In [2]:
```python
def OneHotEncoding(y):
  # Unique class labels
  c = np.unique(y); C = c.size;

  # Mapping class labels from 0 to C-1
  # Create a mapping from values in c to their corresponding indices
  mapping = {value: idx for idx, value in enumerate(c)}
  # Convert y using the mapping
  y = np.vectorize(mapping.get)(y)

  # Generate one-hot encoding using np.eye
  return np.eye(C)[y]
```

ComputeGradient: weight update for logistic regression

Gradient descent applied to logistic regression:

Neg-log-likelihood (NLL): the NLL is a convex objective function

$$\text{NLL}(\mathbf{W}) = \frac{1}{N}\sum_{n=1}^{N} -\log p(\boldsymbol{y}_n \mid \boldsymbol{\mu}_n) \qquad \text{with} \quad \boldsymbol{\mu}_n = \mathcal{S}(\boldsymbol{a}_n) \quad \text{and} \quad \boldsymbol{a}_n = \mathbf{W}^t \boldsymbol{x}_n$$

NLL gradient:

$$\begin{pmatrix} \frac{\partial \text{NLL}}{\partial W_{11}} & \cdots & \frac{\partial \text{NLL}}{\partial W_{1C}} \\ \vdots & \ddots & \vdots \\ \frac{\partial \text{NLL}}{\partial W_{D1}} & \cdots & \frac{\partial \text{NLL}}{\partial W_{DC}} \end{pmatrix} = \frac{\partial \text{NLL}}{\partial \mathbf{W}^t} = \frac{1}{N}\sum_{n=1}^{N} \frac{\partial(-\log p(\boldsymbol{y}_n \mid \boldsymbol{\mu}_n))}{\partial \mathbf{W}^t} = \frac{1}{N}\sum_{n=1}^{N} \boldsymbol{x}_n(\boldsymbol{\mu}_n - \boldsymbol{y}_n)^t$$

In [3]:
```python
def ComputeGradient(X, Y, W):
  N, D = X.shape
  X = np.concatenate((np.ones((N,1)), X),axis=1)
  N, C = Y.shape
  Z = np.zeros((N, C)).astype(float)

  # Compute logits
  a = X @ W
  # Normalizing logits for robust computation to avoid overflow
  a -= a.max(axis=1, keepdims=True)
  # Compute softmax (probabilistic class label)
  mu = np.exp(a)
  mu /= np.sum(mu, axis=1, keepdims=True)
  # Substract actual label from probabilistic class label
  Z = mu - Y
  # Gradient is the sum over all samples of the outer dot product
  return (X.T @ Z)/N
```

LogisticRegressionClassification: classification of samples provided a weight matrix

$$c(\boldsymbol{x}) = \underset{c}{\operatorname{argmax}}\ \boldsymbol{\mu}_c \quad \text{with} \quad \boldsymbol{\mu}_c = \mathcal{S}(\boldsymbol{a})_c, \quad \boldsymbol{a} = f(\boldsymbol{x}; \mathbf{W}) = \mathbf{W}^t \boldsymbol{x}, \quad \mathbf{W} \in \mathbb{R}^{D \times C} \quad \text{and} \quad \boldsymbol{x} \in \mathbb{R}^D$$

In [4]:
```python
def LogisticRegressionClassification(X, W):
  N, D = X.shape
  X = np.concatenate((np.ones((N,1)), X),axis=1)
  # Compute logits
  a = X @ W
  # Normalizing logits for robust computation to avoid overflow
  a -= a.max(axis=1, keepdims=True)
  # Compute softmax (probabilistic class label)
  mu = np.exp(a)
  mu /= np.sum(mu, axis=1, keepdims=True)
  return np.argmax(mu,axis=1)
```

LogisticRegressionTraining: implementation of batch training based on stochastic gradient descent

Gradient descent applied to logistic regression: $\mathbf{W}_0 = \mathbf{0}; \quad \mathbf{W}_{i+1} = \mathbf{W}_i - \eta_i \dfrac{\partial \mathrm{NLL}}{\partial \mathbf{W}^t}\Big|_{\mathbf{W}_i} \quad i = 0, 1, \ldots$

In [5]:
```python
def LogisticRegressionTraining(X, y, W=None, bs=1, maxEpochs=10, eta=1e-2, tol=1e-3):
  rng = np.random.default_rng(seed=23)
  N, D = X.shape;
  Y = OneHotEncoding(y)
  N, C = Y.shape
  if W is None:  # W_0 = 0
    W = np.zeros((1+D, C))
  grad = np.inf; epoch = 0
  while np.max(np.abs(grad)) > tol and epoch < maxEpochs:  # W_i+1 = W_i - eta_i * grad_W_i
    perm = rng.permutation(N); Xperm = X[perm]; Yperm = Y[perm]
    j = 0
    for j in range(N//bs):
      X_batch = Xperm[j*bs:(j+1)*bs]
      Y_batch = Yperm[j*bs:(j+1)*bs]
      grad = ComputeGradient(X_batch, Y_batch, W)
      W = W - eta*grad
    # If there are remaining samples after splitting into batches
    if((j+1)*bs < N):
      X_batch = Xperm[(j+1)*bs:]
```

```
      Y_batch = Yperm[(j+1)*bs:]
      grad = ComputeGradient(X_batch, Y_batch, W)
      W = W - eta*grad
    epoch += 1
  return W
```

Logistic regression applied to the Iris dataset

Reading and partitioning the dataset:

In [6]:
```
import numpy as np;from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
iris = load_iris(); X = iris.data.astype(np.float16); y = iris.target.astype(np.uint)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=True, random_state=23)
N = len(X_train); M = len(X_test)
```

LogisticRegression: training, classification and evaluation

In [7]:
```
W = LogisticRegressionTraining(X_train,y_train, bs=10)
haty_test = LogisticRegressionClassification(X_test,W)
accuracy = np.sum(haty_test==y_test)/M
print(f"Test error: {1.0-accuracy:.1%}")
```

```
Test error: 26.7%
```

Adjusting maximum number of epochs:

In [8]:
```
print("  bs maxEps     eta trainErr testErr")
print("---- ------ ------- -------- -------")
bs=10; eta=1e-2;
for maxEpochs in (5, 10, 20, 50, 100, 200, 500, 1000, 2000):
    W = LogisticRegressionTraining(X_train,y_train, bs=bs, maxEpochs=maxEpochs, eta=eta)
    haty_train = LogisticRegressionClassification(X_train,W)
    acc_train = np.sum(haty_train==y_train)/N
    haty_test = LogisticRegressionClassification(X_test,W)
    acc_test = np.sum(haty_test==y_test)/M
    print(f"{bs:4d} {maxEpochs:6d} {eta:.1e} {1.0-acc_train:8.1%} {1.0-acc_test:7.1%}")
```

```
  bs maxEps     eta trainErr testErr
---- ------ ------- -------- -------
  10      5 1.0e-02    31.7%   23.3%
  10     10 1.0e-02    35.0%   26.7%
  10     20 1.0e-02    17.5%   10.0%
  10     50 1.0e-02     4.2%    6.7%
  10    100 1.0e-02     4.2%    0.0%
  10    200 1.0e-02     3.3%    3.3%
  10    500 1.0e-02     2.5%    3.3%
  10   1000 1.0e-02     2.5%    3.3%
  10   2000 1.0e-02     1.7%    3.3%
```

It can be observed the effect of *over-training* on the error rates, when the error rate in the training set still decreases while the error rate in the test set starts increasing. We must prevent over-training by applying *early stopping*, setting the maximum number of epochs to, for example, 100 epochs.

Adjusting the learning rate (η/eta):

In [9]:
```
print("  bs maxEps     eta trainErr testErr")
print("---- ------ ------- -------- -------")
bs=10; maxEpochs = 100;
for eta in (1e-4, 1e-3, 1e-2, 1e-1, 1, 1e1, 1e2):
    W = LogisticRegressionTraining(X_train,y_train, bs=bs, maxEpochs=maxEpochs, eta=eta)
    haty_train = LogisticRegressionClassification(X_train,W)
    acc_train = np.sum(haty_train==y_train)/N
    haty_test = LogisticRegressionClassification(X_test,W)
    acc_test = np.sum(haty_test==y_test)/M
    print(f"{bs:4d} {maxEpochs:6d} {eta:.1e} {1.0-acc_train:8.1%} {1.0-acc_test:7.1%}")
```

```
  bs maxEps     eta trainErr testErr
---- ------ ------- -------- -------
  10    100 1.0e-04    66.7%   66.7%
  10    100 1.0e-03    28.3%   23.3%
  10    100 1.0e-02     4.2%    0.0%
  10    100 1.0e-01     3.3%    0.0%
  10    100 1.0e+00     6.7%    6.7%
  10    100 1.0e+01     6.7%    6.7%
  10    100 1.0e+02    13.3%   20.0%
```

The default learning rate ($\eta = $ 1e-2) provides good results.

Adjusting the batch size: the *bs* parameter sets every how many samples the weight matrix is updated

In [10]:
```
print("  bs maxEps     eta trainErr testErr")
print("---- ------ ------- -------- -------")
maxEpochs=100; eta=1e-2;
for bs in (1, 2, 5, 10, 20, 50, 100):
    W = LogisticRegressionTraining(X_train,y_train, bs=bs, maxEpochs=maxEpochs, eta=eta)
    haty_train = LogisticRegressionClassification(X_train,W)
    acc_train = np.sum(haty_train==y_train)/N
    haty_test = LogisticRegressionClassification(X_test,W)
    acc_test = np.sum(haty_test==y_test)/M
    print(f"{bs:4d} {maxEpochs:6d} {eta:.1e} {1.0-acc_train:8.1%} {1.0-acc_test:7.1%}")
```

```
  bs maxEps     eta trainErr testErr
---- ------ ------- -------- -------
   1    100 1.0e-02     2.5%    3.3%
   2    100 1.0e-02     2.5%    3.3%
   5    100 1.0e-02     2.5%    0.0%
  10    100 1.0e-02     4.2%    0.0%
  20    100 1.0e-02     5.8%    0.0%
  50    100 1.0e-02    16.7%   10.0%
 100    100 1.0e-02    31.7%   23.3%
```

In this task, a small batch size provides good results.

Logistic regression applied to the Digits dataset

Reading and partitioning the dataset:

In [11]:
```
import numpy as np; from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
digits = load_digits(); X = digits.images.astype(np.float16).reshape(-1, 8*8); X/=np.max(X)
y = digits.target.astype(np.uint);
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=True, random_state=23)
N = len(X_train); M = len(X_test)
```

Adjusting maximum number of epochs:

In [12]:
```
print("  bs maxEps     eta trainErr testErr")
print("---- ------ ------- -------- -------")
bs=10; eta=1e-2;
for maxEpochs in (5, 10, 20, 50, 100, 200, 500, 1000, 2000, 5000):
    W = LogisticRegressionTraining(X_train,y_train, bs=bs, maxEpochs=maxEpochs, eta=eta)
    haty_train = LogisticRegressionClassification(X_train,W)
    acc_train = np.sum(haty_train==y_train)/N
    haty_test = LogisticRegressionClassification(X_test,W)
    acc_test = np.sum(haty_test==y_test)/M
    print(f"{bs:4d} {maxEpochs:6d} {eta:.1e} {1.0-acc_train:8.1%} {1.0-acc_test:7.1%}")
```

```
  bs maxEps     eta trainErr testErr
---- ------ ------- -------- -------
  10      5 1.0e-02    10.6%   13.3%
  10     10 1.0e-02     8.1%   10.8%
  10     20 1.0e-02     6.5%    9.2%
  10     50 1.0e-02     5.0%    6.4%
  10    100 1.0e-02     4.1%    5.0%
  10    200 1.0e-02     2.8%    4.2%
  10    500 1.0e-02     1.7%    3.3%
  10   1000 1.0e-02     1.2%    2.2%
  10   2000 1.0e-02     1.1%    1.9%
  10   5000 1.0e-02     1.1%    1.9%
```

Adjusting learning rate (eta):

In [13]:
```
print("  bs maxEps     eta trainErr testErr")
print("---- ------ ------- -------- -------")
bs=10; maxEpochs = 2000;
for eta in (1e-4, 1e-3, 1e-2, 1e-1, 1, 1e1, 1e2):
    W = LogisticRegressionTraining(X_train,y_train, bs=bs, maxEpochs=maxEpochs, eta=eta)
    haty_train = LogisticRegressionClassification(X_train,W)
    acc_train = np.sum(haty_train==y_train)/N
    haty_test = LogisticRegressionClassification(X_test,W)
    acc_test = np.sum(haty_test==y_test)/M
    print(f"{bs:4d} {maxEpochs:6d} {eta:.1e} {1.0-acc_train:8.1%} {1.0-acc_test:7.1%}")
```

```
  bs maxEps     eta trainErr testErr
---- ------ ------- -------- -------
  10   2000 1.0e-04     6.6%    8.9%
  10   2000 1.0e-03     2.9%    4.2%
  10   2000 1.0e-02     1.1%    1.9%
  10   2000 1.0e-01     0.6%    2.8%
  10   2000 1.0e+00     0.2%    3.3%
  10   2000 1.0e+01     4.0%    6.9%
  10   2000 1.0e+02     5.4%    6.7%
```

Adjusting the batch size: the *bs* parameter sets every how many samples the weight matrix is updated

In [14]:
```
print("  bs maxEps     eta trainErr testErr")
print("---- ------ ------- -------- -------")
maxEpochs=2000; eta=1e-2;
for bs in (1, 2, 5, 10, 20, 50, 100):
    W = LogisticRegressionTraining(X_train,y_train, bs=bs, maxEpochs=maxEpochs, eta=eta)
    haty_train = LogisticRegressionClassification(X_train,W)
    acc_train = np.sum(haty_train==y_train)/N
    haty_test = LogisticRegressionClassification(X_test,W)
    acc_test = np.sum(haty_test==y_test)/M
    print(f"{bs:4d} {maxEpochs:6d} {eta:.1e} {1.0-acc_train:8.1%} {1.0-acc_test:7.1%}")
```

```
  bs maxEps     eta trainErr testErr
---- ------ ------- -------- -------
   1   2000 1.0e-02     1.4%    3.3%
   2   2000 1.0e-02     1.8%    3.6%
   5   2000 1.0e-02     1.3%    2.8%
  10   2000 1.0e-02     1.1%    1.9%
  20   2000 1.0e-02     1.2%    2.2%
  50   2000 1.0e-02     1.8%    3.3%
 100   2000 1.0e-02     2.7%    4.2%
```

Final classifier: Training final classifier with all data available and best parameters, saving and loading to test it

In [15]:
```
bs=10; maxEpochs=2000; eta=1e-2
W = LogisticRegressionTraining(X, y, bs=bs, maxEpochs=maxEpochs, eta=eta)
np.save("DigitsWeights.npy",W)
```

In [16]:
```
with open('DigitsWeights.npy', 'rb') as fd:
    W = np.load(fd)
    fd.close()
haty_test = LogisticRegressionClassification(X_test,W)
accuracy = np.sum(haty_test==y_test)/y_test.size
print(f"Test error of final classifier: {1.0-accuracy:.1%}")
```

```
Test error of final classifier: 1.9%
```

Assignment: Logistic regression applied to MyDigits dataset

Reading and partitioning the dataset:

In [ ]:
```
# Execute this cell only when running in Google Colab
# You need to upload your images and labels files
from google.colab import files
uploaded = files.upload()
```

In [17]:
```
import numpy as np
from sklearn.model_selection import train_test_split

with open('images.npy', 'rb') as fd:
    X = np.load(fd)
    fd.close()

with open('labels.npy', 'rb') as fd:
    y = np.load(fd).astype(int)
    fd.close()

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=True, random_state=23)
N = len(X_train); M = len(X_test)
```

Task: Adjusting maximum number of epochs

In [ ]:
```
print("  bs maxEps     eta trainErr testErr")
print("---- ------ ------- -------- -------")
# Write here the code for this task
```

Task: Adjusting the learning rate (eta)

In [ ]:
```
print("  bs maxEps     eta trainErr testErr")
print("---- ------ ------- -------- -------")
# Write here the code for this task
```

Task: Adjusting the batch size the *bs* parameter sets every how many samples weight matrix is updated

In [ ]:
```
print("  bs maxEps     eta trainErr testErr")
print("---- ------ ------- -------- -------")
# Write here the code for this task
```

Task: Final classifier: Training final classifier with all data available and best parameters, saving and loading to test it

In [ ]:
```
# Task: Replace these default values with the best configuration obtained in the previous experiments
bs=10; maxEpochs=10; eta=1e-2

W = LogisticRegressionTraining(X, y, bs=bs, maxEpochs=maxEpochs)
np.save("MyDigitsWeights.npy",W)
```

In [ ]:
```
with open('MyDigitsWeights.npy', 'rb') as fd:
    W = np.load(fd)
    fd.close()
haty_test = LogisticRegressionClassification(X_test,W)
accuracy = np.sum(haty_test==y_test)/y_test.size
print(f"Test error of final classifier: {1.0-accuracy:.1%}")
```

In [ ]:
```
# Execute this cell only when running in Google Colab
# You need to download MyDigitsWeights.npy
files.download('MyDigitsWeights.npy')
```

Classify your own handwritten digits

The following simple application allows you to classify your own handwritten digits. When you run this application, it shows a basic graphical interface containing a panel on which you can draw your own handwritten digits.

Before you can draw a digit, you need to click on the *pen* located on the left vertical. Then you can draw on the panel. When you need to erase what you have drawn on the panel (in order to draw a new digit), just click on *bin* located on the top menu.

You can classify the image on the panel by clicking on the bottom bar labelled *Classify image*.

In [ ]:
```
# Execute this cell only when running in Google Colab
# You need to upload LogisticRegression.py DigitClassifyGradioApp.py and MyDigitsWeights.npy
from google.colab import files
uploaded = files.upload()
```

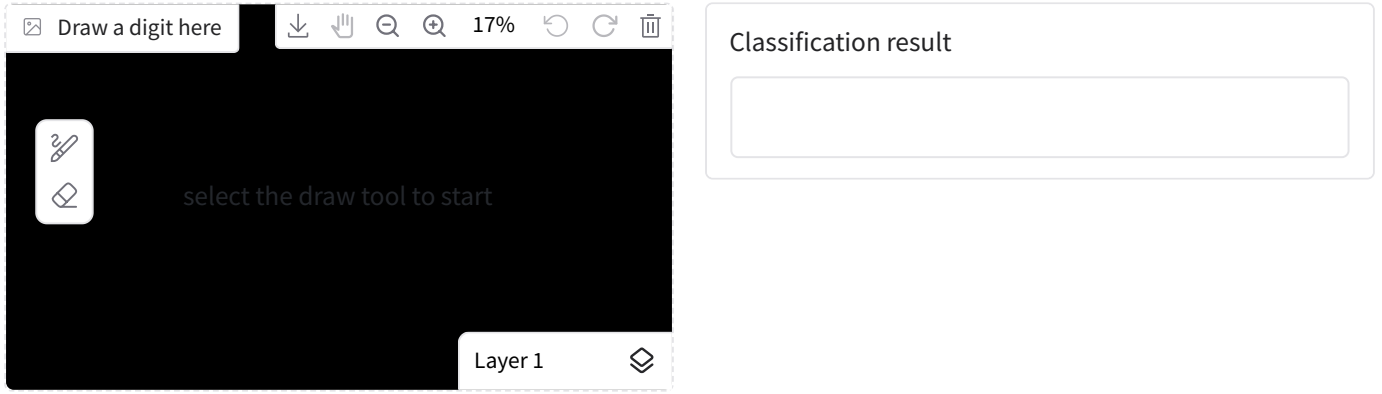
In [19]:
```python
import numpy as np
from DigitClassifyGradioApp import create_interface

fn = input("Please provide filename for weight matrix:")
with open(fn, 'rb') as fd:
    W = np.load(fd)
    fd.close()

demo = create_interface(W, LogisticRegressionClassification)
demo.launch()
```

```
* Running on local URL:  http://127.0.0.1:7860
* To create a public link, set `share=True` in `launch()`.
```

Handwritten Digit Classification



Classify image

Utilitzar via API · Construït amb Gradio · Configuració

Out[19]: