ICT for Health
Neural networks/Tensorflow

Monica Visintin

Politecnico di Torino

2016/17

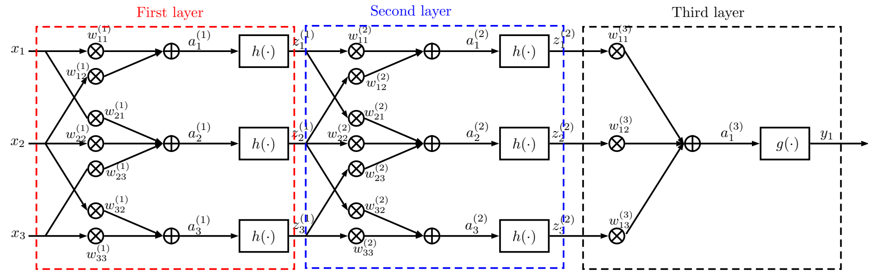
Table of Contents

1. **Neural networks**
   - Structures
   - Some math
   - An example
2. **TensorFlow**
   - Brief description
   - An example of a TensorFlow/Python script
   - The objects of TensorFlow
3. **Laboratory #6**
4. **Laboratory #7**

Table of Contents

1. **Neural networks**
   - Structures
   - Some math
   - An example
2. **TensorFlow**
   - Brief description
   - An example of a TensorFlow/Python script
   - The objects of TensorFlow
3. **Laboratory #6**
4. **Laboratory #7**

An example of a neural network



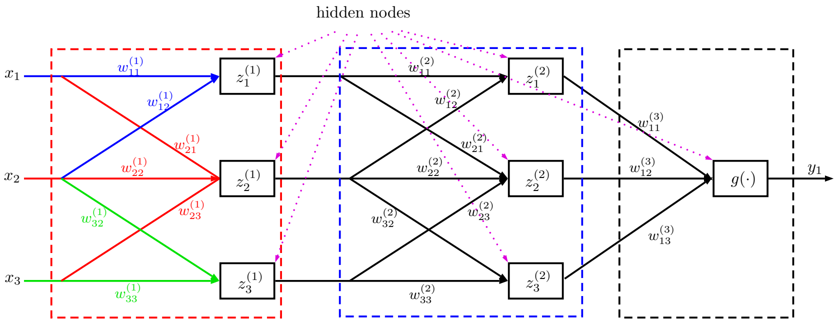
$\boxed{\text{Inputs: } x_1,\ x_2,\ x_3}$ $\boxed{\text{Weights: } w_{ij}^{(s)}}$ $\boxed{\text{Activations: } a_i^{(s)}}$ $\boxed{\text{Activation functions: } h(\cdot),\ g(\cdot)}$ $\boxed{\text{Output: } y_1}$

$$a_1^{(1)} = w_{11}^{(1)}x_1 + w_{12}^{(1)}x_2 \qquad z_1^{(1)} = h\left(a_1^{(1)}\right) \qquad a_1^{(2)} = w_{11}^{(2)}z_1^{(1)} + w_{12}^{(2)}z_2^{(1)} \qquad z_1^{(2)} = h\left(a_1^{(2)}\right) \qquad a_1^{(3)} = w_{11}^{(3)}z_1^{(2)} + w_{12}^{(3)}z_2^{(2)} + w_{13}^{(3)}z_3^{(2)}$$

$$a_2^{(1)} = w_{21}^{(1)}x_1 + w_{22}^{(1)}x_2 \qquad z_2^{(1)} = h\left(a_2^{(1)}\right) \qquad a_2^{(2)} = w_{21}^{(2)}z_1^{(1)} + w_{22}^{(2)}z_2^{(1)} \qquad z_2^{(2)} = h\left(a_2^{(2)}\right) \qquad y_1 = g\left(a_1^{(3)}\right)$$

$$a_3^{(1)} = w_{32}^{(1)}x_2 + w_{33}^{(1)}x_3 \qquad z_3^{(1)} = h\left(a_3^{(1)}\right) \qquad a_3^{(2)} = w_{32}^{(2)}z_2^{(1)} + w_{33}^{(2)}z_3^{(1)} \qquad z_3^{(2)} = h\left(a_3^{(2)}\right)$$

An example (simplified graph)



Inputs: $x_1,\ x_2,\ x_3$

Weights: $w_{ij}^{(s)}$

Activation functions: $h(\cdot),\ g(\cdot)$

Output: y_1

$$z_1^{(1)} = h\left(w_{11}^{(1)} x_1 + w_{12}^{(1)} x_2\right)$$
$$z_2^{(1)} = h\left(w_{21}^{(1)} x_1 + w_{22}^{(1)} x_2\right)$$
$$z_3^{(1)} = h\left(w_{32}^{(1)} x_2 + w_{33}^{(1)} x_3\right)$$

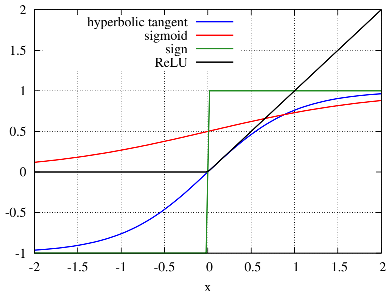
$$z_1^{(2)} = h\left(w_{11}^{(2)} z_1^{(1)} + w_{12}^{(2)} z_2^{(1)}\right)$$
$$z_2^{(2)} = h\left(w_{21}^{(2)} z_1^{(1)} + w_{22}^{(2)} z_2^{(1)}\right)$$
$$z_3^{(2)} = h\left(w_{32}^{(2)} z_2^{(1)} + w_{33}^{(2)} z_3^{(1)}\right)$$

$$y_1 = g\left(w_{11}^{(3)} z_1^{(2)} + w_{12}^{(3)} z_2^{(2)} + w_{13}^{(3)} z_3^{(2)}\right)$$

Nonlinearities [1]

Typical nonlinearities for the hidden nodes:

1. $h(x) = \tanh(x)$ **hyperbolic tangent**
2. $h(x) = \sigma(x) = 1/(1+e^{-x})$ **sigmoid** (also called logistic function)
3. $h(x) =$ sign (x)
4. $h(x) = 0$ if $x < 0$, $h(x) = x$ for $x \geq 0$ **ReLU (Rectified Linear Unit)**



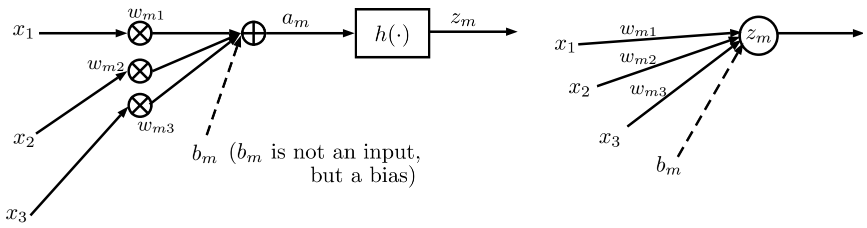
Nonlinearities [2]

Nonlinearities for the output node depend on the problem:

1. for regression, $g(x) = x$ (linear activation function)
2. for two-class classification/clustering $g(x) = \tanh(x)$ or $g(x) = \sigma(x)$ or $g(x) = \text{sign}(x)$
3. for multiple-class classification/clustering, having L output nodes, the **softmax** nonlinearity is

$$g(x_k) = \frac{e^{x_k}}{\sum_{h=1}^{L} e^{x_h}}$$

An example (single hidden node, layer 1)



$$z_m = h(a_m) = h(w_{m1}x_1 + w_{m2}x_2 + w_{m3}x_3 + b_m) \qquad a_m = \mathbf{x}^T\mathbf{w}_m + b_m$$

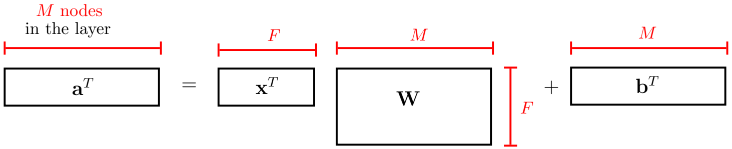
$\boxed{\text{Inputs: } x_1,\, x_2,\, x_3}$ $\boxed{\text{Weights: } w_{Mj}}$ $\boxed{\text{Activations: } a_i^{(s)}}$ $\boxed{\text{Activation function: } h(\cdot)}$

Matrices and vectors [1]

$$a_1 = \mathbf{x}^T\mathbf{w}_1 + b_1, \quad a_2 = \mathbf{x}^T\mathbf{w}_2 + b_2, \quad \ldots \quad a_M = \mathbf{x}^T\mathbf{w}_M + b_M$$

$$[a_1, a_2, \cdots, a_M] = \mathbf{x}^T[\mathbf{w}_1, \mathbf{w}_2, \cdots, \mathbf{w}_M] + [b_1, b_2, \cdots, b_M]$$

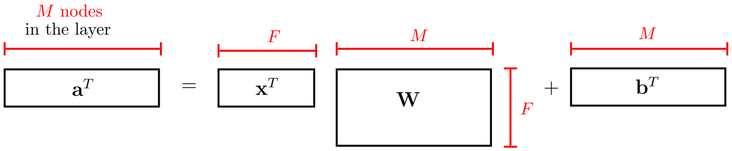
$$\mathbf{a}^T = \mathbf{x}^T\mathbf{W} + \mathbf{b}^T$$



Matrices and vectors [2]

Input made of 1 row vector $\mathbf{x}^T$ with F features (just 1 patient):

$$\mathbf{a}^T = \mathbf{x}^T\mathbf{W} + \mathbf{b}^T$$



Matrices and vectors [3]

Input made of N vectors with F features (N patients), i.e. matrix $\mathbf{X}$ with N rows and F columns (note that the k-th row of matrix $\mathbf{X}$ is $\mathbf{x}^T(k)$):

$$\mathbf{a}^T(1) = \mathbf{x}^T(1)\mathbf{W} + \mathbf{b}^T$$

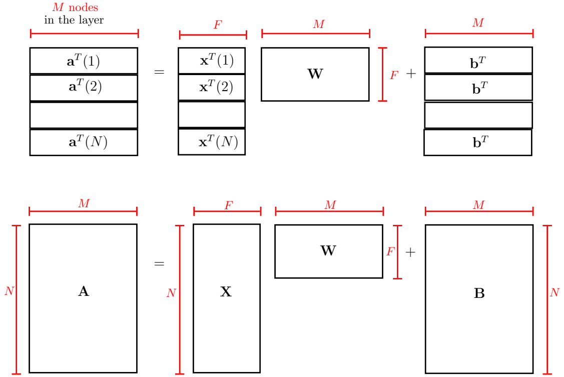
$$\mathbf{a}^T(2) = \mathbf{x}^T(2)\mathbf{W} + \mathbf{b}^T$$

$$\mathbf{a}^T(N) = \mathbf{x}^T(N)\mathbf{W} + \mathbf{b}^T$$

$$\mathbf{A} = \mathbf{X}\mathbf{W} + \mathbf{1}\mathbf{b}^T$$

or

$$\mathbf{A} = \mathbf{X}\mathbf{W} + \mathbf{B}$$



Matrices and vectors [4]

If we assume that all the nodes in layer 1 use nonlinearity $h(\cdot)$ and that $h(\mathbf{X})$ operates element-wise on each of the elements of the matrix $\mathbf{X}$, then the M outputs of layer 1 can be written as

$$\mathbf{Z}^{(1)} = h\left(\mathbf{A}^{(1)}\right) = h\left(\mathbf{X}\mathbf{W}^{(1)} + \mathbf{B}^{(1)}\right)$$

and the outputs of the second layer as

$$\mathbf{Z}^{(2)} = h\left(\mathbf{A}^{(2)}\right) = h\left(\mathbf{Z}^{(1)}\mathbf{W}^{(2)} + \mathbf{B}^{(2)}\right)$$

$$\mathbf{Z}^{(2)} = h\left(h\left(\mathbf{X}\mathbf{W}^{(1)} + \mathbf{B}^{(1)}\right)\mathbf{W}^{(2)} + \mathbf{B}^{(2)}\right)$$

If $h(\cdot)$ were linear, i.e. $h(x) = x$, then we would have a normal regression problem, that could be solved with 1 layer:
$\mathbf{Z}^{(2)} = \mathbf{X}\mathbf{W} + \mathbf{B}$, being $\mathbf{W} = \mathbf{W}^{(1)}\mathbf{W}^{(2)}$ and $\mathbf{B} = \mathbf{B}^{(1)}\mathbf{W}^{(2)} + \mathbf{B}^{(2)}$

Matrices and vectors [5]

Note that

- matrix $\mathbf{W}^{(1)}$ has F rows (as many as the features/columns of $\mathbf{X}$) and M_1 columns (as many as the nodes in layer 1);
- matrix $\mathbf{W}^{(2)}$ has M_1 rows (as many as the nodes in layer 1) and M_2 columns (as many as the nodes in layer 2);
- matrix $\mathbf{B}^{(1)}$ has N rows (all equal, as many as the patients/rows in matrix $\mathbf{X}$) and M_1 columns (as many as the nodes in layer 1)
- matrix $\mathbf{B}^{(2)}$ has N rows (all equal) and M_2 columns (as many as the nodes in layer 2)

In general, we have an input matrix $\mathbf{X}$ and desired outputs, we must find the optimum matrices $\mathbf{W}^{(1)}$, $\mathbf{W}^{(2)}, \ldots,$ $\mathbf{B}^{(1)}$, $\mathbf{B}^{(2)}, \ldots,$ that minimize an objective function.

Neural networks in ICT for health

Neural networks can be used to solve:

- regression problems
- classification problems
- clustering problems

The structure of the neural network and the activation functions/nonlinearities are chosen by the designer, the variables to be optimized, according to the inputs and the desired outputs, are the weights $w_{ij}^{(s)}$. The typical way to find the optimized weights is the **gradient algorithm**.

The advantage of neural networks is that

- they can be implemented in hardware
- they can be simulated using parallel processing in GPUs (optimization becomes very fast)

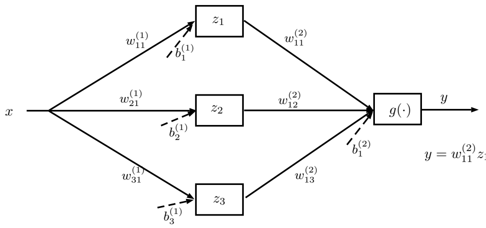
An example of a neural network [1]

Consider the function $f(x) = e^{-x^2}$ and a neural network with input x, three hidden nodes with the **hyperbolic tangent activation function**

$$h(a) = \tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

and the output node with $y = g(a) = a$ (**linear activation function**). We want to find the weights w_{ij} so that $y \simeq f(x)$. Essentially, we want to find a way to approximate a Gaussian function with the sum of weighted hyperbolic tangents

An example of a neural network [2]



$$z_1 = h\left(w_{11}^{(1)}x + b_1^{(1)}\right)$$
$$z_2 = h\left(w_{21}^{(1)}x + b_2^{(1)}\right)$$
$$z_3 = h\left(w_{31}^{(1)}x + b_3^{(1)}\right)$$
$$y = w_{11}^{(2)}z_1 + w_{12}^{(2)}z_2 + w_{13}^{(2)}z_3 + b_1^{(2)}$$

In the training phase, we use N different inputs $x(1), \ldots, x(N)$ to get the outputs $y(1), \ldots, y(N)$. We know $t(1) = f(x(1)), \ldots, t(N) = f(x(N))$. We update the weights $w_{ij}(1)$ and $w_{ij}^{(2)}$ of the neural network in order to minimize the square error $e(\mathbf{w})$:

$$e(\mathbf{w}) = \sum_{n=1}^{N} [y(n) - t(n)]^2 = \sum_{n=1}^{N} e(n)$$

An example of a neural network [3]

Minimization is performed iteratively using the gradient algorithm. Let us evaluate the required partial derivatives:

$$e(n) = [y(n) - t(n)]^2$$

$$y(n) = w_{11}^{(2)} z_1(n) + w_{12}^{(2)} z_2(n) + w_{13}^{(2)} z_3(n) + b_1^{(2)}$$

$$\frac{\partial e(n)}{\partial w_{1k}^{(2)}} = 2[y(n) - t(n)] \frac{\partial y(n)}{\partial w_{1k}^{(2)}} = 2[y(n) - t(n)] z_k(n), \quad k = 1, 2, 3$$

$$\frac{\partial e(n)}{\partial b_1^{(2)}} = 2[y(n) - t(n)] \frac{\partial y(n)}{\partial b_1^{(2)}} = 2[y(n) - t(n)]$$

An example of a neural network [4]

$$y(n) = w_{11}^{(2)} z_1(n) + w_{12}^{(2)} z_2(n) + w_{13}^{(2)} z_3(n) + b_1^{(2)}$$

$$z_m(n) = h\left(a_m^{(1)}(n)\right), \quad a_m^{(1)}(n) = w_{m1}^{(1)} x(n) + b_m^{(1)}$$

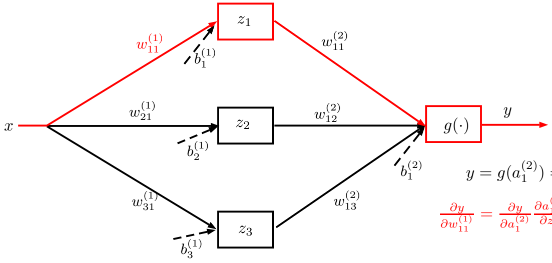
$$\begin{aligned}\frac{\partial e(n)}{\partial w_{m1}^{(1)}} &= 2[y(n) - t(n)] \frac{\partial y(n)}{\partial z_m(n)} \frac{\partial z_m(n)}{\partial a_m^{(1)}(n)} \frac{\partial a_m^{(1)}(n)}{\partial w_{m1}^{(1)}} \\ &= 2[y(n) - t(n)] w_{1m}^{(2)} h'\left(a_m^{(1)}(n)\right) x, \quad m = 1, 2, 3\end{aligned}$$

$$\frac{\partial e(n)}{\partial b_m^{(1)}} = 2[y(n) - t(n)] \frac{\partial y(n)}{\partial z_m(n)} \frac{\partial z_m(n)}{\partial a_m^{(1)}(n)} \frac{\partial a_m^{(1)}(n)}{\partial b_m^{(1)}} = 2[y(n) - t(n)] w_{1m}^{(2)} h'\left(a_m^{(1)}(n)\right)$$

$$h(x) = \frac{1}{1 + e^{-x}}, \quad \Longrightarrow \quad \frac{dh(x)}{dx} = h'(x) = \frac{e^{-x}}{[1 + e^{-x}]^2} = e^{-x} h^2(x)$$

An example of a neural network [5]

To evaluate $\frac{\partial y}{\partial w_{11}^{(1)}}$, it is necessary to find the path that leads backwards from y to $w_{11}^{(1)}$ and multiply the partial derivatives along the path. The network stores the values of x, the activations $a_k^{(i)}$, the weights $w_{kj}^{(i)}$, etc. and stores also the partial derivatives so that it is possible, moving backwards, to evaluate the gradient of $e(n)$ and update the weights.



$$z_1 = h(a_1^{(1)}) = h\left(w_{11}^{(1)}x + b_1^{(1)}\right)$$

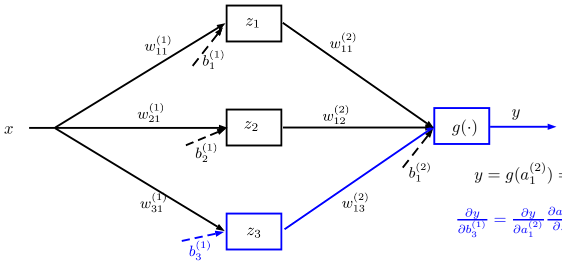
$$z_2 = h(a_2^{(1)}) = h\left(w_{21}^{(1)}x + b_2^{(1)}\right)$$

$$z_3 = h(a_3^{(1)}) = h\left(w_{31}^{(1)}x + b_3^{(1)}\right)$$

$$y = g(a_1^{(2)}) = g(w_{11}^{(2)}z_1 + w_{12}^{(2)}z_2 + w_{13}^{(2)}z_3 + b_1^{(2)})$$

$$\frac{\partial y}{\partial w_{11}^{(1)}} = \frac{\partial y}{\partial a_1^{(2)}}\frac{\partial a_1^{(2)}}{\partial z_1}\frac{\partial z_1}{\partial a_1^{(1)}}\frac{\partial a_1^{(1)}}{\partial w_{11}^{(1)}} = g'(a_1^{(2)})w_{11}^{(2)}h'(a_1^{(1)})x$$

An example of a neural network [6]



$$z_1 = h(a_1^{(1)}) = h\left(w_{11}^{(1)} x + b_1^{(1)}\right)$$

$$z_2 = h(a_2^{(1)}) = h\left(w_{21}^{(1)} x + b_2^{(1)}\right)$$

$$z_3 = h(a_3^{(1)}) = h\left(w_{31}^{(1)} x + b_3^{(1)}\right)$$

$$y = g(a_1^{(2)}) = g(w_{11}^{(2)} z_1 + w_{12}^{(2)} z_2 + w_{13}^{(2)} z_3 + b_1^{(2)})$$

$$\frac{\partial y}{\partial b_3^{(1)}} = \frac{\partial y}{\partial a_1^{(2)}} \frac{\partial a_1^{(2)}}{\partial z_3} \frac{\partial z_3}{\partial a_3^{(1)}} \frac{\partial a_3^{(1)}}{\partial b_3^{(1)}} = g'(a_1^{(2)}) w_{13}^{(2)} h'(a_3^{(1)})$$

If more than one path exists from the input to the weight $w_{kj}^{(i)}$, then it is necessary to add together the partial derivatives evaluated along each path.

An example of a neural network [7]

Note also that

- several minima exist: we can exchange the roles of the hidden nodes z_1 and z_2, for example, and get the same result
- different objective functions can be used, depending on the problem we want to solve:
  1. $$\text{Regression: } e = \sum_{n=1}^{N} [y(n) - t(n)]^2,$$
  2. $$\text{Classification with two classes: } e = \sum_{n=1}^{N} [y(n) - t(n)]^2,$$

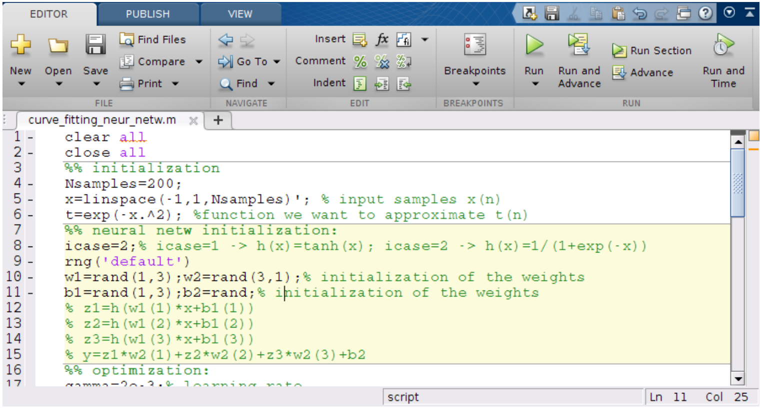
An example of a neural network [8]

3. Multiclass classification (K classes): **cross-entropy**:

$$e = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_k(n)\log y_k(n)$$

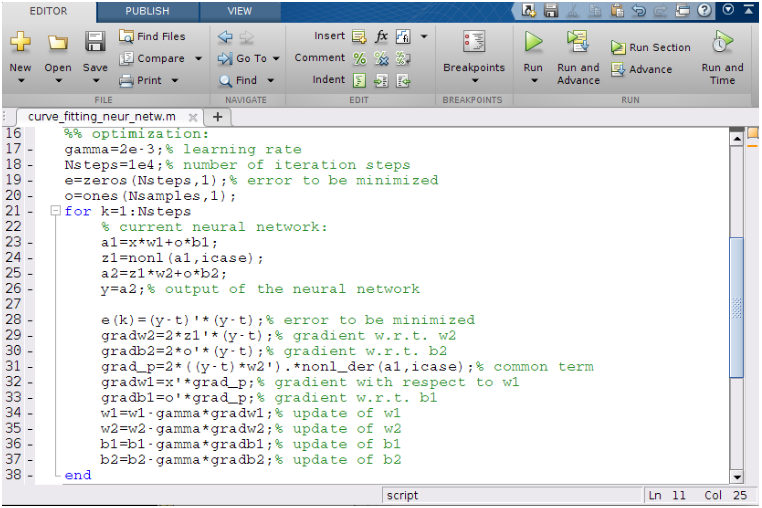
where $y_k(n)$, the output of the k-th neural network output node at time n, is obtained with a softmax nonlinearity and $t_k(n) = 1$ if the true class at time n is k, otherwise $t_k(n) = 0$.

An example of a neural network [9]



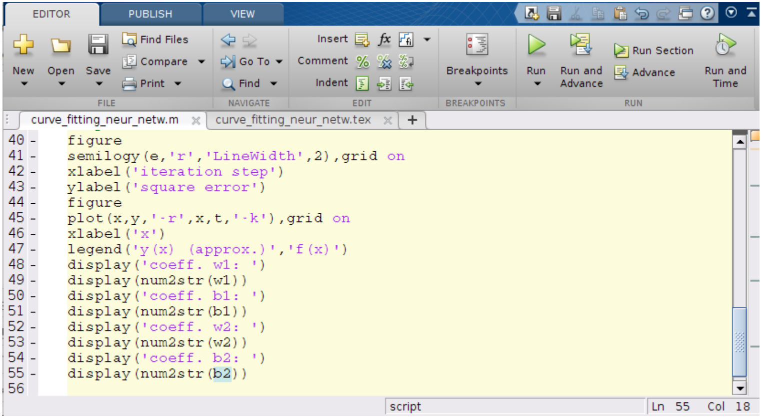
```matlab
clear all
close all
%% initialization
Nsamples=200;
x=linspace(-1,1,Nsamples)'; % input samples x(n)
t=exp(-x.^2); %function we want to approximate t(n)
%% neural netw initialization:
icase=2;% icase=1 -> h(x)=tanh(x); icase=2 -> h(x)=1/(1+exp(-x))
rng('default')
w1=rand(1,3);w2=rand(3,1);% initialization of the weights
b1=rand(1,3);b2=rand;% initialization of the weights
% z1=h(w1(1)*x+b1(1))
% z2=h(w1(2)*x+b1(2))
% z3=h(w1(3)*x+b1(3))
% y=z1*w2(1)+z2*w2(2)+z3*w2(3)+b2
%% optimization:
gamma=2e-3;% learning rate
```

An example of a neural network [10]



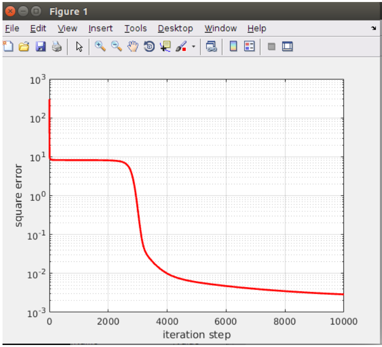
```
%% optimization:
gamma=2e-3;% learning rate
Nsteps=1e4;% number of iteration steps
e=zeros(Nsteps,1);% error to be minimized
o=ones(Nsamples,1);
for k=1:Nsteps
    % current neural network:
    a1=x*w1+o*b1;
    z1=nonl(a1,icase);
    a2=z1*w2+o*b2;
    y=a2;% output of the neural network

    e(k)=(y-t)'*(y-t);% error to be minimized
    gradw2=2*z1'*(y-t);% gradient w.r.t. w2
    gradb2=2*o'*(y-t);% gradient w.r.t. b2
    grad_p=2*((y-t)*w2').*nonl_der(a1,icase);% common term
    gradw1=x'*grad_p;% gradient with respect to w1
    gradb1=o'*grad_p;% gradient w.r.t. b1
    w1=w1-gamma*gradw1;% update of w1
    w2=w2-gamma*gradw2;% update of w2
    b1=b1-gamma*gradb1;% update of b1
    b2=b2-gamma*gradb2;% update of b2
end
```

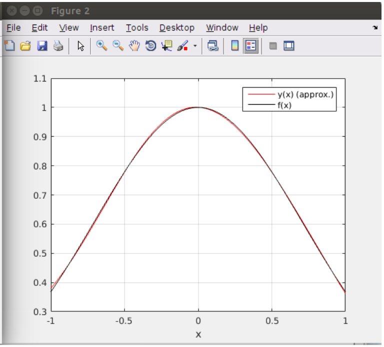
An example of a neural network [11]



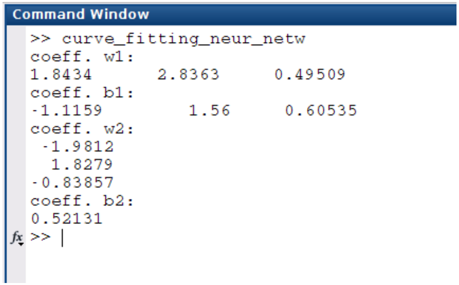
```
    figure
    semilogy(e,'r','LineWidth',2),grid on
    xlabel('iteration step')
    ylabel('square error')
    figure
    plot(x,y,'-r',x,t,'-k'),grid on
    xlabel('x')
    legend('y(x)  (approx.)','f(x)')
    display('coeff. w1: ')
    display(num2str(w1))
    display('coeff. b1: ')
    display(num2str(b1))
    display('coeff. w2: ')
    display(num2str(w2))
    display('coeff. b2: ')
    display(num2str(b2))
```

An example of a neural network [12]





An example of a neural network [13]



```
>> curve_fitting_neur_netw
coeff. w1:
1.8434        2.8363       0.49509
coeff. b1:
-1.1159           1.56       0.60535
coeff. w2:
 -1.9812
  1.8279
-0.83857
coeff. b2:
0.52131
>> |
```

For $x \in [-1, 1]$

$$\begin{aligned} f(x) &= e^{-x^2} \\ &\simeq y(x) = -1.9812\,\tanh(1.8434\,x - 1.1159) \\ &+ 1.8279\,\tanh(2.8363\,x + 1.56) \\ &- 0.83857\,\tanh(0.49509\,x + 0.60535) + 0.52131 \end{aligned}$$

Table of Contents

1 Neural networks
- Structures
- Some math
- An example

2 TensorFlow
- Brief description
- An example of a TensorFlow/Python script
- The objects of TensorFlow

3 Laboratory #6

4 Laboratory #7

Neural networks and TensorFlow

TensorFlow is a tool developed by Google and released in October 2015 as free software. TensorFlow

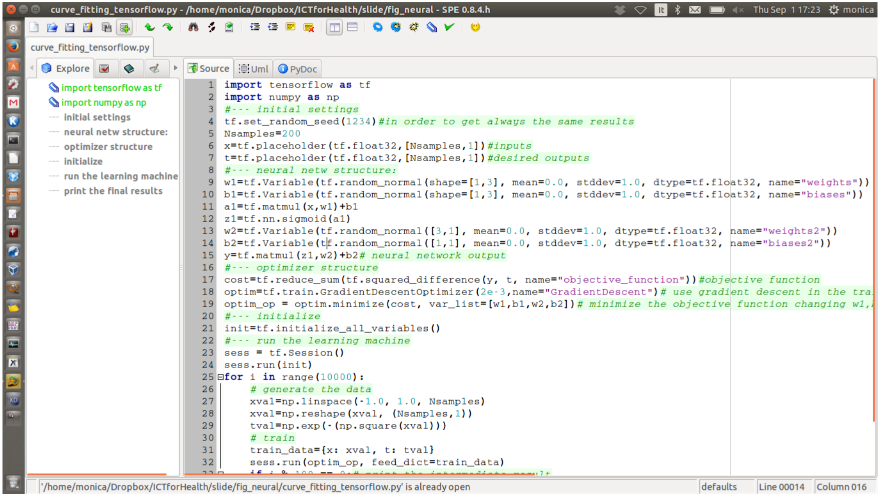
- allows to implement neural networks and run them using in parallel the processors of a PC, the GPU, clusters of PC, clusters of GPUs; it works on mobile phones, tablets, etc.
- has many machine learning algorithms already implemented
- automatically evaluates the gradients/derivatives
- can be seen as a set of APIs to be used in Python or C++ scripts

We will use TensorFlow as a Python extension.

TensorFlow

From `https://www.tensorflow.org/`: “TensorFlow is an open source software library for numerical computation using data flow graphs. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) communicated between them. The flexible architecture allows you to deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device with a single API. TensorFlow was originally developed by researchers and engineers working on the Google Brain Team within Google’s Machine Intelligence research organization for the purposes of conducting machine learning and deep neural networks research, but the system is general enough to be applicable in a wide variety of other domains as well.”

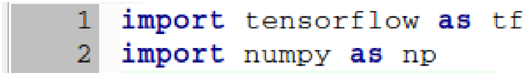
The same example in TensorFlow



```python
import tensorflow as tf
import numpy as np
#--- initial settings
tf.set_random_seed(1234)#in order to get always the same results
Nsamples=200
x=tf.placeholder(tf.float32,[Nsamples,1])#inputs
t=tf.placeholder(tf.float32,[Nsamples,1])#desired outputs
#--- neural netw structure:
w1=tf.Variable(tf.random_normal(shape=[1,3], mean=0.0, stddev=1.0, dtype=tf.float32, name="weights"))
b1=tf.Variable(tf.random_normal(shape=[1,3], mean=0.0, stddev=1.0, dtype=tf.float32, name="biases"))
a1=tf.matmul(x,w1)+b1
z1=tf.nn.sigmoid(a1)
w2=tf.Variable(tf.random_normal([3,1], mean=0.0, stddev=1.0, dtype=tf.float32, name="weights2"))
b2=tf.Variable(tf.random_normal([1,1], mean=0.0, stddev=1.0, dtype=tf.float32, name="biases2"))
y=tf.matmul(z1,w2)+b2# neural network output
#--- optimizer structure
cost=tf.reduce_sum(tf.squared_difference(y, t, name="objective_function"))#objective function
optim=tf.train.GradientDescentOptimizer(2e-3,name="GradientDescent")# use gradient descent in the tra
optim_op = optim.minimize(cost, var_list=[w1,b1,w2,b2])# minimize the objective function changing w1,
#--- initialize
init=tf.initialize_all_variables()
#--- run the learning machine
sess = tf.Session()
sess.run(init)
for i in range(10000):
    # generate the data
    xval=np.linspace(-1.0, 1.0, Nsamples)
    xval=np.reshape(xval, (Nsamples,1))
    tval=np.exp(-(np.square(xval)))
    # train
    train_data={x: xval, t: tval}
    sess.run(optim_op, feed_dict=train_data)
    if i % 100 == 0:# print the intermediate result
```

'/home/monica/Dropbox/ICTforHealth/slide/fig_neural/curve_fitting_tensorflow.py' is already open

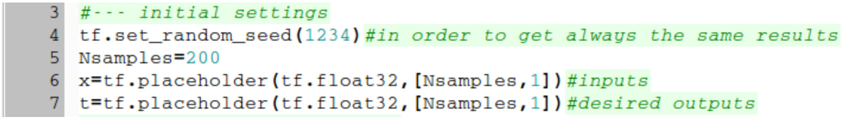
Import the TensorFlow APIs



```python
import tensorflow as tf
import numpy as np
```

You definitely need to import TensorFlow, but you typically need to import also Numpy, to exploit its functions for the generation and manipulation of matrices/vectors. Many Numpy functions are present in TensorFlow, but they are used to work with tensors, i.e. the inputs and outputs of each neural network node (more complex objects than Numpy vectors, as we'll see).

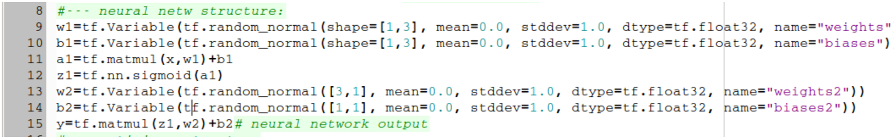
The placeholders



```
#--- initial settings
tf.set_random_seed(1234)#in order to get always the same results
Nsamples=200
x=tf.placeholder(tf.float32,[Nsamples,1])#inputs
t=tf.placeholder(tf.float32,[Nsamples,1])#desired outputs
```

You have to imagine that you build a hardware neural network and in this initial phase you assemble it: the placeholders are objects that will contain the input data $x(n)$ and the desired output data $t(n)$. You can imagine the placeholders as empty boxes in which you'll save (later) the inputs of the network. As in the Matlab version, we specify the seed of the random samples generator to be able to get the same results each time we run the program.

The neural network structure [1]



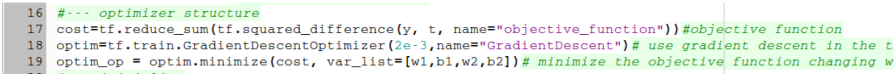
```
#--- neural netw structure:
w1=tf.Variable(tf.random_normal(shape=[1,3], mean=0.0, stddev=1.0, dtype=tf.float32, name="weights"
b1=tf.Variable(tf.random_normal(shape=[1,3], mean=0.0, stddev=1.0, dtype=tf.float32, name="biases")
a1=tf.matmul(x,w1)+b1
z1=tf.nn.sigmoid(a1)
w2=tf.Variable(tf.random_normal([3,1], mean=0.0, stddev=1.0, dtype=tf.float32, name="weights2"))
b2=tf.Variable(tf.random_normal([1,1], mean=0.0, stddev=1.0, dtype=tf.float32, name="biases2"))
y=tf.matmul(z1,w2)+b2# neural network output
```

- `tf.variables` are tensors/arrays that are going to change during the optimization iterations.
- `tf.random.normal` generates a random tensor/array with the specified size and according to a Gaussian distribution with specified mean and variance
- `tf.matmul` multiplies element-wise two tensors
- `tf.nn.sigmoid` applies the sigmoid nonlinearity to the input tensor

The neural network structure [2]

We picked from the shelf the boxes `tf.random.normal`, `tf.matmul`, `tf.nn.sigmoid`, we put them on the laboratory table, we connected them together: we have the “hardware” neural network on the table. From the point of view of TensorFlow, we have a **graph**. If you want you can have several systems/graphs on your laboratory table/TensorFlow .py file; there is a command that specifies that a given box/operation belongs to a given graph. In general you don’t need more than one graph and, in this case, you don’t need to specify its name.

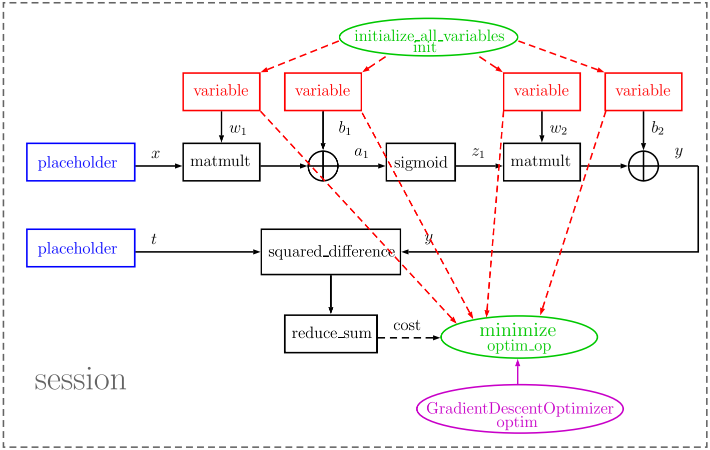
Let’s specify the optimization technique



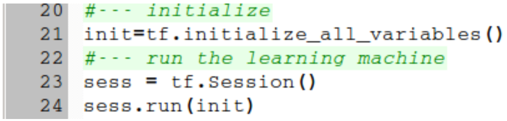
```
#--- optimizer structure
cost=tf.reduce_sum(tf.squared_difference(y, t, name="objective_function"))#objective function
optim=tf.train.GradientDescentOptimizer(2e-3,name="GradientDescent")# use gradient descent in the t
optim_op = optim.minimize(cost, var_list=[w1,b1,w2,b2])# minimize the objective function changing w
```

- `tf.squared_difference(y,t)` evaluates tensor `(y-t).*(y-t)` (element-wise)
- `tf.reduce_sum(x)` evaluates the sum of the elements of `x` and reduces by one the dimensions of the input tensor; we now have `cost`, the objective function
- `tf.train.GradientDescentOptimizer(c)` implements the gradient algorithm, using the learning rate `c`; it is as if we just picked a box from the shelf, we called it “optim”, we did not connect it to the other items we have on the table, yet.
- `optim.minimize(cost, var_list=[w1,b1,w2,b2])`: we now tell “optim” that it has to perform a minimization (the other possibility is maximization), we specify that the objective function is “cost”, and that it has to optimize the tensors `w1, b1, w2, b2`

A pictorial view of the graph
(Not generated by TensorBoard)



Let’s run the machine [1]

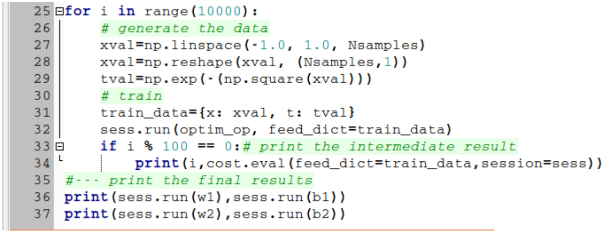


```
#--- initialize
init=tf.initialize_all_variables()
#--- run the learning machine
sess = tf.Session()
sess.run(init)
```

- First we initialize the variables `w1, b1, w2, b2` using `init=tf.initialize_all_variables()`
- Then we start an optimazion session using `sess = tf.Session()`
- Then we tell `sess` to run `init`, which initializes the variables

In the python script you can have several graphs, but each graph must have its own session; on the contrary you might have just one graph and run it under several parallel sessions, with different inputs (in the placeholders). In this case we have just one graph and it is not necessary to specify which graph has to be run by session `sess`.

Let’s run the machine [2]



```
for i in range(10000):
    # generate the data
    xval=np.linspace(-1.0, 1.0, Nsamples)
    xval=np.reshape(xval, (Nsamples,1))
    tval=np.exp(-(np.square(xval)))
    # train
    train_data={x: xval, t: tval}
    sess.run(optim_op, feed_dict=train_data)
    if i % 100 == 0:# print the intermediate result
        print(i,cost.eval(feed_dict=train_data,session=sess))
#--- print the final results
print(sess.run(w1),sess.run(b1))
print(sess.run(w2),sess.run(b2))
```

- We perform ten thousand steps of the gradient algorithm
- Using Numpy, at each step we generate the input vector `xval` and the desired output vector `tval`
- With `train_data={x:  xval, t:  tval}` we specify that `xval` must be put in the placeholder `x` and `tval` in the placeholder `t`, and that these data form the training data for the algorithm

Let's run the machine [3]

- `sess.run(optim_op, feed_dict=train_data)` specifies that the session must run the optimization `optim_op` with the given training data

Note that

1. In this specific case, the generation of `xval` should be performed outside the loop (but we preferred to have it inside the loop, because this is what you typically do during optimization: at each run you generate a new input)
2. we can run several sessions (maybe on different CPUs) in which we have the same objects on the table/the same graph (the neural net with the minimizer), but we use different training sets; this is why the placeholders were invented.

The possibility of "easily" ask the computer to run the optimization on a specific CPU or the GPU or on a specific computer in a cluster is the added value of TensorFlow with respect to other tools devoted to machine learning.

What is a tensor? [1]

- In a nutshell, a tensor is a Numpy array
- More precisely, a tensor is an object, defined in file `.../tensorflow/python/framework/ops.py`. When you write `a=tf.zeros([3, 4], dtype=tf.int32,name=‘‘myzeros’’)` you create an object `a` with the following attributes:
    - `a.shape` (equal in this case to `[3,4]`)
    - `a.dtype` (equal to `int32`, 32 bit integer)
    - `a.name` (equal to `myzeros`) (the name is not mandatory)
    - `a.op` (equal to the operation that generates this tensor, in this case `tf.zeros`)
    - `a.consumers` (equal to the list of operations that use this tensor as input; it is important to know the consumers to be able to evaluate the gradients)

What is a tensor? [2]

From file ops.py, as comment in the definition of class Tensor: “A ”Tensor“ is a symbolic handle to one of the outputs of an ”Operation“. It does not hold the values of that operation’s output, but instead provides a means of computing those values in a TensorFlow ”Session“”. A tensor must be generated by an Operation.
At the link
`https://www.tensorflow.org/versions/r0.11/api_docs/python/framework.html#Tensor`
you find a more complete description of the tensor class and the associated methods

Example [1]

```
>>> import tensorflow as tf
>>> import numpy as np
>>> a=tf.constant(10)
>>> b=tf.constant(5)
>>> a.__dict__
{'_consumers': [], '_value_index': 0, '_shape': TensorShape([]), '_op':
<tensorflow.python.framework.ops.Operation object at 0x7fb8443d8310>, '_dtype':
tf.int32}
>>>b.__dict__
{'_consumers': [], '_value_index': 0, '_shape': TensorShape([]), '_op':
<tensorflow.python.framework.ops.Operation object at 0x7fb8443d82d0>, '_dtype':
tf.int32}
>>> c=tf.add(a,b)
>>> a.__dict__
{'_consumers': [<tensorflow.python.framework.ops.Operation object at 0x7fb8443d8510>],
'_value_index': 0, '_shape': TensorShape([]), '_op':
<tensorflow.python.framework.ops.Operation object at 0x7fb8443d8310>, '_dtype':
tf.int32}
>>> c.__dict__
{'_consumers': [], '_value_index': 0, '_shape': TensorShape([]), '_op':
<tensorflow.python.framework.ops.Operation object at 0x7fb8443d8510>, '_dtype':
tf.int32}
```

Operations that create tensors

1. Constant value tensors:
   tf.zeros(shape,dtype=tf.float32,name=None),
   tf.ones(shape,dtype=tf.float32,name=None),
   tf.fill(dims,value,name=None), tf.constant(value, dtype=None, shape=None, name='Const')
   tf.linspace(start,stop,num,name=None),
   tf.range(start,limit=None,delta=1,name='range')
2. Random Tensors: tf.random_normal(shape, mean=0.0,stddev=1.0,dtype=tf.float32,seed=None,name=None), tf.random_uniform(shape, min-val=0,maxval=None,dtype=tf.float32,seed=None,name=None), tf.random_gamma(shape, alpha, beta=None, dtype=tf.float32, seed=None, name=None)

Variables

Variables are tensors with special features. At each step the tensors in the graph are evaluated from the inputs (so you can clear the value of a tensor at the end of the step), whereas variables must be remembered from step to step (the new value is the old value plus a correction). In general all the parameters that must be optimized must be declared as variables. Of course variables must be initialized, which you typically do with the instruction `tf.initialize_all_variables()`

Operations on tensors [1]

https://www.tensorflow.org/versions/r0.11/api_docs/python/index.html

The following math operations are specifically implemented to be used with tensors:

- binary operators: tf.add, tf.sub, tf.mul, tf.div, tf.truediv, tf.floordiv, tf.mod, tf.lt, tf.le, tf.gt, tf.ge, tf.and, tf.or, tf.xor, tf.getitem, tf.squared_difference,...
- Unary operations: tf.invert, tf.neg, tf.abs, tf.neg, tf.sign, tf.inv, tf.square, tf.round, tf.sqrt, tf.pow, tf.exp, tf.log, tf.ceil, tf.floor, tf.cos, tf.sin, tf.tan, tf.acos, tf.asin, tf.atan, tf.erf, tf.erfc, ...
- Matrix math functions: tf.diag, tf.trace, tf.transpose, tf.matrix_diag, tf.matmul, tf.matrix_determinant, tf.matrix_inverse, tf.cholesky, tf.matrix_solve, tf.svd
- Complex number operations: tf.conj, tf. imag, tf.real, tf.complex_abs
- Fourier Transform Functions: tf.fft, tf.ifft, tf.fft2d, tf.ifft2d, tf.fft3d, tf.ifft3d

Operations on tensors [2]

https://www.tensorflow.org/versions/r0.11/api_docs/python/index.html

- Reduction: tf.reduce_sum, tf.reduce_prod, tf.reduce_min, tf.reduce_max, tf.reduce_mean, tf.cumsum, tf.cumprod
- Sequence comparison
- Logical operators: tf.logical_and, tf.logical_not, tf.logical_or, tf.logical_xor, ...
- Comparison operators: tf.equal, tf.not_equal, tf.less, tf.less_equal, tf.greater, tf.greater_equal, tf. where
- Images: tf.image.encode_jpeg, tf.image.decode_jpeg, tf.image.encode_png, tf.image.decode_png, ..., tf.image.flip_up_down, tf.image.flip_left_right, ...

Other operations/classes [1]

https://www.tensorflow.org/versions/r0.11/api_docs/python/index.html

- Neural networks: tf.nn.relu, tf.nn.softplus, tf.nn.softsign, tf.sigmoid, tf.tanh
- Classification: tf.nn.sigmoid_cross_entropy_with_logits, tf.nn.softmax,...
- Optimizers: tf.train.GradientDescentOptimizer, tf.train.AdadeltaOptimizer, tf.AdamOptimizer, ...
- Gradient computations: tf.gradients, tf.stop_gradient
- tf.contrib.learn.DNNClassifier, tf.contrib.learn.DNNRegressor, tf.contrib.learn.LinearClassifier, tf.contrib.learn.LinearRegressor, tf.contrib.learn.TensorFlowEstimator

References

- `https://www.tensorflow.org/`
- `http://jorditorres.org/first-contact-with-tensorflow/`
- Play with the web application
  `http://playground.tensorflow.org`

Table of Contents

1 Neural networks
- Structures
- Some math
- An example

2 TensorFlow
- Brief description
- An example of a TensorFlow/Python script
- The objects of TensorFlow

3 Laboratory #6

4 Laboratory #7

Laboratory #6 [1]

1. We work again on the data of the Parkinson disease but now using neural networks and TensorFlow (you must have Python and TensorFlow installed on your PC, possibly also an IDE like Spyder or SPE)
2. To import data in Python we suggest to exploit the `scipy.io.loadmat` function which loads files saved as `.mat` by Matlab. The Python lines are:
   ```
   import scipy.io as scio
   mat_file=scio.loadmat('matfilename.mat')
   data=mat_file.get('matrixname')
   ```
   Of course you first have to run a Matlab script that generates `matrixname` and saves it in file `matfilename.mat` through the instruction:
   ```
   save('matfilename.mat','matrixname');
   ```
   The Python structure of `data` is a Numpy array.

Laboratory #6 [2]

3. The matrix will have the averaged 22 features of the 42 patients at several days (990 rows, 22 columns). We will initially work on the regression of feature in column 7 (jitter percentage) and then on the regression of feature in column 5 (motor UPDRS). The regressors are those in columns 2 (age), 3 (sex), 7-22 (the other features) (we exclude feature 1: index of the patient, feature 4: progressive time, features 5, 6, 7, because we use them as regressand or are strictly related to the regressand). **It is very important that the features are normalized (zero mean and unitary variance).** You can choose whether to perform this initial processing in Matlab or in Python using Numpy functions `mean` and `var`. Remember that Python indexes of arrays start from 0, not from 1 as in Matlab.
4. Perform regression in this two cases:

Laboratory #6 [3]

- using initially a neural network without hidden nodes and linear output activation function, i.e. a neural network that outputs $\mathbf{Y} = \mathbf{X}\mathbf{W} + \mathbf{B}$ where $\mathbf{X}$ is the matrix with the regressors (990 rows, 18 columns), $\mathbf{W}$ is a column vector with 18 entries storing the unknown weights, $\mathbf{B}$ is a column vector with 990 rows all equal to the scalar unknown b, $\mathbf{Y}$ is a column vector with 990 entries that should be ideally equal to the regressand (either jitter or motor UPDRS). We are implementing again the same regression we implemented in Lab #1 but with TensorFlow. Use the Python/TensorFlow code described in the lecture, with the necessary changes. Note that there is no PCA function available in TensorFlow and we will not use PCA/PCR in Lab #6.
- using a neural network with two hidden layers (the first one with 18 hidden nodes, the second one with 10 hidden nodes) and tanh activation function (`tf.nn.tanh`). Use the Python/TensorFlow code described in the lecture, with the necessary changes.

Laboratory #6 [4]

- In both cases, appropriately set the learning rate and the number of iterations of the gradient algorithm (trial and error; start with few iterations and small learning rate, then increase these values)
- To view the results, you can import matplotlib:

  ```
  import matplotlib.pyplot as plt
  ```

  get the final result of regression as

  ```
  yval=y.eval(feed_dict=train_data,session=sess)
  ```

  and use, for example, the following lines:

  ```
  plt.plot(data_to_regress,'ro', label='regressand')
  plt.plot(yval,'bx', label='regression')
  plt.xlabel('case number')
  plt.grid(which='major', axis='both')
  plt.legend()
  plt.savefig('one.pdf',format='pdf')
  plt.show()
  ```

  The figure is saved as `one.pdf` in the same folder of the Python script.

Laboratory #6 [5]

5. Draw your conclusions, compare the results obtained with the neural network with those obtained in Lab #1, write the final report on regression

Table of Contents

1 Neural networks
- Structures
- Some math
- An example

2 TensorFlow
- Brief description
- An example of a TensorFlow/Python script
- The objects of TensorFlow

3 Laboratory #6

4 Laboratory #7

Laboratory #7 [1]

1. In laboratory number 7 we repeat the classification problem based on the arrhythmia data, first using only classes 1 (healthy patient) and 2 (patient with some level of arrhythmia), then using the 16 classes as specified by the medical doctors. We will only work on the training phase of the neural network (no testing phase).
2. Load in Python the Matlab matrix that stores the 257 features of the 452 patients and the vector that stores the classes of the 452 patients (same method used in Lab #6). If you did not normalize the data (mean equal to zero and variance equal to one for all the features) in Matlab, do it in Python using Numpy functions `np.mean` and `np.var`.

Laboratory #7 [2]

3. For the case of two classes, use the same two-layer neural network developed in Lab #6, but use the **sigmoid** instead of the tanh activation function (`tf.nn.sigmoid`) **for the hidden nodes and the output node**. Use 257 hidden nodes in the first layer, 128 nodes in the second layer and one output node.
   - We desire an output with two values and the sigmoid has extreme values 0 (for negative activation) and 1 (for positive activation), so that the sigmoid function is a reasonable choice. However, since the sigmoid output is between 0 and 1, it is convenient that the desired classification is not stored as 1 or 2, but 0 or 1. You can then go back to classification 1 and 2 in the plots (by adding 1 to the results).
   - The neural network output generated by the sigmoid function is a number between 0 and 1; if the objective function (i.e. the square norm of the error) has been correctly minimized, then the sigmoid output is very close either to 0 or to 1. In order to get exactly 0 or 1, you can use the Numpy function `np.round(x)`

Laboratory #7 [3]

- Correctly set the values of the learning rate and the number of iteration steps of the gradient algorithm (trial and error; start with few iterations and small learning rate, then increase these values)

4. For the case of 16 classes, we need as many output nodes as the classes, i.e. 16 output nodes. Use 64 hidden nodes in the first layer, 32 hidden nodes in the second layer and 16 output nodes. Use the sigmoid activation function in the hidden nodes and the softmax activation function `tf.nn.softmax(a)` for the output nodes. The output of the network for a given patient is a vector $\mathbf{p}$ with 16 elements: ideally you have just one value equal to 1 and 15 values equal to 0, but in general you have numbers between 0 and 1 in the 16 positions. Value $\mathbf{p}(k)$ practically represents the estimated probability that the patient belongs to class k.

Laboratory #7 [4]

- In order to generate a reasonable objective function to minimize, it is necessary to modify the column vector with the classes given by the doctors into a matrix; if patient n belongs to class k, then the n-th row of the matrix will be equal to 15 zeros and a 1 in position k (remember that Numpy indexes start from 0). This matrix has the same size as the matrix at the output of the neural network and it is possible to evaluate the squared distance (`tf.squared_difference`), but it is necessary to use `tf.reduce_sum` twice to get first a vector and then a scalar, which can be subsequently minimized by TensorFlow.
- Select the learning rate and the number of iterations of the gradient algorithm
- Comment the results in your report.