TELECOM ParisTech **SD210**
2018/2019 **A. Garcia, A. Lambert, G. Staerman, G. Varni**

TP n° 2 : Training a neural network

- Python instructions -

For this practical course, in addition to the usual `numpy` and `sklearn` packages, you will need to install the package `keras`. It is recommended that you use `pip` or `conda` to install this package.

The documentation can be found at https://keras.io/.

Notebook file `tp_neural_net.ipynb` is avalaible on the class website. This file includes the code to complete. This practical session is not graded, yet we encourage you to do it seriously and through the end.

- Introduction -

The goal of this practical course is to train a neural network being able to correctly classify the handwritten numbers of the sklearn dataset `digits`. The first part is about using the high level api `keras` to do so, and play with the different options that are already implemented in this package (optimizer, learning rate, initialization, ...). In the second part, you will have to implement a neural network using `numpy` and derive the backpropagation by hand.

- Training a neural network with keras -

Preprocessing

1) Separate the dataset into a training and a test set. Use the scaler from `sklearn` (https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler) to preprocess the training dataset, and apply the same transformation on the test dataset. Display an example of a sample after preprocessing. How can we get the original sample back ?
2) Transform the training targets from integers into one-hot encoding vectors of the classes using the function `to_categorical` (https://keras.io/utils/). Explain why this is needed when minimizing the cross entropy loss.

Training the neural network

`keras` is a high level api allowing to design neural networks by stacking sequentially the layers one after another. You can find a beginner's guide at https://keras.io/getting-started/sequential-model-guide/.

The three main steps for training a neural network in `keras` are

— Design : stack different layers with activation functions.

— Compilation : configure the learning process by choosing the optimizer, the loss function, and the metrics to compute.

— Training : feed the training data to your model.

You will use a neural network of the following type :

```
from keras.models import Sequential
from keras.layers.core import Dense, Activation
from keras import optimizers

n_i = X_train.shape[1]
n_h = 5
n_o = 10

model = Sequential()
model.add(Dense(n_h, input_dim=n_i))
model.add(Activation("tanh"))
model.add(Dense(n_o))
model.add(Activation("softmax"))

model.compile(optimizer=optimizers.SGD(lr=0.01),
              loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(X_train, Y_train, epochs=10, batch_size=32)
```

The values of the different parameters have been set so that the optimization process is not satisfactory.

3) Show that minimizing the cross entropy loss is equivalent to maximizing the log-likelihood of the observations under a well chosen probabilistic model.
4) Play with the size of the hidden layer. How does the optimization process behave at fixed learning rate and epochs ?
5) Explain the following sentence : "Feed-forward neural networks with one hidden layer are known to be universal approximators". What are the requirements on the activation function for this to be true ? How does the width of such a network evolves ?
6) Set $n_h = 100$. Play with the learning rate : how does the optimization process behave when the learning rate is too high/ too low ?
7) Test different optimization techniques (https://keras.io/optimizers/). What is the main difference between Adam and Adadelta ?
8) What technique is used by default in `keras` to initialise the weights in the dense layers ? Try to fit the model with a random gaussian weights initialization of different standard deviations. What happens when the standard deviation is too high, too low, or zero ?

For the following question, use values of the hyperparameters that give a good accuracy on the train set, i.e a satisfactory optimization.

9) Evaluate the model on the test dataset. Does it generalize well ? Display some examples of samples that your model was wrong about.

- Training a neural network with `numpy` -

Your task is now to implement a neural network using only the `numpy` library to classify handwritten digits. Formally, you have some $(x_j, y_j)_{j=1}^n$ i.i.d. where each $x_j \in \mathbb{R}^{n_i}$ is a sample of size n_i (read it as n_{input}) and the corresponding $y_j \in \{0,1\}^{n_o}$ is the associated class seen as a one-hot encoding vector (for digits classification $n_o = 10$). X refers to the matrix of size (n, n_i).

The neural network will be a single layer feed-forward network using a vectorial extension of the sigmoid function

$$\sigma \overset{\text{def}}{=} x \mapsto \frac{1}{1+e^{-x}}.$$

The network can be summarized as follows, calling respectively z_h, h, z_o the intermediary output produced :

$$X \to z_h = XW_h + b_h \to h = \sigma(z_h) \to z_o = hW_o + b_o \to y = \text{softmax}(z_o)$$

where

— W_h weight matrix of size (n_i, n_h)

— b_h bias vector of size n_h

— $\sigma(z_h)$ is to be understood component-wise

— W_o weight matrix of size (n_h, n_o)

— b_o bias vector of size n_o

The loss considered here is the cross entropy :

$$l(y_{\text{true}}, y_{\text{pred}}) \overset{\text{def}}{=} -\sum_{k=1}^{n_o} \log\left(y^k_{\text{pred}}\right) \cdot y^k_{\text{true}}$$

so that the optimization problem reads, denoting by f the function defined by the network

$$\underset{W_h, b_h, W_o, b_o}{\arg\min} L(W_h, b_h, W_o, b_o) \overset{\text{def}}{=} \sum_{j=1}^{n} l(y_j, f(x_j))$$

10) Find how to express the derivative σ' as a function of σ.

11) Implement the functions `softmax`, `sigmoid`, `dsigmoid` and `nll` in your notebook. (`dsigmoid` is σ', `nll` is the negative log-likelihood).

A few advices about these functions :

— To implement a function `softmax` that works for both an individual sample or a batch of samples, look for the options of the method `np.sum`, especially the axis and the keepdims options in the documentation. Verify that your function is correct on some toy example.

— For the `nll` function, in the same spirit as the previous method, to handle groups of prediction you can use the method `np.atleast_2d` and then do the summation on the correct axis.

— For numerical stability arguments, it can be useful to add a small quantity $\epsilon = 10^{-10}$ in the logarithm.

12) Using the chain rule, find the gradient of L with respect to (W_h, b_h, W_o, b_o).

13) Write the different methods of the class `NeuralNet`.

Again, a few advices about these functions :

— `__init__` : this is the method called when instanciating your model, for example : `model = NeuralNet(n_features, n_hidden, n_classes)`. It shoud not return anything, but it should initialize the weights to vectors of corresponding dimensions. For example, you can choose to use `np.random.uniform` to generate those weights.

— `forward` should return the output $y = f(x)$ of the neural network.

— `forward_with_hidden` should return the output y of the neural network, and also the intermediary results z_h and h (useful when computing the gradient).

— `grad_loss(self,x,y_true)` should return the gradient of L with respect to (W_h, b_h, W_o, b_o) at the point (x, y_{true}) (remember that you will be performing a stochastic gradient descent). You may use a dictionary to store the different values.

— `train_sample` should update the weights with the appropriate learning rate. It corresponds to a single update in the stochastic gradient descent algorithm.

— `fit` should perform the full stochastic gradient algorithm with the correct number of epochs and rely on the `train_sample` method. Store the values of the loss function at each iteration and return them when finishing the learning phase to be able to plot it later.

— `predict` : your method has to handle both single predictions and group of predictions.

14) Plot the evolution of the loss function L during the learning phase. Compare your results to the ones obtained with `keras`.