LELEC2870: Practical Sessions

October 15, 2020

Notations and vocabulary

- a scalar: A number represented with an upper-case letter when defining bounds on a series, or a lower-case one otherwise e.g. x in $1..X$
- a vector: A 1-dimensional array that will always be written with a **bold lower-case letter**
- a matrix: A 2-dimensional array that will always be written with a **bold upper-case letter**
- a tensor: A N-dimensional array that will always be written with a **bold upper-case letter**. While it uses the same notation as the matrix, it should always be clear which one is meant.
- an iteration: A time-measure for *iterative* algorithms. It denotes a pass over a part of the training dataset.
- an epoch: A time-measure for *iterative* algorithms. It denotes *one* pass over the complete training dataset. An epoch is composed of at least 1 iteration, but most of the time of multiple iterations especially when large datasets are involved.

1 Session 2

Objectives

Last session, you implemented linear regression between the target values and one of the features of the dataset. In this second exercise session, you will implement the multivariate linear regression (linear regression between the target and several input features). Next, you will learn how to build a linear model on a complete dataset i.e. using all features, even thought there is no clear linear relation between the features and the target. Finally, you will implement an iterative method named Stochastic Gradient Descent (SGD) useful for optimizing objective function.

1.1 Linear Regression (continued)

Linear Regression Notations

The output prediction of a linear regressor for an element p can then be written as follows:

$$t_p = w_p^0 + w_1.x_p^1 + w_2.x_p^2 + ... + w_D.x_p^D \tag{1}$$

with D the number of dimensions of input vector $\mathbf{x}$, $w_1 ... w_D$ the weights of each feature/dimension, and w_0 the independent term (=bias).

A Linear regressor can be equivalently described as a simple neural network with only one layer and a linear activation. The analytical expression of its output is

$$t_p = \sigma(\mathbf{x}_p\mathbf{w} + w_0). \tag{2}$$

where $\mathbf{w}$ is the weights column vector, w_0 is the bias, σ is the activation function and t_p is the output of the network. In order to simplify notations, the bias is generally included in the inputs and weights vectors, i.e. $\mathbf{x}_p$ becomes

$$\begin{pmatrix} 1 & \mathbf{x}_p \end{pmatrix}$$

and $\mathbf{w}$ becomes

$$\begin{pmatrix} w_0 \\ \mathbf{w} \end{pmatrix}.$$

This convention is used in the rest of this exercise session. Since the activation is linear, the output of one-layer neural networks becomes simply

$$t_p = \mathbf{x}_p\mathbf{w}. \tag{3}$$

Notice that now, $\mathbf{x}_p$ and $\mathbf{w}$ are both vectors of length $(D+1)$.
We can extend this to vectorial outputs (of length T) in the following way

$$\mathbf{t} = \mathbf{X}\mathbf{w}, \tag{4}$$

with the shape of $\mathbf{X}$ being $T \times (D+1)$

1.1.1 Bivariate Linear Regression (5 min)

Choose the 2 best features (still using correlation as criterion) and **train** another linear regression. **Visualize** the results and compare the predicted target values with the actual target values.

RMSE is a formal criterion that can be used to show that linear regression is able to approximate the target values more accurately using two features than only one feature (with respect to training data).

The expression of RMSE is the following:

$$RMSE = \sqrt{\frac{1}{P}\|\mathbf{t} - \mathbf{X}\mathbf{w}\|^2} \tag{5}$$

where **t** is the value of the target (ground truth) and **Xw** is the prediction (output) of a model.

Implement the computation of RMSE to observe that bivariate regression is better that univariate regression.

1.1.2 Multivariate Linear Regression (5 min)

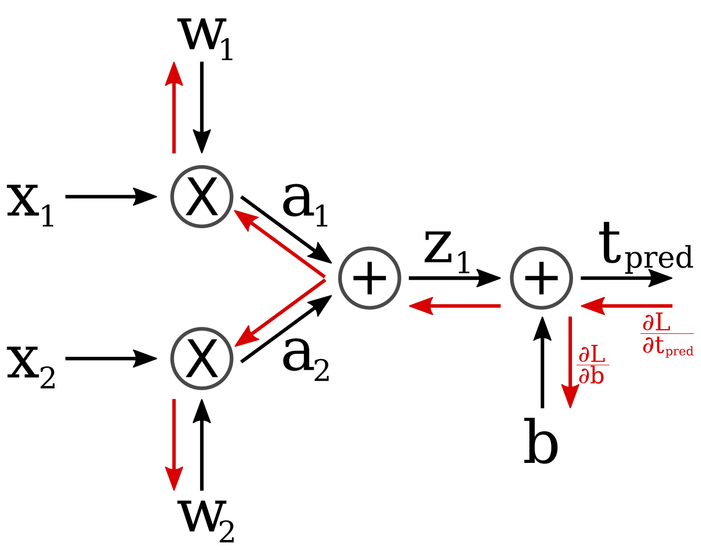
Perform linear regressions with a growing number of features and **plot** the accuracy of the models trained over the growing number of features. What can you conclude?

1.1.3 Train-test Generalization (10 min)

Split the dataset in two equal parts at random ({X1, t1} and {X2,t2}). Build a multivariate regression model on the first sub dataset {X1, t1} and test it on {X1,t1}. Draw a target vs prediction scatterplot and compute the rmse. Is it a good model? Let's see how it generalizes. Test your model on {X2,t2}. Is it as good as the first one? How do you explain the difference between the two? What are the key concepts behind that?

1.1.4 Stochastic Gradient Descent (20 min)

Back-propagation for bi-variate Linear Regression



w_1
x_1
a_1
z_1
t_{pred}
x_2
a_2
$\frac{\partial L}{\partial b}$
$\frac{\partial L}{\partial t_{pred}}$
b
w_2

$$\begin{aligned}
L &= (t_{pred} - t_{true})^2 \\
\frac{\partial L}{\partial t_{pred}} &= 2(t_{pred} - t_{true}) \\
\frac{\partial L}{\partial b} &= \frac{\partial L}{\partial t_{pred}} \frac{\partial t_{pred}}{\partial b} \\
&= \frac{\partial L}{\partial t_{pred}} 1 \\
&\vdots
\end{aligned}$$

Figure 1: Computational graph of a bi-variate linear regression. The red arrows represent the gradients flowing from the loss to the weights. **Compute the missing gradients**

While the pseudo-inverse method gives the exact solution, it isn't always possible to inverse matrices. It remains however possible to update the parameters or weights in an iterative fashion by using Stochastic Gradient Descent (SGD). Through the computation of the gradient of the weights with respect to the objective function (= RMSE in our case) for an input $\mathbf{x}$, it is possible to approach the optimal solution. We also refer to this objective function as the Loss ($\mathcal{L}$). During the SGD process the point is to *minimize* this Loss.
Transcribing this to equations we obtain:

$$t_{pred} = \mathbf{wx} \tag{6}$$

$$\mathcal{L}(t_{pred}, t_{true}) = (t_{pred} - t_{true})^2 \tag{7}$$

$$\mathbf{w} = \mathbf{w} - \alpha \frac{\partial L}{\partial \mathbf{w}} \iff w_i = w_i - \alpha \frac{\partial L}{\partial w_i} \tag{8}$$

with t_{true} the true output associated to $\mathbf{x}$. It is now your turn to implement SGD for bi-variate linear regression *with* bias. Refer to Figure 1 to implement the gradient computation and fill in the missing gradients. Does your solution converge to the optimal solution? Try **tuning** the meta-parameters (`n_epochs`, `lr`, `lr_annealing`). Does this make the solution better? Which meta-parameter is the most important one? And what happens when you skew the *initial* value of the parameters towards their optimal values?

1.2 Radial Basis Function Networks

During this part of the session, you will implement the three learning steps of RBFNs. Thereafter, you will use the RBFN that you have just implemented to approximate the two bivariate functions

$$f : \Re^2 \to \Re : f(\mathbf{x}) = \frac{\sin \|\mathbf{x}\|_2}{\|\mathbf{x}\|_2}$$

$$f : \Re^2 \to \Re : f(\mathbf{x}) = e^{-\frac{1}{2}\|\mathbf{x}\|_2^2}$$

where $\|\mathbf{x}\|_2$ is the Euclidean norm of $\mathbf{x}$. For each function, a dataset is provided which contains a few training instances.

Radial Basis Function Networks Notations

A RBFN model can be seen as an extension of the linear model

$$y = \sum_{i=1}^{D} w_i x_i + w_0$$

where D is the dimensionality of $\mathbf{x}$. Instead of working with the input vectors $\mathbf{x}$, we will work with a non-linear transformation of these inputs: the output of a RBFN is then

$$y = \sum_{i=1}^{M} w_i \phi_i(\mathbf{x}) + w_0$$

where

$$\phi : \mathbf{x} \in \Re^D \to \phi(\mathbf{x}) \in \Re^M$$

is a non-linear transformation and M is the number of components (or basis functions). Theoretically, any non-linear transformation ϕ could be used. In this session, we will use a Gaussian function of the form

$$\phi_i(\mathbf{x}, \mathbf{c}^i, \sigma^i) = \exp\Big(-\frac{\|\mathbf{x} - \mathbf{c}^i\|^2}{2{\sigma^i}^2}\Big)$$

where the index $i = 1, \ldots, M$ indicates the component of the ϕ function. Two parameters are associated with each function and must be learned: a center $\mathbf{c}^i$, belonging to the same space as $\mathbf{x}$, and a gaussian width σ^i.

Learning Procedure A RBFN contains three different types of parameters:

- the position of the centers $\mathbf{c}^i$;
- the width of the Gaussian kernels σ^i;
- the weights of the linear model w_i.

While it is possible to use a single training procedure to learn all these parameters (e.g. by using gradient descent), it is often preferred to divide the learning in three distinct phases, each dedicated to one type of parameters.

1.2.1 Position of the centers

In the first phase of the learning process, the aim is to spread the centers in a way that mimics the density of probability of the input data. This is done by applying a vector quantization method. The centroids the method finds become the centers of your RBFN model. To implement this phase of training, use your code from the first session.

1.2.2 Widths of the Gaussian Functions

The widths of the gaussian kernels should depend on the density of the data around each center. A simple solution consists in choosing the width as the mean of distances between the data points inside a cluster (or Voronoï region) and its center, multiplied by a smoothing factor h (width scaling factor), common to all basis functions.

1.2.3 Non-linear transformation to learn weights (20 min)

Once the centers and widths are fixed, learning the remaining parameters (the weights w_i) becomes a linear problem with inputs $\phi_i(\mathbf{x})$ instead of $\mathbf{x}$. Just like what you have seen for the linear model, it is possible to optimize the weights w_i of the RBFN by optimizing an error criterion. We ask you to **implement** the non-linear transformation described above so that the pseudo-inverse method can be applied and the weights learned.